A fast simple algorithm for computing the potential of charges on a line

Zydrunas Gimbutas

National Institute of Standards and Technology, Boulder, CO 80305, USA

Nicholas F. Marshall[1]

Department of Mathematics, Princeton University, Princeton, NJ 08540, USA

Vladimir Rokhlin[2]

Program in Applied Mathematics, Yale University, New Haven, CT 06511, USA

Abstract

We present a fast method for evaluating expressions of the form

$$u_j = \sum_{i=1, i\neq j}^{n} \frac{\alpha_i}{x_i - x_j}, \quad \text{for} \quad j = 1, \ldots, n,$$

where α_i are real numbers, and x_i are points in a compact interval of $\mathbb{R}$. This expression can be viewed as representing the electrostatic potential generated by charges on a line in $\mathbb{R}^3$. While fast algorithms for computing the electrostatic potential of general distributions of charges in $\mathbb{R}^3$ exist, in a number of situations in computational physics it is useful to have a simple and extremely fast method for evaluating the potential of charges on a line; we present such a method in this paper, and report numerical results for several examples.

Keywords: Fast multipole method, Chebyshev system, generalized Gaussian quadrature

Email addresses: zydrunas.gimbutas@nist.gov (Zydrunas Gimbutas), nicholas.marshall@princeton.edu (Nicholas F. Marshall), vladimir.rokhlin@yale.edu (Vladimir Rokhlin)

[1]N.F.M. was supported in part by NSF DMS-1903015

[2]V.R. was supported in part by AFOSR FA9550-16-1-0175 and ONR N00014-14-1-0797.

2010 MSC: 31C20 (primary), 41A55, 41A50 (secondary)

1. Introduction and motivation

1.1. Introduction

In this paper, we describe a simple fast algorithm for evaluating expressions of the form

$$u_j = \sum_{i=1, i\neq j}^{n} \frac{\alpha_i}{x_i - x_j}, \quad \text{for} \quad j = 1, \ldots, n, \tag{1}$$

where α_i are real numbers, and x_i are points in a compact interval of $\mathbb{R}$. This expression can be viewed as representing the electrostatic potential generated by charges on a line in $\mathbb{R}^3$. We remark that fast algorithms for computing the electrostatic potential generated by general distributions of charges in $\mathbb{R}^3$ exist, see for example the Fast Multipole Method [9] whose relation to the method presented in this paper is discussed in §1.2. However, in a number of situations in computational physics it is useful to have a simple and extremely fast method for evaluating the potential of charges on a line; we present such a method in this paper. Under mild assumptions the presented method involves $\mathcal{O}(n \log n)$ operations and has a small constant. The method is based on writing the potential $1/r$ as

$$\frac{1}{r} = \int_0^\infty e^{-rt} dt.$$

We show that there exists a small set of quadrature nodes $t_1, \ldots, t_m$ and weights $w_1, \ldots, w_m$ such that for a large range of values of r we have

$$\frac{1}{r} \approx \sum_{j=1}^{m} w_j e^{-rt_j}, \tag{2}$$

see Lemma 4.5, which is a quantitative version of (2). Numerically the nodes $t_1, \ldots, t_m$ and weights $w_1, \ldots, w_m$ are computed using a procedure for constructing generalized Gaussian quadratures, see §5.2. An advantage of representing $1/r$ as a sum of exponentials is that the translation operator

$$\frac{1}{r} \mapsto \frac{1}{r + r'} \tag{3}$$

can be computed by taking an inner product of the weights $(w_1, \ldots, w_m)$ with a diagonal transformation of the vector $(e^{-rt_1}, \ldots, e^{-rt_m})$. Indeed, we have

$$\frac{1}{r+r'} \approx \sum_{j=1}^{m} w_j e^{-(r+r')t_j} = \sum_{j=1}^{m} w_j e^{-r't_j} e^{-rt_j}. \tag{4}$$

The algorithm described in §3 leverages the existence of this diagonal translation operator to efficiently evaluate (1).

1.2. Relation to past work

We emphasize that fast algorithms for computing the potential generated by arbitrary distributions of charges in $\mathbb{R}^3$ exist. An example of such an algorithm is the Fast Multipole Method that was introduced by [9] and has been extended by several authors including [7, 10, 16]. In this paper, we present a simple scheme for the special case where the charges are on a line, which occurs in a number of numerical calcuations, see 1.3. The presented scheme has a much smaller runtime constant compared to general methods, and is based on the diagonal form (4) of the translation operator (3). The idea of using the diagonal form of this translation operator to accelerate numerical computations has been studied by several authors; in particular, the diagonal form is used in algorithms by Dutt, Gu and Rokhlin [6], and Yavin and Rokhlin [22] and was subsequently studied in detail by Beylkin and Monzón [1, 2].

The current paper improves upon these past works by taking advantage of robust generalized Gaussian quadrature codes [4] that were not previously available; these codes construct a quadrature rule that is exact for functions in the linear span of a given Chebyshev system, and can be viewed as a constructive version of Lemma 4.2 of Kreĭn [13]. The resulting fast algorithm presented in §3 simplifies past approaches, and has a small runtime constant; in particular, its computational cost is similar to the computational cost of 5-10 Fast Fourier Transforms on data of a similar length, see 5.

1.3. Motivation

Expressions of the form (1) appear in a number of situations in computational physics. In particular, such expressions arise in connection with the Hilbert Transform

$$Hf(x) = \lim_{\varepsilon \to 0} \frac{1}{\pi} \int_{|x-y| \geq \varepsilon} \frac{f(y)}{y-x} dy.$$

For example, the computation of the projection $P_m f$ of a function f onto the first $m+1$ functions in a family of orthogonal polynomials can be reduced to an expression of the form (1) by using the Christoffel–Darboux formula, which is related to the Hilbert transform; we detail the reduction of $P_m f$ to an expression of the form (1) in the following.

