Journal of Computational Physics 344 (2017) 1–22

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

An adaptive fast multipole accelerated Poisson solver for complex geometries

T. Askham [a,*], A.J. Cerfon [b]

[a] Department of Applied Mathematics, University of Washington, Seattle, WA 98195, United States
[b] Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, United States

ARTICLE INFO

Article history:
Received 5 October 2016
Received in revised form 8 March 2017
Accepted 24 April 2017
Available online 2 May 2017

Keywords:
Poisson equation
Fast multipole method
Quadrature by expansion
Integral equations

ABSTRACT

We present a fast, direct and adaptive Poisson solver for complex two-dimensional geometries based on potential theory and fast multipole acceleration. More precisely, the solver relies on the standard decomposition of the solution as the sum of a volume integral to account for the source distribution and a layer potential to enforce the desired boundary condition. The volume integral is computed by applying the FMM on a square box that encloses the domain of interest. For the sake of efficiency and convergence acceleration, we first extend the source distribution (the right-hand side in the Poisson equation) to the enclosing box as a C^0 function using a fast, boundary integral-based method. We demonstrate on multiply connected domains with irregular boundaries that this continuous extension leads to high accuracy without excessive adaptive refinement near the boundary and, as a result, to an extremely efficient "black box" fast solver.

1. Introduction

The solution of the Poisson equation is a critical task in many areas of computational physics. The corresponding solvers need to be able to handle complex, multiply connected geometries, to be fast, adaptive, and to yield high order accuracy. Speed is of particular importance when the Poisson equation is part of a larger system of equations or in the inner loop of an optimization process. And since the physical quantity of interest is often the gradient of the solution, rather than the solution itself [36,44,11,35,29], partial derivatives of the solution must be computable with high accuracy as well.

Integral equation techniques have the potential to address all the challenges mentioned above. Complex geometries may be handled by decomposing the solution to Poisson's equation as the sum of a particular solution v that does not satisfy the proper boundary condition in general, plus a homogeneous solution u^H that solves Laplace's equation and is chosen so that the full solution $u = v + u^H$ satisfies the proper boundary condition. Fast and accurate solvers can be designed based on this construction. Indeed, several efficient and accurate integral equation based schemes exist to compute the solution of Laplace's equation on complex geometries [21,5,6], and fast and accurate evaluation of the particular solution v on fully adaptive grids by use of the Fast Multipole Method (FMM) has also been demonstrated for Poisson's equation [14,27]. Furthermore, in integral equation formulations derivatives do not have to be computed through direct numerical differentiation. Instead, one can analytically differentiate the kernels in the integral representation of the solution, and thus

* Corresponding author.
E-mail address: askham@uw.edu (T. Askham).

http://dx.doi.org/10.1016/j.jcp.2017.04.063
0021-9991/

obtain integral representations for the derivatives of the solution as well. As a result the numerical error for the derivatives often converges at the same rate as the error for the solution itself [35,27].

Remarkably, despite all the strengths described above, we are not aware of an integral equation based Poisson solver for planar problems that combines all the features at once. In [14,18], grid adaptivity and FMM acceleration are demonstrated, but only simple geometries are considered. In contrast, in [32], Poisson's equation is solved for complex geometries and with FMM-accelerated quadratures, but the solver relies on fast methods for uniform grids [30,31]. The purpose of this manuscript is to close this gap and to present an adaptive, FMM-accelerated Poisson solver for complex geometries. We achieve this in the following way. We embed the irregular domain Ω on which Poisson's equation needs to be solved in a larger square domain Ω_B. We decompose the solution to Poisson's equation as $u = v + u^H$, and compute the particular solution v on Ω_B with a fast and accurate solver for square domains [14]. In order to calculate v in this way, we need to extend the source function f on the right-hand side of Poisson's equation beyond the domain Ω where it is originally given. We show that global function extension for f, constructed by solving Laplace's equation or a higher order partial differential equation on the domain $\mathbb{R}^2 \setminus \Omega$, leads to a robust, efficient and accurate algorithm for the evaluation of v. This idea is very similar in spirit to the extension technique recently presented by Stein et al. [43] for the immersed boundary method, but quite different in its implementation. Our approach for computing u^H is standard in its formulation [19], but it relies on numerical tools developed recently for optimized performance. Specifically, we represent u^H as a layer potential whose density solves a second-kind integral equation. We use generalized Gaussian quadrature [7,8] to approximate the integrals, a fast direct solver [23] to compute the density and an FMM accelerated quadrature by expansion (QBX) algorithm [26] to evaluate u^H inside Ω.

The structure of the article is as follows. In Section 2 we present our formulation for the solution to Poisson's equation, which is based on standard potential theory. We stress its computational challenges, which are then addressed in the following sections. In Section 3, we describe an efficient and accurate algorithm for the evaluation of the particular solution v and its derivatives in a square box. While this algorithm plays a central role in our approach to the problem, the section is relatively brief because our solver relies on an implementation of the algorithm and techniques that have been discussed in detail elsewhere [14,27]. In Section 4, we explain how we use a global function extension algorithm in combination with a box Poisson solver for the computation of the particular solution v on the whole square domain Ω_B. This is a key element of our solver, which allows us to deal with complex geometries in an efficient manner. To illustrate the method, we focus on a C^0 global function extension for simplicity. As we will show, for such extensions the super-convergence property of integral equation based schemes mentioned above, in which the error in the derivatives and the solution converge at the same rate, does not hold. We explain the reason for this discrepancy and discuss what is required for a global function extension method to achieve super-convergence in Sections 4.1 and 5.4. In Section 5, we present our numerical method for calculating the homogeneous solution u^H, as well as the function extension. Both are expressed as layer potentials and are computed in very similar ways. In Section 6 we study the performance of our new solver for two Poisson problems on a multiply connected domain. We summarize our work in Section 7 and suggest directions for future work.

2. The potential theoretic approach to Poisson's equation

In this article, we consider the solution u to Poisson's equation with Dirichlet boundary conditions given by

$$\Delta u = f \text{ in } \Omega \tag{1}$$

$$u = g \text{ on } \partial\Omega \tag{2}$$

where Ω is a smooth planar domain, which may or may not be multiply connected. The standard potential theory-based approach to the solution of (1)–(2) proceeds as follows. The first step is to calculate a particular solution, i.e. a function v which satisfies only equation (1) but does in general not satisfy equation (2). A natural candidate for v is given by

$$v(\mathbf{x}) = \int_\Omega G(\mathbf{x}, \mathbf{y}) f(\mathbf{y})\, d\mathbf{y}, \tag{3}$$

where $G(\mathbf{x}, \mathbf{y})$ is the free-space Green's function for Poisson's equation. For planar problems, $G(\mathbf{x}, \mathbf{y}) = -\log(||\mathbf{x} - \mathbf{y}||)/2\pi$. This is the situation we will consider in this article. Once v has been computed, the second step is to compute a homogeneous solution with appropriate boundary conditions. Specifically, one solves the following Dirichlet problem

$$\Delta u^H = 0 \text{ in } \Omega \tag{4}$$

$$u^H = g - v|_{\partial\Omega} \text{ on } \partial\Omega. \tag{5}$$

The solution to (1)–(2) is then the sum, $u = v + u^H$. There are many options for the numerical implementation of these two steps and we will not attempt to provide an exhaustive review of them here. Instead, we focus on our new approach, which is designed to address situations for which the domain Ω may be irregular and where derivatives of the solution are also required with high accuracy. The purpose of this section is to give a short overview of our approach. This overview is divided into two subsections: subsection 2.1 concerns the computation of v, and subsection 2.2 describes the computation

of u^H. The presentation in these two subsections is meant to give a general idea of our numerical scheme, and is brief on purpose. We provide detailed descriptions of our numerical methods to calculate of v and u^H in Sections 3 and 4 for v, and Section 5 for u^H.

2.1. Computing the particular solution

There are two challenges associated with the evaluation of the particular solution v through the integral (3). First, accurate quadratures must be used in order to handle the logarithmic singularity. Second, given a quadrature rule, the naïve numerical approach to computing (3) would require $\mathcal{O}(N^2)$ work for a domain with N discretization nodes. It is now well known that the computational work can in fact be reduced to $\mathcal{O}(N)$ via the fast multipole method [10]. Furthermore, for a fixed domain Ω, the quadrature rules for a weakly singular kernel $G(\mathbf{x}, \mathbf{y})$ can be precomputed using an adaptive, brute-force procedure [16]. As a result, there exist particularly efficient $\mathcal{O}(N)$ algorithms, including optimized versions of the FMM [14], to compute the integral (3) for problems specified on a box. We choose such an algorithm for our solver, and provide some details of this type of method in Section 3.

For irregular domains Ω, however, the situation is quite different. The calculation of appropriate quadratures is much more difficult and fewer optimizations of the FMM are available. A natural strategy, then, for irregular domains is to consider a larger, square domain Ω_B containing Ω and to instead compute

$$v(\mathbf{x}) = \int_{\Omega_B} G(\mathbf{x}, \mathbf{y}) f_e(\mathbf{y}) \, d\mathbf{y}, \tag{6}$$

where f_e is defined on all of Ω_B, and constructed such that $f_e = f$ on Ω. One of the main novelties of our work is to compute f_e via global function extension: f_e restricted to $\mathbb{R}^2 \setminus \Omega$ is the solution of an elliptic partial differential equation with Dirichlet data $f_e = f$ on $\partial\Omega$. The PDE is solved with a standard integral equation representation. We elaborate on this idea in Section 4.

Remark 1. It should be noted here that the solution provided by any Poisson solver is a valid particular solution, though it will not necessarily be equal to the one given by (3). This includes, in particular, the solutions produced by FFT-based solvers for rectangular and circular domains, which are very fast in terms of work per grid point. The method of [32] uses such a particular solution, computed via Buneman's method [9] and the modified stencils developed in [30]. A recent paper, [46], offers an alternative FFT-based algorithm for evaluating the particular solution on uniform grids. By regularizing the Fourier transform of the Green's function, the method is able to utilize the FFT to evaluate the convolution (3) rapidly and with spectral accuracy on a uniform grid. We have chosen the algorithm of [14] because of its facility with adaptive grids, but much of the discussion of this paper would also apply to adapting the algorithm of [46] for use with irregular domains. To demonstrate the efficacy of the algorithm of [14], we compute an example on a highly-adaptive grid in Section 6.

2.2. Computing the homogeneous solution

A standard approach to the solution of Laplace's equation is to represent the solution u^H as a layer potential with unknown density μ on the boundary. The representation should be chosen so that imposing the boundary conditions results in an invertible, second kind integral equation (SKIE) for the density on the boundary. This is a well-studied area and there exist appropriate integral representations for multiply connected domains [34,20], unbounded domains [19], and for situations with other types of boundary conditions [33]. Further references can be found in the previously cited papers, and we recommend [19,4] for very clear treatments of this topic.

Once a suitable representation for u^H is chosen, the discretization of the problem is then simply a matter of quadrature for the resulting SKIE. In general, the integral kernel may be singular and the choice of quadrature requires attention [1, 25,26,21,7,8]. Once discretized, there are many tools available for the fast solution of the resulting linear system, which we briefly discuss in Section 5. For this article, we choose a direct method [23] that is optimized for the type of problems considered here.

After the density σ is computed, the solution u^H can be evaluated in the domain. This step is trivially direct but it is not without its difficulties. With N discretization points in the domain and M discretization points on the boundary, naïve computation of the necessary integrals would require $\mathcal{O}(MN)$ work. This work can be reduced to $\mathcal{O}(M+N)$ with the FMM. Because the integral kernel of the solution representation is nearly singular for discretization points near the boundary, computing the potential u^H to high accuracy at such points requires special quadrature schemes. Such schemes have been developed recently [21,26], and for our solver we choose to rely on the quadrature by expansion method (QBX) [26], which we also briefly discuss in Section 5. In two dimensions, the scheme of [21] would likely offer better performance than that of [26]. Indeed, the method of [21] has been shown to be effective for highly irregular geometries [33]. We have chosen the quadrature by expansion framework because it applies to a large class of integral kernels and extends readily to three dimensions.

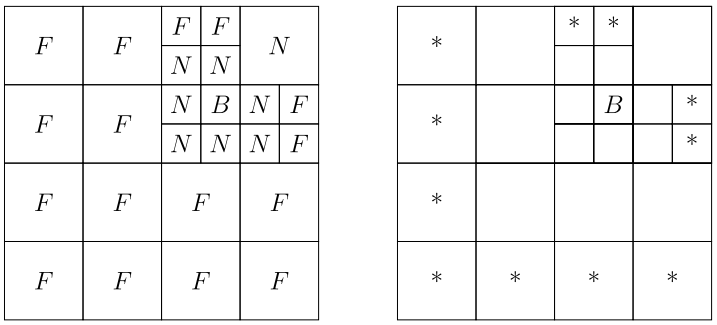

Fig. 1. In the figure on the left, the leaves of a quad-tree are shown and the boxes in the near field of the box B are marked with an N while the boxes in the far field of B are marked with an F. The same quad tree is shown on the right and the boxes for which B is in the far field are marked with an asterisk (*), these boxes being in $\mathcal{F}(B)$.

3. Box codes

This section reviews relevant features of the algorithm of [14], which is the original "box code", and which we have implemented in our solver. By "box codes", we mean a class of fast solvers which are used to evaluate integrals of the form

$$Vf(\mathbf{x}) = \int_{\Omega} G(\mathbf{x},\mathbf{y})\, f(\mathbf{y})\, d\mathbf{y}\,, \tag{7}$$

where the integral kernel $G(\mathbf{x},\mathbf{y})$ is a translation invariant Green's function, the domain Ω is a box, and $f(\mathbf{y})$ is a given density. We take $G(\mathbf{x},\mathbf{y}) = -\log\|\mathbf{x}-\mathbf{y}\|/2\pi$ in what follows.

