Journal of Computational Physics 361 (2018) 424–441

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A source term method for Poisson problems on irregular domains

John D. Towers

MiraCosta College, 3333 Manchester Avenue, Cardiff-by-the-Sea, CA 92007-1516, USA

ARTICLE INFO

Article history:
Received 27 September 2017
Received in revised form 21 January 2018
Accepted 22 January 2018

Keywords:
Poisson problem
Irregular domain
Level set methods
Finite difference
Heaviside function
Delta function

ABSTRACT

This paper presents a finite difference method for solving Poisson problems on a two-dimensional irregular domain. An implicit, level set representation of the domain boundary is assumed, as well as a Cartesian grid that is not fitted to the domain. The algorithm is based on a scheme for interface problems which captures the jump conditions via singular source terms. This paper adapts that method to deal with boundary value problems by employing a simple iterative process that simultaneously enforces the boundary condition and solves for an unknown jump condition. The benefit and novelty of this method is that the boundary condition is captured via easily implemented source terms. The system of equations that results at each iteration can be solved using a FFT-based fast Poisson solver. The scheme can accommodate Dirichlet, Neumann, and Robin boundary conditions. We first address the constant coefficient Poisson equation, and then extend the scheme to accommodate the variable coefficient equation. Numerical examples indicate second order accuracy (or close to it) for the solution. The method also produces useful gradient approximations, but with generally lower convergence rates.

1. Introduction

This paper presents a finite difference method for solving Poisson problems on a two-dimensional irregular domain. These problems arise in several applications including viscous incompressible flow [7,23,33] and shape identification [14,22]. A popular method of describing these domains, and for evolving their motion is the level set method [30,31,34]. With the level set method, the boundary is represented implicitly. This allows for complicated changes of topology to be represented in a natural way. In this setting it is advantageous to use a grid that is not fitted to the domain boundary, avoiding the computational cost of frequently re-fitting a conforming mesh as the geometry changes.

Numerical methods for Poisson problems on irregular domains have been studied for a long time, going back at least to the method of capacitance matrices in the early seventies [9]. More recently reference [27] proposed a method which assumes a Cartesian grid, can be used with fast Poisson solvers on a rectangular region, and is second order accurate. The solution is extended to the embedding rectangular region by solving a Fredholm integral equation. Reference [4] proposes a similar method for Laplace's equation with a Dirichlet boundary condition, and takes advantage of an accurate method for computing nearly singular integrals. Reference [28] uses similar ideas, combined with the fast multipole method. All of these methods assume an explicit, parameterized representation of the domain boundary.

E-mail address: john.towers@cox.net.

https://doi.org/10.1016/j.jcp.2018.01.038
0021-9991/

The Immersed Interface Method (IIM) [19–22,24] is a versatile technique that has been used to solve a wide array of problems, including the problem discussed here. A closely related method is the Explicit Jump Immersed Interface Method (EJIM) [40]. The IIM works with a Cartesian grid, and can be used with either an explicit parameterization, or an implicit level set description of the boundary. With this method, the standard finite difference scheme is modified near the boundary in order to achieve second or fourth order accuracy. The augmented IIM is the version of the IIM that is used to solve boundary value problems of the type considered in this paper. With this method augmented variables are introduced by orthogonally projecting grid points adjacent to the boundary onto the boundary itself, and an enlarged linear system of equations results. By projecting grid points from only one side of the boundary, the projected points are not clustered, and the potential problem of ill-conditioning is avoided. The resulting Schur complement system can be solved using an iterative method such as the Generalized Minimum Residual (GMRES) method. With this approach, it is possible to use a fast Poisson solver at each iteration, which is a benefit of the augmented variable technique. A least squares interpolation scheme is used to approximate the normal derivative of the solution on each side of the interface.

Reference [15] presents a finite difference method for the Dirichlet version of the problem considered here. The authors use a regular Cartesian grid, and by using conservative differencing of the second order fluxes, they achieve second order accuracy. They are able to handle variable coefficients, and their method can be used with adaptive mesh refinement. Their method requires an explicit representation of the boundary.

Reference [18] proposes a boundary integral method for the type of problem considered here. An integral equation is solved for a double layer density (for a Dirichlet boundary condition) or single layer density (for a Neumann boundary condition), and then the solution u is found by using the appropriate double or single layer integral representation. All of this is done within the level set framework, using an implicit representation of the boundary. What makes this possible is a novel method for computing boundary integrals in a level set framework. An advantage of this method is that because the solution is recovered via the integral representation, a uniform grid is not required, and so the method can be used with narrow banding, local level set, or adaptive grid methods. Also, this method handles the three types of boundary conditions addressed by our new algorithm, as well as mixed boundary conditions, where Dirichlet and Neumann boundary conditions are imposed on different portions of a single component of the boundary.

Reference [6] addresses the problems considered here as a special case of a more general interface problem. The authors use a Cartesian grid, and can handle both explicit and implicit representations of the boundary. Away from the boundary, the standard 5-point stencil is used, but near the boundary so-called virtual nodes are introduced. Their method is second order accurate.

The Ghost Fluid Method (GFM) was adapted to the Dirichlet version of the irregular domain problem in a series of papers [25,11,10,12]. The finite difference methods proposed in those papers assume a Cartesian grid, and can use a level set function to locate the boundary of the domain. Reference [25] deals with an interface problem where the elliptic operator may have a variable, discontinuous coefficient. The method is first order accurate, but captures jumps sharply, results in a simple linear algebra problem, and works in any number of spatial dimensions, since the interface is dealt with in a dimension-by-dimension fashion. In [11] this method was adapted to Poisson problems with Dirichlet boundary conditions. Without the interface coupling conditions required in the interface problems of [25], the authors were able to achieve second order accuracy. In [10], the authors devised a fourth order version of their scheme. The linear systems resulting from these methods are symmetric, making it possible to use relatively fast methods such as the Preconditioned Conjugate Gradient (PCG) method. Reference [12] incorporates an adaptive mesh refinement scheme into the GFM for these problems. A number of additional works have proposed methods closely related to, or derived from the GFM [16,26]. Papers that deal specifically with the Robin problem are [17] and [32].

Reference [13] proposes a method that starts with data specified on a uniform cartesian grid (or an adaptive quad-tree or oct-tree grid) but locally creates a Voronoi mesh near the boundary. It uses a fully implicit level set representation of the boundary and achieves second order accuracy in both two and three dimensions.

Reference [35] presents a method that extends the solution smoothly outside of the irregular domain onto a larger computational domain, and then uses Fourier spectral methods to produce an accurate solution, up to fourth order for Dirichlet problems and third order for Neumann problems. Reference [1] also achieves high accuracy using a solution extension method, along with an integral equation formulation and the fast multipole method. Both [35] and [1] employ an explicit representation of the boundary of the irregular domain.

In this paper we use a domain embedding method, along with an iterative approach that simultaneously determines an unknown jump condition and enforces the boundary condition of the original Poisson problem. Starting with an initial guess (of zero) for the undetermined portion of the jump data, we solve a discretization of the embedded Poisson problem. Next that solution (or its partial derivatives) is extrapolated from the interior of the domain onto a narrow strip containing the boundary to produce a new approximation of the unknown jump data, and then the process is repeated. At each iteration, the discrete problem is solved using a fast Poisson solver, and the process yields a fairly accurate solution after a reasonable number of iterations, see Fig. 1. The discretization of the embedded Poisson problem that is the basis for this process was shown in [39] to be second order accurate, and that accuracy seems to carry over to the new algorithm for the original Poisson problem with irregular boundary.