Let $\{p_k\}_{k=0}^{\infty}$ be a family of monic polynomials that are orthogonal with respect to the weight $w(x) \geq 0$ on $(a, b) \subseteq \mathbb{R}$. Consider the projection operator

$$P_m f(x) := \int_a^b \sum_{k=0}^{m} \frac{p_k(x)p_k(y)}{h_k} f(y)w(y)dy,$$

where $h_k := \int_a^b p_k(x)^2 w(x)dx$. Let $x_1, \ldots, x_n$ and $w_1, \ldots, w_n$ be the $n > m/2$ point Gaussian quadrature nodes and weights associated with $\{p_k\}_{k=0}^{\infty}$, and set

$$u_j := \sum_{i=1}^{n} \sum_{k=0}^{m} \frac{p_k(x_j)p_k(x_i)}{h_k} f(x_i)w(x_i), \quad \text{for} \quad j = 1, \ldots, n. \tag{5}$$

By construction the polynomial that interpolates the values $u_1, \ldots, u_n$ at the points $x_1, \ldots, x_n$ will accurately approximate $P_m f$ on (a, b) when f is sufficiently smooth, see for example §7.4.6 of Dahlquist and Björck [5]. Directly evaluating (5) would require $\Omega(n^2)$ operations. In contrast, the algorithm of this paper together with the Christoffel–Darboux Formula can be used to evaluate (5) in $\mathcal{O}(n \log n)$ operations. The Christoffel-Darboux formula states that

$$\sum_{k=0}^{m} \frac{p_k(x)p_k(y)}{h_k} = \frac{1}{h_m} \frac{p_{m+1}(x)p_m(y) - p_m(x)p_{m+1}(y)}{x - y}, \tag{6}$$

see §18.2(v) of [17]. Using (6) to rewrite (5) yields

$$u_j = \frac{1}{h_m} \left(f(x_j) + \sum_{i=1, i \neq j}^{m} \frac{p_{m+1}(x_j)p_m(x_i) - p_m(x_j)p_{m+1}(x_i)}{x_j - x_i} f(x_i)w(x_i) \right), \tag{7}$$

where we have used the fact that the diagonal term of the double summation is equal to $f(x_j)/h_m$. The summation in (7) can be rearranged into two expressions of the form (1), and thus the method of this paper can be used to compute a representation of $P_m f$ in $\mathcal{O}(n \log n)$ operations.

Remark 1.1. Analogs of the Christoffel–Darboux formula hold for many other families of functions; for example, if $J_\nu(w)$ is a Bessel function of the first kind, then we have

$$\sum_{k=1}^{\infty} 2(\nu+k)J_{\nu+k}(w)J_{v+k}(z) = \frac{wz}{w-z}\left(J_{\nu+1}(w)J_\nu(z) - J_\nu(w)J_{\nu+1}(z)\right),$$

see [21]. This formula can be used to write a projection operator related to Bessel functions in an analogous form to (7), and the algorithm of this paper can be similarly applied

Remark 1.2. A simple modification of the algorithm presented in this paper can be used to evaluate more general expressions of the form

$$v_j = \sum_{i=1}^{n} \frac{\alpha_i}{x_i - y_j}, \quad \text{for} \quad j = 1, \ldots, m,$$

where $x_1, \ldots, x_n$ are source points, and $y_1, \ldots, y_m$ are target points. For simplicity, this paper focuses on the case where the source and target points are the same, which is the case in the projection application described above.

2. Main result

2.1. Main result

Our principle analytical result is the following theorem, which provides precise accuracy and computational complexity guarantees for the algorithm presented in this paper, which is detailed in §3.

Theorem 2.1. *Let $x_1 < \ldots < x_n \in [a, b]$ and $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ be given. Set*

$$u_j := \sum_{i=1, i\neq j}^{n} \frac{\alpha_i}{x_i - x_j}, \quad \textit{for} \quad j = 1, \ldots, n.$$

Given $\delta > 0$ and $\varepsilon > 0$, the algorithm described in §3 computes values $\tilde{u}_j$ such

$$\frac{|\tilde{u}_j - u_j|}{\sum_{i=1}^{n} |\alpha_i|} \leq \varepsilon, \quad \textit{for} \quad j = 1, \ldots, n \tag{8}$$

in $\mathcal{O}\left(n \log(\delta^{-1}) \log(\varepsilon^{-1}) + N_\delta\right)$ operations, where

$$N_\delta := \sum_{j=1}^{n} \#\{x_i : |x_j - x_i| < \delta(b-a)\}. \tag{9}$$

The proof of Theorem 2.1 is given in §4. Under typical conditions, the presented algorithm involves $\mathcal{O}(n \log n)$ operations. The following corollary describes a case of interest, where the points $x_1, \ldots, x_n$ are Chebyshev nodes for a compact interval $[a, b]$ (we define Chebyshev nodes in §4.2).

Corollary 2.1. *Fix $\varepsilon = 10^{-15}$, and let the points $x_1, \ldots, x_n$ be Chebyshev nodes on $[a, b]$. If $\delta = 1/n$, then the algorithm of §3 involves $\mathcal{O}(n \log n)$ operations.*

The proof of Corollary 2.1 is given in §4.4. The following corollary states that a similar result holds for uniformly random points.

Corollary 2.2. *Fix $\varepsilon = 10^{-15}$, and suppose that $x_1, \ldots, x_n$ are sampled uniformly at random from $[a, b]$. If $\delta = 1/n$, then the algorithm of §3 involves $\mathcal{O}(n \log n)$ operations with high probability.*

The proof of Corollary 2.2 is immediate from standard probabilistic estimates. The following remark describes an adversarial configuration of points.

Remark 2.1. Fix $\varepsilon > 0$, and let $x_1, \ldots, x_{2n}$ be a collection of points such that $x_1, \ldots, x_n$ and $x_{n+1}, \ldots, x_{2n}$ are evenly spaced in $[0, 2^{-n}]$ and $[1-2^{-n}, 1]$, respectively, that is

$$x_j = 2^{-n} \left(\frac{j-1}{n-1} \right), \quad \text{and} \quad x_{n+j} = 1 + 2^{-n} \left(\frac{j-n}{n-1} \right), \quad \text{for} \quad j = 1, \ldots, n.$$

We claim that Theorem 2.1 cannot guarantee a complexity better than $\mathcal{O}(n^2)$ for this configuration of points. Indeed, if $\delta \geq 2^{-n}$, then $N_\delta \geq n^2/2$, and if $\delta < 2^{-n}$, then $\log_2(\delta^{-1}) > n$. In either case

$$n \log(\delta^{-1}) + N_\delta = \Omega(n^2).$$

This complexity is indicative of the performance of the algorithm for this point configuration; the reason that the algorithm performs poorly is that structures exist at two different scales. If such a configuration were encountered in practice, it would be possible to modify the algorithm of §3 to also involve two different scales to achieve evaluation in $\mathcal{O}(n \log n)$ operations.