3.1. Outline of a box code

As in all fast multipole methods, an FMM-based box code is based on a hierarchical division of space. Specifically, the domain is taken to be the root box (level 0) of a quad-tree. The finer levels are obtained by subdividing boxes from the previous level into four equal parts. After a box is subdivided, the four resulting boxes on the next level are its children. The quad-tree for a box code is thus fairly typical for an FMM. The primary distinction of a quad-tree as used in a box code is that it is typically a level restricted tree, i.e., adjacent leaf boxes are required to be no more than one level apart in the tree hierarchy.

When computing (7), a choice has to be made as to how the function f is represented on each leaf box of the quad-tree. The standard choice, as in [14], is to represent f by collocation points (for monomials, Chebyshev polynomials, etc.) on each leaf box, using the same points scaled for each level. Then, a reasonable subdivision criterion for a box is whether or not the function f is well approximated by its interpolant up to a given tolerance on that box. This criterion makes a box code an adaptive method, with the order of accuracy determined by the order of the polynomial approximation on each box.

After the quad-tree is formed, we have that $\Omega = \cup_j B_j$ where the B_j are leaf boxes and on each leaf box there is a polynomial p_j which approximates the density f. Let $\tilde{f}$, defined by setting $\tilde{f}(\mathbf{x}) = p_j(\mathbf{x})$ for $\mathbf{x} \in B_j$, be the approximation of f over the whole domain. The box code proceeds to evaluate the potential $V\tilde{f}(x)$, where the evaluation points x are taken to be the collocation points of the polynomials p_j. Let $\mathbb{V}\tilde{f}(\mathbf{x})$ be the computed values of $V\tilde{f}(\mathbf{x})$. To evaluate the volume integral at other points in the domain, we evaluate the polynomial which interpolates the values $\mathbb{V}\tilde{f}(\mathbf{x})$ on each box. We denote this piecewise polynomial function by $\tilde{v}$. The distinction between $\tilde{v}$ and $\mathbb{V}\tilde{f}$ is subtle but necessary here. For the sake of speed, a box code only evaluates $\mathbb{V}\tilde{f}$ at the collocation nodes. The operator V is approximated more or less exactly so the error is determined by the interpolation error for $\tilde{f}$. The values of $\tilde{v}$ incur further interpolation error, which depends on the order of the interpolation on leaf boxes and the smoothness of $\mathbb{V}\tilde{f}$. We address the error analysis in more detail in Section 3.3.

For a quad-tree with N total collocation points, computing $\mathbb{V}\tilde{f}$ would require $\mathcal{O}(N^2)$ operations if done naïvely. This cost can be reduced to $\mathcal{O}(N)$ by using the fast multipole method. In the context of this article, it is only necessary to describe the result of the FMM. For a detailed account of the structure of the FMM, see [10,24,14].

Let B_j be a leaf box of the quad-tree with width h. The "near field" of B_j is defined to be any leaf box whose interior intersects the interior of the box of width $3h$ centered at B_j. The boxes which are not in the near field of B_j are said to be in the "far field." Because the boxes in the far field of B_j are separated from B_j by a box of at least the same size as B_j, these boxes are said to be "well separated." Let $\mathcal{F}(B_j) = \{i : B_j \text{ is in the far field of } B_i\}$ be the set of leaf boxes for which B_j is well separated and $\Omega_j = \cup_{i\in\mathcal{F}(B_j)} B_i$ be the union over these leaf boxes. For a non-uniform tree, it is not necessarily the case that the boxes of $\mathcal{F}(B_j)$ and the far field of B_j are the same. See Fig. 1 for examples of these sets. In $\mathcal{O}(N)$ time, the FMM computes functions Φ_j for each leaf box B_j which are expansions (more precisely, the sum of a Taylor expansion

and a number of multipole expansions) approximating the influence of all leaf boxes in $\mathcal{F}(B_j)$ at any point in B_j, i.e. for any $\mathbf{x} \in B_j$

$$\Phi_j(\mathbf{x}) \approx \int_{\Omega_j} G(\mathbf{x},\mathbf{y})\tilde{f}(\mathbf{y})\,d\mathbf{y}\,.$$

With Φ_j computed, it is possible to compute the volume integral (7) by directly adding the influence of leaf boxes for which B_j is in the near field, i.e. for any $\mathbf{x} \in B_j$

$$V\tilde{f}(\mathbf{x}) \approx \Phi_j(\mathbf{x}) + \sum_{i \notin \mathcal{F}(B_j)} \int_{B_i} G(\mathbf{x},\mathbf{y})\tilde{f}(\mathbf{y})\,d\mathbf{y},$$

where the second term on the right-hand side is evaluated by direct computation, using a high order quadrature rule. This step is $\mathcal{O}(1)$ per point because the number of boxes for which B_j is in the near field and the cost of evaluating Φ_j are bounded independent of N. For a given precision ε_V, the computed values

$$\mathbb{V}\tilde{f}(\mathbf{x}) = \Phi_j(\mathbf{x}) + \sum_{i \notin \mathcal{F}(B_j)} \int_{B_i} G(\mathbf{x},\mathbf{y})\tilde{f}(\mathbf{y})\,d\mathbf{y}, \tag{8}$$

satisfy

$$|\mathbb{V}\tilde{f}(\mathbf{x}) - V\tilde{f}(\mathbf{x})| \leq \varepsilon_V \|\tilde{f}\|_{L^1}\,.$$

To achieve this bound for smaller values of ε_V, the FMM uses higher-order expansions to approximate Φ_j. See [17] for more on the error analysis of the FMM.

While $\mathcal{O}(N)$ is indeed optimal in terms of complexity, the numerical scheme presented in [14] is particularly fast in terms of work per gridpoint. For far field interactions, the speed is due in part to the fact that the translation of multipole expansions is diagonalized through the use of plane wave expansions, see [14] and [24] for details. For near field interactions, the speed is due to the use of precomputed tables. Because the tree is level-restricted, there are a limited number of near field interactions possible, up to scale. Therefore, if the possible interactions are stored for a unit box, the influence of any box on a box in its near field can be computed at the cost of a small matrix–vector product.

3.2. Derivatives of the potential

In many physical applications, the derivatives of the volume potential $Vf(\mathbf{x})$ are the quantities of interest, instead of $Vf(\mathbf{x})$ itself. Once the values of the potential $\mathbb{V}\tilde{f}$ are computed, one could differentiate the piecewise polynomial function, $\tilde{v}$, which interpolates the potential on each leaf box to obtain an approximation of the derivatives. This computation results in derivative values which have an order of accuracy that is one less than the order of accuracy for the potential.

Instead, the derivatives can be computed by recognizing that they are given by another volume integral, e.g.

$$\partial_{x_1} Vf(\mathbf{x}) = \int_{\Omega} \partial_{x_1} G(\mathbf{x},\mathbf{y}) f(\mathbf{y})\,d\mathbf{y}\,. \tag{9}$$

In fact, the volume integral for the derivatives can be computed alongside the evaluation of the volume integral for the potential with modest impact on the run time. As in the case of computing the potential, the near-field interactions can be calculated making use of precomputed tables. The far-field interactions can be computed by differentiating the local expansion for the far-field, i.e. by differentiating $\Phi_j(\mathbf{x})$, which is typically a much higher order approximation than the order of the interpolation on leaf boxes. The result of computing the derivatives of the solution with this approach is that the derivatives display the same convergence rate as the potential. For the calculations presented in this article, the authors have implemented such a scheme. A similar approach to computing derivatives was taken in [27].

3.3. Error analysis for smooth f

As above, let $\tilde{f}$ denote the piecewise polynomial approximation to f for a given tree and let $\mathbb{V}\tilde{f}$ denote the computed value of $V\tilde{f}$, as defined in equation (8). Suppose that the fast multipole method is applied with precision ε_V and that the local interaction tables are computed to at least that precision. Then, the error in $V\tilde{f}$ at a collocation node $\mathbf{x}$ has the following bound:

$$|V\tilde{f}(\mathbf{x}) - \mathbb{V}\tilde{f}(\mathbf{x})| \leq \varepsilon_V \|\tilde{f}\|_1\,. \tag{10}$$

That is, the values of $V\tilde{f}(\mathbf{x})$ are computed at the collocation nodes with an error that depends only on the truncation order of the fast multipole method. It then follows that the total error at any given collocation node $\mathbf{x}$ is bounded by

$$|Vf(\mathbf{x}) - \mathbb{V}\tilde{f}(\mathbf{x})| \leq |V\tilde{f}(\mathbf{x}) - \mathbb{V}\tilde{f}(\mathbf{x})| + |V(\tilde{f} - f)(\mathbf{x})| \leq \varepsilon_V \|\tilde{f}\|_1 + C_\Omega \|\tilde{f} - f\|_\infty , \tag{11}$$

for a constant C_Ω which is independent of f and given by

$$C_\Omega = \max_{\mathbf{x}\in\Omega} \int_\Omega |G(\mathbf{x}, \mathbf{y})| \, d\mathbf{y} . \tag{12}$$

The bound (11) provides an *a priori* estimate of the accuracy of the solution which depends only on the values of the data f. This is useful when designing adaptive refinement strategies as one can simply check whether f is well approximated on each leaf in the tree. On a uniform tree, we see that the order of accuracy of the overall scheme depends on the order of accuracy of the local polynomial approximation to f on leaf nodes. Finally, we note that (11) only depends on the fact that V is a bounded operator from L^∞ to L^∞. In particular, it is clear that (11) holds analogously for ∇V and that on a uniform tree we should see the same order of accuracy for the potential and gradient values (this is sometimes referred to as "super-convergence").

4. Box codes for irregular domains

Perhaps the most natural idea to compute a particular solution v to Poisson's equation for an arbitrary irregular domain by using a box code is as follows. Suppose Ω is the irregular domain and Ω_B is a box such that $\Omega \subset \Omega_B$. Then, a particular solution on Ω can be computed as in (6), so long as an extension f_e on $\Omega_B \setminus \Omega$ of the right-hand side f is given.

In many cases of practical interest, a smooth or continuous extension f_e of f is readily available. The density f may for example describe a compactly supported distribution of electric charges which smoothly goes to zero on the boundary of Ω. Likewise, the magnetohydrodynamic equilibrium of a plasma confined in a tokamak is given by a semilinear Poisson equation in which the right-hand side f smoothly goes to zero on the boundary of Ω in most situations [35,29]. In such cases, $f_e \equiv 0$ on $\Omega_B \setminus \Omega$ is a natural and satisfying choice.

In the general case, however, f_e does not have an obvious physical meaning, and the extension f_e is constructed as a purely mathematical artifice required by the box solver. The problem of specifying a function f_e such that $f_e = f$ on Ω is extremely open; we narrow it by looking for an extension f_e that is favorable in terms of the efficiency and accuracy of the box code.

4.1. Error analysis for non-smooth densities

To motivate our construction of f_e, we first perform a heuristic but more detailed analysis of the error bound (11) for densities f_e which are not necessarily smooth on the box Ω_B. Using the standard multi-index notation, let $\partial^{\boldsymbol{\alpha}} = \partial_{x_1}^{\alpha_1} \partial_{x_2}^{\alpha_2}$ and $|\boldsymbol{\alpha}| = \alpha_1 + \alpha_2$. Then the differentiability class $C^k(A)$ of a domain A is defined to be the set of functions g such that $\partial^{\boldsymbol{\alpha}} g$ is continuous for each $\boldsymbol{\alpha}$ with $|\boldsymbol{\alpha}| \leq k$, with the convention that $C^{-1}(A)$ is the set of bounded functions which are possibly discontinuous. For a density $f_e \in C^k(\Omega_B)$ and a uniform tree with leaf boxes of width h, let $\tilde{f}_e$ be the numerical approximation as in the previous section, using interpolants of order p (degree $p-1$) on each box. Standard error estimates imply that

$$\|f_e - \tilde{f}_e\|_\infty = \mathcal{O}(h^m) , \tag{13}$$

where $m = \min(k, p)$. If the additional assumption is made that f_e is piecewise C^{k+l} for some $l \geq 1$ (e.g., for a domain $A \subset \Omega_B$, the density $f_e \in C^{k+l}(A)$ and $f_e \in C^{k+l}(\Omega_B \setminus A)$) and globally C^k (so that any discontinuities in the $k+1$st derivative occur across ∂A) then the approximation order is improved to $m = \min(k+1, p)$, see, *inter alia*, [12].

