A Fast Algorithm for the Evaluation of Legendre Expansions

B. Alpert and V. Rokhlin

Research Report YALEU/DCS/RR-671
January 1989

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

4

An algorithm is presented for the rapid calculation of the values and coefficients of finite Legendre series. Given an n-term Legendre expansion, the algorithm produces its values at n Chebyshev nodes on the interval [-1,1] for a cost proportional to $n \log n$. Similarly, given the values of a function f at n Chebyshev nodes, the algorithm produces the n-term Legendre expansion of the polynomial of degree $n-1$ that is equal to f at these nodes. The cost of the algorithm is roughly 3 times that of the fast Fourier transform of length n, provided that calculations are performed to single precision accuracy. In double precision, the ratio is approximately 5.5.

The method employed admits far-reaching generalizations and is currently being applied to several other problems.

A Fast Algorithm for the Evaluation of Legendre Expansions

B. Alpert and V. Rokhlin

Research Report YALEU/DCS/RR-671
January 1989

The authors were supported in part by the Office of Naval Research under Grant N00014-86-K-0310 and in part by IBM under grant P00038437.

1 Introduction

Legendre polynomials are widely used in applied mathematics. Their applications include quadratures, approximation theory, solution of partial differential equations, analysis of pseudospectral methods, and several other areas. However, attempts to use Legendre polynomials as a numerical tool (as opposed to an analytical apparatus) tend to meet with a serious difficulty: given a function $f : [-1,1] \to \mathbb{R}$ tabulated at n nodes, it takes order $O(n^2)$ operations to obtain the Legendre expansion of f. Similarly, given an n-term Legendre expansion, at takes order $O(n^2)$ operations to evaluate that expansion at n nodes in $\mathbb{R}$. In other words, unlike the Chebyshev expansion or the Fourier series, the Legendre series does not have a fast transform associated with it. Whenever possible, therefore, the Legendre series is avoided in favor of an expansion for which a fast transform exists. In some cases, this substitution causes relatively little difficulty (for example, in the construction of pseudospectral algorithms for the solution of partial differential equations). In other cases, it cannot be done at all (for example, in the solution of partial differential equations by the separation of variables in the spherically symmetric geometry, where the choice of Legendre polynomials as the set of basis functions is dictated by the mathematics of the problem).

In Orszag [8], a method is proposed for the rapid evaluation of a fairly wide class of eigenfunction transforms. The algorithm is based on the combination of certain analytical considerations with the fast Fourier transform, has the asymptotic CPU time estimate of order $n(\log n)^2/\log\log n$, and becomes faster than the direct (order n^2) algorithm at $n \approx 128$ (in the case of the Legendre series).

In this paper, we present a procedure for the rapid evaluation of a Legendre expansion at Chebyshev nodes on the interval $[-1,1]$, and conversely, for the evaluation of the coefficients of a Legendre expansion from a table of its values at Chebyshev nodes. More specifically, given a function f expressed as Legendre expansion of the form

$$f(t) = \sum_{j=0}^{n-1} \alpha_j \cdot P_j(t), \tag{1}$$

the algorithm evaluates f at the n Chebyshev nodes $t_0, t_1, \ldots, t_{n-1}$ on the interval $[-1,1]$ in order $O(n \log n)$ operations. Similarly, given the values of a function $f : [-1,1] \to \mathbb{R}$ tabulated at the nodes $t_0, t_1, \ldots, t_{n-1}$, the algorithm requires order $O(n \log n)$ operations to evaluate the coefficients $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ such that

$$f(t_i) = \sum_{j=0}^{n-1} \alpha_j \cdot P_j(t_i) \qquad \text{for } i = 0, 1, \ldots, n-1. \tag{2}$$

The algorithm we present is based on replacing the Legendre expansion of the form (1) with a Chebyshev expansion of the same length, with subsequent evaluation of the latter via the fast cosine transform (see, *e.g.*, [3] Chap. 10). It turns out that the reduction of a Legendre expansion to a Chebyshev expansion can be performed in order $O(n)$ operations, and it is well known that the cosine transform of length n requires order $O(n \log n)$ operations. Thus, the resulting CPU time estimate of our algorithm is $O(n \log n)$.

Remark 1. While the asymptotic CPU time estimate of our scheme is dominated by that of the fast cosine transform it employs, in most practical situations ($n \leq 20,000$), the conversion of the Legendre series of length n into a Chebyshev series tends to be roughly twice as expensive as one fast Fourier transform of length n, provided the calculations are performed in single precision arithmetic. As a result, evaluating the series of the form (1) at n Chebyshev nodes by means of our algorithm is approximately three times as expensive as a single FFT of length n. In double precision arithmetic, this ratio is roughly 5.5 (see Section 6 below).

In the following section, we summarize several well-known facts from approximation theory to be used in the subsequent sections. In Section 3, the analytical properties of the linear mappings connecting the coefficients of Legendre and Chebyshev expansions are studied. Sections 4 and 5 contain the description of the algorithm and its complexity analysis, and in Section 6 results of several numerical experiments are presented. Finally, Section 7 discusses several straightforward generalizations of the algorithm of this paper.

The paper has two appendices. Appendix A presents details of an efficient scheme for the evaluation of the function $\Gamma(x+\frac{1}{2})/\Gamma(x+1)$, required by our algorithm. In Appendix B we sketch an alternative algorithm, also implemented, which is competitive with the main algorithm presented.

Remark 2. The approach of this paper is closely related to that used in [9] to construct an order $O(n)$ scheme for the evaluation of a polynomial of order n at n arbitrary points in $\mathbb{R}$. Both algorithms can be viewed as particular implementations of a scheme for the rapid application to arbitrary vectors of matrices whose entries are sufficiently smooth functions of their indices. Such a general procedure is discussed in Section 7, and will be reported in detail at a later date.

2 Mathematical and Numerical Preliminaries

2.1 Miscellaneous Facts from Approximation Theory

In this section we summarize several well-known facts, the first being the classical error bound for Chebyshev approximations (see, *e.g.*, [4]).

Lemma 1. *Suppose $k \geq 2$, and let t_i denote the i^{th} Chebyshev node of order k on $[0,1]$ and $u_i(t)$ the i^{th} Lagrange polynomial associated with the t_i's, i.e.,*

$$t_i = \frac{1}{2}\left(1+\cos\left(\frac{(2i+1)\pi}{2k}\right)\right), \qquad u_i(t) = \prod_{j=0,j\neq i}^{k-1} \frac{t-t_j}{t_i-t_j}, \tag{3}$$

for $i = 0, 1, \ldots, k-1$. Suppose further that $f : [a,b] \to \mathbb{R}$ is a function with k continuous derivatives, and that the error $E(f,k,[a,b];t)$ of the k-node Chebyshev expansion for f

at a point $t \in [a,b]$ is defined by the formula

$$E(f,k,[a,b];t) = f(t) - \sum_{i=0}^{k-1} u_i\left(\frac{t-a}{b-a}\right)\cdot f(a+(b-a)t_i). \tag{4}$$

Then for any $t \in [a,b]$,

$$|E(f,k,[a,b];t)| \leq \frac{2(b-a)^k}{4^k k!}\sup_{v\in[a,b]}|f^{(k)}(v)|. \tag{5}$$

Throughout the paper we will retain the notation of t_i for Chebyshev nodes, $u_i(t)$ for the corresponding Lagrange polynomials, and $E(f,k,[a,b];t)$ for the interpolation error.

Next we provide a bound on the derivatives of an analytic function, which follows directly from the Cauchy integral formula.

Lemma 2. *Suppose that $D \subset \mathbb{C}$ is a closed disk of radius r centered at $z \in \mathbb{C}$, and that $f : D \to \mathbb{C}$ is a function continuous on D and analytic on its interior $D\backslash\partial D$. Then*

$$|f^{(k)}(z)| \leq \frac{k!}{r^k}\sup_{\theta\in[0,2\pi]}|f(z+re^{i\theta})|. \tag{6}$$

The next lemma provides an expression for the logarithm of the gamma function (see, *e.g.*, [7], p. 10) which we use in Lemma 4 below.

Lemma 3 (Binet). *For any $z \in \mathbb{C}$ such that* $\mathrm{Re}(z) > 0$,

$$\ln\Gamma(z) = (z-1/2)\ln(z) - z + \ln(2\pi)/2 + I(z), \tag{7}$$

where

$$I(z) = \int_0^\infty \left(\frac{1}{e^t-1} - \frac{1}{t} + \frac{1}{2}\right)\frac{1}{t}e^{-tz}dt. \tag{8}$$

Furthermore,

$$|I(z)| \leq \frac{1}{|6z|}. \tag{9}$$

We define a function $\Lambda : \mathbb{C} \to \mathbb{C}$, by the formula

$$\Lambda(z) = \frac{\Gamma(z+1/2)}{\Gamma(z+1)}. \tag{10}$$

The function Λ will often appear in the remainder of the paper. The following lemma states specific bounds on $|\Lambda(z)|$ that we will need in Section 3.