3. Algorithm

3.1. High level summary

The algorithm involves passing over the points $x_1, \ldots, x_n$ twice. First, we pass over the points in ascending order and compute

$$\tilde{u}_j^+ \approx \sum_{i=1}^{j-1} \frac{\alpha_i}{x_i - x_j}, \quad \text{for} \quad j = 1, \ldots, n, \tag{10}$$

and second, we pass over the points in descending order and compute

$$\tilde{u}_j^- \approx \sum_{i=j+1}^{n} \frac{\alpha_i}{x_i - x_j}, \quad \text{for} \quad j = 1, \ldots, n. \tag{11}$$

Finally, we define $\tilde{u}_j := \tilde{u}_j^+ + \tilde{u}_j^-$ for $j = 1, \ldots, n$ such that

$$\tilde{u}_j \approx \sum_{i=1, i\neq j}^{n} \frac{\alpha_i}{x_i - x_j}, \quad \text{for} \quad j = 1, \ldots, n.$$

We call the computation of $\tilde{u}_1^+, \ldots, \tilde{u}_n^+$ the forward pass of the algorithm, and the computation of $\tilde{u}_1^-, \ldots, \tilde{u}_n^+$ the backward pass of the algorithm. The forward pass of the algorithm computes the potential generated by all points to the left of a given point, while the backward pass of the algorithm computes the potential generated by all points to the right of a given point. In §3.2 and §3.3 we give an informal and detailed description of the forward pass of the algorithm. The backward pass of the algorithm is identical except it considers the points in reverse order.

3.2. Informal description

In the following, we give an informal description of the forward pass of the algorithm that computes

$$\tilde{u}_j^+ \approx \sum_{i=1}^{j-1} \frac{\alpha_i}{x_i - x_j}, \quad \text{for} \quad j = 1, \ldots, n.$$

Assume that a small set of nodes $t_1, \ldots, t_m$ and weights $w_1, \ldots, w_m$ such that

$$\frac{1}{r} \approx \sum_{i=1}^{m} w_i e^{-r t_i} \quad \text{for} \quad r \in [\delta(b-a), b-a], \tag{12}$$

where $\delta > 0$ is given and fixed. The existence and computation of such nodes and weights is described in §4.4 and §5.2. We divide the sum defining u_j^+ into two parts:

$$\tilde{u}_j^+ \approx \sum_{i=1}^{j_0} \frac{\alpha_i}{x_i - x_j} + \sum_{i=j_0+1}^{j-1} \frac{\alpha_i}{x_i - x_j}, \tag{13}$$

where $j_0 = \max\big\{i : x_j - x_i > \delta(b-a)\big\}$. By definition, the points $x_1, \ldots, x_{j_0}$ are all distance at least $\delta(b-a)$ from x_j. Therefore, by (12)

$$\tilde{u}_j^+ \approx -\sum_{i=1}^{j_0} \sum_{k=1}^{m} w_k \alpha_i e^{-(x_j - x_i)t_k} + \sum_{i=j_0+1}^{j-1} \frac{\alpha_i}{x_i - x_j}.$$

If we define

$$g_k(j_0) = \sum_{i=1}^{j_0} \alpha_i e^{-(x_{j_0} - x_i)t_k}, \quad \text{for} \quad k = 1, \ldots, m, \tag{14}$$

then it is straightforward to verify that

$$\tilde{u}_j^+ \approx -\sum_{k=1}^{m} w_k g_k(j_0) e^{-(x_j - x_{j_0})t_k} + \sum_{i=j_0+1}^{j-1} \frac{\alpha_i}{x_i - x_j}. \tag{15}$$

Observe that we can update $g_k(j_0)$ to $g_k(j_0+1)$ using the following formula

$$g_k(j_0+1) = \alpha_{j_0} + e^{-(x_{j_0+1} - x_{j_0})t_k} g_k(j_0), \quad \text{for} \quad k = 1, \ldots, m. \tag{16}$$

We can now summarize the algorithm for computing $\tilde{u}_1^+, \ldots, \tilde{u}_n^+$. For each j, we compute $\tilde{u}_j^+$ by the following three steps:

1. Update $g_1, \ldots, g_m$ as necessary
2. Use $g_1, \ldots, g_m$ to evaluate the potential from x_i such that $x_j - x_i > \delta(b-a)$
3. Directly evaluate the potential from x_i such that $0 < x_j - x_i < \delta(b-a)$

By (16), each update of $g_1, \ldots, g_m$ requires $\mathcal{O}(m)$ operations, and we must update $g_1, \ldots, g_m$ at most n times, so we conclude that the total cost of the first step of the algorithm is $\mathcal{O}(nm)$ operations. For each $j = 1, \ldots, n$,

the second and third step of the algorithm involve $\mathcal{O}(m)$ and $\mathcal{O}(\#\{x_i : 0 < x_j - x_i < \delta(b-a)\})$ operations, respectively, see (15). It follows that the total cost of the second and third step of the algorithm is $\mathcal{O}(nm + N_\delta)$ operations, where N_δ is defined in (9). We conclude that $\tilde{u}_1^+, \ldots, \tilde{u}_n^+$ can be computed in $\mathcal{O}(nm + N_\delta)$ operations. In §4, we complete the proof of the computational complexity guarantees of Theorem 2.1 by showing that there exist $m = \mathcal{O}(\log(\delta^{-1})\log(\varepsilon^{-1}))$ nodes $t_1, \ldots, t_m$ and weights $w_1, \ldots, w_m$ that satisfy (12), where $\varepsilon > 0$ is the approximation error in (12).

3.3. Detailed description

In the following, we give a detailed description of the forward pass of the algorithm that computes $\tilde{u}_1^+, \ldots, \tilde{u}_n^+$. Suppose that $\delta > 0$ and $\varepsilon > 0$ are given and fixed. We describe the algorithm under the assumption that we are given quadrature nodes $t_1, \ldots, t_m$ and weights $w_1, \ldots, w_m$ such that

$$\left| \frac{1}{r} - \sum_{j=1}^{m} w_j e^{-r t_j} \right| \leq \varepsilon \quad \text{for} \quad r \in [\delta(b-a), b-a]. \tag{17}$$

The existence of such weights and nodes is established in §4.4, and the computation of such nodes and weights is discussed in §5.2. To simplify the description of the algorithm, we assume that $x_0 = -\infty$ is a placeholder node that does not generate a potential.

Algorithm 3.1. *Input:* $x_1 < \cdots < x_n \in [a, b]$, $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$. *Output:* $\tilde{u}_1^+, \ldots, \tilde{u}_n^+$.