These bounds suggest that the scheme should have $\mathcal{O}(1)$ error for a piecewise smooth density which is discontinuous across some boundary. However, in practice the observed convergence rate is faster, even for the derivatives of Vf_e. The reason for this is that the bound (11) only makes use of the fact that V and its derivatives are bounded on L^∞. It does not take into account the fact that they are given by weakly singular integrals. One could thus seek a tighter bound for the error $|Vf_e(\mathbf{x}) - V\tilde{f}_e(\mathbf{x})|$, but with our construction of the solution given by Equations (3), (4) and (5), it suffices to see how good of a particular solution $V\tilde{f}_e$ is. For this purpose, let $\mathbf{x}$ be contained in a box B_j and let Ω_j denote the union over all boxes for which B_j is in the far-field, as already defined in Section 3.1. We can write

$$V\tilde{f}_e(\mathbf{x}) = \int_\Omega G(\mathbf{x}, \mathbf{y}) \tilde{f}_e(\mathbf{y}) \, d\mathbf{y} \tag{14}$$

$$= \int_{\Omega \setminus \Omega_j} G(\mathbf{x}, \mathbf{y}) \tilde{f}_e(\mathbf{y}) \, d\mathbf{y} + \int_{\Omega_j} G(\mathbf{x}, \mathbf{y}) \tilde{f}_e(\mathbf{y}) \, d\mathbf{y} . \tag{15}$$

The contribution to $V\tilde{f}_e(\mathbf{x})$ from the second term is harmonic. The contribution from the first term is the relevant one regarding the quality of $V\tilde{f}_e$ as a particular solution, since the particular solution is only defined to within the addition of

a homogeneous solutions. Let h be the side length of the box B_j. If we consider the error without the far-field contribution, we have

$$\left| \int_{\Omega\setminus\Omega_j} G(\mathbf{x},\mathbf{y})\tilde{f}_e(\mathbf{y})\,d\mathbf{y} - \int_{\Omega\setminus\Omega_j} G(\mathbf{x},\mathbf{y})f_e(\mathbf{y})\,d\mathbf{y} \right| \leq \|\tilde{f}_e - f_e\|_\infty \int_{\Omega\setminus\Omega_j} |G(\mathbf{x},\mathbf{y})|\,d\mathbf{y} \tag{16}$$

$$= \mathcal{O}(h^2|\log h|)\|\tilde{f}_e - f_e\|_\infty\,. \tag{17}$$

For the derivative values we have the analogous bound

$$\left| \int_{\Omega\setminus\Omega_j} \nabla G(\mathbf{x},\mathbf{y})\tilde{f}_e(\mathbf{y})\,d\mathbf{y} - \int_{\Omega\setminus\Omega_j} \nabla G(\mathbf{x},\mathbf{y})f_e(\mathbf{y})\,d\mathbf{y} \right| \leq \|\tilde{f}_e - f_e\|_\infty \int_{\Omega\setminus\Omega_j} |\nabla G(\mathbf{x},\mathbf{y})|\,d\mathbf{y} \tag{18}$$

$$= \mathcal{O}(h)\|\tilde{f}_e - f_e\|_\infty\,. \tag{19}$$

There are two main conclusions from the preceding analysis. The first conclusion is the intuitive result that the smoother f_e is the better the approximation of the particular solution. The more interesting conclusion is that f_e may not need to be quite as smooth as we may have initially expected. Specifically, in our implementation, we use 4th order interpolants on each leaf box. Suppose we discretize the domain with a uniform tree and leaf boxes of side length h. For a piecewise smooth density f_e which is discontinuous, the above bounds imply (nearly) 2nd order accuracy in the values of the particular solution and 1st order accuracy in the gradient. Similarly, for a piecewise smooth density f_e which is continuous, they imply (nearly) 3rd and 2nd order accuracy, respectively. These bounds are consistent with our numerical results, as we demonstrate in Section 6.

Finally, note that (17) and (19) also imply that we do not expect to observe "super-convergence" for piecewise smooth densities unless they are of sufficient smoothness globally. While super-convergence would be a desirable property, we have found that adaptive refinement strategies can be advantageously used to obtain the desired high-accuracy (though not necessarily high-order accurate) values for derivatives of the potential. We present these numerical results in more detail in Section 6.

4.2. Global function extension

In previous attempts to use box codes for irregular domains, two main extrapolation techniques were used. The first, which we call "extrapolation by zero", simply sets the density f_e to be zero outside the domain [14]. In this case, the function f_e is as smooth as the original density f inside the domain and trivially smooth outside the domain. Therefore, the estimates for piecewise smooth functions apply and we see that the scheme should converge with a rate $\mathcal{O}(h^2|\log h|)$ for a uniform tree. The reader may keep in mind that for such situations, a box code relying on an adaptive tree is more efficient in terms of degrees of freedom than a code using a uniform tree. Even if so, adaptive refinement for functions with a discontinuity requires trees with a large number of refinement levels and therefore a large number of grid points. This can make the "extrapolation by zero" approach computationally costly. The second extrapolation method uses local polynomial approximations to f to extrapolate f outside the domain over short distances [28]. A major limitation of this method is that it results in a smooth f_e for individual leaf boxes but has no guarantees of smoothness across boxes. Since there can be discontinuities in f_e across boxes near the original domain, the computed potential Vf_e may be unresolved on those boxes. This issue seems to be inherent in local extrapolation methods. That is why we seek out a global extrapolation method which improves on the naïve global "extrapolation by zero" approach, in terms of both speed and accuracy.

The global extrapolation method we adopt is similar in spirit to the one recently proposed in [43] in a different context, and can be explained in a few words. Let w solve the PDE

$$\begin{aligned} \Delta w &= 0 \quad \text{in} \quad \mathbb{R}^2\setminus\Omega \\ w &= f \quad \text{on} \quad \partial\Omega \end{aligned}\,, \tag{20}$$

subject to the condition that $w(\mathbf{x})$ is bounded as $\|\mathbf{x}\|\to\infty$. Then, the function f_e defined by

$$\begin{aligned} f_e(\mathbf{x}) &= f(\mathbf{x}) \quad \text{for} \quad \mathbf{x}\in\Omega \\ f_e(\mathbf{x}) &= w(\mathbf{x}) \quad \text{for} \quad \mathbf{x}\in\Omega_B\setminus\Omega \end{aligned} \tag{21}$$

is globally continuous, as smooth as f on Ω, and smooth on $\Omega_B\setminus\Omega$. While this may at first seem like a computationally expensive way to extrapolate f, the analytical and numerical machinery required to solve this problem is in fact the same as that required to solve the harmonic problem (4)–(5), which is used to enforce the boundary condition of the original Poisson problem ((1)–(2)).

Note that the method can be generalized to compute globally C^k extrapolations of f by solving polyharmonic equations. For example, a C^1 extrapolation can be computed by solving the following biharmonic problem:

$$\begin{aligned} \Delta^2 w &= 0 \quad \text{in} \quad \mathbb{R}^2 \setminus \Omega \\ w &= f \quad \text{on} \quad \partial\Omega \\ \partial_\nu w &= \partial_\nu f \quad \text{on} \quad \partial\Omega \\ w &= 0 \quad \text{on} \quad \partial\Omega_e \\ \partial_\nu w &= 0 \quad \text{on} \quad \partial\Omega_e \end{aligned} \quad , \tag{22}$$

where Ω_e is some domain containing Ω and ∂_ν denotes differentiation in the direction of the unit normal $\boldsymbol{\nu}$ at the given point of the boundary. Once w is computed, f_e is as defined in the continuous case. While the methods for Equation (22) are not as well developed as in the Laplace case, there exist similar potential-theory based integral equations and fast solution methods for the solution of the biharmonic problem. See, for instance, P. Farkas' PhD thesis [15]. There are two main reasons to not consider extrapolations based on polyharmonic equations of higher order than the biharmonic equation: (1) very few numerical tools have been developed for such equations and (2) the equations require to provide high order derivatives of the data f in the direction normal to the boundary, which in most physical applications are not readily available, and can be challenging to compute with high accuracy numerically, even when using integral equation based methods [40].

For our numerical tests, and in the version of the code which will be available online, only the harmonic expansion calculated by solving (20) is implemented. The details of our implementation are given in Section 5 in parallel with the calculation of u^H, since both computations rely on the same mathematical and numerical tools.

5. Computing the homogeneous solution and the harmonic extension

In this section, we describe how we compute the homogeneous solution u^H which solves the harmonic problem ((4)–(5)). Since we use similar numerical techniques to solve this problem and to compute the global function extension through (20), we will also discuss the latter, and highlight the small differences between the two situations.

5.1. Layer potentials

Before we proceed, we should clarify what we mean by a multiply connected domain and the normal direction to the boundary curve. Let Ω be an interior domain with boundary $\partial\Omega$. For a multiply connected domain, $\partial\Omega$ is given as the union over disjoint, closed curves $\partial\Omega = \bigcup_{i=0}^{l} \Gamma_i$, with Γ_0 corresponding to the outer boundary. The normal direction on each component Γ_i is taken to be the direction pointing away from Ω and is denoted by $\boldsymbol{\nu}$. To denote the normal at a specific point $\mathbf{x}$, we use the notation $\boldsymbol{\nu}(\mathbf{x})$. For Γ_0, this vector points to the exterior of the curve and for Γ_i, $i = 1, \ldots, l$, this points to the interior of the curve. To see a simple illustration of such a domain and its normal vectors, see Fig. 2 in Section 6.

We write both the homogeneous solution u^H and the extension w of f in $\Omega_B \setminus \Omega$ as layer potentials [19]. Specifically, for the homogeneous solution we write

$$u^H(\mathbf{x}) = S\mu(\mathbf{x}) + D\mu(\mathbf{x}) \tag{23}$$

where μ is an unknown density, and

$$S\mu(\mathbf{x}) = \int_{\partial\Omega} G(\mathbf{x}, \mathbf{y})\mu(\mathbf{y})\, d\mathbf{y}\,, \tag{24}$$

$$D\mu(\mathbf{x}) = \int_{\partial\Omega} \partial_{\nu_y} G(\mathbf{x}, \mathbf{y})\mu(\mathbf{y})\, d\mathbf{y} \tag{25}$$

with ∂_{ν_y} denoting differentiation in the direction $\boldsymbol{\nu}(\mathbf{y})$. The function $S\mu(\mathbf{x})$ is known as a single layer potential, and $D\mu(\mathbf{x})$ is known as a double layer potential [19].

For the harmonic function extension, we write w as

$$w(\mathbf{x}) = D\sigma(\mathbf{x}) + W\sigma \tag{26}$$

where σ is an unknown density, and

$$W\sigma = \int_{\partial\Omega} \sigma(\mathbf{y})\, d\mathbf{y}. \tag{27}$$

Now, let $\mathcal{S}\mu(\mathbf{x}_0)$, $\mathcal{D}\mu(\mathbf{x}_0)$ and $\mathcal{D}\sigma(\mathbf{x}_0)$ denote the restrictions of S and D to points $\mathbf{x}_0$ on the boundary $\partial\Omega$, where the integrals are taken in the Cauchy principal value sense when necessary. $u_H(\mathbf{x})$ and $w(\mathbf{x})$ reach the following limiting values as $\mathbf{x}$ approaches a point $\mathbf{x}_0$ on the boundary [19]

$$\lim_{\mathbf{x}\to\mathbf{x}_0, \mathbf{x}\in\Omega} u^H(\mathbf{x}) = g(\mathbf{x}_0) - \tilde{v}(\mathbf{x}_0) = -\frac{1}{2}\mu(\mathbf{x}_0) + \mathcal{S}\mu(\mathbf{x}_0) + \mathcal{D}\mu(\mathbf{x}_0) \tag{28}$$

and

$$\lim_{\mathbf{x}\to\mathbf{x}_0,\mathbf{x}\in\Omega} w(\mathbf{x}) = f(\mathbf{x}_0) = \frac{1}{2}\sigma(\mathbf{x}_0) + \mathcal{D}\sigma(\mathbf{x}_0) + W\sigma. \tag{29}$$

(28) is a second kind integral equation (SKIE) for μ, and (29) an SKIE for σ.

At this point, we have the desired integral representations for u^H and for w, and equations for their associated densities. The representation (23) for u^H has been used in commercial software [45] and is known in the integral-equations community [20] but the authors are unaware of any treatment of the Fredholm alternative as applied to the resulting integral equation (28). We consider a proof of the invertibility of (28) to be beyond the scope of this paper but note that the argument of Lemma 29 in [37] can be modified to provide a proof of its invertibility, even on multiply-connected domains. The invertibility of (29) is well-known [19].

We now discuss the numerical methods we chose to solve (28) and (29) and to evaluate the integrals in (23) and (26).

5.2. Solving the second kind integral equations for the densities

In our solver, we discretize $\partial\Omega$ using panels of 16 scaled Legendre nodes. Our numerical methods rely on the following simplifying assumptions concerning the boundary $\partial\Omega$: (1) the boundary is C^n for some large n and (2) the panels are chosen fine enough so that for source and target nodes on distinct, non-adjacent panels the integrals of the layer potentials are computed to high precision using the standard Gaussian weights (the "source" and "target" terminology is explained below). Note that there exist more complex algorithms that would allow us to relax both assumptions, and their implementation in our solver will be the subject of future work. To relax the first assumption, one could use any of the methods described in [22,4,8,42] to allow domains with corners. While the second assumption is not necessarily much of a limitation on the types of domains which can be handled by our solver, the fineness implied by this assumption can lead to too great a computational burden for certain domains, such as domains in which the boundary comes close to intersecting itself. There are methods in development, as in the planned sequel to [38], which provide a more efficient approach for such cases.

Now, let $\partial\Omega$ be discretized into L panels using $M = 16L$ total nodes and denote the ith node by $\mathbf{x}_i$. Using generalized Gaussian quadrature for the interactions between nodes on the same and adjacent panels and the standard, scaled Gaussian weights otherwise, we obtain a Nyström discretization of (28) and (29):

$$g(\mathbf{x}_i) - \tilde{\nu}(\mathbf{x}_i) = -\frac{1}{2}\mu_i + \sum_{j=1}^{M}\left(G(\mathbf{x}_i,\mathbf{x}_j)\mu_j\omega^s_{i,j} + \partial_{\nu_j}G(\mathbf{x}_i,\mathbf{x}_j)\mu_j\omega^d_{i,j}\right), \tag{30}$$

$$f(\mathbf{x}_i) = \frac{1}{2}\sigma_i + \sum_{j=1}^{M}\left(\partial_{\nu_j}G(\mathbf{x}_i,\mathbf{x}_j)\sigma_j\omega^d_{i,j} + \sigma_j\omega_j\right) \tag{31}$$

where $\mu_i = \mu(\mathbf{x}_i)$, $\sigma_i = \sigma(\mathbf{x}_i)$, ∂_{ν_j} denotes differentiation in the direction $\boldsymbol{\nu}(\mathbf{x}_j)$, and the $\omega^s_{i,j}$, $\omega^d_{i,j}$, and ω_j correspond to integration weights. We note that the expressions above are a slight abuse of notation as the Green's function and its derivatives are undefined when $j = i$. The true formula is more generally a function of the boundary and the kernel but we find the above more edifying. In the current context, the relevant piece of information is that there exist weights $\omega^s_{i,j}$, $\omega^d_{i,j}$, and ω_j such that the quadratures appearing in the second kind equations can be evaluated with high-order accuracy. For a more detailed treatment of the generalized Gaussian quadrature framework, see [8]. In the following sections, we will refer to the ω_j, which are given by appropriately scaling the standard Gauss–Legendre weights, as the smooth quadrature weights.