Lemma 4. *Suppose that $z \in \mathbb{C}$ and* $\mathrm{Re}(z) \geq 0$. *Then*

$$\frac{e^{-1/2}}{\sqrt{|z+1|}} \leq |\Lambda(z)| \tag{11}$$

and

$$|\Lambda(z)| \leq \frac{e}{\sqrt{|z+1|}}. \tag{12}$$

Proof. Combining Binet's expression for $\ln\Gamma$ (Eq. 7) with the definition of Λ (Eq. 10), we obtain

$$\ln\Lambda(z) = F(z) + \frac{1}{2}(1 - \ln(z+1)) + Q(z), \tag{13}$$

with the function $F: \mathbb{C} \to \mathbb{C}$ defined by the formula

$$F(z) = z\ln\left(\frac{z+1/2}{z+1}\right), \tag{14}$$

and the function $Q: \mathbb{C} \to \mathbb{C}$ defined by the formula

$$Q(z) = I(z+1/2) - I(z+1), \tag{15}$$

where $I(z)$ is given by Eq. (8). Combining the estimate (9) for $|I(z)|$ with Eq. (15), we see that

$$|\mathrm{Re}(Q(z))| \le \frac{1}{2}, \tag{16}$$

for any z with $\mathrm{Re}(z) \ge 0$, and simple analytical manipulations show that

$$-\frac{1}{2} \le \mathrm{Re}(F(z)) \le 0 \tag{17}$$

for any z with $\mathrm{Re}(z) \ge 0$. Now, combining Eq. (13) with estimates (16) and (17), we obtain

$$-\frac{1}{2} \le \mathrm{Re}(\ln\Lambda(z)) + \frac{1}{2}\ln|z+1| \le 1,$$

from which (11) and (12) follow immediately. ▮

2.2 Definition of Legendre and Chebyshev Polynomials

An orthogonal family of polynomials $\varphi_0, \varphi_1, \varphi_2, \ldots,$ is a set of polynomials of degrees $0, 1, 2, \ldots,$ in which the inner product (φ_i, φ_j) is zero if $i \ne j$ and positive if $i = j$. The inner product is defined by the formula

$$(f,g) = \int_a^b f(t)\, g(t)\, w(t)\, dt,$$

where $[a,b] \subset \mathbb{R}$ and the weight function w is continuous and non-negative on $[a,b]$.

The Legendre and Chebyshev polynomials each form an orthogonal family of polynomials. In each case the inner-product integral is taken over the interval $[-1,1]$; for the Legendre polynomials, the weight function is $w(t) = 1$, while for the Chebyshev polynomials, $w(t) = 1/\sqrt{1-t^2}$.

Any orthogonal polynomial family satisfies a three-term recurrence relation (for $n \ge 0$) of the form

$$\varphi_{-1}(t) = 0, \quad \varphi_0(t) = A_0,$$

$$\frac{A_n}{A_{n+1}}\varphi_{n+1}(t) = \left[t - \frac{(t\varphi_n, \varphi_n)}{(\varphi_n, \varphi_n)}\right]\varphi_n(t) - \frac{(\varphi_n, t\varphi_{n-1})}{(\varphi_{n-1}, \varphi_{n-1})}\varphi_{n-1}(t) \tag{18}$$

where $A_n \neq 0$ is the leading coefficient of $\varphi_n(t)$ and can be chosen arbitrarily. For the Legendre polynomials $P_0, P_1, P_2, \ldots$, Eq. (18) takes the form

$$P_0(t) = 1, \quad P_1(t) = t,$$

$$P_{n+1}(t) = \frac{2n+1}{n+1} t\, P_n(t) - \frac{n}{n+1} P_{n-1}(t), \qquad (n \geq 1). \tag{19}$$

For the Chebyshev polynomials $T_0, T_1, T_2, \ldots$, Eq. (18) becomes

$$T_0(t) = 1, \quad T_1(t) = t,$$

$$T_{n+1}(t) = 2t\, T_n(t) - T_{n-1}(t), \qquad (n \geq 1).$$

In addition to the above recurrences which define P_n and T_n ($n = 0, 1, 2, \ldots$), equivalent closed-form expressions are available. The Legendre polynomials can be given by the equation

$$P_n(t) = \frac{1}{2^n n!} \frac{d^n}{dt^n} (t^2 - 1)^n, \qquad (n \geq 0)$$

and the Chebyshev polynomials by the equation

$$T_n(t) = \cos(n \arccos t), \qquad (n \geq 0).$$

Everything in this section can be found in standard texts (see, *e.g.*, [4] Sec. 4.4).

2.3 Connection between Legendre and Chebyshev Expansions

We will denote by M^n, L^n a pair of $n \times n$-matrices defined by the formulae

$$M^n_{ij} = \begin{cases} \frac{1}{\pi} \Lambda\left(\frac{j}{2}\right)^2 & \text{if } 0 = i \leq j < n \text{ and } j \text{ is even} \\ \frac{2}{\pi} \Lambda\left(\frac{j-i}{2}\right) \Lambda\left(\frac{j+i}{2}\right) & \text{if } 0 < i \leq j < n \text{ and } i + j \text{ is even} \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

$$L^n_{ij} = \begin{cases} 1 & \text{if } i = j = 0 \\ \frac{\sqrt{\pi}}{2\Lambda(i)} & \text{if } 0 < i = j < n \\ \frac{-j(i+1/2)}{(j+i+1)(j-i)} \Lambda\left(\frac{j-i-2}{2}\right) \Lambda\left(\frac{j+i-1}{2}\right) & \text{if } 0 \leq i < j < n \text{ and } i + j \text{ is even} \\ 0 & \text{otherwise,} \end{cases} \tag{21}$$

with Λ defined by Eq. (10).

Remark 3. While Eqs. (20) and (21) define M^n_{ij} and L^n_{ij} for integer values of i, j, it is apparent from Eq. (10) that M^n and L^n can be naturally viewed as functions on $\mathbb{C} \times \mathbb{C}$, and we define $\mathcal{M}, \mathcal{L} : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ by the formulae

$$\mathcal{M}(x, y) = \frac{2}{\pi} \Lambda\left(\frac{y - x}{2}\right) \Lambda\left(\frac{y + x}{2}\right) \tag{22}$$

$$\mathcal{L}(x, y) = \frac{-y(x + 1/2)}{(y + x + 1)(y - x)} \Lambda\left(\frac{y - x - 2}{2}\right) \Lambda\left(\frac{y + x - 1}{2}\right). \tag{23}$$

Clearly, $M^n_{ij} = \mathcal{M}(i,j)$ if $0 < i \leq j < n$ and $i+j$ is even. Similarly, $L^n_{ij} = \mathcal{L}(i,j)$ if $0 \leq i < j < n$ and $i+j$ is even. It easily follows from the well-known properties of the Γ-function (see, *e.g.,* [1]) that $\mathcal{M}$, $\mathcal{L}$ are meromorphic functions of each of their arguments with the poles of $\mathcal{M}$ located at the points $y = \pm x - 1, \pm x - 3, \pm x - 5, \ldots$ and the poles of $\mathcal{L}$ at the points $y = x+1, -x, x-1, -x-2, \ldots$.

The matrices M^n, L^n are inverses of each other; their definition is motivated by the following well-known fact (see, *e.g.,* [5], §8.91-2):

Lemma 5. *Suppose that the function $f: [-1,1] \to \mathbb{R}$ has a finite Legendre expansion of the form*

$$f(\cos\theta) = \sum_{i=0}^{n-1} \alpha_i \cdot P_i(\cos\theta). \tag{24}$$

Then it also has a finite Chebyshev expansion of the form

$$f(\cos\theta) = \sum_{i=0}^{n-1} \beta_i \cdot T_i(\cos\theta), \tag{25}$$

where $\vec{\alpha} = \langle \alpha_0, \ldots, \alpha_{n-1} \rangle$ and $\vec{\beta} = \langle \beta_0, \ldots, \beta_{n-1} \rangle$ are related by the equation

$$\vec{\beta} = M^n \vec{\alpha}. \tag{26}$$

Conversely, if f is a function given by Eq. (25), then it may be expressed in the form of Eq. (24), where $\vec{\alpha}$ is given by

$$\vec{\alpha} = L^n \vec{\beta}. \tag{27}$$

3 Analytical Properties of the Mappings $\mathcal{M}$, $\mathcal{L}$

Definition 1. *Suppose that a square $S \subset \mathbb{R} \times \mathbb{R}$ is defined by the formula $S = [x_0, x_0+a] \times [y_0, y_0+a]$, where $a > 0$. We will say that S is separated from the diagonal if $y_0 - x_0 \geq 2a$.*

The following theorem is the principal analytical tool of this paper. It states that on any square separated from the diagonal, the entries of M^n and L^n are well approximated by Chebyshev expansions of the functions $\mathcal{M}, \mathcal{L}$ with respect to either the first or the second coordinate. For any $\tilde{x} \in \mathbb{C}$ we will define a pair of functions $\mathcal{M}_{\tilde{x}}$ and $\mathcal{L}_{\tilde{x}}$ by the formulae

$$\mathcal{M}_{\tilde{x}}(y) = \mathcal{M}(\tilde{x}, y)$$

$$\mathcal{L}_{\tilde{x}}(y) = \frac{\mathcal{L}(\tilde{x}, y)}{\tilde{x} + \frac{1}{2}}$$

for all $y \in \mathbb{C}$. Similarly, for any $\tilde{y} \in \mathbb{C}$ we will define the functions $\mathcal{M}^{\tilde{y}}$ and $\mathcal{L}^{\tilde{y}}$ by the formulae

$$\mathcal{M}^{\tilde{y}}(x) = \mathcal{M}(x, \tilde{y})$$

$$\mathcal{L}^{\tilde{y}}(x) = \frac{\mathcal{L}(x, \tilde{y})}{x + \frac{1}{2}}$$

for all $x \in \mathbb{C}$.