$j_0 = 0$ and $g_1 = \cdots = g_m = 0$

main loop:

for $j = 1, \ldots, n$

update $g_1, \ldots, g_m$ *and* j_0*:*

while $x_j - x_{j_0+1} > \delta(b-a)$

for i = 1,. . . ,m

$g_i = g_i e^{-(x_{j_0+1}-x_{j_0})t_i} + \alpha_i$

end for

$j_0 = j_0 + 1$

end while

compute potential from x_i such that $x_i \leq x_{j_0}$:

$\tilde{u}_j^+ = 0$

for $i = 1, \ldots, m$

$\tilde{u}_j^+ = \tilde{u}_j^+ - w_i g_i e^{-(x_j - x_{j_0})t_i}$

end for

compute potential from x_i such that $x_{j_0+1} \leq x_i \leq x_{j-1}$

for $i = j_0 + 1, \ldots, j - 1$

$\tilde{u}_j^+ = \tilde{u}_j^+ + \alpha_i/(x_i - x_j)$.

end for

end for

Remark 3.1. In some applications, it may be necessary to evaluate an expression of the form (1) for many different weights $\alpha_1, \ldots, \alpha_n$ associated with a fixed set of points $x_1, \ldots, x_n$. For example, in the projection application described in §1.3 the weights $\alpha_1, \ldots, \alpha_n$ correspond to a function that is being projected, while the points $x_1, \ldots, x_n$ are a fixed set of quadrature nodes. In such situations, pre-computing the exponentials $e^{-(x_j - x_{j_0})t_i}$ used in the Algorithm 3.1 will significantly improve the runtime, see §5.1.

4. Proof of Main Result

4.1. Organization

In this section we complete the proof of Theorem 2.1; the section is organized as follows. In §4.2 we give mathematical preliminaries. In §4.3 we state and prove two technical lemmas. In §4.4 we prove Lemma 4.5, which together with the analysis in §3 establishes Theorem 2.1. In §4.5 we prove Corollary 2.1, and Corollary 2.2.

4.2. Preliminaries

Let $a < b \in \mathbb{R}$ and $n \in \mathbb{Z}_{>0}$ be fixed, and suppose that $f : [a, b] \to \mathbb{R}$, and $x_1 < \cdots < x_n \in [a, b]$ are given. The interpolating polynomial P of the function f at $x_1, \ldots, x_n$ is the unique polynomial of degree at most $n - 1$ such that

$$P(x_j) = f(x_j), \quad \text{for} \quad j = 1, \ldots, n.$$

This interpolating polynomial P can be explicitly defined by

$$P(x) = \sum_{j=1}^{n} f(x_j) q_j(x), \tag{18}$$

where q_j is the nodal polynomial for x_j, that is,

$$q_j(x) = \prod_{k=1, k\neq j}^{n} \frac{x - x_k}{x_j - x_k}. \tag{19}$$

We say $x_1, \ldots, x_n$ are Chebyshev nodes for the interval $[a, b]$ if

$$x_j = \frac{b+a}{2} + \frac{b-a}{2} \cos\left(\pi \frac{j - \frac{1}{2}}{n}\right), \quad \text{for} \quad j = 1, \ldots, n. \tag{20}$$

The following lemma is a classical result in approximation theory. It says that a smooth function on a compact interval is accurately approximated by the interpolating polynomial of the function at Chebyshev nodes, see for example §4.5.2 of Dahlquist and Björck [5].

Lemma 4.1. *Let $f \in C^n([a, b])$, and $x_1, \ldots, x_n$ be Chebyshev nodes for $[a, b]$. If P is the interpolating polynomial for f at $x_1, \ldots, x_n$, then*

$$\sup_{x\in[a,b]} |f(x) - P(x)| \leq \frac{2M}{n!} \left(\frac{b-a}{4}\right)^n,$$

where

$$M = \sup_{x \in [a,b]} |f^{(n)}(x)|.$$

In addition to Lemma 4.1, we require a result about the existence of generalized Gaussian quadratures for Chebyshev systems. In 1866, Gauss [8] established the existence of quadrature nodes $x_1, \ldots, x_n$ and weights $w_1, \ldots, w_n$ for an interval $[a, b]$ such that

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

whenever $f(x)$ is a polynomial of degree at most $2n - 1$. This result was generalized from polynomials to Chebyshev systems by Kreĭn [13]. A collection of functions $f_0, \ldots, f_n$ on $[a, b]$ is a Chebyshev system if every nonzero generalized polynomial

$$g(t) = a_0 f_0(t) + \cdots + a_n f_n(t), \quad \text{for} \quad a_0, \ldots, a_n \in \mathbb{R},$$

has at most n distinct zeros in $[a, b]$. The following result of Kreĭn says that any function in the span of a Chebyshev system of $2n$ functions can be integrated exactly by a quadrature with n nodes and n weights.

Lemma 4.2 (Kreĭn [13])**.** *Let $f_0, \ldots, f_{2n-1}$ be a Chebyshev system of continuous functions on $[a, b]$, and $w : (a, b) \to \mathbb{R}$ be a continuous positive weight function. Then, there exists unique nodes $x_1, \ldots, x_n$ and weights $w_1, \ldots, w_n$ such that*

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

whenever f is in the span of $f_0, \ldots, f_{2n-1}$.

4.3. Technical Lemmas

In this section, we state and prove two technical lemmas that are involved in the proof of Theorem 2.1. We remark that a similar version of Lemma 4.3 appears in [18].

Lemma 4.3. *Fix $a > 0$ and $t \in [0, \infty)$, and let $r_1, \ldots, r_n$ be Chebyshev nodes for $[a, 2a]$. If $P_t(r)$ is the interpolating polynomial for e^{-rt} at $r_1, \ldots, r_n$, then*

$$\sup_{r \in [a,2a]} \left| e^{-rt} - P_t(r) \right| \leq \frac{1}{4^n}.$$

Proof. We have

$$\sup_{r\in[a,2a]}\left|\frac{\partial^n}{\partial r^n}e^{-rt}\right| = \sup_{r\in[a,2a]}|t^n e^{-rt}| = t^n e^{-ta}.$$

By writing the derivative of $t^n e^{-ta}$ as

$$\frac{d}{dt}t^n e^{-ta} = \left(\frac{n}{a} - t\right) a t^{n-1} e^{-at},$$

we can deduce that the maximum of $t^n e^{-ta}$ occurs at $t = n/a$, that is,

$$\sup_{t\in[0,\infty)} t^n e^{-ta} = \left(\frac{n}{a}\right)^n e^{-a(n/a)}. \tag{21}$$

By (21) and the result of Lemma 4.1, we conclude that

$$\sup_{t\in[a,2a]}|e^{-rt} - P_t(r)| \le \frac{2(n/a)^n e^{-a(n/a)}}{n!}\left(\frac{a}{4}\right)^n = \frac{2n^n e^{-n}}{n!}\frac{1}{4^n}.$$

It remains to show that $2n^n e^{-n} \le n!$. Since $\ln(x)$ is a increasing function, we have

$$n\ln n - n + 1 = \int_1^n \ln(x)dx \le \int_1^n \sum_{j=1}^{n-1}\chi_{[j,j+1]}(x)\ln(j+1)dx = \sum_{j=1}^{n}\ln(j).$$

Exponentiating both sides of this inequality gives $en^n e^{-n} \le n!$, which is a classical inequality related to Stirling's approximation. This completes the proof. □

Lemma 4.4. *Suppose that $\varepsilon > 0$ and $M > 1$ are given. Then, there exists*

$$m = \mathcal{O}(\log(M)\log(\varepsilon^{-1}))$$

values $r_1, \ldots, r_m \in [1, M]$ such that for all $r \in [1, M]$ we have

$$\sup_{t\in[0,\infty)}\left|e^{-rt} - \sum_{j=1}^{m} c_j(r)e^{-r_j t}\right| \le \varepsilon, \tag{22}$$

for some choice of coefficients $c_j(r)$ that depend on r.

