For functions tabulated at Chebyshev nodes on an interval, spectral interpolation, integration and differentiation can be performed stably and efficiently via the fast Fourier transform. In this paper, a group of algorithms is presented for the efficient evaluation of Lagrange polynomial interpolants at multiple points on the line, and for the rapid spectral integration and differentiation of functions tabulated at nodes other than Chebyshev. The interpolation scheme requires $O(N \cdot \log(1/\varepsilon))$ arithmetic operations, and $O(N \cdot \log N + N \cdot \log(1/\varepsilon))$ operations are required for the integration and differentiation schemes, where ε is the precision of computations and N is the number of nodes. The algorithms utilize efficient versions of the fast multipole method which have been designed specifically for one-dimensional problems; these are also described in the present paper. Several experiments are included to illustrate the numerical performance of the approach.

Fast Algorithms for Polynomial Interpolation, Integration and Differentiation

A. Dutt†, M. Gu‡ and V. Rokhlin†

Research Report YALEU/DCS/RR-977
July 1993

† The work of this author was supported in part by the Office of Naval Research under grant N00014-89-J-1527 and in part by the National Science Foundation under grant DMS9012751.
‡ The work of this author was supported in part by U.S. Army Research Office under grant DAAL03-91-G-0032.
Approved for public release: distribution is unlimited.
Keywords: *Polynomials, Interpolation, Fast Multipole Method, Approximation Theory*

1 Introduction

Polynomials form the theoretical basis for many areas of applied mathematics. While the mathematical properties of polynomials have been quite well understood for over a century, attempts to use them in practical calculations have met with difficulties. Typical problems accompanying the employment of classical schemes are those of prohibitive computational cost and numerical instability. However, certain classes of orthogonal polynomials do have stable, fast transforms associated with them, the most popular being Chebyshev polynomials which can be manipulated in a stable and efficient manner via the fast cosine transform or FFT.

In this paper we present a group of three algorithms for the interpolation, integration and differentiation of functions tabulated at nodes other than Chebyshev. The first of these algorithms takes as input a set of points, $\{x_1, \ldots, x_N\}$, and a set of function values, $\{f_1, \ldots, f_N\}$, and evaluates the unique interpolating polynomial of degree $N-1$ at the points $\{y_1, \ldots, y_N\}$ for a computational cost proportional to N. The other two algorithms perform spectral integration and differentiation of this interpolant. We will also describe three efficient versions of the Fast Multipole Method (FMM) for one dimensional problems; these algorithms are used by the polynomial interpolation algorithm of this paper. Throughout this paper we will be using the well known Lagrange representation of interpolating polynomials which is defined by the formula

$$P_N(x) = \sum_{j=1}^{N} f_j \cdot \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{x - x_k}{x_j - x_k}. \tag{1}$$

A simple algebraic manipulation converts (1) to the form

$$P_N(x) = \prod_{k=1}^{N} (x - x_k) \cdot \sum_{j=1}^{N} \frac{f_j}{x - x_j} \cdot \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{1}{x_j - x_k}, \tag{2}$$

which can be evaluated at N points in

$$O\left(N \cdot \log\left(\frac{1}{\varepsilon}\right)\right) \tag{3}$$

arithmetic operations using the Fast Multipole Method (FMM) of [10] (ε here is the desired accuracy). In comparison, a direct evaluation of (2) requires $O(N^2)$ operations.

Remark 1.1 A somewhat different classical polynomial interpolation problem consists of determining a set of parameters $\{a_1, \ldots, a_N\}$ such that, for $j = 1, \ldots, N$,

$$\sum_{k=1}^{N} a_k \cdot x_j^{k-1} = f_j \tag{4}$$

where $\{x_1, \ldots, x_N\}$ is a given a set of points and $\{f_1, \ldots, f_N\}$ is a given set of function values. This problem is highly ill-posed for anything other than very small values of N and this formulation is seldom used when actual calculations are being performed.

Remark 1.2 The Lagrange interpolation formula has traditionally been less favored for practical calculations than other classical methods (see, for example, [14]). However, the algorithms of this paper are numerically stable and very efficient, as demonstrated by our numerical experiments, thus affording the Lagrange formula a substantial advantage over other techniques for the manipulation of polynomials.

Following is a plan of the paper. The first three sections are devoted to efficient versions of the FMM which can be used to evaluate expressions of the form (2): Section 2 contains a number of results from analysis which are used in the design of these algorithms, in Section 3 we present the FMM algorithm itself, and in Section 4 we present an adaptive version of this algorithm. A Fast Polynomial Interpolation algorithm utilizing the results of the previous sections is then described in Section 5, and in Section 6 we describe how this algorithm is used to construct fast algorithms for spectral integration and differentiation. In Section 7 we present the results of several of our numerical experiments, and finally, in Section 8 we list several generalizations and conclusions of the results of this paper.

2 Mathematical and Numerical Preliminaries

The algorithms of this paper are based on several results from the Chebyshev approximation theory of the function $1/x$. This analysis is presented in the Lemmas and Theorems of this section, numbered 2.1-2.10. The main results of this section fall into two categories: Theorems 2.5 and 2.7 describe how the function $1/x$ can be approximated on different regions of the real line using Chebyshev expansions, and Theorems 2.8, 2.9 and 2.10 provide three ways of manipulating these expansions which are needed by the fast algorithms of this paper.

We begin with three classical definitions which can be found, for example, in [8], [14].

Definition 2.1 *The n-th degree Chebyshev polynomial, $T_n(x)$, is defined by the following formulae:*

$$T_n(x) = \cos(n \arccos x), \tag{5}$$

$$T_n(x) = \frac{1}{2} \cdot \left((x + \sqrt{x^2 - 1})^n + (x - \sqrt{x^2 - 1})^n \right). \tag{6}$$

Definition 2.2 *The Chebyshev nodes $\{t_1, \ldots, t_n\}$ of order n on the interval $[-1, 1]$ are defined by the formulae*

$$t_k = -\cos\left(\frac{2k-1}{n} \cdot \frac{\pi}{2} \right) \tag{7}$$

for $k = 1, \ldots, n$.

Definition 2.3 *$u_1, \ldots, u_n$ will be a set of polynomials of order $n-1$ defined by the formulae*

$$u_j(t) = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{t - t_k}{t_j - t_k}, \tag{8}$$

for $j = 1, \ldots, n$, where t_k are defined by (7).

For a function $f : [-1,1] \to \mathbf{C}$, the order $n-1$ Chebyshev approximation to f on the interval $[-1,1]$ is the unique polynomial of order $n-1$ which agrees with f at the nodes $t_1, \ldots, t_n$. There exist several standard representations for this polynomial, and the one we will use in this paper is given by the expression

$$\sum_{j=1}^{n} f(t_j) \cdot u_j(t). \tag{9}$$

For the purposes of this paper, Chebyshev expansions for any function will be characterized by values of this function tabulated at Chebyshev nodes.

Lemmas 2.1–2.3 provide estimates involving Chebyshev expansions which are used in the remainder of this section. The proof of Lemma 2.1 is obvious from (5).

Lemma 2.1 *Let $T_n(x)$ be the Chebyshev polynomial of degree n. Then,*

$$|T_n(x)| \leq 1 \tag{10}$$

for any $x \in [-1,1]$.

Lemma 2.2 *Let $T_n(x)$ be the Chebyshev polynomial of degree n. Then,*

$$|T_n(x)| > \frac{1}{2} \cdot \left|\frac{5x}{3}\right|^n \tag{11}$$

for any x such that $|x| \geq 3$.

Proof. From Definition 2.1, we have

$$\begin{aligned} |T_n(x)| &= \frac{1}{2} \cdot \left|(x+\sqrt{x^2-1})^n + (x-\sqrt{x^2-1})^n\right| \\ &> \frac{1}{2} \cdot |x+\sqrt{x^2-(x/3)^2}|^n = \frac{1}{2} \cdot |x \cdot (1+\sqrt{8/9})|^n \\ &> \frac{1}{2} \cdot \left|\frac{5x}{3}\right|^n \end{aligned} \tag{12}$$

for any x such that $|x| \geq 3$. □

Lemma 2.3 *Let $u_j(x)$ be defined by (8). Then, for any $x \in [-1,1]$,*

$$|u_j(x)| \leq 1. \tag{13}$$

Proof. It is obvious from (8) that $u_j(t_j) = 1$, and that $u_j(t_k) = 0$ when $k \neq j$. In addition, the expression

$$\frac{1}{n} \sum_{k=1}^{n} T_k(t_j) \cdot T_k(x) \tag{14}$$

is also equal to 1 at t_j and equal to 0 at all other t_k. Since both u_j and (14) are polynomials of order $n-1$, we have

$$u_j(x) = \frac{1}{n}\sum_{k=1}^{n} T_k(t_j)\cdot T_k(x) \tag{15}$$

for $j = 1,\ldots,n$. Furthermore, due to the combination of (15) and the triangle inequality, we obtain

$$|u_j(x)| = \left|\frac{1}{n}\sum_{k=1}^{n} T_k(t_j)\cdot T_k(x)\right| = \frac{1}{n}\sum_{k=1}^{n} |T_k(t_j)|\cdot|T_k(x)| \le 1 \tag{16}$$

for any $x \in [-1,1]$. □

The following lemma provides an identity which is used in the proof of Theorem 2.5.

Lemma 2.4 *Suppose that $n \ge 2$, and that $b > 0$ and x_0 are real numbers with $|x_0| \ge 3b$. Then, for all x,*

$$1 - (x - x_0)\cdot\sum_{j=1}^{n}\frac{1}{bt_j - x_0}\cdot u_j\left(\frac{x}{b}\right) = \frac{T_n(x/b)}{T_n(x_0/b)}. \tag{17}$$

Proof. Let $Q(x)$ by the polynomial of degree n defined by the formula

$$Q(x) = 1 - (x - x_0)\cdot\sum_{j=1}^{n}\frac{1}{bt_j - x_0}\cdot u_j\left(\frac{x}{b}\right). \tag{18}$$

Using (8) we obtain

$$\begin{aligned} Q(bt_k) &= 1 - (bt_k - x_0)\cdot\sum_{j=1}^{n}\frac{1}{bt_j - x_0}\cdot u_j(t_k) \\ &= 1 - (bt_k - x_0)\cdot\frac{1}{bt_k - x_0} = 0, \end{aligned} \tag{19}$$

for $k = 1,\ldots,n$. Clearly, then, $Q(x)$ satisfies the conditions

$$\begin{aligned} Q(x_0) &= 1 \\ Q(bt_1) &= 0 \\ &\vdots \\ Q(bt_n) &= 0. \end{aligned} \tag{20}$$

It is clear that the function $T_n(x/b)/T_n(x_0/b)$ is also a polynomial of degree n which satisfies the $n+1$ conditions (20). Therefore,

$$Q(x) \equiv \frac{T_n(x/b)}{T_n(x_0/b)}, \tag{21}$$

and (17) follows as an immediate consequence of (18) and (21). □

Theorem 2.5 *Suppose that $n \geq 2$, and that $b > 0$ and x_0 are real numbers with $|x_0| \geq 3b$. Then,*

$$\left| \frac{1}{x - x_0} - \sum_{j=1}^{n} \frac{1}{bt_j - x_0} \cdot u_j\left(\frac{x}{b}\right) \right| < \frac{1}{b \cdot 5^n} \tag{22}$$

for any $x \in [-b, b]$.