Theorem 1. *Suppose the square $S = [x_0, x_0 + a] \times [y_0, y_0 + a]$, $a \geq 2$, is separated from the diagonal and $(x, y) \in S$. Then*

$$|E(\mathcal{M}^y, k, [x_0, x_0 + a]; x)| \leq \frac{8e^3}{3^k} |\mathcal{M}(x, y)| \tag{28}$$

and

$$|E(\mathcal{M}_x, k, [y_0, y_0 + a]; y)| \leq \frac{8e^3}{3^k} |\mathcal{M}(x, y)|. \tag{29}$$

Similarly,

$$|E(\mathcal{L}^y, k, [x_0, x_0 + a]; x)| \leq \frac{128e^3}{3^k} \left| \frac{\mathcal{L}(x, y)}{x + \frac{1}{2}} \right| \tag{30}$$

and

$$|E(\mathcal{L}_x, k, [y_0, y_0 + a]; y)| \leq \frac{224e^3}{3^k} \left| \frac{\mathcal{L}(x, y)}{x + \frac{1}{2}} \right|, \tag{31}$$

where E is the error for Chebyshev interpolation, as defined in Eq. (4).

Proof. We will prove here only the estimate (28), since the proofs of all four statements (28), (29), (30), (31) are quite similar. In order to prove (28), we will prove two inequalities

$$|E(\mathcal{M}^y, k, [x_0, x_0 + a]; x)| \leq \frac{32e^2/\pi}{3^k \sqrt{(y - x + 2)(y + x + 2)}} \tag{32}$$

$$|\mathcal{M}(x, y)| \geq \frac{4e^{-1}/\pi}{\sqrt{(y - x + 2)(y + x + 2)}}, \tag{33}$$

for all $(x, y) \in S$. and observe that estimate (28) is an immediate consequence of (32) and (33).

a) Proof of inequality (32). In order to apply the error estimate (5) of Lemma 1 to the function $\mathcal{M}^y$, we will need a bound on the derivatives of $\mathcal{M}^y$. First, we establish a bound on $\mathcal{M}^y(x + \frac{3}{4}(y - x)e^{i\theta})$ for any $(x, y) \in S$ and $\theta \in [0, 2\pi]$. It immediately follows from the definition of $\mathcal{M}$ (Eq. 22) combined with estimate (12) that

$$\begin{aligned} \left|\mathcal{M}^y \left(x + \frac{3}{4}(y - x)e^{i\theta}\right)\right| &= \left|\mathcal{M}\left(x + \frac{3}{4}(y - x)e^{i\theta}, y\right)\right| \\ &= \frac{2}{\pi} \left|\Lambda\left(\frac{y - x}{2} - \frac{3}{8}(y - x)e^{i\theta}\right)\right| \cdot \left|\Lambda\left(\frac{y + x}{2} + \frac{3}{8}(y - x)e^{i\theta}\right)\right| \\ &\leq \frac{2}{\pi} \frac{e^2}{\sqrt{[(y - x)/8 + 1] \cdot [(y + 7x)/8 + 1]}} \end{aligned} \tag{34}$$

for any $\theta \in [0, 2\pi]$. Combining estimate (34) with the Cauchy integral estimate (6) and noting that $y - x \geq a$, we obtain

$$\left| \frac{\partial^k \mathcal{M}^y}{\partial x^k}(x) \right| \leq \frac{k!}{(3a/4)^k} \frac{16e^2/\pi}{\sqrt{(y - x + 2)(x + y + 2)}}. \tag{35}$$

Now (32) follows from a combination of (35) with estimate (5).

b) Proof of inequality (33). The lower bound on $\mathcal{M}$ is easily established by combining the definition of $\mathcal{M}$ (Eq. 22) with estimate (11), which yields

$$|\mathcal{M}(x,y)| = \frac{2}{\pi}\left|\Lambda\left(\frac{y-x}{2}\right)\right| \cdot \left|\Lambda\left(\frac{y+x}{2}\right)\right| \geq \frac{2}{\pi}\frac{e^{-1}}{\sqrt{[(y-x)/2+1]\cdot[(y+x)/2+1]}},$$

for any $(x,y) \in S$, and inequality (33) immediately follows. ▮

Remark 4. Estimates (28), (29), (30), and (31) in Theorem 1 are quite pessimistic. Numerical experiments indicate that the errors in those estimates all decay approximately as 5^{-k}, as opposed to 3^{-k}. In fact, 8-point Chebyshev expansions will approximate $\mathcal{M}$ and $\mathcal{L}$ with roughly single precision accuracy (7 digits) on any square separated from the diagonal. Similarly, double precision (16 digits) is achieved with 18-point expansions. For our purposes, however, the estimates of Theorem 1 are adequate.

4 Informal Description of the Algorithm

We now define a concept closely analogous to the separation of a square from the diagonal.

Definition 2. *Suppose A is an upper-triangular $n \times n$-matrix with entries $\{A_{ij}\}$, $i,j = 0,1,\ldots,n-1$. Suppose further that T is an $m \times m$-submatrix of A defined by the formula*

$$T_{ij} = A_{p+i,q+j}$$

with p,q two non-negative integers. We will say that the submatrix T of matrix A is separated from the diagonal if

$$q - p \geq 2m.$$

4.1 A Simple Example

Suppose that T is an $m \times m$-submatrix of matrix M^n, and that T is separated from the diagonal. We present a simple example of how the smoothness of M^n enables us to efficiently compute $\vec{w} = T\vec{v}$, where $\vec{v} = \langle v_0, \ldots, v_{m-1}\rangle$ is an arbitrary vector of length m. To compute

$$w_i = \sum_{j=0}^{m-1} \mathcal{M}(i_0+i, j_0+j)\cdot v_j \qquad \text{for } i = 0,\ldots,m-1, \tag{36}$$

we may approximate $\mathcal{M}(i_0+i, j_0+j)$ by its Chebyshev expansion in j. We have

$$\mathcal{M}(i_0+i, j_0+j) \approx \sum_{r=0}^{k-1} \mathcal{M}(i_0+i, j_0+t_r m)\cdot u_r\left(\frac{j}{m}\right), \tag{37}$$

where we know from Theorem 1 that as k grows, the error of this approximation shrinks as 3^{-k}. Substituting (37) into Eq. (36) and changing the order of summation, we obtain

$$w_i \approx \sum_{r=0}^{k-1} \mathcal{M}(i_0+i, j_0+t_r m) \sum_{j=0}^{m-1} u_r\left(\frac{j}{m}\right) \cdot v_j = \sum_{r=0}^{k-1} \mathcal{M}(i_0+i, j_0+t_r m) \cdot b_r$$

where $b_r = \sum_{j=0}^{m-1} u_r(j/m) \cdot v_j$ $(r = 0, 1, \ldots, k-1)$.

The number of operations required to evaluate $\vec{w}$ in this manner is $O(km)$. Indeed, evaluating $b_0, \ldots, b_{k-1}$ requires order $O(km)$ operations (k coefficients at m operations per coefficient). Evaluating the vector $\vec{w}$ given the coefficients $b_0, \ldots, b_{k-1}$ is also an order $O(km)$ procedure (evaluating a k-term expansion at m nodes). For a fixed precision ϵ, the number k of Chebyshev nodes required is $\log_3 \frac{1}{\epsilon}$, and is independent of m. Thus the cost of the evaluation of $\vec{w} = T\vec{v}$ has been reduced from order $O(m^2)$ to $O(m \log \frac{1}{\epsilon})$.