Proof. We construct an explicit set of $m := (\lfloor \log_2 M \rfloor + 1)(\lfloor \log_4 \varepsilon^{-1} \rfloor + 1)$ points and coefficients such that (22) holds. Set $n := \lfloor \log_4 \varepsilon^{-1} \rfloor + 1$. We define the points $r_1, \ldots, r_m$ by

$$r_{in+k} := 2^{i-1} \left(3 + \cos\left(\pi \frac{k - \frac{1}{2}}{n} \right) \right), \tag{23}$$

for $k = 1, \ldots, n$ and $i = 0, \ldots, \lfloor \log_2 M \rfloor$, and define the coefficients $c_1(r), \ldots, c_m(r)$ by

$$c_{in+k}(r) := \chi_{[2^i, 2^{i+1})}(r) \prod_{l=1, l \neq k}^{\lfloor \log_4 \varepsilon^{-1} \rfloor} \frac{r - r_{in+l}}{r_{in+l} - r_{in+k}}, \tag{24}$$

for $k = 1, \ldots, n$ and $i = 0, \ldots, \lfloor \log_2 M \rfloor$. We claim that

$$\sup_{r \in [1, M]} \sup_{t \in [0, \infty)} \left| e^{-rt} - \sum_{j=1}^{m} c_j(r) e^{-r_j t} \right| \leq \varepsilon.$$

Indeed, fix $r \in [1, M]$, and let $i_0 \in \{0, \ldots, \lfloor \log_2 M \rfloor\}$ be the unique integer such that $r \in [2^{i_0}, 2^{i_0+1})$. By the definition of the coefficients, see (24), we have

$$\sum_{j=1}^{m} c_j(r) e^{-r_j t} = \sum_{k=1}^{n} e^{-r_{i_0 n+k} t} \prod_{l=1, l \neq k}^{\lfloor \log_4 \varepsilon^{-1} \rfloor} \frac{r - r_{i_0 n+l}}{r_{i_0 n+l} - r_{i_0 n+k}}.$$

We claim that the right hand side of this equation is the interpolating polynomial $P_{t,i_0}(r)$ for e^{-rt} at $r_{i_0 n+k}, \ldots, r_{(i_0+1)n}$, that is,

$$\sum_{k=1}^{n} e^{-r_{i_0 n+k} t} \prod_{l=1, l \neq k}^{\lfloor \log_4 \varepsilon^{-1} \rfloor} \frac{r - r_{i_0 n+l}}{r_{i_0 n+l} - r_{i_0 n+k}} = P_{t,i_0}(r).$$

Indeed, see (18) and (19). Since the points $r_{i_0 n+k}, \ldots, r_{(i_0+1)n}$ are Chebyshev nodes for the interval $[2^{i_0}, 2^{i_0+1}]$, and since i_0 was chosen such that $r \in [2^{i_0}, 2^{i_0+1})$, it follows from Lemma 4.3 that

$$\left| e^{-rt} - P_{t,i_0}(r) \right| \leq \frac{1}{4^n} \quad \text{for} \quad t \in [0, \infty).$$

Since $n = \lfloor \log_4 \varepsilon^{-1} \rfloor + 1$ the proof is complete. □

Remark 4.1. The proof of Lemma 4.4 has the additional consequence that the coefficients $c_1(r), \ldots, c_m(r)$ in (22) can be chosen such that they satisfy

$$|c_j(r)| \leq \sqrt{2} \quad \text{for} \quad j = 1, \ldots, m.$$

Indeed, in (24) the coefficients $c_j(r)$ are either equal zero or equal to the nodal polynomial, see (19), for Chebyshev nodes on an interval that contains r. The nodal polynomials for Chebyshev nodes on an interval $[a, b]$ are bounded by $\sqrt{2}$ on $[a, b]$, see for example [18]. The fact that e^{-rt} can be approximated as a linear combination of functions $e^{-r_1 t}, \ldots, e^{-r_m t}$ with small coefficients means that the approximation of Lemma 4.4 can be used in finite precision environments without any unexpected catastrophic cancellation.

4.4. Completing the proof of Theorem 2.1

Previously in §3.2, we proved that the algorithm of §3 involves $\mathcal{O}(nm+N_\delta)$ operations. To complete the proof of Theorem 2.1 it remains to show that there exists

$$m = \mathcal{O}(\log(\varepsilon^{-1}) \log(\delta^{-1}))$$

points $t_1, \ldots, t_m$ and weights $w_1, \ldots, w_m$ that satisfy (17); we show the existence of such nodes and weights in the following lemma, and thus complete the proof of Theorem 2.1. The computation of such nodes and weights is described in §5.2.

Lemma 4.5. *Fix $a < b \in \mathbb{R}$, and let $\delta > 0$ and $\varepsilon > 0$ be given. Then, there exists $m = \mathcal{O}(\log(\varepsilon^{-1}) \log(\delta^{-1}))$ nodes $t_1, \ldots, t_m$ and weights $w_1, \ldots, w_m$ such that*

$$\left| \frac{1}{r} - \sum_{j=1}^{m} w_j e^{-r t_j} \right| \leq \varepsilon, \quad \text{for} \quad r \in [\delta(b-a), b-a]. \tag{25}$$

Proof. Fix $a < b \in \mathbb{R}$, and let $\delta, \varepsilon > 0$ be given. By the possibility of rescaling r, w_j, and t_j, we may assume that $b - a = \delta^{-1}$ such that we want to establish (25) for $r \in [1, \delta^{-1}]$. By Lemma 4.4 we can choose $2m = \mathcal{O}(\log(\varepsilon^{-1}) \log(\delta^{-1}))$ points $r_0, \ldots, r_{2m-1} \in [1, \delta^{-1}]$, and coefficients $c_0(r), \ldots, c_{2m-1}(r)$ depending on r such that

$$\sup_{r \in [1, \delta^{-1}]} \sup_{t \in [0, \infty)} \left| e^{-rt} - \sum_{j=0}^{2m-1} c_j(r) e^{-r_j t} \right| \leq \frac{\varepsilon}{2 \log(2\varepsilon^{-1})}. \tag{26}$$