Proof. Due to Lemma 2.4, we have

$$\left| \frac{1}{x - x_0} - \sum_{j=1}^{n} \frac{1}{bt_j - x_0} \cdot u_j\left(\frac{x}{b}\right) \right| = \frac{1}{|x - x_0|} \cdot \frac{|T_n(x/b)|}{|T_n(x_0/b)|}. \tag{23}$$

Also, due to Lemmas 2.1 and 2.2, we have

$$|T_n(x/b)| \leq 1 \tag{24}$$

for any $x \in [-b, b]$, and

$$|T_n(x_0/b)| > \frac{1}{2} \cdot \left|\frac{5x_0}{3b}\right|^n > \frac{5^n}{2} \tag{25}$$

for any $|x_0| \geq 3b$. It follows from the combination of (23), (24) and (25) that

$$\left| \frac{1}{x - x_0} - \sum_{j=1}^{n} \frac{1}{bt_j - x_0} \cdot u_j\left(\frac{x}{b}\right) \right| \leq \frac{1}{2b} \cdot \frac{2}{5^n} \leq \frac{1}{b \cdot 5^n} \tag{26}$$

for any $x \in [-b, b]$. □

The following lemma provides an identity which is used in the proof of Theorem 2.7.

Lemma 2.6 *Suppose that $n \geq 2$, and that $b > 0$ and x_0 are real numbers with $|x_0| \leq b$. Then, for all ξ,*

$$\xi - (3b - \xi x_0) \cdot \sum_{j=1}^{n} \frac{t_j}{3b - t_j x_0} \cdot u_j(\xi) = \frac{3b}{x_0} \cdot \frac{T_n(\xi)}{T_n(3b/x_0)}. \tag{27}$$

Proof. Let $Q(\xi)$ be the polynomial of degree n defined by the formula

$$Q(\xi) = \xi - (3b - \xi x_0) \cdot \sum_{j=1}^{n} \frac{t_j}{3b - t_j x_0} \cdot u_j(\xi). \tag{28}$$

Using (8) we obtain

$$\begin{aligned} Q(t_k) &= t_k - (3b - t_k x_0) \cdot \sum_{j=1}^{n} \frac{t_j}{3b - t_j x_0} \cdot u_j(t_k) \\ &= t_k - (3b - t_k x_0) \cdot \frac{t_k}{3b - t_k x_0} = 0, \end{aligned} \tag{29}$$

for $k = 1, \ldots, n$. Clearly, then, $Q(\xi)$ satisfies the conditions

$$\begin{aligned} Q(3b/x_0) &= 3b/x_0 \\ Q(t_1) &= 0 \\ &\vdots \\ Q(t_n) &= 0. \end{aligned} \tag{30}$$

It is clear that the function

$$\frac{3b}{x_0} \cdot \frac{T_n(\xi)}{T_n(3b/x_0)} \tag{31}$$

is also a polynomial of degree n which satisfies the $n+1$ conditions (30). Therefore,

$$Q(\xi) \equiv \frac{3b}{x_0} \cdot \frac{T_n(\xi)}{T_n(3b/x_0)}, \tag{32}$$

and (27) follows as an immediate consequence of (28) and (32). □

Theorem 2.7 *Suppose that $n \geq 2$, and that $b > 0$ and x_0 are real numbers with $|x_0| \leq b$. Then,*

$$\left| \frac{1}{x - x_0} - \sum_{j=1}^{n} \frac{t_j}{3b - t_j x_0} \cdot u_j\left(\frac{3b}{x}\right) \right| < \frac{3}{b \cdot 5^n} \tag{33}$$

for any x such that $|x| \geq 3b$.

Proof. Writing $\xi = 3b/x$, we have $|\xi| \leq 1$ whenever $|x| \geq 3b$, and

$$\frac{1}{x - x_0} = \frac{1}{3b/\xi - x_0} = \frac{\xi}{3b - \xi x_0}. \tag{34}$$

Due to Lemma 2.6, we have

$$\left| \frac{\xi}{3b - \xi x_0} - \sum_{j=1}^{n} \frac{t_j}{3b - t_j x_0} \cdot u_j(\xi) \right| = \frac{1}{|3b - \xi x_0|} \cdot \frac{3b}{|x_0|} \cdot \frac{|T_n(\xi)|}{|T_n(3b/x_0)|}. \tag{35}$$

In addition, due to Lemmas 2.1 and 2.2 we have

$$3b \cdot |T_n(\xi)| \leq 3b \tag{36}$$

for $\xi \in [-1, 1]$, and

$$|x_0 \cdot T_n(3b/x_0)| > \frac{|x_0|}{2} \cdot \left| \frac{5 \cdot 3b}{3x_0} \right|^n > \frac{5^n \cdot b}{2} \tag{37}$$

for $|x_0| \leq b$. Substituting (36) and (37) into (35), we obtain

$$\left| \frac{\xi}{3b - \xi x_0} - \sum_{j=1}^{n} \frac{t_j}{3b - t_j x_0} \cdot u_j(\xi) \right| < \frac{1}{2b} \cdot \frac{3b \cdot 2}{5^n \cdot b} = \frac{3}{b \cdot 5^n} \tag{38}$$

for $\xi \in [-1,1]$. Now it follows from the combination of (34) and (38) that

$$\left| \frac{1}{x - x_0} - \sum_{j=1}^{n} \frac{t_j}{3b - t_j x_0} \cdot u_j\left(\frac{3b}{x}\right) \right| < \frac{3}{b \cdot 5^n}. \tag{39}$$

□

The following three theorems provide formulae for translating along the real axis Chebyshev expansions of the type described in the previous two theorems. Theorem 2.8 provides a formula for translating expansions described in Theorems 2.5, Theorem 2.9 describes a mechanism of converting the expansion of Theorem 2.5 to the expansion of Theorem 2.7, and Theorem 2.10 provides a way of translating the expansion of Theorem 2.7. These theorems are one-dimensional counterparts of the three theorems in [10] which provide translation operators for multipole expansions in the complex plane.

Theorem 2.8 *Suppose that $n \geq 2$ and let c, d be a pair of real numbers such that $[c-d, c+d] \subset [-1,1]$. Then, for any set of complex numbers $\Psi_1, \ldots, \Psi_n$, and any $x \in [c-d, c+d]$,*

$$\sum_{j=1}^{n} \Psi_j \cdot u_j(x) = \sum_{k=1}^{n} \left(\tilde{\Psi}_k \cdot u_j(c + dt_k) \right) \cdot u_j\left(\frac{x-c}{d}\right). \tag{40}$$

Proof. To prove this theorem we first show that

$$u_j(x) = \sum_{k=1}^{n} u_j(c + dt_k) \cdot u_k\left(\frac{x-c}{d}\right) \tag{41}$$

for $x \in [c-d, c+d]$. Indeed, the right hand side of (41) is simply the $(n-1)$-th degree Lagrange interpolating polynomial for the function $u_j(x)$ at the nodes $c+dt_1, \ldots, c+dt_n$. However, $u_j(x)$ itself is a polynomial of degree $n-1$, and is therefore equal to its Lagrange interpolant of order $n-1$.

The formula (40) then follows as an immediate consequence of (41). □

Theorem 2.9 *Suppose that $n \geq 2$ and let c, d be a pair of real numbers such that $|c| - d > 3$. Let the function $f : \mathbf{R} \to \mathbf{C}$ be defined by the formula*

$$f(x) = \sum_{k=1}^{N} \frac{\alpha_k}{x - x_k} \tag{42}$$

where $x_k \in [-1,1]$ for $k = 1, \ldots, N$, and $\alpha_1, \ldots, \alpha_N$ is a set of complex numbers. Further, let $t_1, \ldots, t_n$ be Chebyshev nodes defined by (7), let $\Phi_1, \ldots, \Phi_n$ be a set of complex numbers defined by the formula

$$\Phi_k = f\left(\frac{3}{t_k}\right) \tag{43}$$

for $k = 1, \ldots, n$, and let $\Psi_1, \ldots, \Psi_n$ be a set of complex numbers defined by the formula

$$\Psi_k = \sum_{j=1}^{n} \Phi_j \cdot u_j\left(\frac{3}{c + dt_k}\right). \tag{44}$$

Then, for any $x \in [c-d, c+d]$,

$$\left| f(x) - \sum_{k=1}^{n} \Psi_k \cdot u_j\left(\frac{x-c}{d}\right)\right| < A \cdot \frac{3n+1}{b \cdot 5^n}, \tag{45}$$

where $A = \sum_{k=1}^{N} |\alpha_k|$.

Proof. Due to the triangle inequality,

$$\left| f(x) - \sum_{k=1}^{n} \Psi_k \cdot u_j\left(\frac{x-c}{d}\right)\right| \leq S_1 + S_2, \tag{46}$$

where

$$S_1 = \left| f(x) - \sum_{k=1}^{n} f(c + dt_k) \cdot u_j\left(\frac{x-c}{d}\right)\right|, \tag{47}$$

and

$$S_2 = \left| \sum_{k=1}^{n} (f(c + dt_k) - \Psi_k) \cdot u_j\left(\frac{x-c}{d}\right)\right|. \tag{48}$$

Combining Theorem 2.5 with the triangle inequality we obtain

$$S_1 < \frac{A}{b \cdot 5^n}, \tag{49}$$

and from the combination of Theorem 2.7, Lemma 2.3 and the triangle inequality, we obtain

$$S_2 \leq \sum_{k=1}^{n} \left| f(c + dt_k) - \sum_{j=1}^{n} \Phi_j \cdot u_j\left(\frac{3}{c + dt_k}\right)\right| < \frac{3An}{b \cdot 5^n}, \tag{50}$$

where $A = \sum_{k=1}^{N} |\alpha_k|$. Finally, substituting (49) and (50) into (46) we have

$$\left| f(x) - \sum_{k=1}^{n} \Psi_k \cdot u_j\left(\frac{x-c}{d}\right)\right| < A \cdot \frac{3n+1}{b \cdot 5^n} \tag{51}$$

for any $x \in [c-d, c+d]$. □

Theorem 2.10 *Suppose that $n \geq 2$ and let c, d be a pair of real numbers such that $[c-d, c+d] \supset [-1, 1]$. Let the function $f : \mathbf{R} \to \mathbf{C}$ be defined by the formula*

$$f(x) = \sum_{k=1}^{N} \frac{\alpha_k}{x - x_k} \tag{52}$$

where $x_k \in [-1,1]$ for $k = 1,\ldots,N$, and $\alpha_1,\ldots,\alpha_N$ is a set of complex numbers. Further, let $t_1,\ldots,t_n$ be Chebyshev nodes defined by (7), let $\Phi_1,\ldots,\Phi_n$ be a set of complex numbers defined by the formula

$$\Phi_k = f\left(\frac{3}{t_k}\right) \tag{53}$$

for $k = 1,\ldots,n$, and let $\tilde{\Phi}_1,\ldots,\tilde{\Phi}_n$ be a set of complex numbers defined by the formula

$$\tilde{\Phi}_k = \sum_{j=1}^{n} \Phi_j \cdot u_j\left(\frac{t_k}{3d + ct_k}\right). \tag{54}$$

Then, for any x such that $|x - c| \geq 3d$,

$$\left| f(x) - \sum_{k=1}^{n} \tilde{\Phi}_k \cdot u_j\left(\frac{3d}{x-c}\right)\right| < A \cdot \frac{3(n+1)}{b \cdot 5^n}, \tag{55}$$

where $A = \sum_{k=1}^{N} |\alpha_k|$.