The “unknown jump data” concept underlying our method is also employed by the IIM, but with two important differences. First, although the unknown jump data can be viewed as an augmented variable, unlike the augmented IIM we do not

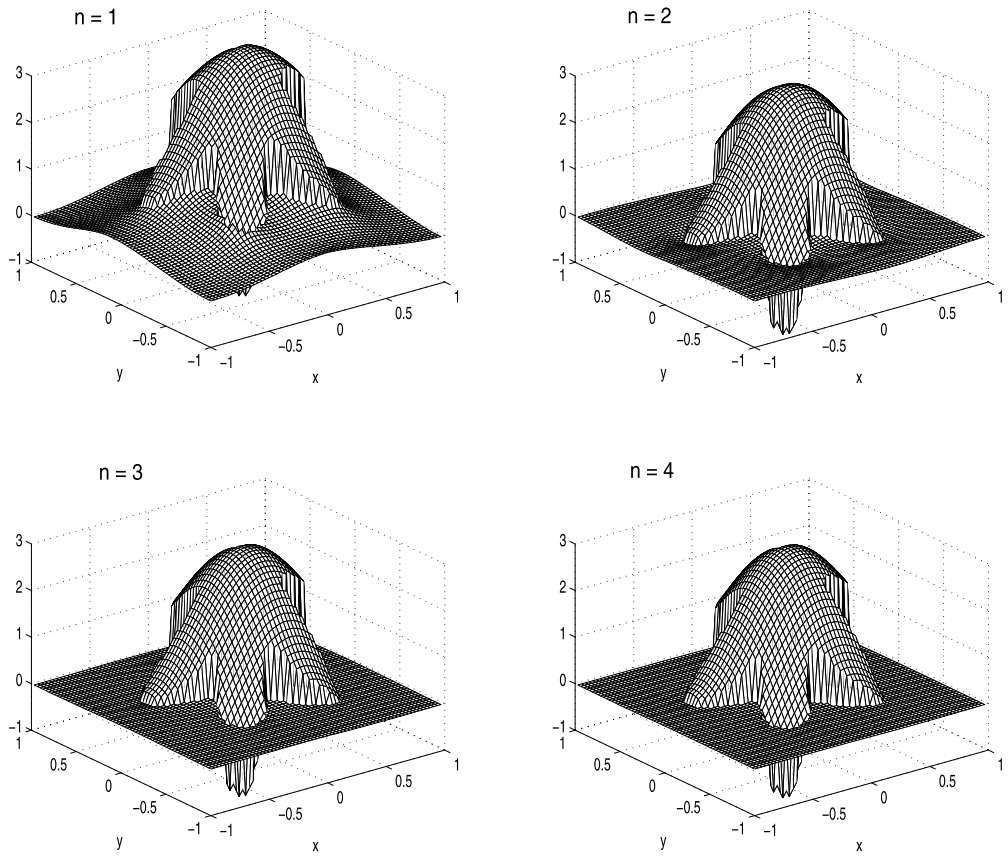

Fig. 1. The approximate solution to Example 3 of Section 4 after each of the first four iterations. The grid size is 64×64. Changes in the approximate solution cannot be detected visually after a small number of iterations. The algorithm executed a total of 19 iterations in this case. The computational effort of each iteration consists mainly of a call to a FFT-based fast Poisson solver.

formulate or solve an augmented linear system. Instead, after each iteration of the interface solver the undetermined jump variable is approximated using the extrapolation step mentioned in the previous paragraph. An advantage of this approach is that the only required linear algebra subroutine is a fast Poisson solver. The other main difference between our method and the IIM is our interface solver which employs singular source terms to capture the desired jump conditions across the interface.

As our admittedly incomplete review of the relevant literature indicates, there are already a number of effective methods for solving Poisson problems on irregular domains. We justify proposing this new algorithm based on its combination of relative simplicity and flexibility, along with the observation that unlike many existing methods the scheme does not require an explicitly defined boundary. Our approach employs the level set framework, meaning that the boundary is represented implicitly. We use a Cartesian grid that is not fitted to the domain boundary. The boundary conditions are captured via easily implemented singular source terms, which is the novel aspect and the benefit of the method. This is accomplished by incorporating a simple iterative process into the interface capturing algorithm of [39]. Since the discretization stencil is not modified near the interface, the matrix associated with the discrete Laplacian retains its symmetry, and moreover does not suffer ill-conditioning due to very small distances between stencil grid points and the boundary. Thus no preconditioning is required when solving the resulting linear system. In fact the system of equations that results at each iteration can be solved using a FFT-based fast Poisson solver. The scheme allows for variable coefficient problems and can accommodate Dirichlet, Neumann, and Robin boundary conditions. Both interior and exterior problems can be solved. All of the formulas and algorithms could be generalized in a straightforward manner to three dimensions. Given smooth data, numerical examples indicate second order accuracy (or close to it) for the solution. The method also produces useful gradient approximations, but with generally lower convergence rates.

The rest of the paper is organized as follows. Section 2 provides the details of the new algorithm for the constant coefficient problem. Section 3 describes the modifications that are required for the variable coefficient problem. Section 4 presents the results of a number of numerical experiments. Finally, Section 5 provides some concluding remarks.

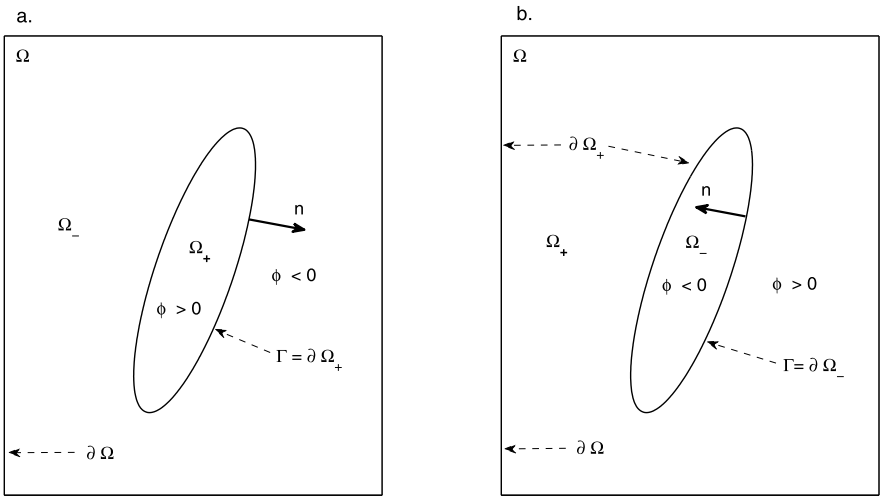

Fig. 2. Geometry of the Poisson problem. Interior problem (left panel) and exterior problem (right panel). The set Ω_+ where the Poisson problem is posed is embedded in the rectangular region Ω. The level set function $\phi(x, y)$ is used to identify $\Omega_\pm$ and to locate the boundary Γ where $\phi = 0$.

2. Discretization and the algorithms

The variable coefficient problem is addressed in Section 3, but for now we focus on the constant coefficient case, the interior version of which is

$$\begin{cases} \Delta u = f(x, y) & \text{in } \Omega_+, \\ B(u, \partial u/\partial n) = g(x, y) & \text{on } \Gamma. \end{cases} \tag{1}$$

For the interior problem Ω_+ is an open bounded domain in $\mathbb{R}^2$. Ω_+ and its boundary $\Gamma = \partial\Omega_+$ lie entirely inside a rectangular region Ω, which will serve as the computational domain. We consider both interior and exterior versions of the problem. In the exterior version, $\partial\Omega$ is part of the boundary of $\partial\Omega_+$, and $\partial\Omega_+ = \Gamma \cup \partial\Omega$. See Fig. 2. For the exterior version of the problem we impose a Dirichlet boundary condition $u = \gamma(x, y)$ on $\partial\Omega$ in addition to the boundary condition shown in (1). For the boundary condition $B(u, \partial u/\partial n) = g$ that appears in (1), we consider three variants:

$$u = g \text{ (Dirichlet)}, \ \partial u/\partial n = g \text{ (Neumann), and } \partial u/\partial n + \sigma u = g \text{ (Robin)}.$$

A level set function $\phi : \Omega \mapsto \mathbb{R}^1$ is assumed to be available to define Γ and the open sets $\Omega_\pm$ on either side of it:

$$\Gamma = \{(x, y) : \phi(x, y) = 0\},$$
$$\Omega_+ = \{(x, y) : \phi(x, y) > 0\} \cap \Omega,$$
$$\Omega_- = \{(x, y) : \phi(x, y) < 0\} \cap \Omega.$$

The interior boundary Γ consists of one or more simple closed curves lying entirely in Ω. The level set function $\phi : \Omega \mapsto \mathbb{R}^1$ is assumed to be smooth near Γ and have a nonvanishing gradient on Γ. The symbol $\partial/\partial n$ is used to denote differentiation in the direction of the normal vector to Γ that points outward from Ω_+. With this convention, $-(\nabla\phi/|\nabla\phi|) \cdot \nabla = \partial/\partial n$.