The collection of functions $e^{-r_0 t}, \ldots, e^{-r_{2m-1} t}$ form a Chebyshev system of continuous functions on the interval $[0, \log(2\varepsilon^{-1})]$, see for example [12]. Thus, by Lemma 4.2 there exists m quadrature nodes $t_1, \ldots, t_m$ and weights $w_1, \ldots, w_m$ such that

$$\int_0^{\log(2\varepsilon^{-1})} f(t)dt = \sum_{j=1}^{m} w_j f(t_j),$$

whenever $f(t)$ is in the span of $e^{-r_0 t}, \ldots, e^{-r_{2m-1} t}$. By the triangle inequality

$$\left| \frac{1}{r} - \sum_{j=1}^{m} w_j e^{-r t_j} \right| \leq \left| \frac{1}{r} - \int_0^{\log(2\varepsilon^{-1})} e^{-rt} dt \right| + \left| \int_0^{\log(2\varepsilon^{-1})} e^{-rt} dt - \sum_{j=1}^{m} w_j e^{r t_j} \right|. \quad (27)$$

Recall that we have assumed $r \in [1, \delta^{-1}]$, in particular, $r \geq 1$ so it follows that

$$\left| \frac{1}{r} - \int_0^{\log(2\varepsilon^{-1})} e^{-rt} dt \right| \leq \varepsilon/2. \quad (28)$$

By (26), the function e^{-rt} can be approximated to error $\varepsilon/(2\log(2\varepsilon^{-1}))$ in the L^∞-norm on $[0, \log(2\varepsilon^{-1})]$ by functions in the span of $e^{-r_0 t}, \ldots, e^{-r_{2m-1} t}$. Since our quadrature is exact for these functions, we conclude that

$$\left| \int_0^{\log(2\varepsilon^{-1})} e^{-rt} dt - \sum_{j=1}^{m} w_j e^{r t_j} \right| \leq \varepsilon/2. \quad (29)$$

Combining (27), (28), and (29) completes the proof. □

4.5. Proof of Corollary 2.1

In this section, we prove Corollary 2.1, which states that the algorithm of §3 involves $\mathcal{O}(n \log n)$ operations when $x_1, \ldots, x_n$ are Chebyshev nodes, $\varepsilon = 10^{-15}$, and $\delta = 1/n$.

Proof of Corollary 2.1. By rescaling the problem we may assume that $[a, b] = [-1, 1]$ such that the Chebyshev nodes $x_1, \ldots, x_n$ are given by

$$x_j = \cos\left(\pi \frac{j - \frac{1}{2}}{n}\right), \quad \text{for} \quad j = 1, \ldots, n.$$

By the result of Theorem 2.1, it suffices to show that $N_\delta = \mathcal{O}(n \log n)$, where

$$N_\delta := \sum_{j=1}^{n} \# \left\{ x_i : |x_j - x_i| < \frac{1}{n} \right\}.$$

It is straightforward to verify that the number of Chebyshev nodes within an interval of radius $1/n$ around the point $-1 < x < 1$ is $\mathcal{O}(1/\sqrt{1-x^2})$, that is,

$$\# \left\{ x_i : |x - x_i| < \frac{1}{n} \right\} = \mathcal{O}\left(\frac{1}{\sqrt{1-x^2}} \right), \quad \text{for} \quad -1 < x < 1.$$

This estimate, together with the fact that the first and last Chebyshev node are distance at least $1/n^2$ from 1 and -1, respectively, gives the estimate

$$\sum_{j=1}^{n} \# \left\{ x_i : |x_j - x_i| < \frac{1}{n} \right\} = \mathcal{O}\left(\int_{1/n^2}^{\pi - 1/n^2} \frac{n}{\sqrt{1-\cos(t)^2}} dt \right). \tag{30}$$

Let $\pi/2 > \eta > 0$ be a fixed parameter; direct calculation yields

$$\int_{\eta}^{\pi-\eta} \frac{1}{\sqrt{1-\cos(t)^2}} dt = 2 \log\left(\cot\left(\frac{\eta}{2} \right) \right) = \mathcal{O}\left(\log\left(\eta^{-1} \right) \right).$$

Combining this estimate with (30) yields $N_\delta = \mathcal{O}(n \log n)$ as was to be shown. □

5. Numerical results and implementation details

5.1. Numerical results

We report numerical results for two different point distributions: uniformly random points in $[1, 10]$, and Chebyshev nodes in $[-1, 1]$. In both cases, we choose the weights $\alpha_1, \ldots, \alpha_n$ uniformly at random from $[0, 1]$, and test the algorithm for

$$n = 1000 \times 2^k \quad \text{points}, \quad \text{for} \quad k = 0, \ldots, 10.$$

We time two different versions of the algorithm: a standard implementation, and an implementation that uses precomputed exponentials. Precomputing exponentials may be advantageous in situations where the expression

$$u_j = \sum_{i=1}^{n} \frac{\alpha_i}{x_i - x_j}, \quad \text{for} \quad j = 1, \ldots, n, \tag{31}$$

must be evaluated for many different weights $\alpha_1, \ldots, \alpha_n$ associated with a fixed set of points $x_1, \ldots, x_n$, see Remark 3.1. We find that using precomputed exponentials makes the algorithm approximately ten times faster, see Tables 1, 2, and 3. In addition to reporting timings, we report the absolute relative difference between the output of the algorithm of §3 and the output of direct evaluation; we define the absolute relative difference ϵ_r between the output $\tilde{u}_j$ of the algorithm of §3 and the output u_j^d of direct calculation by

$$\epsilon_r := \sup_{j=1,\ldots,n} \left| \frac{\tilde{u}_j - u_j^d}{\bar{u}_j} \right|, \quad \text{where} \quad \bar{u}_j := \sum_{i=1}^{n} \left| \frac{\alpha_i}{x_i - x_j} \right|, \tag{32}$$

Dividing by $\bar{u}_j$ accounts were the fact that the calculations are performed in finite precision; any remaining loss of accuracy in the numerical results is a consequence of the large number of addition and multiplication operations that are performed. All calculations are performed in double precision, and the algorithm of §3 is run with $\varepsilon = 10^{-15}$. The parameter $\delta > 0$ is set via an empirically determined heuristic. The numerical experiments were performed on a laptop with a Intel Core i5-8350U CPU and 7.7 GiB of memory; the code was written in Fortran and compiled with gfortran with standard optimization flags. The results are reported in Tables 1, 2, and 3.

To put the run time of the algorithm in context, we additionally perform a time comparison to the Fast Fourier Transform (FFT), which also has complexity $\mathcal{O}(n \log n)$. Specifically, we compare the run time of the algorithm of §3 on random data using precomputed exponentials with the run time of an FFT implementation from FFTPACK [20] on random data of the same length using precomputed exponentials. We report these timings in Table 4; we find that the FFT is roughly 5-10 times faster than our implementation of the algorithm of §3; we remark that no significant effort was made to optimize our implementation, and that it may be possible to improve the run time by vectorization.