Proof. It follows from the triangle inequality that

$$\left| f(x) - \sum_{k=1}^{n} \tilde{\Phi}_k \cdot u_j\left(\frac{3d}{x-c}\right)\right| \leq S_1 + S_2 \tag{56}$$

where

$$S_1 = \left| f(x) - \sum_{k=1}^{n} f\left(c + \frac{3d}{t_k}\right) \cdot u_j\left(\frac{3d}{x-c}\right)\right|, \tag{57}$$

and

$$S_2 = \left| \sum_{k=1}^{n} \left(f\left(c + \frac{3d}{t_k}\right) - \tilde{\Phi}_k \right) \cdot u_j\left(\frac{3d}{x-c}\right)\right|. \tag{58}$$

Combining Theorem 2.7 with the triangle inequality, we obtain

$$S_1 < \frac{3A}{b \cdot 5^n}, \tag{59}$$

and from the combination of Theorem 2.7, Lemma 2.3 and the triangle inequality, we obtain

$$S_2 \leq \sum_{k=1}^{n} \left| f\left(c + \frac{3d}{t_k}\right) - \sum_{j=1}^{n} \Phi_j \cdot u_j\left(\frac{t_k}{3d + ct_k}\right)\right| < \frac{3An}{b \cdot 5^n}, \tag{60}$$

where $A = \sum_{k=1}^{N} |\alpha_k|$. Finally, substituting (59) and (60) into (56) we have

$$\left| f(x) - \sum_{k=1}^{n} \tilde{\Phi}_k \cdot u_j\left(\frac{3d}{x-c}\right)\right| < A \cdot \frac{3(n+1)}{b \cdot 5^n} \tag{61}$$

for any x such that $|x - c| \geq 3d$. □

3 The Fast Multipole Method in One Dimension

In this section we consider the problem of computing the sums

$$f_j = \sum_{k=1}^{N} \frac{\alpha_k}{y_j - x_k} \tag{62}$$

for $j = 1, \ldots, N$, where $\{x_1, \ldots, x_N\}$ and $\{y_1, \ldots, y_N\}$ are sets of real numbers, and $\{\alpha_1, \ldots, \alpha_N\}$ and $\{f_1, \ldots, f_N\}$ are sets of complex numbers. This problem can be viewed as the special case of the N-body problem in physics where we wish to evaluate the electrostatic field due to N charges which lie on a straight line at a set of points on this line.

Remark 3.1 For the remainder of this paper, we shall assume without loss of generality that $x_i, y_i \in [-1, 1]$ for $i = 1, \ldots, N$.

Remark 3.2 The fast multipole algorithm of [10] computes sums of a more general form than (62) in $O(N)$ arithmetic operations. This more general form is described by the formulae

$$f_j = \sum_{k=1}^{N} \frac{\alpha_k}{w_j - z_k} \tag{63}$$

for $j = 1, \ldots, N$, where $\{z_1, \ldots, z_N\}$ and $\{w_1, \ldots, w_N\}$ are sets of complex numbers. From a physical viewpoint, this corresponds to the evaluation of the electrostatic field due to N charges which lie in the plane. While the two and three dimensional scenarios for the N-body problem have been discussed in some depth (see, for example, [4], [10]), the analysis and applications of one dimensional problems appear to have been largely overlooked, with one exception of which we are aware: the application of FMM techniques to various problems in numerical linear algebra (see [11], [12]).

In this section we present an $O(N)$ algorithm for the one-dimensional problem which is based on the two-dimensional FMM, but incorporates a number of modifications which accelerate the scheme significantly. We summarize these modifications below, with the assumption that the reader is familiar with [10].

1. Replacement of all complex by complex multiplications with real by complex multiplications.

2. Replacement of multipole expansions with Chebyshev expansions. Chebyshev series are known to converge more rapidly than multipole expansions (which are actually Taylor series).

3. Further compression of the Chebyshev expansions by a suitable change of basis (see Section 3.4). Using this technique, the function $1/x$ can be accurately represented by about a quarter of the number of coefficients which were required by the original two-dimensional FMM.

4. In one dimension, each subinterval has 3 other subintervals in its interaction list, whereas in two dimensions each box has 27 other boxes in its interaction list.

5. In one dimension, each subinterval has 2 nearest neighbor subintervals, whereas in two dimensions each box has 8 nearest neighbor boxes.

This section is divided into four parts. Sections 3.1-3.3 are devoted to an algorithm for the FMM in one dimension which uses Chebyshev expansions in place of multipole expansions. In Section 3.4, we describe a more efficient algorithm which is based on the algorithm of Section 3.3 but uses a Singular Value Decomposition to further compress the Chebyshev expansions.

Remark 3.3 The algorithms described in this paper are all designed for evaluating sums of the form

$$f(x) = \sum_{k=1}^{N} \frac{\alpha_k}{x - x_k}. \tag{64}$$

However, these algorithms are only mildly dependent on the choice of kernel, i.e. they can be modified to evaluate expressions of a more general form, given by the formula

$$f(x) = \sum_{k=1}^{N} \alpha_k \cdot \phi(x - x_k), \tag{65}$$

where ϕ is singular at 0 but smooth everywhere else. Examples of functions with this property include $\phi(x) = \log(x)$, $\phi(x) = 1/x^2$ and $\phi(x) = 1/\tan(x)$.

3.1 General Strategy

We will illustrate by means of a simple example how Chebyshev expansions can be used to evaluate expressions of the form (62) more efficiently. We will also give an informal description of how the method of this simple example is used in the construction of a fast algorithm for the general case.

First we introduce a definition which formalizes the notion of well-separated intervals on the real line. This is simply the one-dimensional analog of the definition of well-separatedness in [10].

Definition 3.1 *Let $\{x_1, \ldots, x_N\}$ and $\{y_1, \ldots, y_M\}$ be two sets of points in $\mathbf{R}$. We say that the sets $\{x_i\}$ and $\{y_i\}$ are* well-separated *if there exist points $x_0, y_0 \in \mathbf{R}$ and a real $r > 0$ such that*

$$\begin{aligned} &|x_i - x_0| < r \quad \forall \ \ i = 1, \ldots, N, \\ &|y_i - y_0| < r \quad \forall \ \ i = 1, \ldots, M, \ \textit{and} \\ &|x_0 - y_0| > 4r. \end{aligned} \tag{66}$$

Suppose now that $\{x_1, \ldots, x_N\}$ and $\{y_1, \ldots, y_M\}$ are well-separated sets of points in $\mathbf{R}$ (see Figure 1), that $\{\alpha_1, \ldots, \alpha_N\}$ is a set of complex numbers, and that we wish to compute the numbers $f(y_1), \ldots, f(y_M)$ where the function $f : \mathbf{R} \rightarrow \mathbf{C}$ is defined by the formula

$$f(x) = \sum_{k=1}^{N} \frac{\alpha_k}{x - x_k}. \tag{67}$$

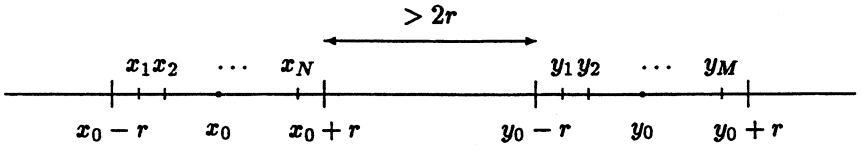


Figure 1: Well-separated intervals on the line.

A direct evaluation of (67) at the points $\{y_1, \ldots, y_M\}$ requires $O(NM)$ arithmetic operations. We will describe two different ways of speeding up this calculation.

Following is the first of these approaches.

Let the function $\tilde{f}_1 : \mathbf{R} \to \mathbf{C}$ be defined by the formula

$$\tilde{f}_1(x) = \sum_{k=1}^{N} \alpha_k \cdot \sum_{j=1}^{p} \frac{t_j}{3r - t_j(x_k - x_0)} \cdot u_j \left(\frac{3r}{x - x_0} \right), \tag{68}$$

where p is an integer and $t_1, \ldots, t_p$ and $u_1, \ldots, u_p$ are given by Definitions 2.2 and 2.3.

Observation 3.4 *From the combination of (67), (68), Theorem 2.7 and the triangle inequality, we see that*

$$\left| f(x) - \tilde{f}_1(x) \right| < \frac{3 \cdot \sum_{k=1}^{N} |\alpha_k|}{r \cdot 5^p} \tag{69}$$

for any x such that $|x - x_0| > 3r$.

Now let $\{\Phi_1, \ldots, \Phi_p\}$ be a set of complex numbers defined by the formulae

$$\Phi_j = \sum_{k=1}^{N} \alpha_k \cdot \frac{t_j}{3r - t_j(x_k - x_0)} \tag{70}$$

for $j = 1, \ldots, p$. Then,

$$\tilde{f}_1(x) = \sum_{j=1}^{p} \Phi_j \cdot u_j \left(\frac{3r}{x - x_0} \right). \tag{71}$$

The vector Φ will be referred to as the far-field expansion for the interval $[x_0 - r, x_0 + r]$.

Computation of the coefficients Φ_j requires $O(Np)$ operations, and a subsequent evaluation of $\tilde{f}_1(y_1), \ldots, \tilde{f}_1(y_M)$ is an $O(Mp)$ procedure. The total computational cost of approximating (67) to a relative precision $1/5^p$ is then $O(Np + Mp)$ operations.

An alternative way of speeding up this calculation is described below.

Let the function $\tilde{f}_2 : \mathbf{R} \to \mathbf{C}$ be defined by the formula

$$\tilde{f}_2(x) = \sum_{k=1}^{N} \alpha_k \cdot \sum_{j=1}^{p} \frac{1}{rt_j - (x_k - y_0)} \cdot u_j \left(\frac{x - y_0}{r} \right), \tag{72}$$

where p is an integer and $t_1, \ldots, t_p$ and $u_1, \ldots, u_p$ are given by Definitions 2.2 and 2.3.

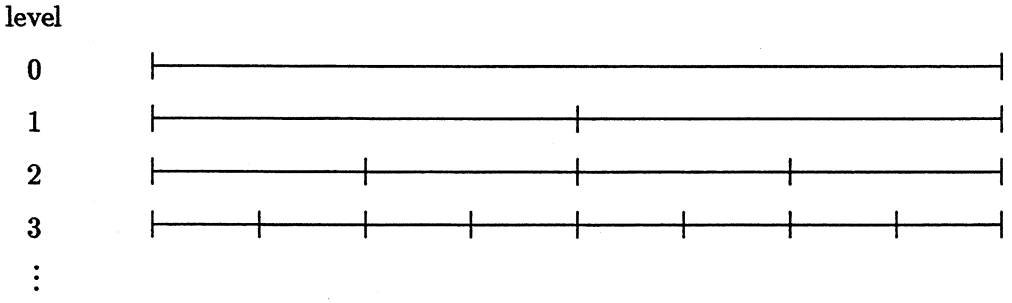


Figure 2: Hierarchy of subintervals.