For a piecewise continuous function $z(x, y)$ that is defined in a neighborhood of Γ, we use the notation $[z] : \Gamma \mapsto \mathbb{R}$ to denote the jump in z along Γ:

$$[z(x, y)] = z^+(x, y) - z^-(x, y),$$

where

$$z^+(x, y) = \lim_{\substack{(\xi,\eta)\to(x,y) \\ (\xi,\eta)\in\Omega_+}} z(\xi, \eta), \quad z^-(x, y) = \lim_{\substack{(\xi,\eta)\to(x,y) \\ (\xi,\eta)\in\Omega_-}} z(\xi, \eta).$$

Closely related to the problem (1) is the Poisson interface problem:

$$\begin{cases} \Delta u = S(x, y) \text{ in } \Omega, \\ u = \rho(x, y) \text{ on } \partial\Omega, \\ \left[\frac{\partial u}{\partial n}\right] = a(x, y) \text{ on } \Gamma, \\ [u] = b(x, y) \text{ on } \Gamma, \end{cases} \qquad S(x, y) = \begin{cases} S^p(x, y), & (x, y) \in \Omega_+, \\ S^m(x, y), & (x, y) \in \Omega_-. \end{cases} \tag{2}$$

Reference [39] derived two source term formulations of (2), and then discretized them, resulting in two finite difference algorithms. This paper employs those two algorithms as the basis for a new algorithm for the Poisson problem (1). The original Poisson problem (1) is viewed within the framework of the Poisson interface problem (2), using a so-called domain embedding technique. We seek a solution u defined on all of the embedding rectangle Ω, but are ultimately only interested in the restriction of u to Ω_+, which by design will solve the original problem (1). The idea is to replace the boundary condition of the original problem (1) by jump conditions of the type appearing in the interface problem (2). In order to determine the jump conditions on Γ, we impose the requirement that the extended solution u should vanish on the set Ω_- as in e.g., [22] and [40]. We set $S^p(x,y) = f(x,y)$, $S^m(x,y) = 0$. For interior problems we take $\rho = 0$, and for exterior problems $\rho = \gamma = u|_{\partial\Omega}$.

This zero-extension approach implies the following jump data for (2):

$$\begin{aligned}
&\text{Dirichlet:} \begin{cases} a(x,y) = (\partial u/\partial n)^+ \text{ on } \Gamma, \\ b(x,y) = g(x,y) \text{ on } \Gamma, \end{cases} \\
&\text{Neumann:} \begin{cases} a(x,y) = g(x,y) \text{ on } \Gamma, \\ b(x,y) = u^+ \text{ on } \Gamma, \end{cases} \\
&\text{Robin:} \quad \begin{cases} a(x,y) = g(x,y) - \sigma u^+ \text{ on } \Gamma, \\ b(x,y) = u^+ \text{ on } \Gamma. \end{cases}
\end{aligned} \tag{3}$$

In (3) the function g is specified, but the function with a "+" superscript is unknown. Thus in each of the three cases shown in (3), a portion of the jump data (a,b) of the associated interface problem (2) must be found along with the solution u. For Dirichlet problems the unknown is a, and for Neumann problems it is b. For Robin problems both a and b are unknown, but we view b as the primary unknown.

Remark 2.1. The idea of formulating (1) in terms of (2) with unknown jump data is not new, and has been used extensively within the framework of the Immersed Interface Method [21,22,24], where the unknown jump data are referred to as augmented variables.

For simplicity of notation, we take the rectangular region Ω to be a square centered at the origin: $\Omega = (-L,L)\times(-L,L)$. The mesh size $h = \Delta x = \Delta y$ is chosen so that grid points are located on $\partial\Omega$. The spatial grid points are $(x_j, y_k) = (jh, kh)$, $-\mathcal{J} \le j,k \le \mathcal{J}$, where $h = L/\mathcal{J}$. For a function $Z(x_j, y_k)$ defined at the grid points (x_j, y_k), we use the abbreviation $Z_{j,k}$. We use standard second order discrete partial derivatives

$$\partial_x^h Z_{j,k} = (Z_{j+1,k} - Z_{j-1,k})/2h, \quad \partial_y^h Z_{j,k} = (Z_{j,k+1} - Z_{j,k-1})/2h.$$

The discrete gradient operator is defined by

$$\nabla^h Z_{j,k} = \left(\partial_x^h Z_{j,k}, \partial_y^h Z_{j,k}\right),$$

and the discrete normal derivative to Γ is defined by

$$\partial_n^h Z_{j,k} = -\nabla^h Z_{j,k} \cdot \frac{\nabla^h \phi_{j,k}}{\left|\nabla^h \phi_{j,k}\right|} \approx \frac{\partial Z}{\partial n}(x_j, y_k).$$

We use the standard five-point discrete Laplacian:

$$\Delta^h Z_{j,k} = \frac{1}{h^2}\left(Z_{j-1,k} + Z_{j+1,k} + Z_{j,k-1} + Z_{j,k+1} - 4Z_{j,k}\right).$$

The (extended) approximate solution produced by the algorithms is denoted $U_{j,k}$, i.e., $U_{j,k}|_{\Omega_+} \approx u(x_j, y_k)$, where u is the exact solution of (1).

Let $\mathcal{N}_1$ (the so-called set of irregular points) denote the set of grid points (x_j, y_k) which are on Γ or adjacent to it:

$$\mathcal{N}_1 = \{(x_j, y_k) : \phi_{j,k}\phi_{j\pm1,k} \le 0 \text{ or } \phi_{j,k}\phi_{j,k\pm1} \le 0\}.$$

The characteristic function of $\mathcal{N}_1$ is denoted $\chi_{j,k}$:

$$\chi_{j,k} = \begin{cases} 1, & (x_j, y_k) \in \mathcal{N}_1, \\ 0, & (x_j, y_k) \notin \mathcal{N}_1. \end{cases}$$

In addition to $\mathcal{N}_1$ we will also employ the sets $\mathcal{N}_2$, $\mathcal{N}_3$, $\mathcal{N}_4$, where

$$\mathcal{N}_2 := \mathcal{N}_1 \cup \{(x_j, y_k) : (x_{j\pm1}, y_k) \in \mathcal{N}_1 \text{ or } (x_j, y_{k\pm1}) \in \mathcal{N}_1\},$$

and $\mathcal{N}_3$, $\mathcal{N}_4$ are defined similarly in terms of $\mathcal{N}_2$, $\mathcal{N}_3$.

Let H denote the Heaviside function: $H(z) = 0$ if $z < 0$, $H(z) = 1$ if $z \geq 0$. The algorithms require a discrete regularized version of $H(\phi(x, y))$, and also the Dirac delta function $\delta(\phi(x, y)) = H'(\phi(x, y))$. To this end, define

$$I(z) = \max(0, z), \quad J(z) = \frac{1}{2}\max(0, z)^2, \quad K(z) = \frac{1}{6}\max(0, z)^3.$$

We use the following discretized versions of $H(\phi)$ and $\delta(\phi)$ [39]:

$$\begin{aligned} H^h_{j,k} &= \chi_{j,k} \cdot \left(\frac{\Delta^h J_{j,k}}{\left|\nabla^h \phi_{j,k}\right|^2} - \frac{\left(\Delta^h K_{j,k} - J_{j,k}\Delta^h \phi_{j,k}\right)\Delta^h \phi_{j,k}}{\left|\nabla^h \phi_{j,k}\right|^4} \right) + (1 - \chi_{j,k}) \cdot H(\phi_{j,k}), \\ \delta^h_{j,k} &= \chi_{j,k} \cdot \left(\frac{\Delta^h I_{j,k}}{\left|\nabla^h \phi_{j,k}\right|^2} - \frac{\left(\Delta^h J_{j,k} - I_{j,k}\Delta^h \phi_{j,k}\right)\Delta^h \phi_{j,k}}{\left|\nabla^h \phi_{j,k}\right|^4} \right). \end{aligned} \tag{4}$$

Here $I_{j,k}$ is an abbreviation for $I(\phi_{j,k})$, and likewise for $J_{j,k}$ and $K_{j,k}$. In the sequel we will similarly use $H_{j,k}$ as an abbreviation for $H(\phi_{j,k})$. In (4) $H^h_{j,k} = H_{j,k}$ for $(x_j, y_k) \notin \mathcal{N}_1$, meaning that the regularization is confined to one mesh width on either side of Γ. Also $\delta^h_{j,k} = 0$ for $(x_j, y_k) \notin \mathcal{N}_1$.