5.2. Computing nodes and weights

The algorithm of §3 is described under the assumption that nodes $t_1, \ldots, t_m$ and weights $w_1, \ldots, w_m$ are given such that

$$\left| \frac{1}{r} - \sum_{j=1}^{m} w_j e^{-r t_j} \right| \leq \varepsilon \quad \text{for} \quad r \in [\delta(b-a), b-a], \tag{33}$$

Label	Definition
n	number of points
t_w	time of algorithm of §3 without precomputation in seconds
t_p	time of precomputing exponentials for algorithm of §3 in seconds
t_u	time of algorithm of §3 using precomputed exponentials in seconds
t_d	time of direct evaluation in seconds
ϵ_r	maximum absolute relative difference defined in (32)
t_f	time of FFT using precomputed exponentials (for time comparison only)

Table 1: Key for column labels of Tables 2, 3, and 4.

n	t_w	t_p	t_u	t_d	ϵ_r
1000	0.74 E −03	0.18 E −02	0.93 E −04	0.66 E −03	0.19 E −14
2000	0.19 E −02	0.31 E −02	0.19 E −03	0.25 E −02	0.30 E −14
4000	0.42 E −02	0.61 E −02	0.43 E −03	0.10 E −01	0.52 E −14
8000	0.85 E −02	0.10 E −01	0.89 E −03	0.37 E −01	0.72 E −14
16000	0.18 E −01	0.25 E −01	0.18 E −02	0.14 E +00	0.92 E −14
32000	0.38 E −01	0.49 E −01	0.37 E −02	0.59 E +00	0.19 E −13
64000	0.84 E −01	0.98 E −01	0.78 E −02	0.23 E +01	0.21 E −13
128000	0.16 E +00	0.19 E +00	0.18 E −01	0.95 E +01	0.35 E −13
256000	0.37 E +00	0.53 E +00	0.34 E −01	0.40 E +02	0.59 E −13
512000	0.75 E +00	0.10 E +01	0.71 E −01	0.19 E +03	0.88 E −13
1024000	0.17 E +01	0.23 E +01	0.15 E +00	0.81 E +03	0.14 E −12

Table 2: Numerical results for uniformly random points in $[1, 10]$.

where $\varepsilon > 0$ and $\delta > 0$ are fixed parameters. As in the proof of Lemma 4.5 we note that by rescaling r it suffices to find nodes and weights satisfying

$$\left| \frac{1}{r} - \sum_{j=1}^{m} w_j e^{-rt_j} \right| \leq \varepsilon \quad \text{for} \quad r \in [1, \delta^{-1}]. \tag{34}$$

Indeed, if the nodes $t_1, \ldots, t_m$ and weights $w_1, \ldots, w_m$ satisfy (34), then the nodes $t_1/(b-a), \ldots, t_m/(b-a)$ and weights $w_1/(b-a), \ldots, w_m/(b-a)$ will satisfy (33). Thus, in order to implement the algorithm of §3 it suffices to tabulate nodes and weights that are valid for $r \in [1, M]$ for various values of

n	t_w	t_p	t_u	t_d	ϵ_r
1000	0.54 E −03	0.12 E −02	0.74 E −04	0.60 E −03	0.11 E −14
2000	0.15 E −02	0.26 E −02	0.15 E −03	0.24 E −02	0.14 E −14
4000	0.38 E −02	0.51 E −02	0.37 E −03	0.99 E −02	0.39 E −14
8000	0.83 E −02	0.10 E −01	0.85 E −03	0.38 E −01	0.35 E −14
16000	0.19 E −01	0.23 E −01	0.17 E −02	0.14 E +00	0.58 E −14
32000	0.41 E −01	0.48 E −01	0.37 E −02	0.62 E +00	0.89 E −14
64000	0.98 E −01	0.90 E −01	0.82 E −02	0.24 E +01	0.12 E −13
128000	0.22 E +00	0.19 E +00	0.23 E −01	0.10 E +02	0.19 E −13
256000	0.44 E +00	0.47 E +00	0.32 E −01	0.40 E +02	0.26 E −13
512000	0.84 E +00	0.94 E +00	0.73 E −01	0.19 E +03	0.52 E −13
1024000	0.19 E +01	0.19 E +01	0.14 E +00	0.84 E +03	0.64 E −13

Table 3: Numerical results for Chebyshev nodes on $[-1, 1]$.

n	t_u	t_f
1000	$0.91E-04$	$0.16E-04$
2000	$0.28E-03$	$0.37E-04$
4000	$0.41E-03$	$0.44E-04$
8000	$0.93E-03$	$0.85E-04$
16000	$0.18E-02$	$0.24E-03$
32000	$0.38E-02$	$0.41E-03$
64000	$0.81E-02$	$0.88E-03$
128000	$0.18E-01$	$0.19E-02$
256000	$0.38E-01$	$0.59E-02$
512000	$0.71E-01$	$0.12E-01$
1024000	$0.14E+00$	$0.25E-01$

Table 4: Time comparison with FFT.

M. In the implementation used in the numerical experiments in this paper, we tabulated nodes and weights valid for $r \in [1, M]$ for

$$M = [1, 4^k] \quad \text{for} \quad k = 1, \ldots, 10.$$

For example, in Tables 5 and 6 we have listed $m = 33$ nodes $t_1, \ldots, t_{33}$ and weights $w_1, \ldots, w_{33}$ such that

$$\left| \frac{1}{r} - \sum_{j=1}^{33} w_j e^{-rt_j} \right| \leq 10^{-15},$$

for all $r \in [1, 1024]$.

```
0.2273983006898589D-03,0.1206524521003404D-02,0.3003171636661616D-02,
0.5681878572654425D-02,0.9344657316017281D-02,0.1414265501822061D-01,
0.2029260691940998D-01,0.2809891134697047D-01,0.3798133147119762D-01,
0.5050795277167632D-01,0.6643372693847560D-01,0.8674681067847460D-01,
0.1127269233505314D+00,0.1460210820252656D+00,0.1887424688689547D+00,
0.2435986924712581D+00,0.3140569015209982D+00,0.4045552087678740D+00,
0.5207726670656921D+00,0.6699737362118449D+00,0.8614482005965975D+00,
0.1107074709906516D+01,0.1422047253849542D+01,0.1825822499573290D+01,
0.2343379511131976D+01,0.3006948272874077D+01,0.3858496861353812D+01,
0.4953559345813267D+01,0.6367677940017810D+01,0.8208553424367139D+01,
0.1064261195532074D+02,0.1396688222191633D+02,0.1889449184151398D+02
```