Observation 3.5 *From the combination of (67), (72), Theorem 2.5 and the triangle inequality, we see that*

$$\left|f(x) - \tilde{f}_2(x)\right| < \frac{\sum_{k=1}^{N} |\alpha_k|}{r \cdot 5^p} \tag{73}$$

for any x such that $|x - y_0| < r$.

Now let $\{\Psi_1, \ldots, \Psi_p\}$ be a set of complex numbers defined by the formulae

$$\Psi_j = \sum_{k=1}^{N} \alpha_k \cdot \frac{1}{rt_j - (x_k - x_0)}. \tag{74}$$

for $j = 1, \ldots, p$. Then,

$$\tilde{f}_2(x) = \sum_{j=1}^{p} \Psi_j \cdot u_j \left(\frac{x - y_0}{r}\right). \tag{75}$$

The vector Ψ will be referred to as the local expansion for the interval $[y_0 - r, y_0 + r]$.

Computation of the coefficients Ψ_j requires $O(Np)$ operations, and a subsequent evaluation of $\tilde{f}_2(y_1), \ldots, \tilde{f}_2(y_M)$ is an $O(Mp)$ procedure. Again the total computational cost of approximating (67) to a relative precision $1/5^p$ is $O(Np + Mp)$ operations.

Consider now the general case, where the points $\{x_1, \ldots, x_N\}$ and $\{y_1, \ldots, y_M\}$ are arbitrarily distributed on the interval $[-1, 1]$ (see Remark 3.1). We use a hierarchy of grids to subdivide the computational domain $[-1, 1]$ into progressively smaller subintervals, and to subdivide the sets $\{x_i\}$ and $\{y_i\}$ according to subinterval (see Figure 2). A tree structure is imposed on this hierarchy, so that the two subintervals resulting from the bisection of a larger (parent) interval are referred to as its children. Two Chebyshev expansions are associated with each subinterval: a far-field expansion for the points within the subinterval, and a local expansion for the points which are well-separated from the subinterval. Interactions between pairs of well-separated subintervals can be computed via these Chebyshev expansions in the manner described above, and all other interactions at the finest level can be computed directly. Once the precision has been fixed, the computational cost of the entire procedure is $O(N)$ operations (for a detailed description and complexity analysis, see Section 3.3).

3.2 Notation

In this section we introduce the notation to be used in the description of the algorithm below.

- p will be an integer denoting the size of Chebyshev expansions used by the algorithm. Normally, $p = \lceil -\log_5(\varepsilon) \rceil$, where ε is the desired precision of computations.
- $t_1, \ldots, t_p$ will denote Chebyshev nodes of order p on the interval $[-1, 1]$, defined by the formulae

$$t_i = \cos\left(\frac{2i-1}{p} \cdot \frac{\pi}{2}\right) \tag{76}$$

  for $i = 1, \ldots, p$.
- $u_1(t), \ldots, u_p(t)$ will denote the set of polynomials defined by the formulae

$$u_j(t) = \prod_{\substack{k=1 \\ k \neq j}}^{p} \frac{t - t_k}{t_j - t_k}, \tag{77}$$

  for $j = 1, \ldots, p$.
- s will be an integer denoting the number of points in each subinterval at the finest level of subdivision. Normally, $s \approx 2p$ (see Remark 3.6).
- $nlevs = \lceil \log_2(N/s) \rceil$ will denote the level of finest subdivision of the interval $[-1, 1]$.
- $\Phi_{l,i}$ will be a p-vector denoting the far-field expansion for the subinterval i at level l.
- $\Psi_{l,i}$ will be a p-vector denoting the local expansion for the subinterval i at level l.
- M_L and M_R will be $p \times p$ matrices for obtaining far-field expansions for subintervals in terms of the far-field expansions of their children. These will be defined by the formulae

$$\begin{aligned} M_L(i,j) &= u_j\left(\frac{t_i}{2+t_i}\right), \\ M_R(i,j) &= u_j\left(\frac{t_i}{2-t_i}\right), \end{aligned} \tag{78}$$

  which are obtained from the formula (54) of Theorem 2.10 applied to the cases $c = -1, d = 2$ and $c = 1, d = 2$.
- S_L and S_R will be $p \times p$ matrices for obtaining local expansions for subintervals in terms of the local expansions of their parent. These will be defined by the formulae

$$\begin{aligned} S_L(i,j) &= u_j\left(\frac{t_i - 1}{2}\right), \\ S_R(i,j) &= u_j\left(\frac{t_i + 1}{2}\right). \end{aligned} \tag{79}$$

  which are obtained from the formula (40) of Theorem 2.8 applied to the cases $c = -\frac{1}{2}, d = \frac{1}{2}$ and $c = \frac{1}{2}, d = \frac{1}{2}$.

- T_1, T_2, T_3 and T_4 will be $p \times p$ matrices for obtaining local expansions from far-field expansions. These will be defined by the formulae

$$\begin{aligned} T_1(i,j) &= u_j\left(\frac{3}{t_i - 6}\right), \\ T_2(i,j) &= u_j\left(\frac{3}{t_i - 4}\right), \\ T_3(i,j) &= u_j\left(\frac{3}{t_i + 4}\right), \\ T_4(i,j) &= u_j\left(\frac{3}{t_i + 6}\right). \end{aligned} \tag{80}$$

  which are obtained from the formula (44) of Theorem 2.9 applied to the cases $c = -6, d = 1$, $c = -4, d = 1$, $c = 4, d = 1$ and $c = 6, d = 1$.

3.3 Description of the Algorithm

Following is a formal description of an algorithm for the efficient evaluation of expressions of the form (62).

Algorithm 3.1

Step	Complexity	Description
1	$O(1)$	**Comment** [Input problem size, N, and a real number $\varepsilon > 0$.] Set size of Chebyshev expansions $p = \lceil -\log_5(\varepsilon) \rceil$, choose s and set level of refinement $nlevs = \lceil \log_2(N/s) \rceil$.
2	$O(Np)$	**Comment** [Determine far-field expansions for subintervals at finest level.] **do** $i = 1, \ldots, 2^{nlevs}$ Compute p-term far-field expansion $\Phi_{nlevs,i}$ due to the subset of $\{x_j\}$ which lie in subinterval i at level $nlevs$. **enddo**
3	$O(2Np^2/s)$	**Comment** [Determine p-term far-field expansions for each subinterval at every level by shifting and adding far-field expansions of the subinterval's children.] **do** $l = nlevs - 1, \ldots, 1$ **do** $i = 1, \ldots, 2^l$ $\Phi_{l,i} = M_L \cdot \Phi_{l+1,2i-1} + M_R \cdot \Phi_{l+1,2i}$ **enddo** **enddo**

4 $O(8Np^2/s)$ **Comment** [Determine p-term local expansions for each subinterval at every level by first shifting local expansion of the subinterval's parent, and then adding the interactions with the three subintervals which are well-separated from the subinterval, but which have not been accounted for at the parent's level.]

do $l = 1, \ldots, nlevs - 1$
  do $i = 1, \ldots, 2^l$
    $\Psi_{l+1,2i-1} = S_L \cdot \Psi_{l,i} + T_1 \cdot \Phi_{l+1,2i-3} + T_3 \cdot \Phi_{l+1,2i+1} + T_4 \cdot \Phi_{l+1,2i+2}$
    $\Psi_{l+1,2i} = S_R \cdot \Psi_{l,i} + T_1 \cdot \Phi_{l+1,2i-3} + T_2 \cdot \Phi_{l+1,2i-2} + T_4 \cdot \Phi_{l+1,2i+2}$
  enddo
enddo

5 $O(Np)$ **Comment** [Evaluate local expansions for subintervals at finest level.]

do $i = 1, \ldots, 2^{nlevs}$
  Evaluate p-term local expansions $\Psi_{nlevs,i}$ at the subset of $\{y_j\}$ which lie in subinterval i at level $nlevs$.
enddo

6 $O(3Ns)$ **Comment** [Add nearest neighbour interactions which have not yet been accounted for via expansions.]

do $i = 1, \ldots, 2^{nlevs}$
  For each y_k in subinterval i at level $nlevs$, compute interactions with all x_j in subintervals $i-1, i, i+1$, and add to well-separated values.
enddo

Total $O(N \cdot (2p + 10p^2/s + 3s))$

Remark 3.6 The number s can be chosen to minimize the total operation count of the algorithm, which yields $s \approx 2p$. The above algorithm then requires order

$$13 \cdot N \cdot p \tag{81}$$

arithmetic operations.

Remark 3.7 The operation count for Step 6 assumes that the points $\{x_1, \ldots, x_N\}$ are reasonably uniformly distributed. Highly non-uniform distributions are discussed in Section 4.

3.4 A More Efficient Algorithm

Chebyshev expansions are not the most efficient means of representing interactions between well-separated intervals. All the matrix operators of the algorithm are numerically rank-deficient, and can be further compressed by a suitable change of basis. The orthogonal matrices required for this basis change are obtained via singular value decompositions (SVDs) of appropriate matrices.

In this subsection we describe a more efficient version of the one dimensional FMM which is based upon this observation. The algorithm is very similar to Algorithm 3.1 with one important

modification: separate changes of basis are needed for interactions from the left and interactions from the right, so for every subinterval we maintain two sets of expansions, leftward expansions and rightward expansions.

Remark 3.8 SVDs can also be used for the efficient representation and application of much broader classes of smooth linear operators (see Section 8).

We will require some additional notation for the algorithm description of this section. The following denotations use the notation of Section 3.2.

- $\bar{p}$ will be an integer denoting the numerical rank of the operators to be compressed.
- t_j^L will denote the j-th Chebyshev node of order p on the interval $[0,1]$, and t_j^R will denote the j-th Chebyshev node of order p on the interval $[-1,0]$.
- U_L and V_L will denote $p \times \bar{p}$ matrices, each of whose columns forms an orthonormal set, and Σ will denote a $\bar{p} \times \bar{p}$ diagonal matrix such that

$$||U_L \Sigma V_L^T - \mathcal{M}_L|| < \varepsilon \cdot ||\mathcal{M}_L||, \tag{82}$$

where

$$\mathcal{M}_L(i,j) = \frac{1}{3/t_j^L - t_i}, \tag{83}$$

for $i,j = 1,\ldots,p$. U_L and V_L are used to compress leftward expansions, and $U_L \Sigma V_L^T$ is effectively the numerical SVD of $\mathcal{M}_L$.

- U_R and V_R will denote $p \times \bar{p}$ matrices, each of whose columns forms an orthonormal set, such that

$$||U_R \Sigma V_R^T - \mathcal{M}_R|| < \varepsilon \cdot ||\mathcal{M}_R||, \tag{84}$$

where

$$\mathcal{M}_R(i,j) = \frac{1}{3/t_j^R - t_i}, \tag{85}$$

for $i,j = 1,\ldots,p$. An examination of the matrices $\mathcal{M}_L$ and $\mathcal{M}_R$ reveals that U_R and U_L are closely related, and V_R and V_L are closely related. U_R and V_R are used to compress rightward expansions, and $U_R \Sigma V_R^T$ is effectively the numerical SVD of $\mathcal{M}_R$.