There exist many tools available for the fast solution of the linear systems (30) and (31). There are iterative solution techniques, e.g. GMRES [41], which perform well for linear systems discretized from SKIEs on simple domains. The computational cost of such a scheme is typically dominated by a term of the order qT where q is the number of iterations required to converge and T is the amount of work for a matrix–vector product. For well-conditioned problems with M boundary nodes, typically $q = \mathcal{O}(1)$ and the cost of T can be reduced to $\mathcal{O}(M)$ with an FMM. There are also fast-direct solution methods, i.e., methods which construct, in $\mathcal{O}(M)$ or $\mathcal{O}(M\log M)$ time, a representation of the inverse of the system matrix which can be applied in $\mathcal{O}(M)$ or $\mathcal{O}(M\log M)$ time. For such direct methods, the cost of forming the representation of the inverse is often much greater than that of the FMM, while the speed of applying the inverse, once computed, is often faster than the FMM. Fast-direct solvers can be particularly useful for problems in highly-irregular domains, in which the iteration count of an iterative solver may be too high or unpredictable. They are also advantageous for cases in which several Laplace problems need to be solved for a fixed domain, since the high initial cost only has to be paid once. For our solver, we implemented the direct method developed by [23], which is optimized for the type of problems considered here, and found that it gave very satisfactory performance.

5.3. Evaluation of u^H and w by quadrature by expansion

Once μ and σ are computed, we evaluate u^H and w by direct computation of the integrals (23) and (26). This step can at first appear complicated because the integral kernels are near singular for the evaluation of points near the boundary

of the domain. We resolve the difficulty by computing the integrals for points near the boundary using the quadrature by expansion (QBX) method. We will not present the fundamentals of the QBX scheme here, since clear presentations for situations very closely related to the one we encounter here can be found in [26,5,40,3]. We will however stress two modifications to the standard QBX scheme which we implemented in our solver. First, we accelerated the evaluation of the layer potentials with the FMM (a similar but more sophisticated acceleration scheme is presented in [38]). Second, we developed a variant of QBX which allows, after precomputation of the field at a fixed number of points, the evaluation of the field anywhere in the domain in $\mathcal{O}(1)$ time [3]. This is particularly convenient for the evaluation of the function extension when we construct the adaptive tree, since the grid points at which the values of the layer potential are desired may not be known a priori.

5.3.1. Explanation of the algorithm

Let us discuss these two modifications to the standard QBX method in more detail. Consider, for example, the evaluation of the layer potential $u^H = S\mu + D\mu$. We use the notation as above for the discretization nodes $\mathbf{x}_i$, the smooth quadrature weights ω_i, and the boundary normals $\boldsymbol{\nu}_i = \boldsymbol{\nu}(\mathbf{x}_i)$ of $\partial\Omega$. Let $\mathbf{c}_i$ be the QBX centers, located at a distance r_i from the boundary: $\mathbf{c}_i = \mathbf{x}_i - r_i\boldsymbol{\nu}_i$. In the QBX method, the potential u is approximated by a power series in the disc of radius r_i about $\mathbf{c}_i$, denoted by $B_{r_i}(\mathbf{c}_i)$. For any $\mathbf{x}$ in $B_{r_i}(\mathbf{c}_i)$, we write

$$u^H(\mathbf{x}) \approx \mathrm{Re}\left(\sum_{l=0}^{p} \alpha_{l,i}(z-\xi)^l\right), \tag{32}$$

where $z = x_1 + ix_2$ and $\xi = c_{i,1} + ic_{i,2}$.

We will present a simple method for computing the QBX coefficients, $\alpha_{l,i}$, with FMM acceleration. For points $\mathbf{x}$ which are sufficiently far from the boundary, the values of u^H can be evaluated with high accuracy using the smooth integration weights ω_j, i.e. the formula

$$u^H(\mathbf{x}) \approx \sum_{j=1}^{M} \left(G(\mathbf{x},\mathbf{x}_j)\mu_j\omega_j + \partial_{\nu_j}G(\mathbf{x},\mathbf{x}_j)\mu_j\omega_j\right) \tag{33}$$

is accurate for such points. Naïvely, the computational cost of evaluating the sum (33) is $\mathcal{O}(MN)$ for N targets $\mathbf{x}$. This sum can be computed at N targets with cost $\mathcal{O}(M+N)$ using a standard FMM [17,10,24]. Therefore, if the coefficients $\alpha_{l,i}$ can be recovered from function evaluations of u^H, they can be computed with FMM acceleration.

Let $0 < \delta < 1$ be given. Consider the integral of the power series approximation to u^H (32) about the circle of radius δr_i about $\mathbf{c}_i$. We have

$$\frac{1}{2\pi}\int_0^{2\pi} u^H\left(\mathbf{c}_i + \delta r_i(\cos\theta,\sin\theta)\right)\,d\theta \approx \frac{1}{2\pi}\mathrm{Re}\left(\int_0^{2\pi}\sum_{l=0}^{p}\alpha_{l,i}\left(\delta r_i\right)^l e^{il\theta}\,d\theta\right) = \mathrm{Re}\left(\alpha_{0,i}\right), \tag{34}$$

where we have used the orthogonality properties of complex exponentials. Note that $\mathrm{Re}\left(\alpha_{0,i}\right)$ is precisely what is needed to evaluate the first term in the expansion (32). By similar reasoning, the coefficients $\alpha_{l,i}$ can be recovered from the following integral on the circle of radius δr_i about $\mathbf{c}_i$:

$$\alpha_{l,i} = \frac{1}{\pi\delta^l r_i^l}\int_0^{2\pi} u^H\left(\mathbf{c}_i + \delta r_i(\cos\theta,\sin\theta)\right)e^{-il\theta}\,d\theta\,. \tag{35}$$

The layer potential u^H is smooth on the circle of radius δr_i, so the $\alpha_{l,i}$ can be computed with high order accuracy using the trapezoidal rule to discretize the integral (35). Let M_{QBX} equispaced points $\mathbf{y}_{i,j}$ be placed on the circle $\partial B_{\delta r_i}(\mathbf{c}_i)$. The values $u^H(\mathbf{y}_{i,j})$ can be computed accurately using the smooth quadrature weights for $\partial\Omega$ to approximate the single and double layer potentials there, assuming that $(1-\delta)r_i$ is large enough (this is the closest that $\mathbf{y}_{i,j}$ will be to the boundary). For sufficient sampling, M_{QBX} should be taken larger than $2p$.

Once the coefficients are computed, the power series (32) can be used to approximate the potential at targets which are close to the boundary. High accuracy can be obtained when r_i is sufficiently small.

With these preliminaries in place, the FMM-accelerated algorithm for the evaluation of the potential at N targets $\mathbf{t}_i$ can be described in the following steps:

- Place M centers at the points $\mathbf{c}_i = \mathbf{x}_i - r_i\boldsymbol{\nu}_i$.
- Define M_{QBX} equispaced points $\mathbf{y}_{i,j}$ for $j = 1,\ldots,M_{QBX}$ on the circle of radius δr_i about each center $\mathbf{c}_i$.
- Call the FMM to evaluate u^H at the targets $\mathbf{t}_i$ and the points $\mathbf{y}_{i,j}$, where the layer potentials are approximated using the smooth quadrature weights ω_i. This is an $\mathcal{O}(MM_{QBX}+N)$ procedure.

- Compute the coefficients $\alpha_{l,i}$ for each center as in (35), using the trapezoidal rule. This takes $\mathcal{O}(MM_{QBX}\log M_{QBX})$ work for $M_{QBX} > 2p$ using the FFT.
- For each target which is within $(1-\delta)r_i$ of any boundary node $\mathbf{x}_i$, let $\mathbf{c}_j$ be the nearest QBX center. The smooth rule might not be accurate for this target, so instead use the value given by the power series (32) about $\mathbf{c}_i$. The cost for this is $\mathcal{O}(p)$ per target.

The scheme presented above is satisfactory if the targets $\mathbf{t}_i$ are all known in advance. However, when constructing the adaptive tree, the potential associated with the function extension may have to be evaluated at new targets $\mathbf{t}$. For the new targets which are close to the boundary, the potential can be computed using the expansion about the nearest QBX center. For the targets further from the boundary, we avoid calling the FMM again to compute the potential there. Instead, we store the multipole and local expansion coefficients computed for all boxes in the hierarchy during the initial call to the FMM. The values of the potential at the new targets $\mathbf{t}$ can then be evaluated in $\mathcal{O}(p_{FMM})$ work for each target, where p_{FMM} is the order of the multipole and local expansions in the FMM [3].

5.3.2. Choosing the QBX parameters

In the above, we have avoided the key topic of how to select r_i, the radius of the QBX expansion. For the sake of argument, we will assume for now that $\delta \leq 3/4$. This must be done to balance two competing concerns: (1) that $(1-\delta)r_i \geq r_i/4$ is sufficiently large so that $u^H(\mathbf{y}_{i,j})$ can be computed accurately using the smooth quadrature weights, ω_i, and (2) that r_i is sufficiently small so that the truncated power series (32) is an accurate approximation of u^H. While it may seem unclear whether choosing an appropriate r_i is indeed possible, this fact was proven in [13]. Now, if the discretization node $\mathbf{x}_i$ is on a boundary panel of length h_i, it can be shown that setting $r_i = 4h_i$ provides high accuracy when using 16 Legendre nodes on each panel. However, having the center so far places further restrictions on the discretization of the domain, since no boundary points are allowed to be in the interior of the QBX disc. Thus, in practice one often takes $r_i = h_i$ and computes $u^H(\mathbf{y}_{i,j})$ using the smooth weights for an oversampled version of the boundary.

We finally get to the choice of the parameter δ. Two issues must be considered. The first is that $(1-\delta)r_i$ must be sufficiently large, as noted above. For larger δ, the boundary must be oversampled more in order to compute u^H accurately at the $\mathbf{y}_{i,j}$. The second issue is numerical stability. It is clear that for a very small choice of δ, there may be numerical issues in determining the difference between various points $\mathbf{y}_{i,j}$. However, there is a more important source of error to consider for small δ. By examining the formulas for the QBX coefficients (35) and the power series (32), we see that any error in computing the $\alpha_{l,i}$ can be amplified by a factor of $1/\delta^p$ in the worst case, i.e. for points on the boundary of the QBX disc. From this, we can see that for a higher-order scheme (larger p) it is desirable to have a larger δ and therefore more oversampling of the boundary. This phenomenon — the need to oversample more for a higher order scheme — will be familiar to practitioners of QBX.

Let $\mathbf{x}_i$ be a boundary point and h_i the length of its boundary panel. For the computations in Section 6, we used the following parameters in the QBX scheme:

- (Distance to QBX center) $r_i = h_i$
- (QBX order) $p = 8$
- (Number of points in integral) $M_{QBX} = 40$
- (Radius for integral) $\delta r_i = 3r_i/4$, i.e. $\delta = 3/4$
- (FMM order) $p_{FMM} = 52$
- (Boundary oversampling factor) $n_{over} = 4$

Note that by "oversampling the boundary" we mean that we place $16n_{over}$ Legendre nodes on each panel instead of 16. The parameters above were found to work well in practice but more optimal choices may be possible.

Remark 2. The idea of using equispaced points on a circle to form a power series expansion of a harmonic function is reminiscent of the "fast multipole method without multipoles" of [2].

5.4. Derivatives of u^H

As mentioned in the introduction, the values of the derivatives of the solution are important for many physical applications. Unfortunately, there are two sources of difficulty for obtaining high accuracy values of the derivative with our scheme when solving with Dirichlet boundary conditions. The first is that the standard QBX method loses precision when computing derivatives of a double layer potential. The second source of difficulty stems from the nature of the Dirichlet problem and the way that we interpolate function values in an embedded boundary method. We address these issues in detail in the next two sections, and show that while the first issue can be avoided with a clever use of the Cauchy–Riemann equations, the second issue is intrinsic to the fact that the scheme we present in this article relies on C^0 extension.

5.4.1. Derivatives of a double layer potential

In the QBX setting, the derivatives of the layer potentials can be obtained by differentiating the QBX power series expansion. This works well for the single layer potential but has been observed to result in the loss of precision for the double layer potential [26]. The cause of this loss of precision is unclear but may result from the hyper-singular nature of the derivatives of the double layer potential as operators on the boundary.

It was pointed out to us by Manas Rachh and Leslie Greengard [39] that the evaluation of the derivatives of the double layer potential can be accomplished by differentiating the density along the boundary and using the QBX algorithm for the Cauchy kernel (which behaves like the double layer kernel). The key observations are that

$$D\mu(\mathbf{x}) = \operatorname{Re}\left(-\frac{1}{2\pi i}\int \frac{\mu(\xi)}{\xi - z}\,d\xi\right), \tag{36}$$

$$\left(\partial_{x_1} - i\partial_{x_2}\right) D\mu(\mathbf{x}) = -\frac{1}{2\pi i}\int \frac{\mu(\xi)}{(\xi - z)^2}\,d\xi, \tag{37}$$

$$-\frac{1}{2\pi i}\int \frac{\mu(\xi)}{(\xi - z)^2}\,d\xi = -\frac{1}{2\pi i}\int \frac{\mu'(\xi)}{\xi - z}\,d\xi, \tag{38}$$

where $z = x_1 + ix_2$ and $\xi = y_1 + iy_2$ for $\mathbf{y} \in \partial\Omega$. Let $\boldsymbol{\tau}$ denote the unit tangent on the boundary and ∂_τ denote differentiation along the boundary. The following steps summarize the algorithm for computing $\nabla D\mu$ implied by the above observations:

- Compute the derivative $\partial_\tau \mu$ using spectral differentiation on each panel of Legendre nodes.
- Compute

$$F(\mathbf{x}) = -\frac{1}{2\pi i}\int \frac{\partial_\tau \mu(\xi)}{\xi - z}\,d\xi, \tag{39}$$

  with $z = x_1 + ix_2$ for each target $\mathbf{x}$ using a standard QBX algorithm.
- Obtain the values of the gradient by

$$\partial_{x_1}(D\mu(\mathbf{x})) = \operatorname{Re}(F(\mathbf{x})), \tag{40}$$

$$\partial_{x_2}(D\mu(\mathbf{x})) = -\operatorname{Im}(F(\mathbf{x})). \tag{41}$$

With the above algorithm, the difficulties associated with evaluating derivatives of the double layer potential are avoided by instead evaluating the derivative of a function along a curve, which is relatively simple with our discretization of the boundary with panels of Legendre nodes. We have implemented this method for the computations in Section 6.