The discrete Heaviside and delta functions in (4) were derived using the finite difference approach of [37,38]. But they are somewhat different from the ones appearing in [37,38] because they were designed specifically for the interface problem (2). To clarify this difference take the case of the delta function; a similar discussion applies to the Heaviside function. From the second formula of (4) one recovers the discrete delta function of [37] if $\Delta^h J_{j,k} - I_{j,k}\Delta^h \phi_{j,k}$ is replaced by the simpler quantity $\nabla^h I_{j,k} \cdot \nabla^h \phi_{j,k}$, which follows from the identity $\nabla I \cdot \nabla \phi = \Delta J - I\Delta\phi$. Section 4 of [39] contains a heuristic explanation of the preference for the more complicated formula. That discussion can be roughly summarized as follows: for the singular Poisson problems to be described in Section 2.1, the $\Delta^h J_{j,k}$ portion of the more complicated formula can be ignored as a contribution to the order of truncation error, essentially yielding a discrete Poisson problem with a less singular source term and resulting in a more accurate computed approximation.

2.1. A pair of algorithms for the Poisson interface problem

Reference [39] derived the following pair of source term / level set formulations of the Poisson interface problem (2):

$$\begin{cases} \Delta u = \Delta(bH) - H\Delta b - \left(a - \frac{\partial b}{\partial n}\right)\delta(\phi)\left|\nabla\phi\right| + S \text{ in } \Omega, \\ u = \rho \text{ on } \partial\Omega, \end{cases} \tag{5}$$

and

$$\begin{cases} \Delta u = \Delta\left(\hat{b}H\right) - H\Delta\hat{b} + S \text{ in } \Omega, \\ \hat{b} := b + \left(\frac{\partial b}{\partial n} - a\right)\phi/\left|\nabla\phi\right|, \\ u = \rho \text{ on } \partial\Omega. \end{cases} \tag{6}$$

The same paper proposed the following discretization of (5):

$$\begin{cases} \Delta^h U_{j,k} = \Delta^h\left(b_{j,k}H_{j,k}\right) - H^h_{j,k}\Delta^h b_{j,k} - \left(a_{j,k} - \partial^h_n b_{j,k}\right)\delta^h_{j,k}\left|\nabla^h\phi_{j,k}\right| + S^h_{j,k}, & (x_j, y_k) \in \Omega, \\ S^h_{j,k} := H^h_{j,k}S^p_{j,k} + (1 - H^h_{j,k})S^m_{j,k}, & \\ U_{j,k} = \rho_{j,k}, & (x_j, y_k) \in \partial\Omega, \end{cases} \tag{7}$$

and the following discretization of (6):

$$\begin{cases} \Delta^h U_{j,k} = \Delta^h\left(\hat{b}_{j,k}H_{j,k}\right) - H^h_{j,k}\Delta^h\hat{b}_{j,k} + S^h_{j,k}, & (x_j, y_k) \in \Omega, \\ \hat{b}_{j,k} := b_{j,k} + \left(\partial^h_n b_{j,k} - a_{j,k}\right)\phi_{j,k}/\left|\nabla^h\phi_{j,k}\right|, & \\ S^h_{j,k} := H^h_{j,k}S^p_{j,k} + (1 - H^h_{j,k})S^m_{j,k}, & \\ U_{j,k} = \rho_{j,k}, & (x_j, y_k) \in \partial\Omega. \end{cases} \tag{8}$$

In (7) and (8) the first appearance of the Heaviside function $H_{j,k}$ in the discrete PDE's is not regularized in any way. This produces a sharp jump in the solution, and allows for *pointwise* (as opposed to L^1 or L^2) second order accuracy.

For the discretization (8), [39] proved $O(h^2)$ convergence in the maximum norm using a result of [5], and numerical experiments confirmed this rate of convergence. Numerical experiments also indicated second order convergence for the discretization (7). The discretizations (7) and (8) are the basis for the method proposed in this paper for solving the Poisson problem (1).

The use of discrete delta functions to capture jumps in the normal derivative of the solution of a differential equation is a widely used and well studied computational device, see e.g., [36] and the references therein. However the interface problem (2) also requires a jump in the solution itself. The use of singular source terms for this purpose as in (7) and (8) has been much less explored.

2.2. Algorithms for the Poisson boundary value problem

What follows is a description of the algorithm for (1) and its exterior version. The scheme boils down to two main operations performed iteratively: first solve an interface problem using one of the interface solvers from Section 2.1, and second update the unknown jump data via extrapolation from the interior of Ω_+ onto Γ.

There are three variants of the algorithm, one for each type of boundary condition. For Dirichlet problems, the algorithm is based on the discretization (8). For Neumann and Robin problems, we use the discretization (7). The discretization (8) is generally more accurate than (7), with a truncation error at irregular points of $O(h)$ versus $O(1)$ [39]. The superior accuracy of (8) comes at the price of needing a more accurate $\partial_n^h b_{j,k}$. This can be understood by noting that in (8) $\partial_n^h b_{j,k}$ is differenced two additional times in the terms $\Delta^h(\hat{b}_{j,k}H_{j,k})$ and $H_{j,k}^h\Delta^h\hat{b}_{j,k}$. On the other hand, it is possible to extend $b_{j,k}$ near the boundary Γ so that it is approximately constant in the normal direction. One can then simply approximate $\partial_n^h b_{j,k}\approx 0$. We found that for Dirichlet problems this approach makes it possible to take advantage of the better accuracy of (8). For Neumann and Robin problems we obtained better results using (7).

By setting $S_{j,k}^m = 0$, $S_{j,k}^p = f_{j,k}$ the discretization (7) becomes

$$\begin{cases} \Delta^h U_{j,k} = \mathcal{S}_{j,k}^N, & (x_j, y_k) \in \Omega, \\ U_{j,k} = \rho_{j,k}, & (x_j, y_k) \in \partial\Omega, \end{cases} \tag{9}$$

where

$$\mathcal{S}_{j,k}^N = \underbrace{\Delta^h\left(b_{j,k}H_{j,k}\right) - H_{j,k}^h\Delta^h b_{j,k} - \left(a_{j,k} - \partial_n^h b_{j,k}\right)\delta_{j,k}^h\left|\nabla^h\phi_{j,k}\right|}_{\text{singular part}} + H_{j,k}^h f_{j,k}. \tag{10}$$

The singular part of $\mathcal{S}_{j,k}^N$ is identically zero for $(x_j, y_k)\notin\mathcal{N}_1$, i.e., it only needs to be computed in $\mathcal{N}_1$.

By setting $S_{j,k}^m = 0$, $S_{j,k}^p = f_{j,k}$ and $\partial_n^h b_{j,k} = 0$, the discretization (8) becomes

$$\begin{cases} \Delta^h U_{j,k} = \mathcal{S}_{j,k}^D, & (x_j, y_k) \in \Omega, \\ U_{j,k} = \rho_{j,k}, & (x_j, y_k) \in \partial\Omega, \end{cases} \tag{11}$$

where

$$\begin{aligned} \mathcal{S}_{j,k}^D &= \underbrace{\Delta^h\left(\hat{b}_{j,k}H_{j,k}\right) - H_{j,k}^h\Delta^h\hat{b}_{j,k}}_{\text{singular part}} + H_{j,k}^h f_{j,k}, \\ \hat{b}_{j,k} &:= b_{j,k} - a_{j,k}\phi_{j,k}/\left|\nabla^h\phi_{j,k}\right|. \end{aligned} \tag{12}$$

We set $\partial_n^h b_{j,k} = 0$ due to our method of extending $b(x, y)$ off of Γ (see the discussion above and Section 2.3). The singular part of $\mathcal{S}_{j,k}^D$ is identically zero for $(x_j, y_k)\notin\mathcal{N}_1$.

In (9) and (11), the boundary data on $\partial\Omega$ is

$$\rho_{j,k} = \begin{cases} 0 & \text{for interior problems,} \\ \gamma(x_j, y_k) & \text{for exterior problems.} \end{cases}$$

All three variants of the algorithm require a solution of the discrete Poisson system of linear equations

$$\begin{cases} \Delta^h U_{j,k} &= Q_{j,k}, \quad (x_j, y_k) \in \Omega, \\ U_{j,k} &= \rho_{j,k}, \quad (x_j, y_k) \in \partial\Omega, \end{cases} \tag{13}$$

once per iteration, which can be computed efficiently with a fast Poisson solver.