- $\Phi_{l,i}^L$ and $\Phi_{l,i}^R$ will be $\bar{p}$-vectors denoting respectively the left and right far-field expansions for subinterval i at level l. These will be defined by the formulae:

$$\begin{aligned} \Phi_{l,i}^L &= U_L^T \cdot \Phi_{l,i}, \\ \Phi_{l,i}^R &= U_R^T \cdot \Phi_{l,i}. \end{aligned} \tag{86}$$

- $\Psi_{l,i}^L$ and $\Psi_{l,i}^R$ will be $\bar{p}$-vectors denoting respectively the left and right local expansions for subinterval i at level l. These will be defined by the formulae:

$$\begin{aligned} \Psi_{l,i}^L &= V_L^T \cdot \Psi_{l,i}, \\ \Psi_{l,i}^R &= V_R^T \cdot \Psi_{l,i}. \end{aligned} \tag{87}$$

- M_L^L, M_R^L, M_L^R and M_R^R will be $\bar{p} \times \bar{p}$ matrices for obtaining left and right far-field expansions for subintervals in terms of the left and right far-field expansions of their children. These will be defined by the formulae

$$\begin{aligned} M_L^L &= U_L^T \cdot M_L \cdot U_L, \\ M_R^L &= U_L^T \cdot M_R \cdot U_L, \\ M_L^R &= U_R^T \cdot M_L \cdot U_R, \\ M_R^R &= U_R^T \cdot M_R \cdot U_R. \end{aligned} \tag{88}$$

- S_L^L, S_R^L, S_L^R and S_R^R will be $\bar{p} \times \bar{p}$ matrices for obtaining left and right local expansions for subintervals in terms of the left and right local expansions of their parent. These will be defined by the formulae

$$\begin{aligned} S_L^L &= V_L^T \cdot S_L \cdot V_L, \\ S_R^L &= V_L^T \cdot S_R \cdot V_L, \\ S_L^R &= V_R^T \cdot S_L \cdot V_R, \\ S_R^R &= V_R^T \cdot S_R \cdot V_R. \end{aligned} \tag{89}$$

- T_1^L and T_2^L will be $\bar{p} \times \bar{p}$ matrices for obtaining left local expansions from right far-field expansions. These will be defined by the formulae

$$\begin{aligned} T_1^L &= V_L^T \cdot T_3 \cdot U_L, \\ T_2^L &= V_L^T \cdot T_4 \cdot U_L. \end{aligned} \tag{90}$$

- T_1^R and T_2^R will be $\bar{p} \times \bar{p}$ matrices denoting for obtaining right local expansions from left far-field expansions. These will be defined by the formulae

$$\begin{aligned} T_1^R &= V_R^T \cdot T_2 \cdot U_R, \\ T_2^R &= V_R^T \cdot T_1 \cdot U_R. \end{aligned} \tag{91}$$

Following is a step-by-step description of a modified version of Algorithm 3.1.

Algorithm 3.2

Step	Complexity	Description
1	$O(1)$	**Comment** [Input problem size, N, and a real number $\varepsilon > 0$.] Set size of Chebyshev expansions $p = \lceil -\log_5(\varepsilon) \rceil$, choose $\bar{p}$, choose s and set level of refinement $nlevs = \lceil \log_2(N/s) \rceil$.
2	$O(2N\bar{p})$	**do** $i = 1, \ldots, 2^{nlevs}$ Form a $\bar{p}$-term left far-field expansion $\Phi^L_{nlevs,i}$. Form a $\bar{p}$-term right far-field expansion $\Phi^R_{nlevs,i}$. **enddo**

3 $O(4N\bar{p}^2/s)$ **do** $l = nlevs - 1, \ldots, 1$

do $i = 1, \ldots, 2^l$

$\Phi^L_{l,i} = M^L_L \cdot \Phi^L_{l+1,2i-1} + M^L_R \cdot \Phi^L_{l+1,2i}$

$\Phi^R_{l,i} = M^L_L \cdot \Phi^R_{l+1,2i-1} + M^R_R \cdot \Phi^R_{l+1,2i}$

enddo

enddo

4 $O(10N\bar{p}^2/s)$ **do** $l = 1, \ldots, nlevs - 1$

do $i = 1, \ldots, 2^l$

$\Psi^R_{l+1,2i-1} = S^R_L \cdot \Psi^R_{l,i} + T^R_1 \cdot \Phi^L_{l+1,2i+1} + T^R_2 \cdot \Phi^L_{l+1,2i+2}$

$\Psi^R_{l+1,2i} = S^R_R \cdot \Psi^R_{l,i} + T^R_1 \cdot \Phi^L_{l+1,2i+2}$

$\Psi^L_{l+1,2i-1} = S^L_L \cdot \Psi^L_{l,i} + T^L_1 \cdot \Phi^R_{l+1,2i-3}$

$\Psi^L_{l+1,2i} = S^L_R \cdot \Psi^L_{l,i} + T^L_1 \cdot \Phi^R_{l+1,2i-2} + T^L_2 \cdot \Phi^R_{l+1,2i-3}$

enddo

enddo

5 $O(2N\bar{p})$ **do** $i = 1, \ldots, 2^{nlevs}$

Evaluate and add left and right $\bar{p}$-term local expansions $\Psi^L_{nlevs,i} + \Psi^R_{nlevs,i}$.

enddo

6 $O(3Ns)$ **do** $i = 1, \ldots, 2^{nlevs}$

For each point in subinterval i at level *nlevs*, compute interactions with all other points in subintervals $i-1, i, i+1$, and add to far-field values.

enddo

Total $O(N \cdot (4\bar{p} + 14\bar{p}^2/s + 3s))$

Remark 3.9 We can choose s to minimize the total operation count of the algorithm, which yields $s \approx 2\bar{p}$. The above algorithm then requires order

$$17 \cdot N \cdot \bar{p} \tag{92}$$

arithmetic operations.

Remark 3.10 The results of our numerical experiments indicate that, for a fixed precision ε and corresponding p, $\bar{p}$ (the numerical rank of the matrix operators to be compressed) is approximately $p/2$. This condition together with (81) and (92) leads us to expect that Algorithm 3.2 will require about two-thirds of the number of arithmetic operations needed by Algorithm 3.1.

Remark 3.11 The operation count for Step 5 assumes that the points $\{x_1, \ldots, x_N\}$ are reasonably uniformly distributed. An adaptive version of this algorithm is described in Section 4, and is capable of handling highly non-uniform distributions while preserving computational efficiency and accuracy.

4 The Adaptive FMM in One Dimension

The algorithms of the previous section have one drawback: their operation count is quite sensitive to the distribution of points, and they become inefficient for highly non-uniform distributions. We now describe an adaptive version of Algorithm 3.2 which overcomes this deficiency, its complexity being $O(N)$ independently of the spacing of the nodes. This versatility is achieved by using different levels of subdivision for different parts of the computational domain. An integer s is fixed, and at each level of refinement, we subdivide only those intervals which contain more than s points. At each level, then, a list of *non-empty* subintervals is maintained whose members are the result of the selective subdivision of intervals at the previous level. This policy eliminates the inefficiency of the non-adaptive version, where, at the finest level of subdivision, we may encounter intervals with very few or very many points. In the non-adaptive version, each interval has two nearest neighbors of the same size, whereas in the adaptive version, intervals are permitted to have neighbors of differing size. Figure 3 depicts a subdivision of the computational domain for a non-uniform distribution of points.

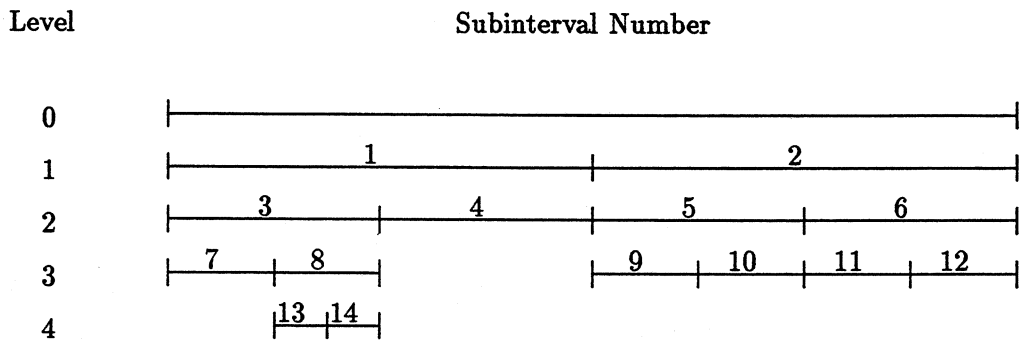
Remark 4.1 The idea of selectively subdividing the computational domain is taken from the two dimensional adaptive FMM of [4]. This algorithm requires a somewhat more elaborate data structure than its non-adaptive counterpart to account for interactions between all the different sized boxes. The adaptive algorithm for problems in one dimension also needs additional bookkeeping to keep track of interactions between all the different sized subintervals. However, the simplified geometry of the real line suggests the use of a somewhat more efficient data structure than that of the two dimensional case (see Figure 3).

Remark 4.2 It is clear that for a fixed machine precision ε, only certain distributions of points will yield meaningful results. For example, the points x_1 and x_2 are indistinguishable if $|x_2 - x_1| < \frac{\varepsilon}{2} \cdot |x_1 + x_2|$. To avoid such cases we will impose that the minimum distance between two points must be greater than $\varepsilon \cdot (b - a)$ where $[a, b]$ is the computational domain. Under this condition, the highest level of refinement of the computational domain is bounded above by $|\log_2(\varepsilon)|$.

4.1 Notation

We require some additional notation for the description of the adaptive algorithm to supplement that of Sections 3.2 and 3.4.

- *nlevs* will denote the level of finest subdivision of any part of the interval $[-1, 1]$.
- For a fixed precision, ε, $m = |\log_2(\varepsilon)|$ will denote the maximum level of refinement of the interval $[-1, 1]$ (see Remark 4.2).
- I_l will denote the set of non-empty subintervals at level l, i.e. the set of subintervals at level l resulting from the bisection of a larger interval at level $l - 1$.
- If subinterval *isub* contains more than s points, it is called a parent subinterval, and *ilchild*(*isub*) and *irchild*(*isub*) will denote its left child and its right child which are the



Interval	Neighbor		Child	
	Left	Right	Left	Right
1	-	2	3	4
2	1	-	5	6
3	-	4	7	8
4	3	5	-	-
5	4	6	9	10
6	5	-	11	12
7	-	8	-	-
8	7	4	13	14
9	4	10	-	-
10	9	11	-	-
11	10	12	-	-
12	11	-	-	-
13	7	14	-	-
14	13	4	-	-

Figure 3: Hierarchy of subintervals for non-uniform distribution.

subintervals resulting from its bisection. Otherwise, *isub* is a called a childless subinterval and *ilchild*(*isub*) and *irchild*(*isub*) are set to 0.