This example represents only a part of the computation required to apply matrix M^n to an arbitrary vector, since the submatrix T is assumed to be separated from the diagonal. The actual algorithm is constructed by extending the above example in order to apply the entire matrix. The matrix M^n can be divided into square submatrices, each of which is separated from the diagonal (Figure 1). To apply the matrix M^n to the vector

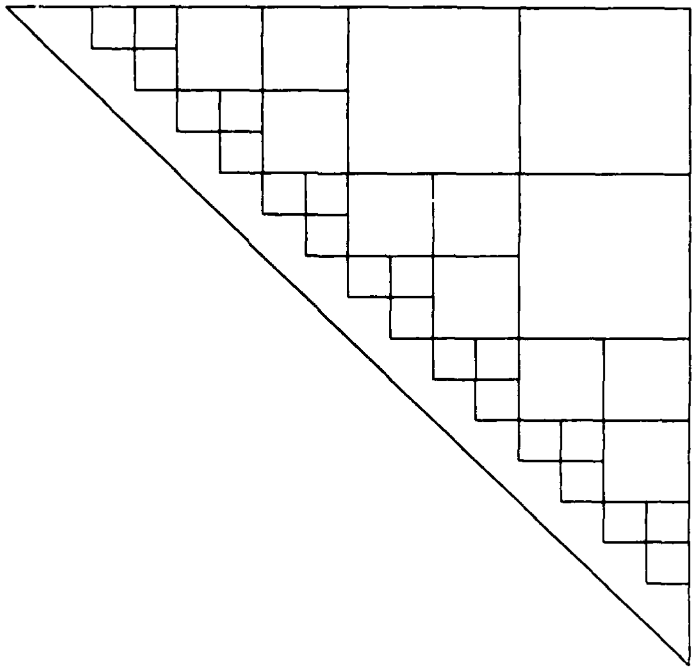

Figure 1: Each submatrix T in the subdivision of M^n is separated from the diagonal (see Definition 2). Here the subdivision is shown to three levels.

$\vec{\alpha}$, each submatrix T is applied separately, as suggested in the example. The remaining undivided portion of M^n near the diagonal is applied directly. Before we can introduce the algorithm, however, we will need some additional notation.

4.2 Notation

The algorithm to be described will input an arbitrary vector $\vec{\alpha} = \langle \alpha_0, \ldots, \alpha_{n-1} \rangle$ and compute an approximation $\vec{\gamma} = \langle \gamma_0, \ldots, \gamma_{n-1} \rangle$ to the vector $\vec{\beta} = M^n \vec{\alpha}$. Theorem 1 will be used to ensure the desired quality of approximation.

Suppose that s is a positive integer such that $h = \log_2(n/s) - 1$ is also a positive integer. For any integer $l \in \{1, \ldots, h\}$ and $j \in \{0, \ldots, n/(2^{l-1}s) - 1\}$, we define the interval $I_{l,j} \subset \mathbb{R}$ by the formula

$$I_{l,j} = [j \cdot 2^{l-1}s, (j+1) \cdot 2^{l-1}s].$$

For any integer $l \in \{1, \ldots, h\}$, $i \in \{0, \ldots, n/(2^{l-1}s) - 3\}$, and $j \in \{i+2, i+3\}$ (for i even), $j = i + 2$ (for i odd) we define the square ${}_lS_{i,j} \subset \mathbb{R}^2$ by the formula

$${}_lS_{i,j} = I_{l,i} \times I_{l,j}.$$

The definition of the squares ${}_lS_{i,j}$ is illustrated in Figure 2.

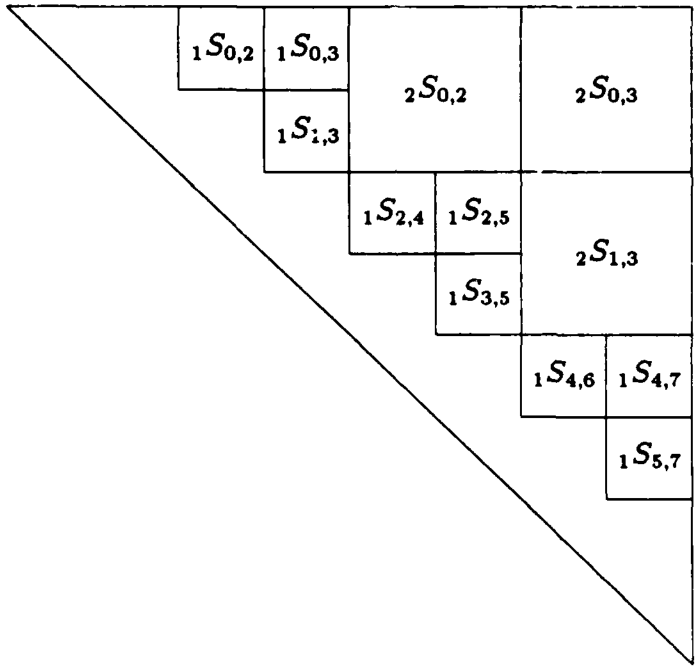

Figure 2: The upper triangle of the square $[0, n] \times [0, n]$ is subdivided into squares, each of which is separated from the diagonal (Definition 1). Here the number h of levels is equal to 2.

For $l \in \{1, \ldots, h\}$, $m \in \{0, \ldots, 2^{l-1}s - 1\}$, and $r \in \{0, \ldots, k-1\}$, we define the Chebyshev interpolation coefficient $u^l_{r,m}$ by the formula

$$u^l_{r,m} = u_r \left(\frac{m}{2^{l-1}s} \right), \tag{38}$$

where u_r is given by Eq. (3). For each interval $I_{l,j}$ and $r \in \{0, \ldots, k-1\}$ we define the coefficient $b^r_{l,j}$ by the formula

$$b^r_{l,j} = \sum_{m=0}^{2^{l-1}s-1} u^l_{r,m} \cdot \alpha_{m+j\cdot 2^{l-1}s} \tag{39}$$

and observe that the definition of $b^r_{l,j}$ is analogous to the definition of b_r in the example above. For each square ${}_lS_{i,j}$ we define a $k \times k$-matrix ${}_l\mathcal{M}_{i,j}$ of which each element ${}_l\mathcal{M}^{r,m}_{i,j}$ $(r, m \in \{0, \ldots, k-1\})$ is given by the formula

$${}_l\mathcal{M}^{r,m}_{i,j} = \mathcal{M}((i + t_r) \cdot 2^{l-1}s, (j + t_m) \cdot 2^{l-1}s). \tag{40}$$

We further define, for each square ${}_lS_{i,j}$ and $r \in \{0, \ldots, k-1\}$, the coefficient ${}_lc^r_{i,j}$ by the formula

$${}_lc^r_{i,j} = \sum_{m=0}^{k-1} {}_l\mathcal{M}^{r,m}_{i,j} \cdot b^m_{l,j}.$$

For each interval $I_{l,j}$ and $r \in \{0, \ldots, k-1\}$, we define the coefficient $c^r_{l,j}$ by the formulae

$$\begin{array}{rcll} c^r_{l,2j} & = & {}_lc^r_{2j,2j+2} + {}_lc^r_{2j,2j+3} & (j \le n/(2^l s) - 2) \\ c^r_{l,2j+1} & = & {}_lc^r_{2j+1,2j+3} & \\ c^r_{l,n/(2^{l-1}s)-2} & = & c^r_{l,n/(2^{l-1}s)-1} \;=\; 0 & \end{array} \tag{41}$$

For each interval $I_{l,j}$ and $m \in \{0, \ldots, 2^{l-1}s-1\}$ we define the coefficient $a^m_{l,j}$ by the formula

$$a^m_{l,j} = \sum_{r=0}^{k-1} u^l_{r,m} \cdot c^r_{l,j}.$$

Finally, for $i \in \{0, \ldots, n-1\}$, we define γ_i by the formula

$$\gamma_i = \sum_{l=1}^{\lceil \log_2(\frac{n-i}{2s}) \rceil} a^{m_{l,i}}_{l,j_{l,i}} + \sum_{r=i}^{\lfloor i/s \rfloor s + 2s - 1} M^n_{ir} \cdot \alpha_r, \tag{42}$$

where $j_{l,i} = \lfloor i/(2^{l-1}s) \rfloor$ and $m_{l,i} = i \bmod (2^{l-1}s)$.

The notation introduced so far allows us to extend the example of Section 4.1 to apply the entire matrix M^n to an arbitrary vector. The simplest algorithm, which is described in the next section, requires order $O(n \log n)$ operations. We have introduced, however, one change from the example of Section 4.1: the values of $\mathcal{M}$ are interpolated with respect to both the first and the second coordinates, rather than just the second coordinate. This change allows us to construct an algorithm requiring order $O(n)$ operations. This latter algorithm is described in Section 4.3, and will require some additional notation.

We note that for any integer $l \in \{2, \ldots, h\}$, $m \in \{0, \ldots, 2^{l-1}s-1\}$, and $r \in \{0, \ldots, k-1\}$, the Chebyshev interpolation coefficient $u^l_{r,m}$ defined in Eq. (38) can be equivalently given by the formula

$$u^l_{r,m} = \sum_{i=0}^{k-1} u_r\left(\frac{t_i}{2}\right) \cdot u^{l-1}_{i,m}. \tag{43}$$

Remark 5. Eq. (43) is an instance of the more general formula

$$u_r(t) = \sum_{i=0}^{k-1} u_r\left(\frac{t_i}{2}\right) \cdot u_i(2t) \quad \text{for any } t \in \mathbb{R}, \tag{44}$$

which immediately follows from the observation that the summation in Eq. (44) is the $(k-1)^{\text{st}}$-degree interpolating polynomial which agrees with the function u_r at points $t_0/2, t_1/2, \ldots, t_{k-1}/2$, and that u_r itself is a polynomial of degree $k-1$ (see Eq. 3).