Algorithm 1: Algorithm for the Dirichlet problem.

- 1 Extend $g(x,y)=u(x,y)$ off of Γ using (14). The result is $b_{j,k}$ defined throughout $\mathcal{N}_2$;
- 2 Extrapolate $f_{j,k}$ so that it is defined throughout $\Omega_+\cup\mathcal{N}_1$ (using the algorithm of Section 2.4 with $\mathcal{E}_0=\Omega_+\setminus\mathcal{N}_1$, $\mathcal{T}_0=\mathcal{N}_1\setminus\mathcal{E}_0$);
- 3 Initialize $a_{j,k}=0$ throughout $\mathcal{N}_2$;
- 4 Set the boundary data for (13): $\rho_{j,k}=0$ for an interior problem, and $\rho_{j,k}=\gamma(x_j,y_k)$ for an exterior problem;

while *not done* **do**

- A1 Compute the source term $\mathcal{S}^D_{j,k}$ using (12);
- A2 Compute $Q_{j,k}=\mathcal{S}^D_{j,k}+L_{j,k}$ where $L_{j,k}$ is the forcing term given by (15);
- A3 Call a Poisson solver to solve (13), resulting in $U_{j,k}$ defined throughout Ω;
- A4 Compute $\partial^h_x U_{j,k}$, $\partial^h_y U_{j,k}$ for $(x_j,y_k)\in(\Omega_+\cap\mathcal{N}_4)\setminus\mathcal{N}_1$;
- A5 Extrapolate $\partial^h_x U_{j,k}$, $\partial^h_y U_{j,k}$ so that they are defined throughout $\mathcal{N}_2$ (using the algorithm of Section 2.4 with $\mathcal{E}_0=\Omega_+\setminus\mathcal{N}_1$, $\mathcal{T}_0=\mathcal{N}_2\setminus\mathcal{E}_0$);
- A6 Compute the new $a_{j,k}=-(\partial^h_x U_{j,k},\partial^h_y U_{j,k})\cdot\nabla^h\phi_{j,k}/|\nabla^h\phi_{j,k}|$ defined throughout $\mathcal{N}_2$;
- A7 Check for termination, see Section 2.6;

end

Algorithm 2: Algorithm for the Neumann problem.

- 1 Extend $g(x,y)=(\partial u/\partial n)(x,y)$ off of Γ using (14), resulting in $a_{j,k}$ defined throughout $\mathcal{N}_1$;
- 2 Extrapolate $f_{j,k}$ so that it is defined throughout $\Omega_+\cup\mathcal{N}_1$ (using the algorithm of Section 2.4 with $\mathcal{E}_0=\Omega_+\setminus\mathcal{N}_1$, $\mathcal{T}_0=\mathcal{N}_1\setminus\mathcal{E}_0$);
- 3 Initialize $b_{j,k}=0$ throughout $\mathcal{N}_2$;
- 4 Set the boundary data for (13): $\rho_{j,k}=0$ for an interior problem, and $\rho_{j,k}=\gamma(x_j,y_k)$ for an exterior problem;

while *not done* **do**

- A1 Compute the source term $\mathcal{S}^N_{j,k}$ using (10). Set $Q_{j,k}=\mathcal{S}^N_{j,k}$;
- A2 Call a Poisson solver to solve (13), resulting in $U_{j,k}$ defined throughout Ω;
- A3 Extrapolate $U_{j,k}|_{\Omega_+\setminus\mathcal{N}_1}$ so that it is defined throughout $\mathcal{N}_2$, which gives the new $b_{j,k}$ (using the algorithm of Section 2.4 with $\mathcal{E}_0=\Omega_+\setminus\mathcal{N}_1$, $\mathcal{T}_0=\mathcal{N}_2\setminus\mathcal{E}_0$);
- A4 Check for termination, see Section 2.6;

end

Algorithm 3: Algorithm for the Robin problem.

- 1 Extend $\sigma(x,y)$ off of Γ using (14), the result is $\sigma_{j,k}$ defined throughout $\mathcal{N}_1$;
- 2 Extend $g(x,y)=(\partial u/\partial n)(x,y)+\sigma(x,y)u(x,y)$ off of Γ using (14), the result is $g_{j,k}$ defined throughout $\mathcal{N}_1$;
- 3 Extrapolate $f_{j,k}$ so that it is defined throughout $\Omega_+\cup\mathcal{N}_1$ (using the algorithm of Section 2.4 with $\mathcal{E}_0=\Omega_+\setminus\mathcal{N}_1$, $\mathcal{T}_0=\mathcal{N}_1\setminus\mathcal{E}_0$);
- 4 Initialize $b_{j,k}=0$ throughout $\mathcal{N}_2$;
- 5 Set the boundary data for (13): $\rho_{j,k}=0$ for an interior problem, and $\rho_{j,k}=\gamma(x_j,y_k)$ for an exterior problem;

while *not done* **do**

- A1 Compute $a_{j,k}=g_{j,k}-\sigma_{j,k}b_{j,k}$, defined throughout $\mathcal{N}_1$;
- A2 Compute the source term $\mathcal{S}^N_{j,k}$ using (10). Set $Q_{j,k}=\mathcal{S}^N_{j,k}$;
- A3 Call a Poisson solver to solve (13), resulting in $U_{j,k}$ defined throughout Ω;
- A4 Extrapolate $U_{j,k}|_{\Omega_+\setminus\mathcal{N}_1}$ so that it is defined throughout $\mathcal{N}_2$, which gives the new $b_{j,k}$ (using the algorithm of Section 2.4 with $\mathcal{E}_0=\Omega_+\setminus\mathcal{N}_1$, $\mathcal{T}_0=\mathcal{N}_2\setminus\mathcal{E}_0$);
- A5 Check for termination, see Section 2.6;

end

Remark 2.2. After the first iteration only the singular parts of $\mathcal{S}^D_{j,k}$, $\mathcal{S}^N_{j,k}$ need to be updated, contributing $O(1/h)$ operations per iteration.

Remark 2.3. The FFT-based fast Poisson solver contributes $O((1/h^2)\log(1/h))$ operations to the processing inside the loop of each algorithm. All of the rest of the processing within the loop is restricted to a narrow band surrounding Γ and contributes $O(1/h)$ operations per iteration.

Remark 2.4. The Neumann problem is a special case of the Robin problem, and clearly one could include the Neumann algorithm in the Robin algorithm. We leave them separate in order to keep the Neumann algorithm as simple as possible, since Neumann problems are more common.

2.3. Extending data off of the interface Γ

For each type of problem there is a function $g(x,y)$ that is a priori only specified on the interface Γ. (For Robin problems there is σ in addition to g.) The algorithms require an extension of g to a narrow band of grid points surrounding Γ. We use the following linear projection:

$$
\begin{aligned}
&g^p_{j,k} = g(x^p_{j,k}, y^p_{j,k}) \text{ where} \\
&x^p_{j,k} = x_j + n^x_{j,k}\phi_{j,k}/\left|\nabla^h\phi_{j,k}\right|, \quad y^p_{j,k} = y_k + n^y_{j,k}\phi_{j,k}/\left|\nabla^h\phi_{j,k}\right|, \\
&n^x_{j,k} = -\partial^h_x\phi_{j,k}/\left|\nabla^h\phi_{j,k}\right|, \quad n^y_{j,k} = -\partial^h_y\phi_{j,k}/\left|\nabla^h\phi_{j,k}\right|.
\end{aligned} \tag{14}
$$

The projected point $(x^p_{j,k}, y^p_{j,k})$ is generally only near Γ, not exactly on it, but (14) is accurate enough for our purposes. Reference [22] provides a more accurate quadratic projection method, and reference [8] contains an iterative scheme for locating the closest point on Γ to third order accuracy, which is also used in [10].

A consequence of the extension (14) is that $\partial^h_n g^p_{j,k} \approx 0$. As mentioned earlier, based on this approximation we set $\partial^h_n b_{j,k} = 0$ in the Dirichlet algorithm. This is not strictly necessary, but it simplifies the algorithm and it turns out to improve accuracy.