- *ilnbr*(*isub*) and *irnbr*(*isub*) will denote the left and right neighbors for subinterval *isub*, which are the smallest adjacent subintervals at the same level of refinement or a coarser one.
- Φ_i^L and Φ_i^R will be $\bar{p}$-vectors denoting respectively the left and right far-field expansions for subinterval i.
- Ψ_i^L and Ψ_i^R will be $\bar{p}$-vectors denoting respectively the left and right local expansions for subinterval i.

4.2 Description of the Algorithm

This section contains a detailed description and a complexity analysis of an adaptive version of Algorithm 3.2.

Algorithm 4.1
Initialization Step
Comment [Geometrical preprocessing]
Comment [Input problem size, N, and a real number $\varepsilon > 0$.]
Set size of Chebyshev expansions $p = \lceil -\log_5(\varepsilon) \rceil$, choose $\bar{p}$, choose s and set maximum level of refinement $m = \lceil \log_2(N/s) \rceil$.

$I_1 = \{[-1, 0], [0, 1]\}$
do $l = 1, \ldots, m$ **while** I_l is non-empty
  do $isub \in I_l$
    if *isub* contains more than s points **then**
      Add *ilchild*(*isub*) and *irchild*(*isub*) to I_{l+1}.
    else
      ilchild(*isub*) = 0
      irchild(*isub*) = 0
    endif
  enddo
enddo
nlevs = l
do $l = 1, \ldots, nlevs$
  do $isub \in I_l$
    if *isub* is not childless **then**
      ilnbr(*irchild*(*isub*)) = *ilchild*(*isub*)
      irnbr(*ilchild*(*isub*)) = *irchild*(*isub*)
      if *ilnbr*(*isub*) is not childless **then**
        ilnbr(*ilchild*(*isub*)) = *irchild*(*ilnbr*(*isub*))
      else
        ilnbr(*ilchild*(*isub*)) = *ilnbr*(*isub*)
      endif
      if *irnbr*(*isub*) is not childless **then**

$irnbr(irchild(isub)) = ilchild(irnbr(isub))$
else
$irnbr(irchild(isub)) = irnbr(isub)$
endif
endif
enddo
enddo

Step 1
Comment [Upward Pass]

do $l = nlevs, \ldots, 1$
do $isub \in I_l$
if *isub* is a childless subinterval **then**
Form a $\bar{p}$-term far-field expansion Φ^L_{isub}.
Form a $\bar{p}$-term far-field expansion Φ^R_{isub}.
else
$\Phi^L_{isub} = M^L_L \cdot \Phi^L_{ilchild(isub)} + M^L_R \cdot \Phi^L_{irchild(isub)}$
$\Phi^R_{isub} = M^R_L \cdot \Phi^R_{ilchild(isub)} + M^R_R \cdot \Phi^R_{irchild(isub)}$
endif
enddo
enddo

Step 2
Comment [Downward Pass]

do $l = 1, \ldots, nlevs$
do $isub \in I_l$
if *isub* is a childless subinterval **then**
Evaluate and add $\bar{p}$-term local expansions $\Psi^L_{isub} + \Psi^R_{isub}$.
else
$\Psi^L_{ilchild(isub)} = S^R_L \cdot \Psi^L_{isub}$
$\Psi^L_{irchild(isub)} = S^R_R \cdot \Psi^L_{isub}$
$\Psi^R_{ilchild(isub)} = S^R_L \cdot \Psi^R_{isub}$
$\Psi^R_{irchild(isub)} = S^R_R \cdot \Psi^R_{isub}$
if $ilnbr(isub)$ is a childless subinterval **then**
Add contribution of $\Phi^L_{irchild(isub)}$ to each point in $ilnbr(isub)$
Add contribution of each point in $ilnbr(isub)$ to $\Psi^L_{irchild(isub)}$
else
$\Psi^L_{ilchild(isub)} = \Psi^L_{ilchild(isub)} + T^L_1 \cdot \Phi^R_{ilchild(ilnbr(isub))}$
$\Psi^L_{irchild(isub)} = \Psi^L_{irchild(isub)} + T^L_1 \cdot \Phi^R_{irchild(ilnbr(isub))} + T^L_2 \cdot \Phi^R_{ilchild(ilnbr(isub))}$
endif
if $irnbr(isub)$ is a childless subinterval **then**
Add contribution of $\Phi^R_{ilchild(isub)}$ to each point in $irnbr(isub)$
Add contribution of each point in $irnbr(isub)$ to $\Psi^R_{ilchild(isub)}$
else
$\Psi^R_{ilchild(isub)} = \Psi^R_{ilchild(isub)} + T^R_1 \cdot \Phi^R_{ilchild(irnbr(isub))} + T^R_2 \cdot \Phi^R_{irchild(irnbr(isub))}$
$\Psi^R_{irchild(isub)} = \Psi^R_{irchild(isub)} + T^R_1 \cdot \Phi^R_{irchild(irnbr(isub))}$
endif

```
      endif
   enddo
enddo

Step 3
Comment [Direct Interactions]

do isub = 1, ..., nsub
   if isub is childless then
      For each point in subinterval isub, compute interactions with
      all other points in this subinterval and in adjacent childless
      subintervals, and add to far-field values.
   endif
enddo
```

Following is a complexity analysis of Algorithm 4.1. Before presenting a step-by-step breakdown of the operation counts, we need two lemmas. These lemmas provide upper bounds on the numbers of subintervals which can be created by the process of selective subdivision.

Lemma 4.1 *For any subdivision of the computational domain produced by Algorithm 4.1, the number of childless intervals is bounded by*

$$2 \cdot \log_2\left(\frac{1}{\varepsilon}\right) \cdot \frac{N}{s}. \tag{93}$$

Proof. Each parent interval at level l contains more than s points (otherwise it would not be further subdivided). Therefore, the total number of parent intervals at level l is bounded above by N/s. Each parent interval has two children by definition, so the number of childless boxes at any level l is bounded above by $2N/s$. The bound (93) follows from the fact that the number of levels of subdivision is bounded above by $\log_2(1/\varepsilon)$ (see Remark 4.2). □

Lemma 4.2 *For any subdivision of the computational domain produced by Algorithm 4.1, the total number of intervals is bounded by*

$$3 \cdot \log_2\left(\frac{1}{\epsilon}\right) \cdot \frac{N}{s}. \tag{94}$$

Proof. The number of parent intervals at level l is bounded above by N/s. Each of these parent intervals has two children, so the number of childless boxes at any level l is bounded above by $2N/s$. Thus, the total number of intervals at all levels (childless and parent) is bounded by $\log_2(1/\varepsilon) \cdot (N/s + 2N/s)$. □

Following is a step-by-step breakdown of the operation counts of Algorithm 4.1.

Step	Complexity	Description
1	$O(2\bar{p}N)+$	Each point contributes to the left and right $\bar{p}$-term far-field expansions of the childless subinterval in which it lies.
	$O(4\bar{p}^2mN/s)$	For each parent subinterval (there are at most mN/s) we perform 4 $\bar{p} \times \bar{p}$ matrix-vector products.
2	$O(2\bar{p}N)+$	We evaluate and add the left and right $\bar{p}$-term local expansions for each of the N points.
	$O(4\bar{p}^2mN/s)+$	For each parent subinterval (there are at most mN/s) we perform 4 $\bar{p} \times \bar{p}$ matrix-vector products.
	$O(6\bar{p}^2mN/s)+$	For each parent subinterval (there are at most mN/s) we perform at most 6 $\bar{p} \times \bar{p}$ matrix-vector products.
	$O(4\bar{p}mN)$	For each parent subinterval (there are at most mN/s) we perform at most 4 $\bar{p} \times s$ matrix-vector products.
3	$O(3Ns)$	Each of the y_j interacts directly with those x_k which lie in its own subinterval and the two nearest neighbors. There are at most $3s$ of these points x_k.
Total	$O(14\bar{p}^2mN/s + 4\bar{p}mN + 4\bar{p}N + 3Ns)$	

Remark 4.3 We choose $s \approx 2\bar{p}$, as in Algorithm 3.2 (see Remark 3.10). The above algorithm then requires order

$$N \cdot (11\bar{p}m + 10\bar{p}) \tag{95}$$

arithmetic operations.

5 A Fast Algorithm for Polynomial Interpolation

In this section we return to the original problem of this paper: given a set of points $\{x_1, \ldots, x_N\}$ and function values $\{f_1, \ldots, f_N\}$, evaluate the unique interpolating polynomial at the points $\{y_1, \ldots, y_N\}$. Recall from (2) that the Lagrange interpolating polynomial defined by the formula

$$P_N(x) = \sum_{j=1}^{N} f_j \cdot \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{x - x_k}{x_j - x_k} \tag{96}$$

can be rewritten in the form

$$P_N(x) = \prod_{k=1}^{N} (x - x_k) \cdot \sum_{j=1}^{N} \frac{f_j}{x - x_j} \cdot \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{1}{x_j - x_k}. \tag{97}$$

Furthermore,

$$P_N(y_l) = r_l \cdot \sum_{j=1}^{N} \frac{f_j \cdot s_j}{y_l - x_j} \tag{98}$$

for $l = 1, \ldots, n$, where r_l and s_j are defined by the formulae

$$r_l = \prod_{k=1}^{N} (y_l - x_k) = e^{\sum_{k=1}^{N} \ln(y_l - x_k)}, \tag{99}$$

and,

$$s_j = \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{1}{x_j - x_k} = e^{-\sum_{k=1, k\neq j}^{N} \ln(x_j - x_k)}. \tag{100}$$

Observation 5.1 *Sums of the form $\sum \ln(x - x_k)$ can also be evaluated using the algorithms of this paper (see Remark 3.3). The numbers $\{r_l\}$ and $\{s_j\}$ can therefore be computed in $O(N \log(\frac{1}{\varepsilon}))$ operations according to (99) and (100).*

Following is a description of an algorithm for the efficient evaluation of expressions of the form (98).

Algorithm 5.1

Step	Complexity	Description
Init	$O(N \log(\frac{1}{\varepsilon}))$	Compute the numbers $\{r_l\}$ and $\{s_j\}$.
1	$O(N)$	**do** $j = 1, n$ $g_j = f_j \cdot s_j$ **end do**
2	$O(N \log(\frac{1}{\varepsilon}))$	Compute $\tilde{p}_l = \sum_{j=1}^{n} g_j/(y_l - x_j)$ using Algorithm 4.1 (or Algorithm 3.2).
3	$O(N)$	**do** $l = 1, n$ $\tilde{p}_l = \tilde{p}_l \cdot r_l$ **end do**
Total	$O(N \log(\frac{1}{\varepsilon}))$	

Remark 5.2 It is well known that the polynomial interpolant of a function is spectrally accurate when the function is tabulated at Chebyshev or Legendre nodes (which are clustered near the interval ends), whereas the interpolation errors can be arbitrarily large when the function is tabulated at general distributions of points (see, for example, [5], [14]). It is expected that many practical applications of Algorithm 5.1 will assume nonuniformly spaced nodes which are clustered near the extremities of the interval.