Eq. (43) combined with Eq. (39) produces a recursive expression for $b^r_{l,j}$ given by the formulae

$$b^r_{1,j} = \sum_{i=0}^{s-1} u_r\left(\frac{i}{s}\right) \cdot \alpha_{i+js} \tag{45}$$

$$b^r_{l,j} = \sum_{i=0}^{k-1} \left[u_r\left(\frac{t_i}{2}\right) \cdot b^i_{l-1,2j} + u_r\left(\frac{1+t_i}{2}\right) \cdot b^i_{l-1,2j+1} \right] \quad (l \in \{2, \ldots, h\}). \tag{46}$$

Following a similar procedure, for each $I_{l,j}$ and $r \in \{0, \ldots, k-1\}$, we recursively define the coefficient $d^r_{l,j}$ by the formulae

$$\begin{aligned} d^r_{h,j} &= c^r_{h,j} \\ d^r_{l,2j} &= c^r_{l,2j} + \sum_{i=0}^{k-1} u_i\left(\frac{t_r}{2}\right) \cdot d^i_{l+1,j} \quad (l \in \{1, \ldots, h-1\}) \\ d^r_{l,2j+1} &= c^r_{l,2j+1} + \sum_{i=0}^{k-1} u_i\left(\frac{1+t_r}{2}\right) \cdot d^i_{l+1,j} \end{aligned} \tag{47}$$

where $c^r_{l,j}$ is given by Eq. (41). Finally, combining Eqs. (45), (46), and (47), we see that the vector $\vec{\gamma}$, defined by Eq. (42), can be equivalently given by the formula

$$\gamma_{m+js} = \sum_{i=0}^{k-1} u_i\left(\frac{m}{s}\right) \cdot d^i_{1,j} + \sum_{i=m}^{2s-1} M^n_{m+js,i+js} \cdot \alpha_{i+js} \quad \text{for} \quad \begin{array}{l} m = 0, \ldots, s-1, \\ j = 0, \ldots, n/s-1. \end{array} \tag{48}$$

4.3 Description of an O($n \log n$) Algorithm

A simple algorithm for the application of the matrix M^n to an arbitrary vector $\vec{\alpha} = \langle \alpha_0, \ldots, \alpha_{n-1} \rangle$ can be constructed by combining the procedure of Section 4.1 and the subdivision of M^n shown in Figures 1 and 2. The procedure of Section 4.1 works only for a submatrix which is separated from the diagonal. The matrix M^n can be divided, however, into a collection of submatrices, each of which is separated from the diagonal, plus an undivided part near the diagonal (see Figure 1). By applying the scheme of Section 4.1 to each submatrix, we immediately obtain an order O($n \log n$) algorithm for the application of the matrix M^n (or L^n) to $\vec{\alpha}$.

4.4 Description of an $O(n)$ Algorithm

Now we describe an order $O(n)$ algorithm for the application of the matrix M^n (or matrix L^n) to an arbitrary vector. The matrix M^n is divided into submatrices, each of which is separated from the diagonal. The scheme of such a subdivision is shown in Figure 1. There are 3 squares of side length $n/4$, $3 \cdot 3$ of side length $n/8$, $3 \cdot 7$ of side length $n/16$, and so forth down to $3 \cdot (2^h - 1)$ squares of side length $s = n/2^{1+h}$. The size $s \times s$ of the smallest squares in the subdivision is fixed, independent of n. For each square the separation condition of Theorem 1 holds.

The most direct application of Chebyshev expansions, to each square independently, leads to an order $O(n \log n)$ algorithm. This operation count is due to order $O(n)$ operations for all squares of one size, multiplied by $O(\log n)$ different sizes. The undivided part of the matrix near the diagonal is handled directly in order $O(n)$ operations.

We improve on this simple method by using in each square the Chebyshev expansion for M^n in both row and column directions. We then "gather" coefficients used in each row interval up from those for the next smaller intervals, compute $k \times k$ matrix-vector products, then "spread" the results down to the next smaller column intervals. To describe this procedure, we employ the notation introduced in Section 4.2. The coefficients $b_{l,j}^r$ are computed from level 1 to level h according to Eqs. (45) and (46). Then the matrix-vector products ${}_ic_{i,j}^r$ are computed and summarized to the values $c_{l,j}^r$ (Eq. 41). These values are used to compute the coefficients $d_{l,j}^r$ from level h to level 1, as specified in Eq. (47). Finally, the vector $\vec{\gamma}$ is computed according to Eq. (48).

This incremental computation of the coefficients $b_{l,j}^r$ and the coefficients $d_{l,j}^r$ leads to the reduction in complexity from order $O(n \log n)$ to $O(n)$ operations. Now instead of order $O(n)$ operations for all squares of one size, we expend order $O(n)$ operations for all squares of the smallest size, half as many for the next larger size, one fourth as many for the second larger size, and so forth. The sum of these operation counts remains order $O(n)$.

We divide the computation into an initialization phase, which is independent of $\vec{\alpha} = \langle \alpha_0, \ldots, \alpha_{n-1} \rangle$, depending only on n, and an evaluation phase, which does the rest of the work. This partitioning of the algorithm leads to substantial savings when one computes many Legendre transformations of the same size.

In the initialization phase, the values of ${}_i\mathcal{M}_{i,j}^{r,m}$ for each square, as defined in Eq. (40), are computed and stored; the near-diagonal values of $\mathcal{M}$, appearing in Eq. (48), are also computed. Details of an efficient scheme for the evaluation of $\mathcal{M}$ are contained in Appendix A. The values of u_r appearing in Eqs. (45) and (46) are also computed during the initialization phase, and later used in the evaluation phase. The evaluation phase consists of using the stored values of $\mathcal{M}$ and u_r in computing, in succession, $b_{l,j}^r$, $c_{l,j}^r$, $d_{l,j}^r$, and γ_i, as described above.

5 Detailed Description and Complexity Analysis of the Algorithm

5.1 Description of the Algorithm

Initialization Phase

Comment [Input to this phase is the number of values n].
Set the number of Chebyshev nodes per interval $k \approx \log(1/\epsilon)$,
where ϵ is the desired precision. Set the smallest interval
size $s \approx 4k$. (See Section 5.2, below.)

Step 1.

Comment [Construct Chebyshev nodes $t_0, t_1, \ldots, t_{k-1}$ on the interval $[0,1]$, Chebyshev nodes $t'_0, \ldots, t'_{k-1}$ on the interval $[0, \frac{1}{2}]$, and Chebyshev nodes $t'_k, \ldots, t'_{2k-1}$ on the interval $[\frac{1}{2}, 1]$.]

```
do $r = 0, \ldots, k-1$
    set $t_r = [1 - \cos((r + .5)\pi/k)]/2$
    set $t'_r = t_r/2$ and $t'_{k+r} = (1 + t_r)/2$
enddo
```

Step 2.

Comment [Evaluate the denominators in the expressions for the Chebyshev interpolation coefficients $u_0, u_1, \ldots, u_{k-1}$.]

```
do $r = 0, \ldots, k-1$
    set $den_r = \prod_{l=0, l\neq r}^{k-1}(t_r - t_l)$
enddo
```

Comment [Evaluate the Chebyshev interpolation coefficients $u_0, u_1, \ldots, u_{k-1}$ at the uniformly spaced nodes $0, \frac{1}{s}, \frac{2}{s}, \ldots, \frac{s-1}{s}$.]

```
do $l = 0, \ldots, s-1$
    set $x = \prod_{r=0}^{k-1}(l/s - t_r)$
    do $r = 0, \ldots, k-1$
        set $u_r(l/s) = [x/(l/s - t_r)]/den_r$
    enddo
enddo
```

Comment [Evaluate the Chebyshev interpolation coefficients $u_0, u_1, \ldots, u_{k-1}$ at the Chebyshev nodes $t'_0, \ldots, t'_{2k-1}$.]

```
do $l = 0, \ldots, 2k-1$
    set $x = \prod_{r=0}^{k-1}(t'_l - t_r)$
    do $r = 0, \ldots, k-1$
        set $u_r(t'_l) = [x/(t'_l - t_r)]/den_r$
    enddo
enddo
```

Step 3.

```
Comment [Evaluate the values ${}_l\mathcal{M}^{r,m}_{i,j}$ of the $k \times k$
matrix on each square ${}_lS_{i,j}$ of the subdivision.]
$h = \log_2(n/s) - 1$
do $l = 1, \ldots, h$
    do $i = 0, \ldots, n/(2^{l-1}s) - 3$
        do $j = i + 2, \ldots, i + 3 - i \bmod 2$
            do $r = 0, \ldots, k - 1$
                do $m = 0, \ldots, k - 1$
                    Calculate ${}_l\mathcal{M}^{r,m}_{i,j}$ according to Eq. (40).
                enddo
            enddo
        enddo
    enddo
enddo
```

Step 4.

```
Comment [Evaluate $\mathcal{M}$ in the undivided part of the matrix,
near the diagonal.]
do $j = 0, \ldots, n/s - 1$
    do $m = 0, \ldots, s - 1$
        set $p = 2s - 2 + m \bmod 2$
        do $i = m, m + 2, m + 4, \ldots, p$
            Calculate $\mathcal{M}(m + js, i + js)$ using Eq. (22).
        enddo
    enddo
enddo
End of Initialization Phase
```

Evaluation Phase

Comment [Input to this phase is $\vec{\alpha} = \langle \alpha_0, \ldots, \alpha_{n-1} \rangle$.]