Remark 2.5. In some applications the function $g(x, y)$ is already defined at surrounding grid points, not just on the interface. In that case one can skip the projection step described above. For Dirichlet problems this would mean that the second line of (12) should be replaced by

$$\hat{b}_{j,k} := b_{j,k} + \left(\partial^h_n b_{j,k} - a_{j,k}\right)\phi_{j,k}/\left|\nabla^h\phi_{j,k}\right|.$$

In the particular case of $g(x, y) = b(x, y)$, even if b is defined at grid points, it may still be advantageous (from the standpoint of accuracy) to use the projection described above. Then there would be an additional step, where one approximates $g(x^p_{j,k}, y^p_{j,k})$ by interpolation using nearby grid values. This interpolation step was not modeled in the numerical results of Section 4. However we checked on a small subset of those examples that this approach is compatible with our algorithms. We found that it is necessary to use at least quadratic interpolation [29]; with bilinear interpolation there is a significant loss of accuracy at the finer mesh sizes.

2.4. Extrapolation across the interface

The algorithms above require extrapolated values of $(\partial^h_x U_{j,k}, \partial^h_y U_{j,k})$ throughout the set $\mathcal{N}_2$ for the Dirichlet problem, and $U_{j,k}$ throughout $\mathcal{N}_2$ for the Neumann and Robin problems. Extrapolated values of $f_{j,k}$ are required throughout $\Omega_+ \cup \mathcal{N}_1$ for all three problem types. In all cases we extrapolate from the "+" side of Γ, using values from the set $\Omega_+ \setminus \mathcal{N}_1$.

In order to achieve second order accuracy, numerical experience indicates that one must use at least linear extrapolation for $(\partial^h_x U_{j,k}, \partial^h_y U_{j,k})$ and $f_{j,k}$, and quadratic extrapolation for $U_{j,k}$. In practice, we use quadratic extrapolation in all cases because it is more accurate and requires only slightly more computational effort than linear extrapolation.

The extrapolation algorithm is described next. Alternative approaches to solving this type of extrapolation problem can be found in [2,3,30]. Let $z_{j,k}$ denote the quantity that is to be extrapolated across Γ, and let $\tilde{z}_{j,k}$ denote the extrapolated quantity. We start by initializing the set of eligible points, $\mathcal{E} = \mathcal{E}_0 = \Omega_+ \setminus \mathcal{N}_1$, and setting $\tilde{z}_{j,k} = z_{j,k}$ for $(x_j, y_k) \in \mathcal{E}_0$. Let $\mathcal{T} = \mathcal{T}_0$ denote the initial set of target points. The goal is to give each of the target points an extrapolated value, using data from the eligible points. After a target point has received an extrapolated value its status changes from "target" to "eligible". As the algorithm proceeds, the eligible set $\mathcal{E}$ grows and the target set $\mathcal{T}$ shrinks. At completion $\mathcal{T} = \emptyset$ and $\mathcal{E} = \mathcal{E}_0 \cup \mathcal{T}_0$.

The target set for extrapolating $U_{j,k}$ is initialized to $\mathcal{T}_0 = \mathcal{N}_2 \setminus \mathcal{E}_0$. The target set for extrapolating $(\partial^h_x U_{j,k}, \partial^h_y U_{j,k})$ is also initialized to $\mathcal{T}_0 = \mathcal{N}_2 \setminus \mathcal{E}_0$. When extrapolating $f_{j,k}$, the target set is initialized to $\mathcal{T}_0 = \mathcal{N}_1 \setminus \mathcal{E}_0$.

Remark 2.6. For $f_{j,k}$ one could include irregular points in $\mathcal{E}_0$, i.e., set $\mathcal{E}_0 = \Omega_+$. This potentially gives a more accurate result, but we found that the improvement is negligible.

Quadratic extrapolation algorithm. Given a target point $(x_j, y_k) \in \mathcal{T}$, we check to see if it has a triplet of adjacent eligible grid points. This means that we check four triplets of points:

East: $(x_{j+1}, y_k), (x_{j+2}, y_k), (x_{j+3}, y_k)$ West: $(x_{j-1}, y_k), (x_{j-2}, y_k), (x_{j-3}, y_k)$

North: $(x_j, y_{k+1}), (x_j, y_{k+2}), (x_j, y_{k+3})$ South: $(x_j, y_{k-1}), (x_j, y_{k-2}), (x_j, y_{k-3})$.

For example, if it is found that all three points in the East triplet above are in $\mathcal{E}$, then $\tilde{z}_{j,k}$ is computed by quadratic extrapolation: $\tilde{z}_{j,k} = 3\tilde{z}_{j+1,k} - 3\tilde{z}_{j+2,k} + \tilde{z}_{j+3,k}$. If it turns out that there are other adjacent eligible triplets, we take the average of the extrapolated values for $\tilde{z}_{j,k}$. After visiting all current target points in $\mathcal{T}$, we add to the set $\mathcal{E}$ those target points (x_j, y_k) where we found one or more eligible triplets, and we remove all such points from $\mathcal{T}$. In addition, the new extrapolated value $\tilde{z}_{j,k}$ becomes available for use in extrapolating to other points still in $\mathcal{T}$. The process is then repeated until the target set $\mathcal{T}$ is empty. We found that five iterations were sufficient to leave the target set empty in all of the examples in Section 4. See Fig. 3 and Table 1 for an example of the extrapolation algorithm.

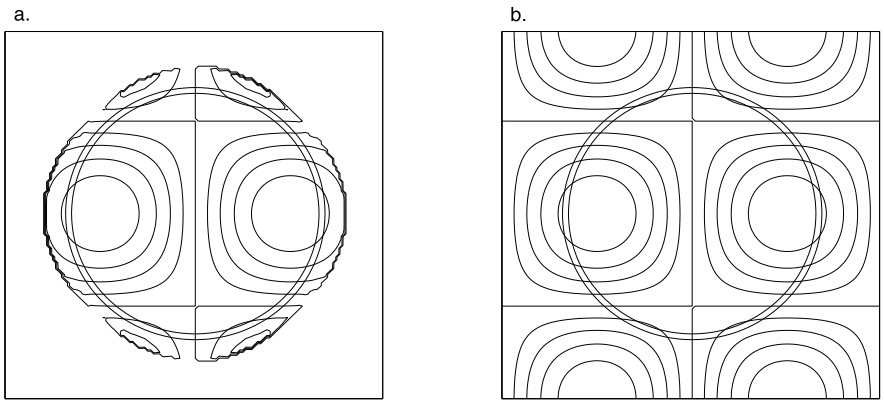

Fig. 3. The left plot shows the result after 10 iterations of the quadratic extrapolation algorithm of Section 2.4 on a 128×128 grid. The box is the computational domain $\Omega = [-1, 1]^2$, and the level set function is $\phi(x, y) = .65 - r$. The function $\psi(x, y) = H(\phi)\sin(\pi x)\cos(\pi y)$ is extrapolated outward from the inside of the inner circle which marks the boundary Γ. The algorithm was run for 10 iterations to make this plot, but for the Poisson algorithms 5 iterations suffices in all of the examples in Section 4. The outer circle is $\phi(x, y) = -2h$, provided for reference. The annular region between the two circles is indicative of the size of band in which one would need to compute extrapolated values. The right plot shows the exact $\psi(x, y)$ for comparison.

Table 1
Test of quadratic extrapolation. Maximum error on $\mathcal{N}_2$ for the example shown in Fig. 3. Rate of convergence ≈ 3, as desired for a quadratic extrapolation algorithm.

Grid	32×32	64×64	128×128	256×256	512×512
Error	6.210e−2	8.981e−3	1.113e−3	1.424e−4	1.808e−5
Rate		2.79	3.01	2.97	2.97

The number of iterations required depends on the shape of the boundary, and also the level of grid refinement. For a given problem, the number of iterations generally decreases with decreasing mesh size h, until h is small enough that the boundary is adequately resolved. Moreover, it is possible to construct examples where more than five iterations are required. As an example of this, we performed some tests (not included in Section 4) with a diagonally oriented and slender elliptical boundary. With a sufficiently slender ellipse and coarse mesh, we observed an instance where the algorithm terminated (successfully, with the target set empty) after eleven iterations. Going a step further, it is possible to have the algorithm fail (the target set reaches a state where it is not empty and also not diminishing from one iteration to the next) if the ellipse is too slender and the mesh is too coarse. In this case there are not enough triplets of adjacent eligible points.