6 Applications in Numerical Integration and Differentiation

The fast polynomial interpolation algorithm of this paper can be applied to a variety of problems. One such example is discussed in this section. Here we will consider the folowing problem: given a set of points $\{x_1, \ldots, x_N\}$ and function values $\{f_1, \ldots, f_N\}$, evaluate the integrals and

derivatives of the interpolating polynomial at the points $\{x_k\}$. In other words, we wish to compute

$$\int_{-1}^{x_k} P_N(x)dx \quad \text{and} \quad P_N'(x_k) \tag{101}$$

for $k = 1, \ldots, N$, where P_N is the interpolating polynomial for the function values $\{f_k\}$ at the points $\{x_k\}$, defined by the Lagrange formula

$$P_N(x) = \sum_{j=1}^{N} f_j \cdot \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{x - x_k}{x_j - x_k}. \tag{102}$$

We will make use of the following lemma, which may be found in the appendix to [7]. This lemma describes formulae for the integration and differentiation of Chebyshev expansions.

Lemma 6.1 *Let P_N be a polynomial given by a Chebyshev series*

$$P_N(x) = \sum_{k=0}^{N-1} a_k \cdot T_k(x). \tag{103}$$

Then, the integral of P_N has a series expansion of the form

$$\int_{-1}^{x} P_N(t)dt = \sum_{k=0}^{N} b_k \cdot T_k(x), \tag{104}$$

where

$$\begin{aligned} b_k &= \frac{1}{2k} \cdot (a_{k-1} - a_{k+1}) \quad \text{for } 2 \leq k \leq N, \\ b_1 &= \frac{1}{2} \cdot (2a_0 - a_2), \\ b_0 &= -2 \cdot \sum_{j=1}^{N} (-1)^j \cdot b_j, \end{aligned} \tag{105}$$

and the derivative of P_N has a series expansion of the form

$$\frac{d}{dx} P_N(x) = \sum_{k=0}^{N-2} d_k \cdot T_k(x), \tag{106}$$

where

$$\begin{aligned} d_k &= \sum_{\substack{j=k+1 \\ j+k \text{ odd}}}^{N} j \cdot a_j, \quad \text{for } 1 \leq k \leq N-2, \\ d_0 &= \frac{1}{2} \cdot \sum_{\substack{j=1 \\ j \text{ odd}}}^{N} j \cdot a_j. \end{aligned} \tag{107}$$

Remark 6.1 It can be shown that the process of numerical differentiation via Chebyshev series has a condition number proportional to N^2, whereas the process of numerical integration via Chebyshev series has a condition number bounded by 2. Thus, numerical differentiation of this type is not usually favored when large scale calculations are being performed. On the other hand, numerical integration is virtually insensitive to problem size, and is a powerful tool in the solution of certain classes of differential equations, for example (for a more detailed discussion, see [9]).

The two algorithms described below perform the integration and differentiation of the Lagrange interpolating polynomial of a function which is tabulated at nodes other than Chebyshev. In these descriptions we wil assume that $x_k \in [-1,1]$ for $k = 1,\ldots,N$, and that $t_1,\ldots,t_N$ are Chebyshev nodes of order N on the interval $[-1,1]$.

Algorithm 6.1

Step	Complexity	Description
1	$O(N\log(\frac{1}{\epsilon}))$	Interpolate from $\{x_k\}$ to $\{t_k\}$ using Algorithm 5.1.
2	$O(N\log N)$	Compute Chebyshev coefficients using fast cosine transform.
3	$O(N)$	Integrate Chebyshev series using (105).
4	$O(N\log N)$	Evaluate new series at Chebyshev nodes using fast cosine transform.
5	$O(N\log(\frac{1}{\epsilon}))$	Interpolate from $\{t_k\}$ to $\{x_k\}$ using Algorithm 5.1.
Total	$O(N\cdot\log N + N\cdot\log(\frac{1}{\epsilon}))$	

Algorithm 6.2

Step	Complexity	Description
1	$O(N\log(\frac{1}{\epsilon}))$	Interpolate from $\{x_k\}$ to $\{t_k\}$ using Algorithm 5.1.
2	$O(N\log N)$	Compute Chebyshev coefficients using fast cosine transform.
3	$O(N)$	Differentiate Chebyshev series using (107).
4	$O(N\log N)$	Evaluate new series at Chebyshev nodes using fast cosine transform.
5	$O(N\log(\frac{1}{\epsilon}))$	Interpolate from $\{t_k\}$ to $\{x_k\}$ using Algorithm 5.1.
Total	$O(N\cdot\log N + N\cdot\log(\frac{1}{\epsilon}))$	

7 Implementation and Numerical Results

We have written FORTRAN implementations of the algorithms of this paper using double precision arithmetic, and have applied these programs to a variety of situations.

Two technical details of our implementations appear to be worth mentioning here:

1. Each implementation consists of two main subroutines: the first is an initialization stage in which the elements of the various matrices employed by the algorithms are precomputed and stored, and the second is an evaluation stage in which these matrices are applied. Successive application of the linear transformations to multiple vectors requires the initialization to be performed only once.

2. The parameters for each algorithm were chosen to retain maximum precision while minimizing the CPU time requirements. The values we used for the numerical examples of this section were $p = 16$, $\hat{p} = 9$, $s = 16$ and $nlevs = \log_2(n/s)$.

Our implementations of the algorithms of this paper have been tested on the the Sun SPARCstation 1 for a variety of input data. Four experiments are described in this section, and their results are summarized in Tables 1–9. These tables contain error estimates and CPU time requirements for the algorithms, with all computations performed in double precision arithmetic.

The table entries are described below.

- The first column in each table contains the problem size N, which was chosen to be a power of 2 ranging from 64 to 4096 for each example.
- The second and third columns in each table contain the relative ∞-norm error E_∞ and the relative 2-norm error E_2 for each result.
- The fourth and fifth columns in each table contain CPU timings for the initialization and evaluation stages of the algorithm.
- The sixth column in each of Tables 1–4 contains CPU timings for the corresponding direct calculation.
- The last column in each of Tables 1–4 contains CPU timings for an FFT of the same size.

Following are the descriptions of the experiments, and the tables of numerical results.

Example 1.

The purpose of this example is to demonstrate the performance of the one dimensional FMM algorithms of this paper when applied to uniform distributions of points in the interval $[-1, 1]$. We considered the problem of evaluating the expressions

$$f_j = \sum_{k=1}^{N} \frac{\alpha_k}{y_j - x_k} \tag{108}$$

for $j = 1, \ldots, N$, where the numbers $\{x_k\}$ and $\{y_k\}$ were chosen according to the formulae

$$x_k = -1 + \frac{2k-1}{N}, \tag{109}$$

$$y_k = -1 + \frac{2 \cdot (k + 0.1 \cdot \delta_k) - 1}{N}, \tag{110}$$

for $k = 1, \ldots, N$, and $\{\delta_k\}$ were random numbers in $[-1, 1]$. In addition, $\{\alpha_k\}$ were randomly chosen from the interval $[0, 1]$. This calculation was performed in three ways:

1. via Algorithm 3.1,
2. via Algorithm 3.2, and,
3. via a direct implementation of the formula (108).

Results of this experiment are presented in Tables 1 and 2.

Example 2.

Here we again considered the problem of evaluating (108), as in Example 1, but we replaced the equispaced nodes with non-uniform distributions, and applied both the adaptive and non-adaptive versions of the one dimensional FMM algorithms to this problem. The numbers $\{x_k\}$ were chosen to be Legendre nodes of order N (i.e. the roots of the N-th order Legendre polynomial) on the interval $[-1,1]$, and $\{y_k\}$ were chosen to be Chebyshev nodes of order N on the same interval. Once again, $\{\alpha_k\}$ were randomly chosen from the interval $[0,1]$. This calculation was performed in three ways:

1. via Algorithm 3.2,
2. via Algorithm 4.1, and,
3. via a direct implementation of the formula (108).

Results of this experiment are presented in Tables 3 and 4.

Example 3.

Here we considered the problem of evaluating the Lagrange interpolant of the function

$$f(x) = e^{-4x^2} \tag{111}$$

tabulated at Legendre nodes $\{x_1, \ldots, x_N\}$ on the interval $[-1,1]$. The interpolant, defined by the formula

$$P_N(y) = \sum_{j=1}^{N} f(x_j) \cdot \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{y - x_k}{x_j - x_k}, \tag{112}$$

was evaluated at Chebyshev nodes $\{y_1, \ldots, y_N\}$ on the same interval. This calculation was performed in three ways:

1. via the formula

$$P_N(y_l) = e^{\sum_{k=1}^{N} \ln(y_l - x_k)} \cdot \sum_{j=1}^{N} \frac{f(x_j)}{y_l - x_j} \cdot e^{-\sum_{k=1, k \neq j}^{N} \ln(x_j - x_k)} \tag{113}$$

   using Algorithm 5.1,

2. via a direct implementation of the formula

$$P_N(y_l) = \prod_{k=1}^{N} (y_l - x_k) \cdot \sum_{j=1}^{N} \frac{f(x_j)}{y_l - x_j} \cdot \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{1}{x_j - x_k}, \tag{114}$$

   and,

3. via a direct implementation of the formula

$$P_N(y_l) = e^{\sum_{k=1}^{N} \ln(y_l - x_k)} \cdot \sum_{j=1}^{N} \frac{f(x_j)}{y_l - x_j} \cdot e^{-\sum_{k=1, k \neq j}^{N} \ln(x_j - x_k)}. \tag{115}$$

Results of this experiment are presented in Tables 5–7.

Remark 7.1 Table 6 only contains results for $N \leq 1024$ because for larger problem sizes the computation of the products in the expression (114) generates overflow and underflow errors for this particular computer software and hardware.

Example 4.
In this example we tested the integration and differentiation algorithms of Section 6 on the functions

$$f(x) = 4x \cdot (x^2 - 1), \tag{116}$$

and

$$g(x) = \int_{-1}^{x} f(t)dt = (x^2 - 1)^2, \tag{117}$$

which were tabulated at Legendre nodes of order N on the interval $[-1, 1]$. Algorithm 6.1 was used to compute the integrals of f, and Algorithm 6.2 was used to compute the derivatives of g at these nodes. Results of this experiment are presented in Tables 8 and 9.

The following observations can be made from Tables 1–9, and are in agreement with results of our more extensive experiments for the particular computer architecture, implementations, and values of N under consideration.

1. The algorithms permit high accuracy to be attained, and the observed errors are in accordance with the theoretically obtained error bounds.
2. The CPU timings for each algorithm grow linearly, as expected, with the problem size N.
3. Algorithms 3.2 and 4.1 are about 10 times as costly as an FFT of the same size.
4. The algorithms break even with the corresponding direct methods at around $N = 32$ if the initialization time is ignored, and at about $N = 512$ if it is included.
5. The evaluation stage for Algorithm 3.1 is approximately 50 percent slower than the evaluation stage for Algorithm 3.2, as expected (see Remark 3.10). However, if the initialization times are included, Algorithm 3.1 is more than twice as fast as Algorithm 3.2. Thus, Algorithm 3.1 should be used whenever the linear transformation of this example is to be applied to a single vector, and Algorithm 3.2 should be used whenever the same linear transformation is to be applied to many different vectors.
6. Tables 3 and 4 indicate that the evaluation stage of the adaptive algorithm is only about 30 percent faster than the non-adaptive one for Legendre nodes. These nodes were chosen because functions tabulated there are very well approximated by their polynomial interpolants (see Remark 5.2). For highly non-uniform distributions, however, the adaptive algorithm will be significantly faster.
7. It is apparent from Tables 6 and 7 that the first of the direct methods used in Example 3 is the more accurate and more efficient of the two. However, we could not use this method for values of N larger than 1024 due to numerical instability (see Remark 7.1).