Table 5: A list of 33 nodes $t_1, \ldots, t_{33}$.

```
0.5845245927410881D-03,0.1379782337905140D-02,0.2224121503815854D-02,
0.3150105276431181D-02,0.4200370923383030D-02,0.5431379037435571D-02,
0.6918794756934398D-02,0.8763225538492927D-02,0.1109565843047196D-01,
0.1408264766413004D-01,0.1793263393523491D-01,0.2290557147478609D-01,
0.2932752351846237D-01,0.3761087060298772D-01,0.4828044150885936D-01,
0.6200636888239893D-01,0.7964527252809662D-01,0.1022921587521237D+00,
0.1313462348178323D+00,0.1685948994092301D+00,0.2163218289369589D+00,
0.2774479391081561D+00,0.3557192797195578D+00,0.4559662159666857D+00,
0.5844792718191478D+00,0.7495918095861060D+00,0.9626599456939077D+00,
0.1239869481076760D+01,0.1605927580173348D+01,0.2102583514906888D+01,
0.2811829220697454D+01,0.3937959064316012D+01,0.6294697335695096D+01
```

Table 6: A list of 33 weights $w_1, \ldots, w_{33}$.

The nodes and weights satisfying (34) can be computed by using a procedure for generating generalized Gaussian quadratures for Chebyshev systems together with the proof of Lemma 4.4. Indeed, Lemma 4.4 is constructive

with the exception of the step that invokes Lemma 4.2 of Kreĭn. The procedure described in [4] is a constructive version of Lemma 4.2: given a Chebyshev system of functions, it generates the corresponding quadrature nodes and weights. We remark that generalized Gaussian quadrature generation codes are a powerful tools for numerical computation with a wide range of applications. The quadrature generation code used in this paper was an optimized version of [4] recently developed by Serkh for [19].

Acknowledgements

The authors would like to thank Jeremy Hoskins for many useful discussions. Certain commercial equipment is identified in this paper to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that equipment identified is necessarily the best available for the purpose.

References

[1] Gregory Beylkin and Lucas Monzón, *Approximation by exponential sums revisited*, Appl. Comput. Harmon. Anal. **28** (2010), no. 2, 131–149. MR2595881

[2] Gregory Beylkin and Lucas Monzón, *On approximation of functions by exponential sums*, Appl. Comput. Harmon. Anal. **19** (2005), no. 1, 17–48. MR2147060

[3] Dietrich Braess, *Nonlinear approximation theory*, Springer Series in Computational Mathematics, vol. 7, Springer-Verlag, Berlin, 1986. MR866667

[4] James Bremer, Zydrunas Gimbutas, and Vladimir Rokhlin, *A nonlinear optimization procedure for generalized Gaussian quadratures*, SIAM J. Sci. Comput. **32** (2010), no. 4, 1761–1788. MR2671296

[5] Germund Dahlquist and Åke Björck, *Numerical methods*, Dover Publications, Inc., Mineola, NY, 2003, Translated from the Swedish by Ned Anderson, Reprint of the 1974 English translation. MR1978058

[6] A. Dutt, M. Gu, and V. Rokhlin, *Fast algorithms for polynomial interpolation, integration, and differentiation*, SIAM J. Numer. Anal. **33** (1996), no. 5, 1689–1711. MR1411845

[7] William Fong and Eric Darve, *The black-box fast multipole method*, J. Comput. Phys. **228** (2009), no. 23, 8712–8725. MR2558773

[8] C. F. Gauss. *Methodus nova integralium valores per approximationen inveniendi*, Werke, **3** (1866), 1630–196.

[9] Leslie Greengard, *The rapid evaluation of potential fields in particle systems*, ACM Distinguished Dissertations, MIT Press, Cambridge, MA, 1988. MR936632

[10] Leslie Greengard and Vladimir Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, Acta numerica, 1997, Acta Numer., vol. 6, Cambridge Univ. Press, Cambridge, 1997, pp. 229–269. MR1489257

[11] Rüdiger Jakob-Chien and Bradley K. Alpert, *A fast spherical filter with uniform resolution*, Journal of Computational Physics **136** (1997), no. 2, 580–584.

[12] Samuel Karlin and William J. Studden, *Tchebycheff systems: With applications in analysis and statistics*, Pure and Applied Mathematics, Vol. XV, Interscience Publishers John Wiley & Sons, New York-London-Sydney, 1966. MR0204922

[13] M. G. Kreĭn, *The ideas of P. L. Čebyšev and A. A. Markov in the theory of limiting values of integrals and their further development*, Amer. Math. Soc. Transl. (2) **12** (1959), 1–121. MR0113106

[14] J. Ma, V. Rokhlin, and S. Wandzura, *Generalized Gaussian quadrature rules for systems of arbitrary functions*, SIAM J. Numer. Anal. **33** (1996), no. 3, 971–996. MR1393898

[15] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert, *On interpolation and integration in finite-dimensional spaces of bounded functions*, Commun. Appl. Math. Comput. Sci. **1** (2006), 133–142. MR2244272

[16] K. Nabors, F. T. Korsmeyer, F. T. Leighton, and J. White, *Preconditioned, adaptive, multipole-accelerated iterative methods for three-dimensional first-kind integral equations of potential theory*, SIAM J. Sci. Comput. **15** (1994), no. 3, 713–735, Iterative methods in numerical linear algebra (Copper Mountain Resort, CO, 1992). MR1273161

[17] *NIST Digital Library of Mathematical Functions*. http://dlmf.nist.gov/, Release 1.0.22 of 2019-03-15. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller and B. V. Saunders, eds

[18] V. Rokhlin, *A fast algorithm for the discrete Laplace transformation*, J. Complexity **4** (1988), no. 1, 12–32. MR939693

[19] Kirill Serkh, *On the Solution of Elliptic Partial Differential Equations on Regions with Corners*, ProQuest LLC, Ann Arbor, MI, 2016, Thesis (Ph.D.)–Yale University. MR3564124

[20] P. N. Swarztrauber, *Vectorizing the FFTs*, Parallel Computations (G. Rodrigue, ed.), Academic Press, 1982, pp. 51–83.

[21] M. Tygert. Analogues for Bessel Functions of the Christoffel-Darboux Identity. *Yale Tech. Rep.* (2016).

[22] Norman Yarvin and Vladimir Rokhlin, *An improved fast multipole algorithm for potential fields on the line*, SIAM J. Numer. Anal. **36** (1999), no. 2, 629–666. MR1675269

[23] N. Yarvin and V. Rokhlin, *Generalized Gaussian quadratures and singular value decompositions of integral operators*, SIAM J. Sci. Comput. **20** (1998), no. 2, 699–718. MR1642612