2.5. Convergence forcing term

For Dirichlet problems it is possible to accelerate convergence and improve accuracy by adding to the right side of (13) an additional source term whose purpose is to force the solution toward zero on Ω_-. The Dirichlet algorithm uses a forcing term of the form

$$L_{j,k} = \begin{cases} qh^{-1} \cdot U_{j,k}, & \text{if } x_{j,k} \in \Omega_- \cap \mathcal{N}_2, \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

Here q is a parameter, h is the mesh size, and $U_{j,k}$ is the extended solution computed at the previous iteration. On the first iteration $L_{j,k} = 0$. The value $q = .75$ worked well on all of the Dirichlet problems that we tested. See Fig. 4.

A motivation for the forcing term comes from the following observations. First, for a Dirichlet problem, where the jump b is known and corresponds to the Dirichlet boundary data, forcing an approximate solution nearer to zero on Ω_- should force the Dirichlet boundary condition to be more nearly satisfied. The second observation is that the interior Dirichlet problem

$$\Delta u = \mathcal{S} \text{ in } \Omega, \qquad u|_{\partial\Omega} = 0$$

can also be viewed as the problem of minimizing

$$\frac{1}{2}\int_\Omega |\nabla u|^2 \, dx dy + \int_\Omega \mathcal{S} u \, dx dy, \quad u|_{\partial\Omega} = 0. \tag{16}$$

Since we seek a solution that vanishes on Ω_-, we may equivalently minimize

$$\frac{1}{2}\int_\Omega |\nabla u|^2\,dx\,dy + \int_\Omega \mathcal{S}u\,dx\,dy + \frac{\nu}{2}\int_\Lambda u^2\,dx\,dy, \quad u|_{\partial\Omega} = 0, \tag{17}$$

where ν is a nonnegative constant and $\Lambda \subseteq \Omega_-$. When the source term $\mathcal{S}$ is only approximate (as is the case for the discrete version during the iterations), the solution of (17) will generally be closer to zero on Λ (and hopefully also on the larger set Ω_-) than the solution of (16). The Poisson problem associated with (17) is

$$\Delta u = \mathcal{S} + \nu\kappa u \text{ in } \Omega, \qquad u|_{\partial\Omega} = 0, \tag{18}$$

where $\kappa = \kappa(x, y)$ is the characteristic function of the set Λ. The source term $L_{j,k}$ then comes from the $\nu\kappa u$ term when (18) is discretized. We take Λ to be a thin strip near Γ (but in Ω_-) so that there is only $O(1/h)$ additional processing per iteration. The specific form $\nu = q/h^\alpha$, $\alpha > 0$ can be motivated by assuming that ν should be large, as measured by the level of grid refinement. The specific values $q = .75$ and $\alpha = 1$ result after some numerical experimentation.

The forcing term (15) was added in all of the Dirichlet problems in Section 4, but a forcing term was not used on the Neumann or Robin problems. Focusing the discussion on Neumann problems, where a is the known quantity, one would want to force $|\nabla u|$ toward zero on Ω_-, but we did not find a satisfactory method for this. We obtained some ad hoc but promising results (not shown in Section 4) on Neumann problems by using a modified version of (15). The forcing parameter q (which is now negative) is allowed to ramp up gradually as the iterations progress:

$$q = -.75\min(1, m/10),$$

where m is the iteration number.

2.6. Termination criterion

A termination criterion is required for the iterative processes described in Algorithms 1, 2, and 3. Let $U^m = \{U_{j,k} : (x_j, y_k) \in \Omega\}$ denote the approximate solution at the mth iteration. The fact that the desired solution is required to vanish on Ω_- suggests a stopping rule that is based on the progress of U^m toward zero on a subset of Ω_-. Let

$$\mathcal{Q}^m = \max\left\{\left|U^m_{j,k}\right| : (x_j, y_k) \in \Omega_- \cap \mathcal{N}_2\right\}.$$

The iterations are terminated when $m > 5$ and $\mathcal{Q}^m \geq (99/100)\cdot\mathcal{Q}^{m-1}$. This criterion was used in all of the examples in Section 4. See Fig. 4.

The question arises as to whether the sequence $\{U^m\}$ converges as $m \to \infty$. Let m^* denote the index of the final iteration, determined by the criterion described above. By performing tests where we let the algorithm continue iterating beyond m^*, we found that there is not always convergence, but there is behavior that is similar enough to convergence for our purposes. More specifically our tests indicate that there is a grid function $\{W_{j,k}\}$, a real number $\sigma > 0$, and an integer $M > 0$ (all depending on the problem and on the mesh size h) such that

$$\max_{(x_j,y_k)\in\Omega}\left|U^m_{j,k} - W_{j,k}\right| \leq \sigma \text{ for } m \geq M, \tag{19}$$

where σ is smaller (by orders of magnitude if the boundary is well-resolved by the mesh) than the error in the approximation $\{W_{j,k}\}$,

$$\max_{(x_j,y_k)\in\Omega}\left|W_{j,k} - u(x_j, y_k)\right|.$$

In a number of cases where the boundary is well-resolved by the mesh we observed that there is actually convergence of the sequence $\{U^m\}$ (modulo floating point precision). In our tests the stopping criterion above resulted in $U^{m^*} \approx W$, with $U^{m^*} - W$ negligible except in a few cases where the boundary was very poorly resolved by the mesh.

The question also arises as to why we use a somewhat nonstandard stopping criterion, as opposed to stopping for example when the difference between successive approximations is less than some number ϵ. The main reason is that in some cases we do not have convergence, but only the weaker condition (19), making it difficult to choose the number ϵ. A secondary reason is based on efficiency. Even in those cases where there is convergence, where one could choose ϵ based on floating point precision, our numerical experiments indicate that the result is generally a solution whose accuracy is very close to that of U^{m*}, but at the cost of a number of iterations much greater than m^*.

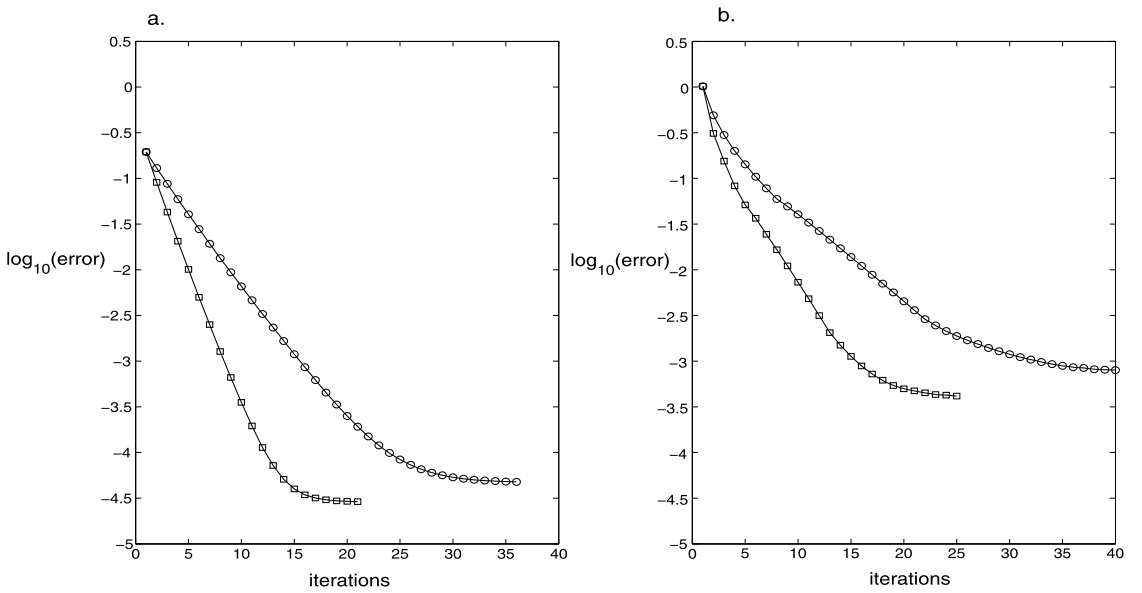

Fig. 4. $\log_{10}$(max error) versus number of iterations for two examples in Section 4. Left panel: Example 1. Right panel: Example 3. Line with circle markers: Without convergence forcing term $L_{j,k}$. Line with square markers: With convergence forcing term. The grid for both examples was 256×256. The stopping rule of Section 2.6 was used. As shown in these examples, convergence is usually faster with the forcing term, and the resulting error is often smaller. The forcing term is only used for Dirichlet problems.