N	Errors		Timings (sec.)			
	E_∞	E_2	Init.	Eval.	Direct	FFT
64	0.176 E-14	0.192 E-14	0.04	0.017	0.02	0.001
128	0.289 E-15	0.631 E-15	0.07	0.043	0.08	0.002
256	0.719 E-15	0.133 E-14	0.13	0.101	0.34	0.005
512	0.888 E-16	0.649 E-15	0.27	0.225	1.34	0.012
1024	0.104 E-14	0.104 E-14	0.48	0.465	5.37	0.026
2048	0.176 E-14	0.145 E-14	0.94	0.958	21.82	0.059
4096	0.321 E-14	0.330 E-14	1.93	1.953	87.79	0.132

Table 1: Example 1, Numerical Results for Algorithm 3.1.

N	Errors		Timings (sec.)			
	E_∞	E_2	Init.	Eval.	Direct	FFT
64	0.204 E-14	0.298 E-14	0.22	0.012	0.02	0.001
128	0.351 E-15	0.560 E-15	0.32	0.030	0.08	0.002
256	0.998 E-15	0.332 E-14	0.49	0.070	0.34	0.005
512	0.517 E-15	0.553 E-14	0.94	0.147	1.34	0.012
1024	0.944 E-15	0.679 E-14	1.71	0.306	5.37	0.026
2048	0.190 E-14	0.955 E-14	3.40	0.630	21.82	0.059
4096	0.401 E-14	0.328 E-13	6.58	1.283	87.79	0.132

Table 2: Example 1, Numerical Results for Algorithm 3.2.

N	Errors		Timings (sec.)			
	E_∞	E_2	Init.	Eval.	Direct	FFT
64	0.813 E-15	0.202 E-14	0.29	0.012	0.02	0.001
128	0.612 E-15	0.180 E-14	0.39	0.033	0.08	0.002
256	0.623 E-15	0.197 E-14	0.63	0.079	0.34	0.005
512	0.725 E-15	0.203 E-14	1.12	0.176	1.34	0.012
1024	0.108 E-14	0.588 E-14	2.06	0.378	5.32	0.026
2048	0.287 E-14	0.111 E-13	4.09	0.795	21.58	0.059
4096	0.323 E-14	0.892 E-14	8.05	1.727	88.71	0.132

Table 3: Example 2, Numerical Results for Algorithm 3.2.

N	Errors		Timings (sec.)			
	E_∞	E_2	Init.	Eval.	Direct	FFT
64	0.746 E-15	0.171 E-14	0.32	0.012	0.02	0.001
128	0.758 E-15	0.216 E-14	0.50	0.032	0.08	0.002
256	0.146 E-14	0.316 E-14	0.72	0.071	0.34	0.005
512	0.325 E-14	0.897 E-14	1.15	0.145	1.34	0.012
1024	0.128 E-13	0.264 E-13	2.02	0.295	5.32	0.026
2048	0.700 E-14	0.211 E-13	3.88	0.606	21.58	0.059
4096	0.775 E-14	0.318 E-13	7.50	1.209	88.71	0.132

Table 4: Example 2, Numerical Results for Algorithm 4.1.

N	Errors		Timings (sec.)	
	E_∞	E_2	Init.	Eval.
64	0.351 E-13	0.330 E-13	0.44	0.012
128	0.542 E-13	0.453 E-13	0.74	0.032
256	0.628 E-13	0.340 E-13	1.30	0.072
512	0.877 E-13	0.459 E-13	2.49	0.147
1024	0.136 E-12	0.497 E-13	4.97	0.299
2048	0.160 E-12	0.555 E-13	10.30	0.611
4096	0.204 E-12	0.692 E-13	20.99	1.239

Table 5: Example 3, Numerical Results for Algorithm 5.1.

N	E_∞	E_2	Timings (sec.)
64	0.134 E-14	0.789 E-15	0.04
128	0.222 E-14	0.126 E-14	0.14
256	0.688 E-14	0.263 E-14	0.58
512	0.161 E-13	0.598 E-14	2.33
1024	0.318 E-13	0.105 E-13	9.52
2048	-	-	-
4096	-	-	-

Table 6: Example 3, Numerical Results for Direct Method 1.

N	E_∞	E_2	Timings (sec.)
64	0.740 E-14	0.433 E-14	0.14
128	0.266 E-13	0.141 E-13	0.54
256	0.976 E-13	0.319 E-13	2.19
512	0.198 E-12	0.951 E-13	8.70
1024	0.619 E-12	0.295 E-12	34.92
2048	0.199 E-11	0.789 E-12	141.84
4096	0.588 E-11	0.218 E-11	567.92

Table 7: Example 3, Numerical Results for Direct Method 2.

N	Errors		Timings (sec.)	
	E_∞	E_2	Init.	Eval.
64	0.766 E-13	0.384 E-13	0.90	0.031
128	0.135 E-12	0.419 E-13	1.48	0.077
256	0.309 E-12	0.529 E-13	2.62	0.155
512	0.383 E-12	0.637 E-13	5.03	0.329
1024	0.268 E-12	0.596 E-13	10.01	0.661
2048	0.566 E-12	0.729 E-13	20.36	1.325
4096	0.683 E-12	0.102 E-12	40.50	2.651

Table 8: Example 4, Numerical Results for Algorithm 6.1.

N	Errors		Timings (sec.)	
	E_∞	E_2	Init.	Eval.
64	0.101 E-10	0.225 E-11	0.94	0.032
128	0.520 E-10	0.890 E-11	1.53	0.079
256	0.221 E-09	0.320 E-10	2.71	0.164
512	0.348 E-08	0.338 E-09	4.98	0.331
1024	0.272 E-07	0.193 E-08	10.04	0.716
2048	0.190 E-06	0.960 E-08	20.85	1.309
4096	0.801 E-06	0.528 E-07	40.90	2.621

Table 9: Example 4, Numerical Results for Algorithm 6.2.

8. Tables 8 and 9 show rapidly growing errors for spectral differentiation, but not for spectral integration, demonstrating the well known difference in stability between these two processes (see Remark 6.1). Timings for the two methods are similar, as expected.

9. The initialization stage is more costly than the evaluation stage for all of the algorithms. Implementing the algorithms in two stages gives considerable time savings whenever the same linear transformation is to be applied to multiple vectors.

8 Conclusions and Generalizations

In this paper, we have described a collection of algorithms which includes:

1. efficient versions of the Fast Multipole Method in one dimension,

2. a fast algorithm for polynomial interpolation on the line, and

3. fast algorithms for the integration and differentiation of functions tabulated at nodes other than Chebyshev.

The number of arithmetic operations required by each algorithm is proportional to either N or $N \cdot \log N$, where the constant of proportionality depends on the desired accuracy.

Several obvious generalizations of the results of this paper are discussed below.

1. The algorithms described in Sections 3 and 4 are all designed for evaluating sums of the form

$$f(x) = \sum_{k=1}^{N} \frac{\alpha_k}{x - x_k}. \tag{118}$$

However, they can be modified to evaluate expressions of a more general form, given by the formula

$$f(x) = \sum_{k=1}^{N} \alpha_k \cdot \phi(x - x_k), \tag{119}$$

where ϕ is singular at 0 but smooth everywhere else. Examples of functions with this property include $\phi(x) = \log(x)$ and $\phi(x) = 1/x^2$, which are readily obtained by integrating and differentiating the expression (64). Another example of interest is $\phi(x) = 1/\tan(x)$ which arises in the formulation of the trigonometric interpolation problem. This case is currently being investigated, and results will be reported at a later date.

While the algorithmic procedures for different kernels ϕ will be virtually identical, different sets of formulae will be needed for the manipulation of the Chebyshev far-field and local expansions which are required by the algorithms. These formulae are obtained by constructing analogs of Theorems 2.5, 2.7, 2.8, 2.9 and 2.10 for the particular function ϕ.

2. The problems considered in this paper all involve the evaluation of an N-term series at N points. Straightforward modifications to the algorithms of this paper will allow the efficient evaluation of these N-term series at M points, where $M \neq N$. These modifications have been implemented.

3. The use of the Singular Value Decomposition in the approximation of smooth functions can be extended to many families of linear operators. Specifically, this approach may be used to substantially accelerate the Fast Laplace Transform of [13] and the Fast Legendre Transform of [2].

References

[1] B. ALPERT, G. BEYLKIN, R. COIFMAN AND V. ROKHLIN, *Wavelet-like Bases for the Fast Solution of Second-Kind Integral Equations*, SIAM J. Sci. Stat. Comp., 14 (1993).

[2] B. ALPERT AND V. ROKHLIN, *A Fast Algorithm for the Evaluation of Legendre Expansions*, Technical Report 671, Yale Computer Science Department, 1988.

[3] G. BEYLKIN, R. COIFMAN AND V. ROKHLIN, *Fast Wavelet Transforms and Numerical Algorithms I*, Comm. on Pure and Applied Mathematics, 44 (1991), pp. 141-183.

[4] J. CARRIER, L. GREENGARD AND V. ROKHLIN, *A Fast Adaptive Multipole Algorithm for Particle Simulations*, SIAM J. Sci. Stat. Comp., 9 (1988), pp. 669–686.

[5] G. DAHLQUIST AND A. BJORCK, *Numerical Methods*, Prentice Hall Inc., Englewood Cliffs, N.J., 1974.

[6] D. GOTTLIEB, M. Y. HUSSAINI AND S. ORSZAG, in *Spectral Methods for Partial Differential Equations*, edited by R. G. Voigt, D. Gottlieb and M. Y. Hussaini, SIAM, Philadelphia PA, 1984, p.1.

[7] D. GOTTLIEB AND S. ORSZAG, *Numerical Analysis of Spectral Methods*, SIAM, Philadelphia PA, 1977.

[8] I. S. GRADSHTEYN AND I. M. RYZHIK, *Table of Integrals, Series and Products*, Academic Press Inc., 1980.

[9] L. GREENGARD, *Spectral Integration and Two-Point Boundary Value Problems*, Technical Report 646, Yale Computer Science Department, 1988.

[10] L. GREENGARD AND V. ROKHLIN, *A Fast Algorithm for Particle Simulations*, J. Comp. Phys., 73 (1987), pp. 325–348.

[11] M. GU AND S. C. EISENSTAT, *A Divide-And-Conquer Algorithm for the Symmetric Tridiagonal Eigenproblem*, Technical Report 932, Yale Computer Science Department, 1992.

[12] M. GU AND S. C. EISENSTAT, *A Divide-And-Conquer Algorithm for the Bidiagonal SVD*, Technical Report 933, Yale Computer Science Department, 1992.

[13] V. ROKHLIN, *A Fast Algorithm for the Discrete Laplace Transformation*, Journal of Complexity, 4 (1988), pp. 12-32.

[14] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer Verlag, New York, 1980.