5.4.2. Embedded boundary methods and the Dirichlet boundary condition

We will introduce some notation in order to discuss the effect of the boundary correction on the overall error. Let $\widehat{u}(\mathbf{x})$ be the computed value of the solution u at a point $\mathbf{x}$ and $\widehat{u^H}(\mathbf{x})$ be the computed value of $u^H(\mathbf{x})$. Let Vf_e be the volume integral of f_e, $\mathbb{V}\tilde{f}_e$ be the computed values of Vf_e at the grid points, and $\tilde{v}(\mathbf{x})$ be the interpolation of those computed values at any given point $\mathbf{x}$, as in Section 3. Likewise, let ∇Vf_e be the gradient of the volume integral of f_e, $\nabla\mathbb{V}\tilde{f}_e$ be the computed values of ∇Vf_e at the grid points, and $\tilde{g}(\mathbf{x})$ be the interpolation of those computed values at any given point $\mathbf{x}$. Recall that u is given by

$$u(\mathbf{x}) = Vf_e(\mathbf{x}) + u^H(\mathbf{x}), \tag{42}$$

where $u^H(\mathbf{x})$ solves

$$\Delta u^H = 0 \text{ in } \Omega \tag{43}$$

$$u^H = g - Vf_e|_{\partial\Omega} \text{ on } \partial\Omega. \tag{44}$$

A conservative estimate of the error is then given by

$$|u(\mathbf{x}) - \widehat{u}(\mathbf{x})| \leq |Vf_e(\mathbf{x}) - \tilde{v}(\mathbf{x})| + |u^H(\mathbf{x}) - \widehat{u^H}(\mathbf{x})|. \tag{45}$$

Likewise, we have

$$|\nabla u(\mathbf{x}) - \nabla\widehat{u}(\mathbf{x})| \leq |\nabla Vf_e(\mathbf{x}) - \tilde{g}(\mathbf{x})| + |\nabla u^H(\mathbf{x}) - \nabla\widehat{u^H}(\mathbf{x})|. \tag{46}$$

We have addressed the error due to the volume integral (the first term on the right-hand sides of (45) and (46)) in Sections 3 and 4. We will now discuss the effect of the error in the boundary correction (the second term on the right-hand sides of (45) and (46)).

In most of the numerical tests we perform in Section 6, the discretization error, solution error, and QBX error associated with the boundary integral are made to be so small that the error in the boundary correction u^H is dominated by the error

in the interpolated values of the volume potential on the boundary. In other words, we assume that $\widehat{u^H}$ is exactly a solution of

$$\Delta \widehat{u^H} = 0 \text{ in } \Omega \tag{47}$$

$$\widehat{u^H} = g - \tilde{v}|_{\partial\Omega} \text{ on } \partial\Omega\,, \tag{48}$$

and the source of the error is that we are solving for the wrong boundary condition. By the maximum principle, we have that

$$|u^H(\mathbf{x}) - \widehat{u^H}(\mathbf{x})| \leq \max_{\mathbf{y}\in\partial\Omega} |Vf_e(\mathbf{y}) - \tilde{v}(\mathbf{y})|\,, \tag{49}$$

so that this error is of the same order as the contribution from the volume integral. Let us now turn to the error in $\nabla\widehat{u^H}$. We may make the same assumption about the discretization error, but need to have a close look at the boundary value problem $\nabla\widehat{u^H}$ solves. For a function ψ defined on the boundary $\partial\Omega$, let $DtN[\psi]$ denote the Dirichlet-to-Neumann map applied to ψ, i.e. if φ is the solution of

$$\Delta\varphi = 0 \text{ in } \Omega \tag{50}$$

$$\varphi = \psi|_{\partial\Omega} \text{ on } \partial\Omega\,, \tag{51}$$

then $DtN[\psi] = \partial_\nu\varphi|_{\partial\Omega}$. Having introduced this notation, we may write that $\nabla\widehat{u^H}$ is a solution of the following

$$\Delta\nabla\widehat{u^H} = 0 \text{ in } \Omega \tag{52}$$

$$\nabla\widehat{u^H} = \boldsymbol{\nu} DtN[g - \tilde{v}] + \boldsymbol{\tau}\partial_\tau(g - \tilde{v})|_{\partial\Omega} \text{ on } \partial\Omega\,. \tag{53}$$

Again, we may apply the maximum principle to obtain

$$|\nabla u^H(\mathbf{x}) - \nabla\widehat{u^H}(\mathbf{x})| \leq \max_{\mathbf{y}\in\partial\Omega} |\nabla Vf_e(\mathbf{y}) - \boldsymbol{\nu}(\mathbf{y})DtN[\tilde{v}](\mathbf{y}) - \boldsymbol{\tau}(\mathbf{y})\partial_\tau\tilde{v}(\mathbf{y})|\,. \tag{54}$$

We see that the accuracy of the gradient of the potential will be affected more strongly than the accuracy of the potential by the error in the boundary correction because the gradient of the solution of the Laplace Dirichlet problem depends on the accuracy of the tangential derivative (and Dirichlet-to-Neumann map) of the boundary values. This is typically not a concern in the integral equations context, for two reasons. First, one usually assumes that one has high order accurate values for the boundary data. Second, the evaluation of the gradient of a layer potential is smoothing for points sufficiently far from the boundary. However, in the context of the Poisson solver we present in this article, the tangential derivative of $\tilde{v}$ is, as an analytic matter, one order lower than the order of $\tilde{v}$ (this is because $\tilde{v}$ is a polynomial on each box) and the value of the gradient may be requested arbitrarily close to (or even on) the boundary. This loss of accuracy for the boundary data results in a similar loss of accuracy for the computed gradient.

For a smoothly extended function f_e, a potential way to address this problem is as follows. The box code can be used to compute $\nabla\mathbb{V}\tilde{f}_e$ at the collocation points in the same way that $\mathbb{V}\tilde{f}_e$ is computed. The interpolant of the gradient computed this way, which we call $\tilde{\mathbf{g}}$, is the same order as $\tilde{v}$. We can then construct a new approximation to Vf_e on the boundary by first computing $\tilde{\mathbf{g}}\cdot\boldsymbol{\tau}$ along the boundary, where $\boldsymbol{\tau}$ is the tangent vector of the curve, and then computing its indefinite integral panel-wise (we correct for the constant using $\tilde{v}$, again panel-wise). The resulting function is the same order accuracy as $\tilde{v}$ but its derivative is a better approximation to the derivative of $V\tilde{f}_e$. We investigate the merits of this alternative approach numerically in Section 6.

Unfortunately, this approach does not appear to improve the order of accuracy for non-smooth f_e. As noted in Section 4.1, we expect the convergence of the computed gradient to be one order lower than the potential for an extended density f_e obtained through continuous extension or extension by zero, i.e. there is no advantage to using $\tilde{\mathbf{g}}$ as it is already one order of accuracy lower than $\tilde{v}$. This was evident in numerical experiments, where the accuracy in the derivatives was comparable using this new approximation to $V\tilde{f}_e$ instead of $\tilde{v}$. We therefore leave these results out of the next section. In the case that a C^k extension is available for sufficiently large k, this technique may prove important to achieving super-convergence.

6. Numerical results

In order to verify the preceding analysis and test the performance of the numerical method we propose in this article, we have implemented a Poisson solver in Fortran which combines all the different modules we presented in the previous sections. The volume integral code is a modified version of the original Fortran code of [14] (using fourth order interpolants on leaf boxes), with some added OpenMP parallelism and the modification for computing gradient values discussed in Section 3.2. The codes for the boundary correction and the continuous extension were written specifically for this work, and are based on the methods described in Sections 4 and 5. We are currently documenting the numerical solver we used to generate the results shown below, and will make it freely available online at a later date.

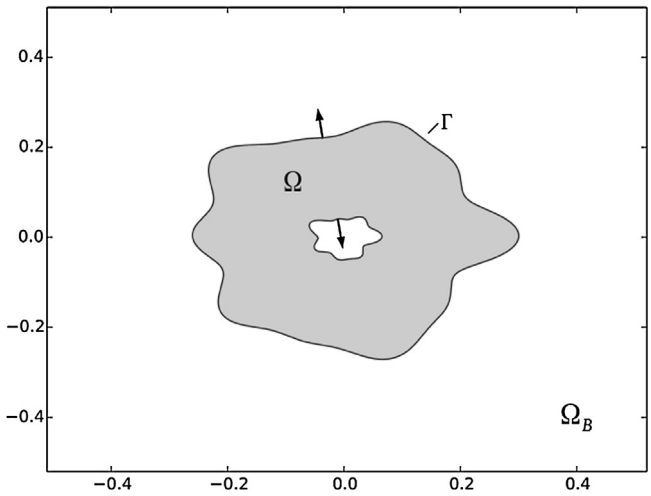

Fig. 2. The domain Ω and its boundary Γ. The axes coincide with the boundary of the containing box Ω_B. Two outward-pointing boundary normal vectors are indicated by arrows.

For each numerical test, we have used the domain Ω shown in Fig. 2, which has an irregular boundary and is multiply connected. The interfaces of this domain are specified by parametric equations in polar coordinates. Specifically, each interface is given by a set of points $(\theta, r(\theta))$ for $\theta \in [0, 2\pi)$, where $r(\theta) = c_0 + \sum_j (c_j \cos(j\theta) + d_j \sin(j\theta))$. The choice of the coefficients was arbitrary. For reference, the non-zero coefficients for the outer boundary were $c_0 = 0.25$, $d_3 = c_6 = c_8 = c_{10} = 0.01$, $c_5 = 0.02$. The non-zero coefficients for the inner boundary were $c_0 = 0.05$, $c_2 = d_3 = c_5 = c_7 = 0.005$.

Let N_p denote the number of panels used in the discretization of the boundary and $M = 16 * N_p$ denote the total number of boundary points (we use 16 Legendre nodes on each panel throughout). For the volume integral nodes, let N_V denote the total number of points in Ω_B and N_Ω denote the number of points inside Ω. For each test, we approximated the relative L_∞ error,

$$E(\psi) = \frac{\max_\Omega |\psi_{exact} - \psi_{numerical}|}{\max_\Omega |\psi_{exact}|},$$

where ψ is either the potential or its derivatives, by sampling at 10^6 points randomly placed in Ω. These points were kept the same for each discretization level for the sake of convergence tests. We report the error in the gradient below as $\sqrt{E(u_x)^2 + E(u_y)^2}$. In the error analysis of this section, the density for the boundary correction is computed with high accuracy (say 12 digits) and the corresponding layer potential is evaluated with high accuracy as well (say 12 digits for the potential and 8–9 digits for its gradient). With this assumption, the error will be primarily a function of the number of discretization nodes in the volume, i.e. N_Ω.

All computations were performed on a desktop computer with an Intel® Xeon(R) CPU E3-1220 v5 (3.00 GHz, 4 core) and 16 Gb of memory. A few of the computations depend only on the boundary and therefore take the same amount of time for each discretization level. In the first example, the boundary was discretized with $M = 9{,}280$ nodes. The precomputation time for the direct solver took 1.20 and 1.80 seconds for the continuous function extension and boundary correction linear systems, respectively. The precomputation time to allow for $\mathcal{O}(1)$ access to the layer potential took 0.60 and 1.20 seconds for the continuous function extension and boundary correction layer potentials, respectively. The solution in the second example is much more irregular than in the first and thus more boundary points were required. For this case, the boundary was discretized with $M = 14{,}208$ nodes. The precomputation time for the direct solver took 1.57 and 2.89 seconds for the continuous function extension and boundary correction linear systems, respectively. The precomputation time to allow for $\mathcal{O}(1)$ access to the layer potential took 1.10 and 1.99 seconds for the continuous function extension and boundary correction layer potentials, respectively. We consider these computational costs to be modest and emphasize that the precomputation of the fast-direct solver must only be done once per domain. We report on the speed of the volume integral code and the evaluation of the layer potentials below.

6.1. A note on adaptivity

When a smooth extension f_e is known, the bound (11) of Section 3.3 implies a rather straightforward *a priori* adaptive discretization strategy: for a given tolerance, refine the tree until the local polynomial interpolant on each leaf box approximates f_e within that tolerance, which can be tested by comparing f_e and the interpolant on a finer grid. It turns out that this strategy will result in an overall error well below the desired tolerance. A modification which, in practice, gets closer

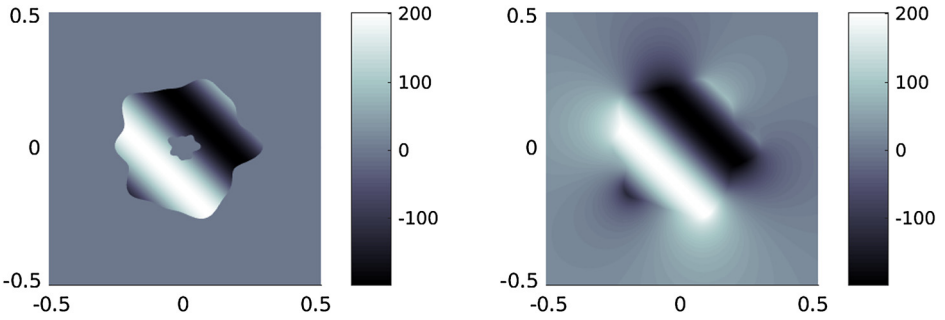

Fig. 3. The extended density f_e for Example 1 using extension by zero (left) and continuous extension (right).

to the desired tolerance is to refine until the error in the local polynomial interpolation times the area of the box is within the tolerance on each leaf box.