Step 5.

```
Comment [Evaluate the coefficients $b^r_{1,j}$ from the input
vector $\langle \alpha_0, \ldots, \alpha_{n-1} \rangle$ and the interpolation coefficients $u_r(i/s)$.]
do $j = 0, \ldots, n/s - 1$
    do $r = 0, \ldots, k - 1$
        set $b^r_{1,j} = \sum_{i=0}^{s-1} u_r(i/s) \cdot \alpha_{i+js}$
    enddo
enddo
```

Step 6.

```
Comment [Evaluate the coefficients $b^r_{l,j}$ for $l \geq 2$, which
correspond to larger interval sizes, from the coefficients for
smaller interval sizes and the interpolation coefficients $u_r(t'_i)$.]
do $l = 2, \ldots, h$
    do $j = 0, \ldots, n/(2^{l-1}s) - 1$
        do $r = 0, \ldots, k-1$
            set $b^r_{l,j} = \sum_{i=0}^{k-1} \left[u_r(t'_i) \cdot b^i_{l-1,2j} + u_r(t'_{k+i}) \cdot b^i_{l-1,2j+1}\right]$
        enddo
    enddo
enddo
```

Step 7.

```
Comment [Evaluate the coefficients $c^r_{l,j}$ from the
values ${}_l\mathcal{M}^{r,m}_{i,j}$ and the coefficients $b^r_{l,j}$.]
do $l = 1, \ldots, h$
    do $j = 0, \ldots, n/(2^l s) - 2$
        do $r = 0, \ldots, k-1$
            set $c^r_{l,2j} = \sum_{i=0}^{k-1} \left[{}_l\mathcal{M}^{r,i}_{2j,2j+2} \cdot b^i_{l,2j+2} + {}_l\mathcal{M}^{r,i}_{2j,2j+3} \cdot b^i_{l,2j+3}\right]$
            set $c^r_{l,2j+1} = \sum_{i=0}^{k-1} {}_l\mathcal{M}^{r,i}_{2j+1,2j+3} \cdot b^i_{l,2j+3}$
        enddo
    enddo
enddo
```

Step 8.

```
Comment [Evaluate the coefficients $d^r_{l,j}$ from the
coefficents $c^r_{l,j}$ and the interpolation coefficients $u_r(t'_i)$.]
do $l = h, h-1, \ldots, 1$
    do $j = 0, \ldots, n/(2^{l-1}s) - 2$
        do $r = 0, \ldots, k-1$
            set $d^r_{l,2j} = c^r_{l,2j} + \sum_{i=0}^{k-1} u_i(t'_r) \cdot d^i_{l+1,j}$
            set $d^r_{l,2j+1} = c^r_{l,2j+1} + \sum_{i=0}^{k-1} u_i(t'_{k+r}) \cdot d^i_{l+1,j}$
        enddo
    enddo
enddo
```

```
                    Step 9.
Comment [Evaluate the final result $\vec{\gamma} = \langle \gamma_0, \ldots, \gamma_{n-1} \rangle$ from
the coefficients $d^r_{1,j}$, the interpolation coefficients $u_r(m/s)$, the
values of $\mathcal{M}$ near the diagonal, and the input vector $\vec{\alpha}$.]
do $j = 0, \ldots, n/s - 1$
    do $m = 0, \ldots, s - 1$
        set $\gamma_{m+js} = \sum_{i=0}^{k-1} u_i(m/s) \cdot d^i_{1,j}$
        set $p = 2s - 2 + m \bmod 2$
        do $i = m, m+2, m+4, \ldots, p$
            set $\gamma_{m+js} = \gamma_{m+js} + \mathcal{M}(m + js, i + js) \cdot \alpha_{i+js}$
        enddo
    enddo
enddo
End of Evaluation Phase
End of Algorithm
```

The algorithm for the opposite direction, namely, computing Legendre expansion coefficients from Chebyshev expansion coefficients, is identical to the algorithm given above, with $\mathcal{L}$ substituted for $\mathcal{M}$.

Remark 6. In Theorem 1, error bounds for Chebyshev expansions in a square separated from the diagonal are given for $\mathcal{L}(x,y)/(x+\frac{1}{2})$, rather than for $\mathcal{L}(x,y)$. In principle, then, the algorithm for computing Legendre coefficients from Chebyshev coefficients should apply the matrix corresponding to $\mathcal{L}(x,y)/(x+\frac{1}{2})$ using the method given above and then apply the diagonal matrix corresponding to $x+\frac{1}{2}$. Doing so would produce an algorithm of the same asymptotic time complexity as that for $\mathcal{M}$, but slightly more expensive. In practice, this alteration produces virtually no improvement in accuracy, so our implementation uses the same algorithm in both directions.

5.2 Complexity Analysis

In the following table, we provide the operation count for each step of the algorithm.

Initialization Phase

Step	Complexity	Explanation
1.	$O(k)$	Chebyshev nodes $t_0, \dots, t_{k-1}$ and $t'_0, \dots, t'_{2k-1}$ (a total of $3k$ nodes) are computed.
2.	$O(k^2 + ks)$	Interpolation coefficients $u_r(l/s)$, for $l = 0, 1, \dots, s-1$ and $u_r(t'_l)$, for $l = 0, \dots, 2k-1$ are computed ($r = 0, \dots, k-1$). The average cost per coefficient is constant.
3.	$O(nk^2/s)$	Matrix entries ${}_l\mathcal{M}^{r,m}_{i,j}$ are computed. There are k^2 of these values per square; there are $\frac{3}{2}(n/s - 2)$ squares of the smallest size, $\frac{3}{2}(n/(2s) - 2)$ squares of the next size, and so forth, up to 3 squares of the largest size. Thus the total number of squares is order $O(n/s)$, at a cost of order $O(k^2)$ per square.
4.	$O(ns)$	Matrix entries $\mathcal{M}(m + js, i + js)$ are computed. There are no more than $2s$ of these values per row of the matrix M^n, and M^n has n rows.
Total	$O(n(s + k^2/s))$	

Evaluation Phase

Step	Complexity	Explanation
5.	$O(nk)$	Coefficients $b^r_{1,j}$, for $j = 0, 1, \dots, n/s - 1$ and $r = 0, \dots, k-1$ are computed, at order $O(s)$ operations each.
6.	$O(nk^2/s)$	Coefficients $b^r_{l,j}$ for $l > 1$ are computed. There are $(n/s - 4) \cdot k$ of these, and each requires $O(k)$ operations to compute.
7.	$O(nk^2/s)$	Coefficients $c^r_{l,j}$ are computed. There are $2[n/s - \log_2(n/s) - 1] \cdot k$ of these, and each is computed in order $O(k)$ operations.
8.	$O(nk^2/s)$	Coefficients $d^r_{l,j}$ are computed. Again there are $2[n/s - \log_2(n/s) - 1] \cdot k$ of these, and each is computed in order $O(k)$ operations.
9.	$O(nk + ns)$	Final results γ_i are computed. There are n of these and each requires order $O(k + s)$ operations to compute.
Total	$O(n(s + k + k^2/s))$	

The total computation time for the initialization phase is

$$t_{\text{init}} = a_1 \cdot k + a_2 \cdot k^2 + a_3 \cdot ks + a_4 \cdot nk^2/s + a_5 \cdot ns$$

and the time for the evaluation phase is

$$t_{\text{eval}} = a_6 \cdot ns + a_7 \cdot nk + a_8 \cdot nk^2/s,$$

where $a_1, a_2, \ldots, a_8$ depend on the implementation, language, and computer system. The length s of the smallest interval, however, is not determined by the problem, and can be chosen arbitrarily. Choosing s to minimize the evaluation time t_{eval} yields

$$s = k\sqrt{a_8/a_6}.$$

For this choice of s, the initialization time t_{init} and evaluation time t_{eval} each are order $O(nk)$. The actual values of k and s used in our implementation are given in the next section, which also includes running times and accuracies.

6 Numerical Results

We have implemented the $O(n)$ algorithm described above. It has been combined with a cosine transform (order $n \log n$) to produce a code converting coefficients of Legendre expansions (of functions in the interval [-1,1]) into values of such functions at Chebyshev nodes, and vice versa. In the forward direction, the algorithm transforms function values at Chebyshev nodes to Legendre expansion coefficients, while in the backward direction, Legendre expansion coefficients are converted to function values at Chebyshev nodes. Here we present the results of applying the algorithm to inputs of varying size, giving accuracies and running times. These results are compared to a direct calculation.

The algorithm has been implemented both in single precision and double precision arithmetic. The number k of Chebyshev nodes per interval was chosen to retain maximum precision. The size s of the smallest interval was chosen to maximize efficiency. For the single precision version, we chose k to be 8 and s to be 32. For double precision, k is 18 and s is 64.