For piecewise smooth f_e, we saw in Section 4.1 that the bound (11) may be pessimistic. However, the analysis of that section offers little in terms of an *a priori* discretization strategy. If the above strategy for smooth f_e is implemented, the accuracy of the resulting scheme is often not even competitive with a uniform grid. We consider the problem of efficient *a priori* adaptive discretization to be open in this setting but have empirically found the following scheme to compare favorably to uniform discretization in our tests: weight the error approximation using the size of the given leaf box as described above but using the area of the leaf box for boxes which intersect the boundary (where f_e is less smooth) and using the sidelength of the leaf box otherwise. In this sense, we are more forgiving of the approximation error for boxes where f_e is merely continuous.

It seems that in many situations an *a posteriori* discretization strategy would be more efficient in terms of accuracy per grid point. While this may be an intuitive statement, it is not clear whether an *a posteriori* scheme would be more efficient in terms of accuracy per flop because such a scheme may require several successive iterations. We do not attempt to answer this question here but emphasize that the issue with (11) is a matter of efficiency rather than correctness, i.e. if an *a priori* bound is required, (11) provides one, it just may be an over-estimate.

6.2. Example 1

For Example 1, we choose a known, relatively smooth solution u given by

$$u(\mathbf{x}) = \sin(10(x_1 + x_2)) + x_1^2 - 3x_2 + 8 \tag{55}$$

and calculate f analytically by direct differentiation, and g by evaluating u on $\partial\Omega$. Fig. 3 shows heat maps of the corresponding f_e obtained by zero extension and by continuous extension. Example 1 is relatively simple on purpose, in order to test the validity of the analysis of the previous sections.

First, consider the question of super-convergence for a smooth extension f_e. This is simple to test numerically as the formula (55) for u is smooth on $\mathbb{R}^2$. In Section 5, we noted that the boundary correction can be computed with two different types of boundary data. Let version 1 denote the boundary data obtained from $\tilde{v}$ and version 2 denote the boundary data obtained by integrating $\boldsymbol{\tau} \cdot \tilde{\mathbf{g}}$, where we have reused the notation of Section 5. We perform a convergence test on uniform trees for both versions 1 and 2. According to the analysis of the preceding sections, version 1 should display fourth order convergence for the potential and sub-fourth order convergence for the gradient, while version 2 should display super-convergence, i.e. fourth order for both the potential and gradient.

In Fig. 4, we see that the analysis is largely confirmed. While we cannot conclude decisively regarding the convergence order of the gradient for version 1, it is indeed fourth order for version 2. Note that the slope seems to taper off for the last point, which is likely due to the fact that one is approaching the accuracy of the QBX evaluation of the derivative. In terms of accuracy per grid point, version 2 is clearly superior to version 1.

Next, we consider the question of the convergence order using extension-by-zero and continuous extension with a layer potential. The analysis of Section 4.1 suggests that we should see second order convergence for the potential and first order convergence for the gradient using extension-by-zero. This should be improved to third order for the potential and second order for the gradient by using continuous extension. As a reminder, these rates are to be compared with the rates implied by the coarser error bound (11), which suggests that the extension-by-zero scheme would not converge and that the continuous extension scheme would be merely first order in the potential and derivative. To test the reasoning of Section 4.1, we performed a convergence test of the extension by zero and continuous extension methods on uniform trees.

The results are shown in Fig. 5, and confirm that the analysis of Section 4.1 gives a better sense of the convergence rate than a naïve application of the bound (11).

Next, we consider the question of adaptive grid refinement. An adaptive grid should be able to provide significant gains, especially for the nonsmooth f_e. For the results presented here, we use an adaptive tree based on *a priori* error estimates, as described in the previous subsection. This refinement rule tends to place more boxes near the boundary because of the

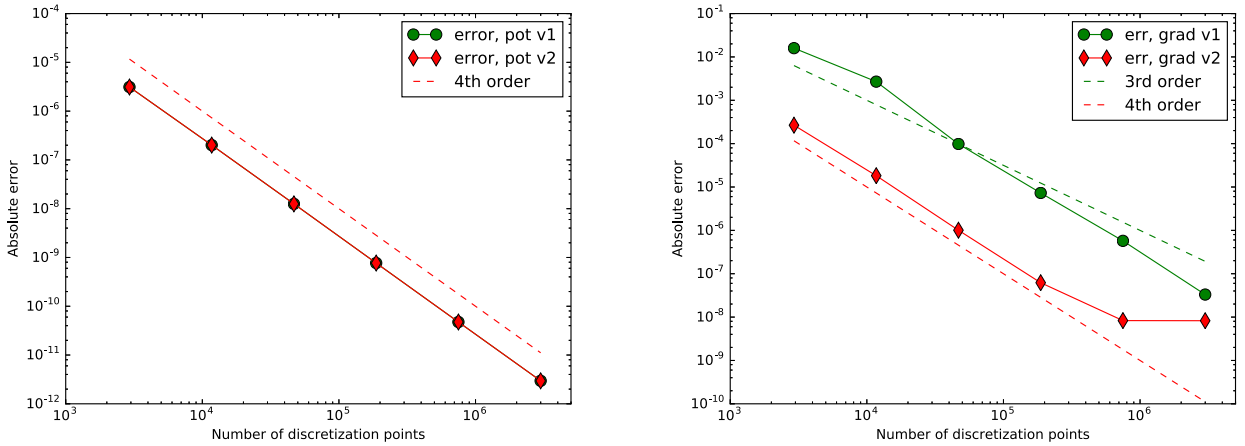

Fig. 4. Example 1, smooth extension. Accuracy of versions 1 (green circles) and 2 (red diamonds) for the potential (left) and gradient (right).

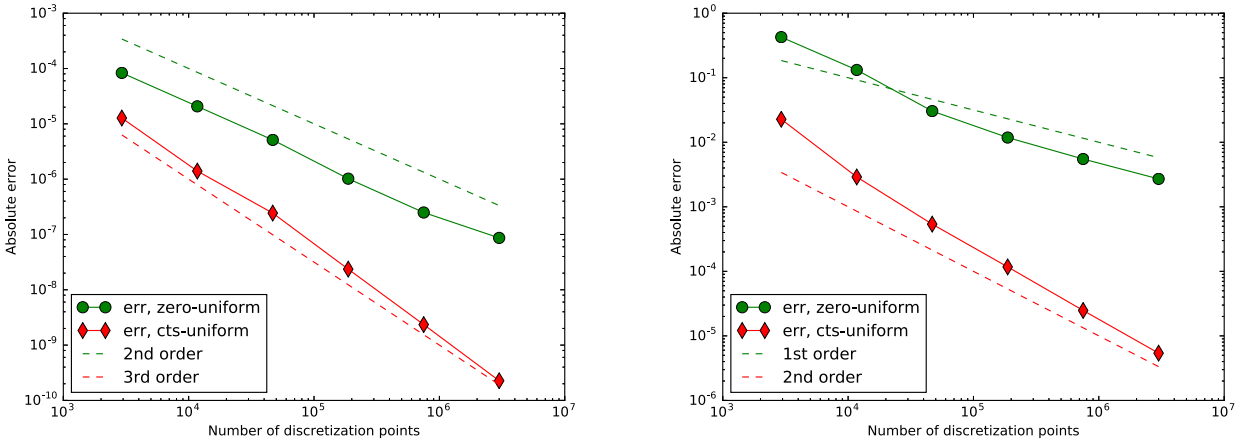

Fig. 5. Example 1, convergence rates on a uniform tree. Accuracy of the potential (left) and gradient (right) versus the number of discretization nodes N_Ω, using either extension-by-zero (green circles) or continuous extension (red diamonds).

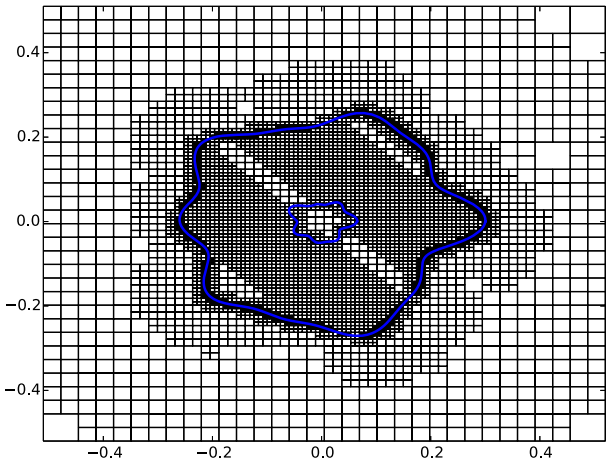

Fig. 6. Example 1. An example adaptive tree for the continuously extended f_e.

irregularity of f_e across the boundary, as shown in Fig. 6. We only present results corresponding to continuous extension here, as our refinement rule did not work well with zero extension, and is not relevant in the case of the smooth f_e since there is little difference between adaptive and uniform discretization in that case.

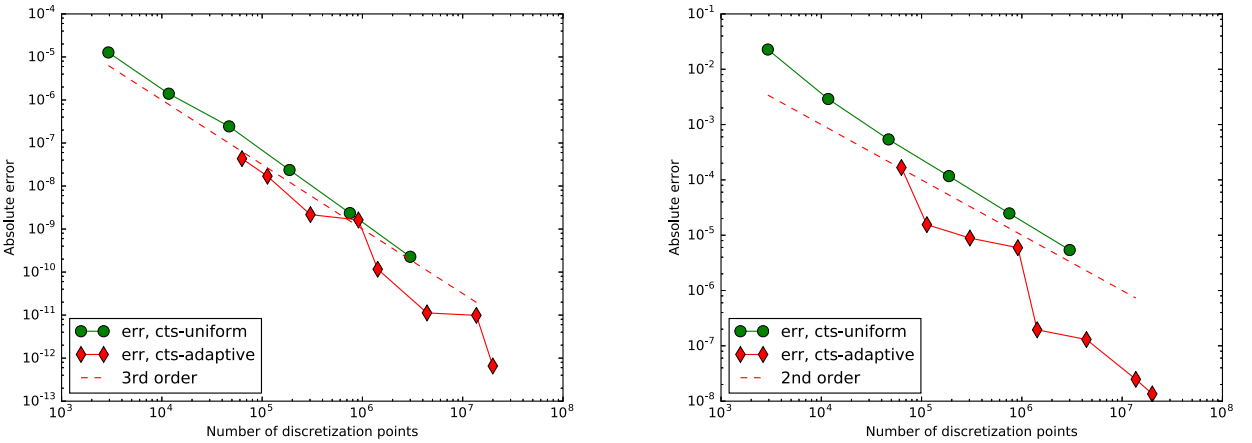

Fig. 7. Example 1. Using continuous extension, a plot of the error in the potential (left) and error in the gradient (right) versus the number of discretization nodes N_Ω, for both uniform (green circles) and adaptive trees (red diamonds).

Table 1
Box code timing information for Example 1 with continuous function extension and a uniform tree.

N_Ω	N_V	t_V	N_Ω/t_V	N_V/t_V
2.9290e+03	1.6384e+04	3.0193e−02	9.7009e+04	5.4264e+05
1.1717e+04	6.5536e+04	4.4461e−02	2.6353e+05	1.4740e+06
4.6846e+04	2.6214e+05	1.2574e−01	3.7256e+05	2.0848e+06
1.8739e+05	1.0486e+06	3.9343e−01	4.7629e+05	2.6652e+06
7.4955e+05	4.1943e+06	1.4926e+00	5.0218e+05	2.8101e+06
2.9983e+06	1.6777e+07	7.3144e+00	4.0991e+05	2.2937e+06

Table 2
Box code timing information for Example 1 with continuous function extension and an adaptive tree.

N_Ω	N_V	t_V	N_Ω/t_V	N_V/t_V
6.2928e+04	1.1685e+05	7.5608e−02	8.3229e+05	1.5454e+06
1.1291e+05	2.3666e+05	1.2941e−01	8.7251e+05	1.8287e+06
3.0310e+05	5.6781e+05	2.7364e−01	1.1077e+06	2.0750e+06
9.1144e+05	1.5124e+06	6.4442e−01	1.4144e+06	2.3468e+06
1.4207e+06	2.7318e+06	1.1932e+00	1.1906e+06	2.2895e+06
4.4043e+06	7.3749e+06	3.0303e+00	1.4534e+06	2.4337e+06

In Fig. 7, we see modest improvement in the accuracy of the potential and significant improvement in the accuracy of the gradient using adaptive discretization. We note that for the tests with adaptive grids much larger values of N_Ω could be achieved. This is because the memory consumption of the volume integral code depends on N_V, the total number of nodes in the box Ω_B. The uniform tree rather inefficiently places many points outside of Ω, whereas the adaptive tree places relatively few points because the extended function is quite smooth outside of Ω, where it is harmonic.

We conclude this section on Example 1 by analyzing the run time performance of the box code and of the evaluation of layer potentials. Figures for the box code are given in Tables 1 and 2, and figures for the evaluation of layer potentials are given in Tables 3 and 4. t_V denotes the time for the box code, t_{QP} denotes the time for QBX precomputation (forming the expansions for $\mathcal{O}(1)$ access to the field, as described above), and t_{QE} denotes the time for QBX evaluations at each node in the domain. Each of these times includes the time required to evaluate both the potential and the gradient. The performance is only reported for continuous extension; the results for extension by zero and smooth extension are similar.

There are a few points to highlight from Tables 1 and 2. We see that N_V/t_V is roughly constant for large N_V, indicating that the FMM indeed scales linearly in terms of the total number of FMM nodes. One of the strengths of a box code is that this ratio is similar for uniform and adaptive trees. Further, the throughput is quite good, at about 2.5 million points per second. We include the ratio N_Ω/t_V because the number of grid points inside the domain seems to be the more natural figure of merit. For a uniform tree (Table 1), we have that N_Ω is a fixed fraction of N_V, so that N_Ω/t_V is some fraction of N_V/T_V; here it is typically around 470 thousand points per second. In the adaptive case (Table 2), the nodes can be placed more intelligently inside the domain and we see that the throughput — in terms of N_Ω/t_V — is better than in the uniform case.

Table 3
QBX timing information for Example 1 with continuous function extension and a uniform tree.

N_Ω	t_{QP}	t_{QE}	$N_\Omega/(t_{QP}+t_{QE})$	N_Ω/t_{QE}
2.9290e+03	1.1848e+00	1.3120e−03	2.4694e+03	2.2325e+06
1.1717e+04	1.1697e+00	3.2675e−03	9.9892e+03	3.5859e+06
4.6846e+04	1.1825e+00	1.2799e−02	3.9192e+04	3.6601e+06
1.8739e+05	1.2034e+00	5.4077e−02	1.4902e+05	3.4652e+06
7.4955e+05	1.1677e+00	1.8898e−01	5.5249e+05	3.9663e+06
2.9983e+06	1.1896e+00	7.4644e−01	1.5487e+06	4.0168e+06

Table 4
QBX timing information for Example 1 with continuous function extension and an adaptive tree.

N_Ω	t_{QP}	t_{QE}	$N_\Omega/(t_{QP}+t_{QE})$	N_Ω/t_{QE}
6.2928e+04	1.1912e+00	1.7972e−02	5.2042e+04	3.5014e+06
1.1291e+05	1.1744e+00	3.1084e−02	9.3665e+04	3.6325e+06
3.0310e+05	1.1708e+00	7.9357e−02	2.4245e+05	3.8195e+06
9.1144e+05	1.1733e+00	2.3131e−01	6.4889e+05	3.9404e+06
1.4207e+06	1.1770e+00	3.6280e−01	9.2264e+05	3.9159e+06
4.4043e+06	1.1733e+00	1.1130e+00	1.9264e+06	3.9572e+06

Tables 3 and 4 show that the run time for QBX is similar for volume nodes arranged in uniform or adaptive trees, as one might expect. If one only considers the cost of the evaluations, we see that the throughput, N_Ω/t_{QE}, is roughly constant at about 3.9 million points per second. The precomputation time, t_{QP}, depends only on the number of boundary nodes M and is large relative to t_{QE} until N_Ω is of the order of a few millions. When this precomputation time is included, the throughput, $N_\Omega/(t_{QP}+t_{QE})$, is still quite high, on the same order as the box code for large N_Ω. Of course, for a boundary with many discretization nodes M, one expects this to no longer be the case.

6.3. Example 2

For Example 2, we choose an exact solution u with a sharp ridge along the x_2 axis, given by

$$u(\mathbf{x}) = \sin(10(x_1+x_2)) + x_1^2 - 3x_2 + 8 + e^{-500x_1^2}. \tag{56}$$

As before, we obtain a closed form formula for f by calculating the Laplacian of u. Observe that f has very sharp variations. This example was chosen on purpose to specifically illustrate and analyze the value of adaptive mesh refinement. As in Example 1, g is computed with arbitrary accuracy by evaluating u on $\partial\Omega$. The function g also has sharp variations, and so does the volume integral. In order to better resolve the boundary data we thus use $M = 14{,}208$ boundary nodes in this example, as opposed to $M = 9{,}280$ in Example 1.

First, consider the question of super-convergence for a smooth extension f_e. Let version 1 and version 2 of the boundary data be defined as in Example 1. We perform a convergence test on uniform trees for both versions 1 and 2. As before, version 1 should display fourth order convergence for the potential and sub-fourth order convergence for the gradient, while version 2 should display super-convergence. This is precisely what we see in Fig. 8. For each version, the initial convergence order is slow, likely a result of the irregularity of f. It is unclear what the eventual convergence order of the gradient is for version 1 but it is fourth order for version 2. As in Example 1, the accuracy of version 2 is much better.

Next, we consider the question of the convergence order using extension-by-zero and continuous extension with a layer potential. In Fig. 9, we plot the error for the potential and gradient for increasing N_Ω on uniform trees with both extension-by-zero and continuous extension. For this example, the two methods have similar error until N_Ω is large because the irregularity in the solution is unresolved by the grid for small N_Ω. Once N_Ω is sufficiently large, we see that the convergence rate for continuous extension is faster, though the specific rates are not as clear as they were for Example 1.

Fig. 9 also demonstrates that a uniform grid does a poor job of giving high accuracy for the gradient. We now test the effect of adaptive mesh refinement as in Example 1. Fig. 10 shows a representative adaptive tree for Example 2. The *a priori* refinement strategy places many boxes near the irregularity in f_e. Because the continuous extension is smooth outside of Ω, the effect of the "ridge" on the x_2 axis does not extend far outside of the domain.

As in Example 1, the adaptive discretization strategy provides modest improvement in the accuracy of the potential (and eventually no improvement at all). An explanation for this is that the solution u is much smoother than f, so we greatly over-resolve u when we construct the tree with the goal of resolving f. In other words, the *a priori* refinement strategy is eventually less efficient than the uniform strategy in terms of the accuracy of the potential. In contrast, adaptive discretization provides significant gains in the accuracy of the gradient. This is because the gradient is less smooth and more difficult to resolve than u so that the additional boxes used to resolve f are not as wasteful. Note also that once again, for the tests with adaptive grids much larger values of N_Ω could be achieved, for the same reasons as in Example 1 (Fig. 11).

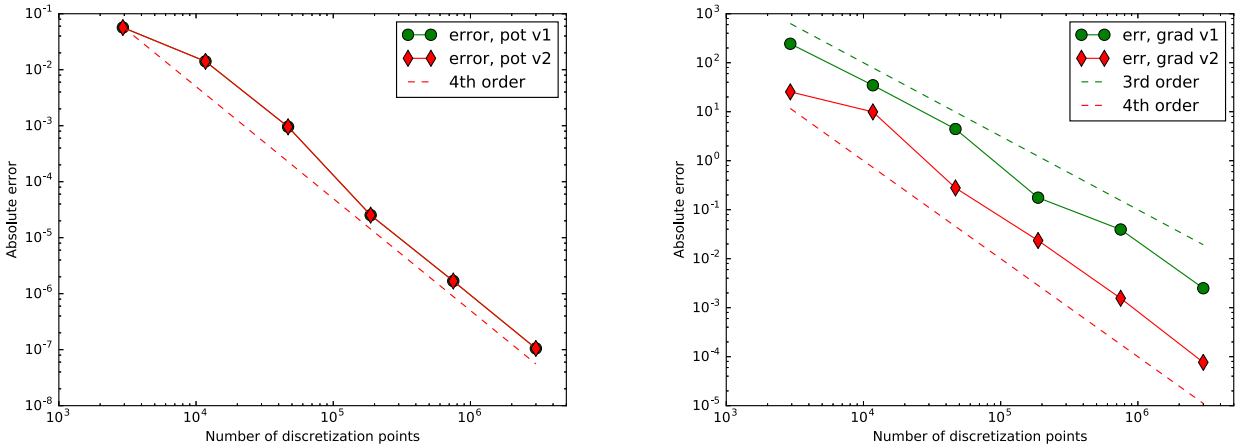

Fig. 8. Example 2, smooth extension. Accuracy of versions 1 (green circles) and 2 (red diamonds) for the potential (left) and gradient (right).

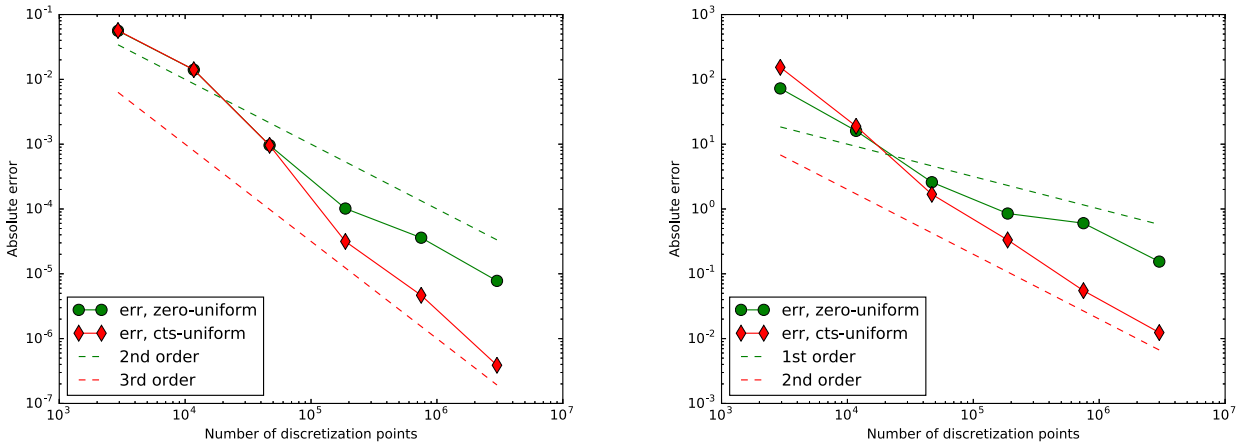

Fig. 9. Example 2, convergence rates on a uniform tree. Accuracy of the potential (left) and gradient (right) versus the number of discretization nodes N_Ω, using either extension-by-zero (green circles) or continuous extension (red diamonds).

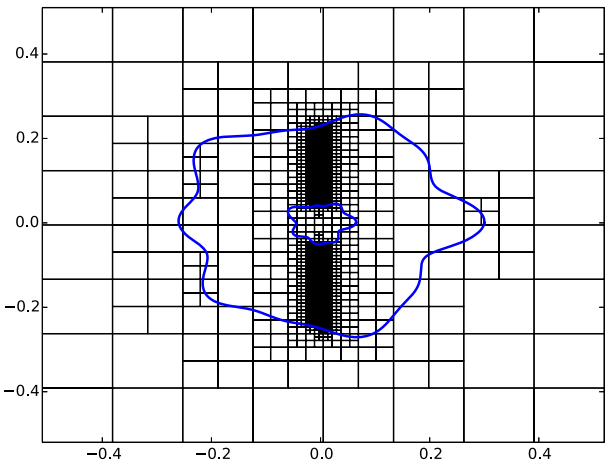

Fig. 10. Example 2. An example adaptive tree for the continuously extended f_e.

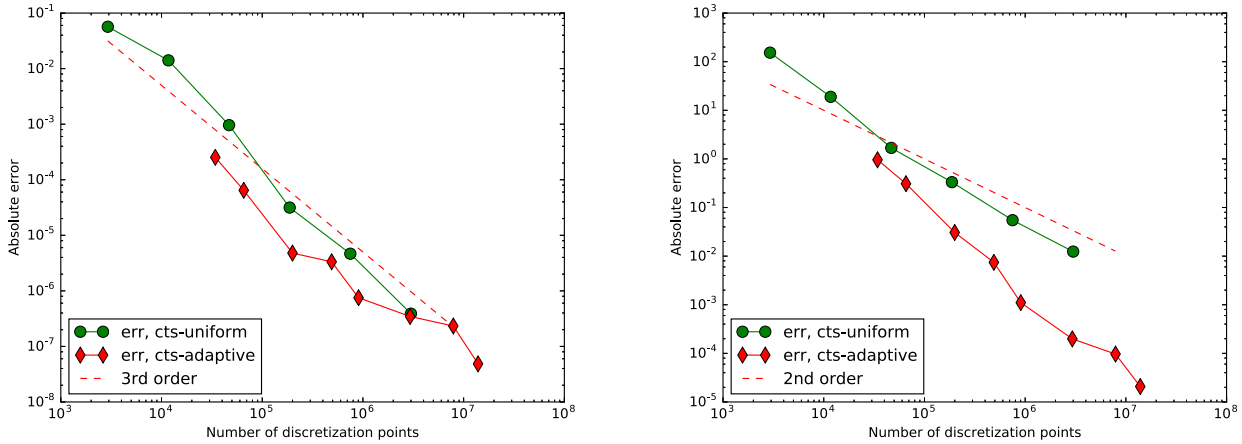

Fig. 11. Example 2. Using continuous extension, a plot of the error in the potential (left) and error in the gradient (right) versus the number of discretization nodes N_Ω, for both uniform (green circles) and adaptive trees (red diamonds).

Table 5
Box code timing information for Example 2 with continuous function extension and a uniform tree.

N_Ω	N_V	t_V	N_Ω/t_V	N_V/t_V
2.9290e+03	1.6384e+04	1.0521e−01	2.7840e+04	1.5573e+05
1.1717e+04	6.5536e+04	4.5237e−02	2.5901e+05	1.4487e+06
4.6846e+04	2.6214e+05	1.1961e−01	3.9166e+05	2.1917e+06
1.8739e+05	1.0486e+06	3.9992e−01	4.6856e+05	2.6220e+06
7.4955e+05	4.1943e+06	1.4935e+00	5.0188e+05	2.8084e+06
2.9983e+06	1.6777e+07	7.0171e+00	4.2728e+05	2.3909e+06

Table 6
Box code timing information for Example 2 with continuous function extension and an adaptive tree.

N_Ω	N_V	t_V	N_Ω/t_V	N_V/t_V
3.4204e+04	3.8032e+04	4.5255e−02	7.5581e+05	8.4039e+05
6.5547e+04	7.0480e+04	5.4312e−02	1.2069e+06	1.2977e+06
1.9972e+05	2.0987e+05	1.1204e−01	1.7826e+06	1.8732e+06
4.8924e+05	5.1256e+05	2.3064e−01	2.1212e+06	2.2223e+06
9.0490e+05	9.6006e+05	4.0450e−01	2.2371e+06	2.3735e+06
2.9398e+06	3.0676e+06	1.1827e+00	2.4857e+06	2.5937e+06

Table 7
QBX timing information for Example 2 with continuous function extension and a uniform tree.

N_Ω	t_{QP}	t_{QE}	$N_\Omega/(t_{QP}+t_{QE})$	N_Ω/t_{QE}
2.9290e+03	2.5967e+00	1.2099e−03	1.1274e+03	2.4209e+06
1.1717e+04	1.8191e+00	3.3716e−03	6.4292e+03	3.4752e+06
4.6846e+04	1.7931e+00	1.2859e−02	2.5940e+04	3.6431e+06
1.8739e+05	1.7918e+00	4.8408e−02	1.0183e+05	3.8710e+06
7.4955e+05	1.7931e+00	2.0054e−01	3.7597e+05	3.7377e+06
2.9983e+06	1.8273e+00	7.3797e−01	1.1688e+06	4.0629e+06

Finally, we present run time performance results as we did for Example 1 (Tables 5–8). The conclusions here are the same as the ones for Example 1. Observe in particular that the performance of the box code is nearly the same here as it was for Example 1, even though the trees used in this example are highly adaptive. This is one of the major advantages of the numerical method we present in this article.