We ran our FORTRAN implementations on a Sun 3/50 equipped with an MC68881 floating-point coprocessor. For comparison, times required to compute a complex FFT on this hardware are also given. All accuracies were determined by comparing to results computed in quadruple precision (on a microVax), using the direct method. The measure of error was chosen to be the L_2-norm:

$$E = \sqrt{\sum_{i=0}^{n-1}(\gamma_i - \lambda_i)^2 / \sum_{i=0}^{n-1}\lambda_i^2},$$

where $\vec{\gamma}$ is the result of the computation being tested and $\vec{\lambda}$ is the result of the quadruple precision computation. Input for the computations in the backward direction (converting from Legendre coefficients to function values) was a vector whose components were independent pseudo-random numbers, each drawn from a distribution uniformly distributed in the interval $[0, 1]$. Input for the forward direction was the function values from the backward quadruple precision computation.

Table 1

Single Precision Computations

Input	Error			Time (sec)			
Size	Algorithm		Direct	Algorithm		Direct	
(n)	Forward	Backward	Method	Initial.	Eval.	Method	FFT
64	0.335e-06	0.133e-06	0.112e-05	0.92	0.07	0.30	0.02
128	0.629e-06	0.107e-06	0.290e-05	2.34	0.17	1.16	0.06
256	0.888e-06	0.117e-06	0.762e-04	5.48	0.47	4.78	0.16
512	0.130e-05	0.167e-06	0.242e-03	11.60	1.07	18.26	0.32
1024	0.236e-05	0.125e-06	0.278e-03	24.24	2.31	73.96	0.72
2048	0.285e-05	0.171e-06	0.396e-02	48.76	4.90	297.56	1.66
4096	0.503e-05	0.262e-06	0.121e-01	100.18	9.84	1168.18	3.48

The direct computation of $\sum \alpha_i \cdot P_i(t)$ was accomplished by applying the three-term recurrence relation for Legendre polynomials (Eq. 19). This procedure produces significant round-off errors for t near -1 and 1, which impairs the accuracy of the direct computation. It is given here primarily for CPU time comparisons.

Table 2

Double Precision Computations

Input	Error			Time (sec)			
Size	Algorithm		Direct	Algorithm		Direct	
(n)	Forward	Backward	Method	Initial.	Eval.	Method	FFT
64	0.152e-14	0.673e-15	0.756e-15	1.86	0.11	0.30	0.02
128	0.201e-14	0.695e-15	0.209e-14	4.56	0.28	1.18	0.08
256	0.312e-14	0.723e-15	0.134e-13	10.80	0.71	4.80	0.16
512	0.495e-14	0.725e-15	0.127e-12	24.32	1.96	19.22	0.36
1024	0.689e-14	0.768e-15	0.455e-12	52.30	4.62	76.92	0.78
2048	0.908e-14	0.787e-15	0.827e-12	108.50	10.12	307.40	1.76
4096	0.139e-13	0.840e-15	0.716e-11	223.00	21.38	1229.28	3.70

Several observations may be made from Tables 1 and 2.

1. The algorithm permits very high accuracy. In single precision the relative error is less than 6×10^{-6} for all input sizes up to 4096. For double precision, the relative error is less than 2×10^{-14}.

2. The error of the algorithm increases roughly as the square root of the length of the input vector.

3. The time required to make the single precision computation is less than 3 times that for an FFT of the same size. For double precision, the ratio is about 5.5 for n in the range we have tabulated. (For larger n, the ratio would be less than 5.5.) The initialization time is roughly 10 times the computation time, for both single and double precision.

4. In single precision computations, for n as low as 64, the time required by the algorithm is less than one-fourth of the time required by the direct method. For $n = 4096$, the speedup is 120-fold. For double precision, the speedups are roughly 3-fold at $n = 64$ and 60-fold at $n = 4096$.

7 Generalizations and Conclusions

An algorithm has been presented for the rapid conversion of the coefficients of a Legendre expansion of a function on the interval $[-1,1]$ into its values at the Chebyshev nodes on that interval, and vice versa. The algorithm requires order $O(n \log n)$ operations to transform an n-term expansion into n function values, or n function values into an n-term expansion.

The method admits several straightforward generalizations.

1. As described in Section 4.4, the scheme requires that n (the number of Chebyshev nodes on the interval $[-1,1]$, and also the length of the Legendre expansion) be a power of 2. Clearly, this is not an essential requirement. A simple modification of the scheme will convert an n-term Chebyshev expansion into an n-term Legendre expansion, and vice-versa, for any positive integer n. On the other hand, the fast Fourier transform used to evaluate the Chebyshev series at the Chebyshev nodes does impose certain algebraic requirements on n. Thus, the scheme of this paper will perform efficiently whenever the FFT does.

2. Clearly, the problem solved by the algorithm of this paper is a particular case of the following problem, regularly encountered in the numerical treatment of equations of mathematical physics.

   Problem 1. Given the coefficients α_{nm} for $n = 0, 1, \ldots, N-1$, and $m = -n, -n+1, \ldots, n$, evaluate the expression

   $$f(\theta,\varphi) = \sum_{n=0}^{N-1} \sum_{m=-n}^{n} \alpha_{nm} \cdot P_n^m(\cos\theta) \cdot e^{im\varphi}, \tag{49}$$

   at the points (θ_i, φ_j), for $i, j = 0, 1, \ldots, N-1$, defined by the formulae

   $$\theta_i = \frac{2i+1}{N}\pi, \qquad \varphi_j = \frac{j}{N}2\pi.$$

   (In Eq. (49), P_n^m denotes the associated Legendre function of degree n and order m, as defined in, *e.g.*, [1].)

The algorithm of this paper admits a generalization that solves Problem 1 in order $O(N^2 \log N))$ operations, and the paper describing such a scheme is in preparation.

3. The heart of this paper is an order $O(n)$ algorithm converting an n-term Legendre expansion of a function into an n-term Chebyshev expansion of that function, and vice-versa. The algorithm is based on the fact that elements of the matrices M^n, L^n connecting the two expansions are a smooth function of the indices, except near the diagonal. In this respect, the scheme of this paper is somewhat similar to those of [9], [6], and several others. A radical generalization of this approach is to observe that any matrix whose elements are a sufficiently smooth non-oscillatory function of their indices can be applied to an arbitrary vector for a cost proportional to n (with a fixed precision). As the matrix becomes less smooth, the procedure becomes less efficient. For matrices singular near the diagonal (such as the matrices M^n, L^n of this paper), the scheme is still of order $O(n)$. The same is true for matrices with singularities along a fixed number of bands of fixed width. It is easy to construct examples of matrices for which the method will be of order $O(n \log n)$, and other examples for which the method will fail completely. An investigation of these issues is in progress, and the authors are aware of at least one other such algorithm [2]. This latter scheme, however, is based on a somewhat different set of techniques.

Appendices

A Numerical Evaluation of the Function Λ

The definitions of the functions $\mathcal{M}$, $\mathcal{L}$ (Eqs. 22 and 23) involve the function Λ, defined as

$$\Lambda(z) = \frac{\Gamma(z+1/2)}{\Gamma(z+1)},$$

for $z \in \mathbb{C}$. The algorithm, described in Sections 4 and 5, actually requires computation of $\Lambda(x)$ for $x \in \mathbb{R}^+$. The function Λ may of course be computed using the Sterling asymptotic expansion for Γ, but for an efficient computation we use the asymptotic formula

$$\Lambda(x) \sim \frac{1}{\sqrt{x}} \qquad \text{as } x \to \infty. \tag{50}$$

For large values of x, therefore, the function $\Lambda(x)\sqrt{x}$ may be well approximated by a polynomial in $1/x$. Defining the change of variable $y = 1/x$, we obtain the approximate formula

$$\frac{\Lambda\left(\frac{1}{y}\right)}{\sqrt{y}} \approx a_0 + a_1 y + \cdots + a_5 y^5,$$

where the coefficients $a_0, a_1, \ldots, a_5$ have been determined by evaluating the function $\Lambda(1/y)/\sqrt{y}$ at Chebyshev nodes on each of four intervals. The values of the coefficients are shown in Table A.1.