7. Conclusion

We have demonstrated that continuous global function extension constructed as the solution of an exterior Laplace problem provided an effective framework to apply adaptive FMM based Poisson solvers to problems with complex geometries. We found that the desirable properties of the FMM are kept intact with such a method: the amount of work still scales

Table 8
QBX timing information for Example 2 with continuous function extension and an adaptive tree.

N_Ω	t_{QP}	t_{QE}	$N_\Omega/(t_{QP}+t_{QE})$	N_Ω/t_{QE}
3.4204e+04	2.1972e+00	1.1984e−02	1.5483e+04	2.8541e+06
6.5547e+04	2.1816e+00	2.5388e−02	2.9700e+04	2.5818e+06
1.9972e+05	2.1215e+00	5.0410e−02	9.1958e+04	3.9620e+06
4.8924e+05	2.0789e+00	1.2520e−01	2.2197e+05	3.9077e+06
9.0490e+05	1.8968e+00	2.2485e−01	4.2651e+05	4.0245e+06
2.9398e+06	2.1047e+00	7.7307e−01	1.0216e+06	3.8028e+06

linearly with the number of degrees of freedom in the computational domain and is competitive with classical FFT-based solvers in terms of work per grid point, despite the flexibility of adaptive mesh refinement. This holds even for multiply connected domains with irregular boundaries. The adaptive refinement capability of our new solver plays a crucial role in guaranteeing an efficient use of the degrees of freedom in the system, and in obtaining high accuracy for the gradient of the potential. Finally, for the particular situations in which a smooth global extension is readily available without resorting to numerical computation, as is for example the case of an extension by zero in plasma physics applications [29], we have presented a numerical method which leads to the same order of convergence for the gradient of the potential as the potential itself. In our implementation of the FMM, this translates to 4th order convergence for both the potential and the gradient, and the order of convergence can be increased by choosing higher order basis functions [14].

Of course, when continuous extension is employed, the *convergence order* of the method is not particularly high. We demonstrated above that adaptive refinement can help improve the accuracy per degree of freedom in this case, particularly for the gradient, but the low order of accuracy is really a result of compromise. The method of this paper emphasizes ease of use, domain flexibility, speed, and compatibility with adaptive refinement strategies. To achieve these goals we have chosen an embedded boundary method (for ease of use and domain flexibility) built on a box code (for speed and handling highly adaptive grids). Because it is an embedded boundary method, high order accuracy is more difficult to achieve. However, the method asks for very little from the user. Only a parametric description of the boundary and a method for evaluating f accurately in the domain must be provided. In particular, no special quadrature rules are required, as is the case for a boundary fitted mesh, and there are no requirements on the accuracy of derivatives of the user-provided f. As noted in Section 4.2, when accurate derivatives of f are available, an extension computed as the solution of a polyharmonic equation would result in a higher order method.

The capabilities of our solver can be extended in a number of ways. First, C^1 function extension provided by the solution of an exterior biharmonic problem would lead to faster convergence for the solution and gradient than we have obtained with C^0 extension, provided that accurate values for the gradient of f are available on the boundary. Second, one could allow for boundaries with corners and which nearly self-intersect. Numerical tools addressing these two challenges have recently been developed, but have not yet been implemented in the Poisson context. Fortunately, the overall method is largely agnostic as to how the function extension and harmonic correction are computed, so that new methods may be swapped in when they become available. Finally, much of the technology and analysis required for this work extends to three dimensions in a straightforward manner. This is the subject of ongoing work, with progress to be reported at a later date.

Acknowledgements

The authors would like to thank Prof. Leslie Greengard (NYU) and Dr. Manas Rachh (Yale) for many insightful conversations, and Dr. Zydrunas Gimbutas (NIST) for helping with the generation of tables. A.J.C. would like to thank the Institut de Recherche Mathématique Avancée (IRMA) at the University of Strasbourg for their hospitality and for providing an office in the summer 2016, during which much of this article was written. T.A. was partially supported by the U.S. Department of Energy under contract DEFG0288ER25053, by the Air Force Office of Scientific Research under NSSEFF Program Award FA9550-10-1-0180 and FA9550-15-1-0385, and by a GSAS Dissertation Fellowship from NYU. A.J.C. was supported by the U.S. Department of Energy, Office of Science, Fusion Energy Sciences under Award Nos. DE-FG02-86ER53223 and DE-SC0012398.

References

[1] B.K. Alpert, Hybrid Gauss-trapezoidal quadrature rules, SIAM J. Sci. Comput. 20 (5) (1999) 1551–1584, http://dx.doi.org/10.1137/S1064827597325141.
[2] C.R. Anderson, An implementation of the fast multipole method without multipoles, SIAM J. Sci. Stat. Comput. 13 (4) (1992) 923–947.
[3] T. Askham, Integral-Equation Methods for Inhomogeneous Elliptic Partial Differential Equations in Complex Geometry, PhD thesis, New York University, 2016.
[4] K.E. Atkinson, The Numerical Solution of Integral Equations of the Second Kind, Cambridge University Press, 1997, Cambridge Books Online.
[5] A.H. Barnett, Evaluation of layer potentials close to the boundary for Laplace and Helmholtz problems on analytic planar domains, SIAM J. Sci. Comput. 36 (2) (2014) A427–A451, http://dx.doi.org/10.1137/120900253.
[6] A.H. Barnett, B. Wu, S. Veerapaneni, A fast direct solver for elliptic partial differential equations on adaptively refined meshes, SIAM J. Sci. Comput. 37 (4) (2015) B519–B542, http://dx.doi.org/10.1137/140990826.

[7] J. Bremer, Z. Gimbutas, V. Rokhlin, A nonlinear optimization procedure for generalized Gaussian quadratures, SIAM J. Sci. Comput. 32 (4) (2010) 1761–1788.
[8] J. Bremer, V. Rokhlin, I. Sammis, Universal quadratures for boundary integral equations on two-dimensional domains with corners, J. Comput. Phys. 229 (22) (2010) 8259–8280.
[9] B.L. Buzbee, G.H. Golub, C.W. Nielson, On direct methods for solving Poisson's equations, SIAM J. Numer. Anal. 7 (4) (1970) 627–656.
[10] J. Carrier, L. Greengard, V. Rokhlin, A fast adaptive multipole algorithm for particle simulations, SIAM J. Sci. Stat. Comput. 9 (4) (1988) 669–686, http://dx.doi.org/10.1137/0909044.
[11] S. Celestin, Z. Bonaventura, B. Zeghondy, A. Bourdon, P. Ségur, The use of the ghost fluid method for Poisson's equation to simulate streamer propagation in point-to-plane and point-to-point geometries, J. Phys. D, Appl. Phys. 42 (2009) 065203, http://dx.doi.org/10.1088/0022-3727/42/6/065203.
[12] S.S. Dragomir, Approximating real functions which possess *n*th derivatives of bounded variation and applications, Comput. Math. Appl. 56 (9) (2008) 2268–2278.
[13] C.L. Epstein, L. Greengard, A. Klöckner, On the convergence of local expansions of layer potentials, SIAM J. Numer. Anal. 51 (5) (2013) 2660–2679.
[14] F. Ethridge, L. Greengard, A new fast-multipole accelerated Poisson solver in two dimensions, SIAM J. Sci. Comput. 23 (3) (2001) 741–760, http://dx.doi.org/10.1137/S1064827500369967.
[15] P. Farkas, Mathematical Foundations for Fast Methods for the Biharmonic Equation, PhD thesis, The University of Chicago, ProQuest LLC, Ann Arbor, MI, 1989.
[16] A.C. Genz, A.A. Malik, Remarks on algorithm 006: an adaptive algorithm for numerical integration over an *n*-dimensional rectangular region, J. Comput. Appl. Math. 6 (4) (1980) 295–302.
[17] L. Greengard, The Rapid Evaluation of Potential Fields in Particle Systems, MIT Press, 1988.
[18] L. Greengard, J-Y. Lee, A direct adaptive Poisson solver of arbitrary order accuracy, J. Comput. Phys. 125 (1996) 415–424, http://dx.doi.org/10.1006/jcph.1996.0103.
[19] R.B. Guenther, J.W. Lee, Partial Differential Equations of Mathematical Physics and Integral Equations, Prentice Hall, Englewood Cliffs, NJ, 1988.
[20] J. Helsing, E. Wadbro, Laplace's equation and the Dirichlet–Neumann map: a new mode for Mikhlin's method, J. Comput. Phys. 202 (2) (2005) 391–410.
[21] J. Helsing, R. Ojala, On the evaluation of layer potentials close to their sources, J. Comput. Phys. 227 (5) (2008) 2899–2921, http://dx.doi.org/10.1016/j.jcp.2007.11.024.
[22] J. Helsing, R. Ojala, Corner singularities for elliptic problems: integral equations, graded meshes, quadrature, and compressed inverse preconditioning, J. Comput. Phys. 227 (20) (2008) 8820–8840.
[23] K.L. Ho, L. Greengard, A fast direct solver for structured linear systems by recursive skeletonization, SIAM J. Sci. Comput. 34 (5) (2012) A2507–A2532.
[24] T. Hrycak, V. Rokhlin, An improved fast multipole algorithm for potential fields, SIAM J. Sci. Comput. 19 (6) (1998) 1804–1826, http://dx.doi.org/10.1137/S106482759630989X.
[25] S. Kapur, V. Rokhlin, High-order corrected trapezoidal quadrature rules for singular functions, SIAM J. Numer. Anal. 34 (4) (1997) 1331–1356.
[26] A. Klöckner, A. Barnett, L. Greengard, M. O'Neil, Quadrature by expansion: a new method for the evaluation of layer potentials, J. Comput. Phys. 252 (2013) 332–349, http://dx.doi.org/10.1016/j.jcp.2013.06.027.
[27] M.H. Langston, L. Greengard, D. Zorin, A free-space adaptive FMM-based PDE solver in three dimensions, Commun. Appl. Math. Comput. Sci. 6 (2011) 79–122, http://dx.doi.org/10.2140/camcos.2011.6.79.
[28] M.H. Langston, An Adaptive Fast Multipole Method-Based PDE Solver in Three Dimensions, PhD thesis, New York University, 2012.
[29] J.P. Lee, A.J. Cerfon, ECOM: a fast and accurate solver for toroidal axisymmetric MHD equilibria, Comput. Phys. Commun. 190 (2015) 72–88, http://dx.doi.org/10.1016/j.cpc.2015.01.015.
[30] A. Mayo, The fast solution of Poisson's and the biharmonic equations on irregular regions, SIAM J. Numer. Anal. 21 (2) (1984) 285–299, http://dx.doi.org/10.1137/0721021.
[31] A. Mayo, A. Greenbaum, Fast parallel iterative solution of Poisson's and the biharmonic equations on irregular regions, SIAM J. Sci. Stat. Comput. 13 (1) (1992) 101–118, http://dx.doi.org/10.1137/0913006.
[32] A. McKenney, L. Greengard, A. Mayo, A fast Poisson solver for complex geometries, J. Comput. Phys. 118 (1995) 348–355, http://dx.doi.org/10.1006/jcph.1995.1104.
[33] R. Ojala, A robust and accurate solver of Laplace's equation with general boundary conditions on general domains in the plane, J. Comput. Math. 30 (4) (2012) 433–448.
[34] A. Greenbaum, L. Greengard, G.B. McFadden, Laplace's equation and the Dirichlet–Neumann map in multiply connected domains, J. Comput. Phys. 105 (2) (1993) 267–278.
[35] A. Pataki, A.J. Cerfon, J.P. Freidberg, L. Greengard, M. O'Neil, A fast, high-order solver for the Grad–Shafranov equation, J. Comput. Phys. 243 (2013) 28–45, http://dx.doi.org/10.1016/j.jcp.2013.02.045.
[36] S. Popinet, Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries, J. Comput. Phys. 190 (2003) 572–600, http://dx.doi.org/10.1088/0022-3727/42/6/065203.
[37] M. Rachh, Integral Equation Methods for Problems in Electrostatics, Elastostatics and Viscous Flow, PhD thesis, New York University, 2015.
[38] M. Rachh, A. Klöckner, M. O'Neil, Fast algorithms for quadrature by expansion I: globally valid expansions, arXiv preprint arXiv:1602.05301, 2016.
[39] M. Rachh, personal communication.
[40] L.F. Ricketson, A.J. Cerfon, M. Rachh, J.P. Freidberg, Accurate derivative evaluation for any Grad–Shafranov solver, J. Comput. Phys. 305 (2016) 744, http://dx.doi.org/10.1016/j.jcp.2015.11.015.
[41] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (3) (1986) 856–869.
[42] K. Serkh, V. Rokhlin, On the solution of elliptic partial differential equations on regions with corners, J. Comput. Phys. 305 (2016) 150–171.
[43] D.B. Stein, R.D. Guy, B. Thomases, Immersed boundary smooth extension: a high-order method for solving PDE on arbitrary smooth domains using Fourier spectral methods, J. Comput. Phys. 304 (2016) 252–274, http://dx.doi.org/10.1016/j.jcp.2015.10.023.
[44] H.R. Strauss, Nonlinear, three-dimensional magnetohydrodynamics of noncircular tokamaks, Phys. Fluids 19 (1976) 134, http://dx.doi.org/10.1063/1.861310.
[45] User's Manual FMM Toolbox (2D) for MATLAB Version 1.1, MadMax Optics Inc., Hamden, CT, 2002.
[46] F. Vico, L. Greengard, M. Ferrando, Fast convolution with free-space green's functions, J. Comput. Phys. 323 (2016) 191–203.