Table A.1
Coefficients $a_0, a_1, \ldots, a_5$ in the formula
$\Lambda\left(\frac{1}{y}\right) \approx \sqrt{y}\,[a_0 + a_1 y + \cdots + a_5 y^5]$, for four separate intervals

Coefficient	$y \in (0, .02)$	$y \in (.02, .04)$
a_0	$0.99999999999999999d+00$	$0.99999999999974725d+00$
a_1	$-0.12499999999996888d+00$	$-0.12499999994706490d+00$
a_2	$0.78124999819509772d-02$	$0.78124954632722315d-02$
a_3	$0.48828163023526451d-02$	$0.48830152125039076d-02$
a_4	$-0.64122689844951054d-03$	$-0.64579205161159155d-03$
a_5	$-0.15070098356496836d-02$	$-0.14628616278637035d-02$

Coefficient	$y \in (.04, .058)$	$y \in (.058, .067)$
a_0	$0.99999999999298844d+00$	$0.99999999996378690d+00$
a_1	$-0.12499999914033463d+00$	$-0.12499999657282661d+00$
a_2	$0.78124565447111342d-02$	$0.78123659464717666d-02$
a_3	$0.48839648427432678d-02$	$0.48855685911602214d-02$
a_4	$-0.65752249058053233d-03$	$-0.67176366234107532d-03$
a_5	$-0.14041419931494052d-02$	$-0.13533949520771154d-02$

For small values of x, formula (50) is no longer useful. Fortunately, however, the evaluation of Λ at small values of x, which corresponds to the evaluation of $\mathcal{M}$ and $\mathcal{L}$ near the diagonal, is necessary only at half-integer values of x. In particular, it suffices to obtain Λ at values $x = 0, .5, 1.0, 1.5, \ldots, (s-3)/2$ and $x \in [s/2-1, \infty)$, where s is the size of the smallest interval, as defined in Section 4.2. For our implementation, $s \geq 32$ (see Section 6). Table A.1 shows the coefficients from the division of the interval $(0, 1/15)$ for y into four subintervals. The subintervals were chosen so as to limit the approximation's relative error to 2×10^{-15}. The coefficient values were computed using quadruple precision arithmetic.

Table A.2 contains the tabulation of Λ at the half-integer values up to 14.5.

Table A.2

Values of $\Lambda(x)$ for Small x

x	$\Lambda(x)$	x	$\Lambda(x)$
0	$0.17724538509055159d+01$	0.5	$0.11283791670955126d+01$
1	$0.88622692545275794d+00$	1.5	$0.75225277806367508d+00$
2	$0.66467019408956851d+00$	2.5	$0.60180222245094006d+00$
3	$0.55389182840797380d+00$	3.5	$0.51583047638652002d+00$
4	$0.48465534985697706d+00$	4.5	$0.45851597901023999d+00$
5	$0.43618981487127934d+00$	5.5	$0.41683270819112728d+00$
6	$0.39984066363200604d+00$	6.5	$0.38476865371488672d+00$
7	$0.37128061622971992d+00$	7.5	$0.35911741013389425d+00$
8	$0.34807557771536241d+00$	8.5	$0.33799285659660638d+00$
9	$0.32873804562006448d+00$	9.5	$0.32020375888099550d+00$
10	$0.31230114333906123d+00$	10.5	$0.30495596083904336d+00$
11	$0.29810563682364938d+00$	11.5	$0.29169700601995452d+00$
12	$0.28568456862266400d+00$	12.5	$0.28002912577915634d+00$
13	$0.27469670059871537d+00$	13.5	$0.26965767667622464d+00$
14	$0.26488610414876124d+00$	14.5	$0.26035913610118244d+00$

B An Alternative Approach

In this appendix we will briefly describe an alternative approach to convert between Legendre and Chebyshev expansions. The algorithm we described above is designed to cope with the singularities in $\mathcal{M}$, $\mathcal{L}$ where the two arguments are nearly equal. The definitions of $\mathcal{M}$, $\mathcal{L}$ (Eqs. 23 and 22) reveal that each is roughly the product of two lambda's, one a function of the sum of the arguments, another of the difference. The singular behavior near the diagonal is due to the function of the difference. Our second algorithm is based on "factoring out" this component.

We denote by T^n, S^n the pair of $n \times n$-matrices defined by the formulae

$$T^n_{ij} = \begin{cases} \Lambda(j-i) & \text{if } 0 \le i \le j < n \\ 0 & \text{otherwise} \end{cases} \tag{51}$$

$$S^n_{ij} = \begin{cases} \frac{-1}{2\pi(j-i)}\Lambda(j-i-1) & \text{if } 0 \le i \le j < n \\ 0 & \text{otherwise,} \end{cases} \tag{52}$$

with Λ defined by Eq. (10). We note that the entries along a forward diagonal of T^n or S^n are all equal, and it is not difficult to show that the matrices T^n and S^n are inverses of each other. We may therefore write

$$\begin{aligned} M^n &= T^n(S^n M^n) \\ L^n &= S^n(T^n L^n), \end{aligned}$$

where the matrix products in parentheses are both extremely smooth everywhere. We will not prove this assertion here; we will instead describe how this fact may be used to create our alternative algorithm.

The task of applying the transformation M^n is reduced to applying $R^n = S^n M^n$ then applying T^n to the result. The latter task may be accomplished by use of the fast Fourier transform for the application of T^n. Thus our task is reduced to applying R^n. The key to applying R^n is that an entire column of the matrix, excluding the diagonal element, may be well-approximated by a polynomial of small degree. In fact, the degree does not depend on the column number. This observation leads to a very simple algorithm.

We introduce the notation $r_{l,j}$ to denote the l^{th} polynomial coefficient for column j, *i.e.*,

$$R^n_{ij} \approx \sum_{l=0}^{k} r_{l,j} i^l \qquad \text{for } j = 1, 2, \ldots \text{ and } i = 0, \ldots, j-1. \tag{53}$$

Here we use k to denote the degree of the polynomial sufficient to maintain a given level of precision ($k = 5$ gives single precision and $k = 17$ gives double precision). Given the matrix size n, we define

$$q_{l,i} = \sum_{j=i}^{n-1} r_{l,j}\alpha_j \qquad \text{for } i = 1, \ldots, n-1 \text{ and } l = 0, \ldots, k. \tag{54}$$

We want to compute $\vec{\gamma} = R^n\vec{\alpha}$, where $\vec{\alpha} = \langle \alpha_0, \ldots, \alpha_{n-1} \rangle$. Since $R^n = S^n M^n$ is upper-triangular, γ_i $(i = 0, \ldots, n-1)$ satisfies the formula

$$\gamma_i = \sum_{j=i}^{n-1} R^n_{ij}\alpha_j. \tag{55}$$

Combining Eq. (55) with (53), changing the order of summation, and using the expression for $q_{l,i}$ in Eq. (54), yields

$$\gamma_i \approx \sum_{j=i}^{n-1} \left(\sum_{l=0}^{k} r_{l,j} i^l \right) \alpha_j = \sum_{l=0}^{k} i^l \left(\sum_{j=i}^{n-1} r_{l,j}\alpha_j \right) = \sum_{l=0}^{k} i^l q_{l,i} \tag{56}$$

It should now be clear that all $q_{l,i}$ $(l = 0, \ldots, k$ and $i = 0, \ldots, n-1)$ can be computed from the $r_{l,j}$ in order $O(nk)$ operations and then $\vec{\gamma} = \langle \gamma_0, \ldots, \gamma_{n-1} \rangle$ in an additional $O(nk)$ operations. The degree of the polynomials, k, depends only on the precision required; with the precision specified, the total time complexity needed to apply R^n is order $O(n)$.

The algorithm may be used, without modification, for the application of the matrix $Q^n = T^n L^n$ to an arbitrary vector of length n.

An issue arises in computing the polynomial coefficients $r_{l,j}$. This computation is complicated by the form of the definition of R^n. Each matrix element is expressed as a sum:

$$R^n_{ij} = \sum_{l=i}^{j} S^n_{il} M^n_{lj}.$$

Whereas each term in the sum may be computed efficiently, the sum itself may contain order $O(n)$ terms. On the face of it, the computation of the elements of R^n, and therefore the polynomial coefficients $r_{l,j}$, is prohibitively time-consuming. We solve this problem by computing instead a corresponding integral, using Gauss quadrature, then determining the sum by using the Euler-Maclaurin summation formula. This method produces an $O(\log n)$ algorithm for computing each coefficient $r_{l,j}$. We omit further details of this algorithm.

References

[1] M. Abramowitz and I. E. Stegun, editors. *Handbook of Mathematical Functions*. Applied Mathematics Series, No. 55. National Bureau of Standards, 1972.

[2] G. Beylkin and R. Coifman. Personal communication.

[3] E. O. Brigham. *The Fast Fourier Transform*. Prentice Hall, Inc., Englewood Cliffs, N.J., 1974.

[4] G. Dahlquist and Å. Björck. *Numerical Methods*. Prentice Hall, Inc., Englewood Cliffs, N.J., 1974.

[5] I. S. Gradshteyn and I. M. Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, Inc., 1980.

[6] L. Greengard and V. Rokhlin. "A Fast Algorithm for Particle Simulations," *Journal of Computational Physics*. Vol. 73, No. 2, Academic Press, Inc., December, 1987.

[7] N. N. Lebedev. *Special Functions and Their Applications*. Dover Publications, Inc., New York, 1972.

[8] S. A. Orszag. "Fast Eigenfunction Transforms," *Science and Computers, Advances in Mathematics Supplementary Studies*. G. C. Rota, editor. Vol. 10, pp. 23-30, Academic Press, Inc., 1986.

[9] V. Rokhlin. "A Fast Algorithm for the Discrete Laplace Transformation," *Journal of Complexity* 4, pp. 12-32, Academic Press, Inc., 1988.