2.7. Level set function

The scheme requires that the level set function ϕ be defined throughout Ω and smooth in $\mathcal{N}_3$. It does not need to be a signed distance function ($|\nabla\phi| = 1$ for a signed distance function), but we found that using a signed distance function generally improves accuracy, especially on the coarsest grids.

In all of the example problems of Section 4 the level set function ϕ is initially defined by a formula, see Table 2. A level set reinitialization algorithm was used to replace each level set function ϕ by a signed distance function in a narrow band containing Γ. The reinitialization method that we used is based on the two-step Newton-type closest point algorithm of [8], see also [10]. We obtained very similar results using signed distance functions produced by the reinitialization algorithm of [30], where one employs fifth order WENO spatial differencing and third order Runge–Kutta time stepping to solve

$$\phi_t + \mathcal{S}(\phi)\left(|\nabla\phi| - 1\right) = 0, \quad \mathcal{S}(\phi) := \phi/\sqrt{\phi^2 + |\nabla\phi|^2 h^2}.$$

3. The variable coefficient equation

This section describes how to modify the algorithms to handle the variable coefficient equation

$$\nabla \cdot (\beta \nabla u) = f. \tag{20}$$

Here $\beta = \beta(x, y)$ is smooth and positive, making it possible to rewrite (20) in the form

$$\Delta u = (f - \nabla\beta \cdot \nabla u)/\beta. \tag{21}$$

The constant coefficient scheme is applied to (21), relying on the iterative loop to resolve the unknown ∇u appearing on the right side of the equation.

With respect to the algorithms of Section 2, this means that the term $f_{j,k}$ appearing in $\mathcal{S}^N_{j,k}$ and $\mathcal{S}^D_{j,k}$ is replaced by

$$\tilde{f}_{j,k} := \left(f_{j,k} - \nabla^h\beta_{j,k} \cdot \overline{\nabla^h U_{j,k}}\right)/\beta_{j,k}. \tag{22}$$

Here $\overline{\nabla^h U_{j,k}}$ is an extended version of $\nabla^h U_{j,k}$, computed using $U_{j,k}$ from the most recent iteration. (On the first iteration, we set $\overline{\nabla^h U_{j,k}} = 0$.) In order to obtain second order accuracy, $\overline{\nabla^h U_{j,k}}$ must be regular throughout $\Omega_+ \cup \mathcal{N}_1$. Since $U_{j,k}$ generally has a jump along the interface, one cannot simply use $\partial^h_x U_{j,k}$ and $\partial^h_y U_{j,k}$. Instead $\partial^h_x U_{j,k}$ and $\partial^h_y U_{j,k}$ are computed in $\Omega_+ \setminus \mathcal{N}_1$ and then extended to $\Omega_+ \cup \mathcal{N}_1$ using the quadratic extrapolation algorithm of Section 2.4. Computing $\tilde{f}_{j,k}$ adds $O(1/h^2)$ processing within the iterative loop, but the variable coefficient algorithms are not noticeably slower than their constant coefficient counterparts.

Table 2
List of numerical examples. We use r and θ to denote polar coordinates, centered at $(x, y) = (0, 0)$. In the "type" column, D, N, R stand for Dirichlet, Neumann, Robin, and I,E stand for interior, exterior.

No.	Type	$u(x, y) =$ solution	$\phi(x, y) =$ level set function
1	D,I	$u = 1 + \sin(Ax)\cos(By)$ $A = .5\pi,\ B = .75\pi$	$\phi = .9 - \sqrt{(\alpha x)^2 + (\beta y)^2}$ $\alpha = 4,\ \beta = 1.25$
2	D,I	$u = 1 + \sin(Ax)\cos(By)$ $A = 2.7,\ B = 3.1$	$\phi = 0.5 + 0.1\sin(4\theta + .17\pi) - r$
3	D,I	$u = x^6 + y^6 + \sin(\pi x) + \sin(\pi y)$ $+ \cos(\pi x) + \cos(\pi y)$	$\phi = \rho^2 - ((x-\alpha)^2 + (y-\alpha)^2)$ $\rho := .5 + .2\sin(5\theta),\ \alpha := .02\sqrt{5}$
4	D,I	$u = 1 + \sin(Ax)\cos(By)$ $A = .6\pi,\ B = .7\pi$	$\phi = (r - .35)(.7 - r)$
5	N,I	$u = 1 + \sin(Ax)\cos(By)$ $A = .5\pi,\ B = .75\pi$	$\phi = .9 - \sqrt{(\alpha x)^2 + (\beta y)^2}$ $\alpha = 4,\ \beta = 1.25$
6	N,I	$u = 1 + \sin(Ax)\cos(By)$ $A = 2.7,\ B = 3.1$	$\phi = 0.5 + 0.1\sin(4\theta + .17\pi) - r$
7	N,I	$u = x^6 + y^6 + \sin(\pi x) + \sin(\pi y)$ $+ \cos(\pi x) + \cos(\pi y)$	$\phi = \rho^2 - ((x-\alpha)^2 + (y-\alpha)^2)$ $\rho := .5 + .2\sin(5\theta),\ \alpha := .02\sqrt{5}$
8	N,I	$u = 1 + \sin(Ax)\cos(By)$ $A = .6\pi,\ B = .7\pi$	$\phi = (r - .35)(.7 - r)$
9	R,I	$u = 1 + \sin(Ax)\cos(By)$ $A = 2.7,\ B = 3.1$	$\phi = 0.5 + 0.1\sin(4\theta + .17\pi) - r$
10	R,I	$u = 1 + \sin(Ax)\cos(By)$ $A = .6\pi,\ B = .7\pi$	$\phi = (r - .35)(.7 - r)$
11	D,E	$u = \log(r) + .5\sin(\pi x)\sin(\pi y)$	$\phi = r - (.5625 + .15\sin(3\theta + .17\pi))$
12	N,E	$u = -0.5\log r + r^2$	$\phi = (x/a)^2 + (y/b)^2 - 1$ a = 0.5, b=0.15

It is also possible to formulate a self-adjoint version of the above discretization via

$$\begin{aligned}
&\nabla\cdot\left(\beta(x_j, y_k)\nabla u(x_j, y_k)\right) \approx \Delta_+^x\left(\beta_{j-1/2,k}\Delta_-^x U_{j,k}\right)/h^2 + \Delta_+^y\left(\beta_{j,k-1/2}\Delta_-^y U_{j,k}\right)/h^2\\
&= \beta_{j,k}\Delta^h U_{j,k}\\
&+ \underbrace{\frac{1}{2h^2}\Delta_+^x\beta_{j,k}\Delta_+^x U_{j,k} + \frac{1}{2h^2}\Delta_-^x\beta_{j,k}\Delta_-^x U_{j,k} + \frac{1}{2h^2}\Delta_+^y\beta_{j,k}\Delta_+^y U_{j,k} + \frac{1}{2h^2}\Delta_-^y\beta_{j,k}\Delta_-^y U_{j,k}}_{=:(\nabla\beta\cdot\nabla u)^h_{j,k}}.
\end{aligned} \tag{23}$$

Here

$$\beta_{j-1/2,k} = \left(\beta_{j-1,k} + \beta_{j,k}\right)/2, \quad \beta_{j,k-1/2} = \left(\beta_{j,k-1} + \beta_{j,k}\right)/2,$$

and $\Delta_\pm^{x,y}$ are difference operators, e.g.,

$$\Delta_+^x U_{j,k} = U_{j+1,k} - U_{j,k}, \quad \Delta_-^x U_{j,k} = U_{j,k} - U_{j-1,k}.$$

Based on (23) the term $\nabla^h\beta_{j,k}\cdot\overline{\nabla^h U_{j,k}}$ in (22) is replaced by the extended version of $(\nabla\beta\cdot\nabla u)^h_{j,k}$ that results by extrapolating from $\Omega_+\setminus\mathcal{N}_1$ to $\Omega_+\cup\mathcal{N}_1$. Numerical tests indicate that the self-adjoint discretization gives results that are very similar to (22).

4. Numerical examples

This section presents the results of numerical experiments. The exact solutions $u(x, y)$ and level set functions $\phi(x, y)$ are listed in Table 2. In each case the computational domain is the square $\Omega = [-1, 1]\times[-1, 1]$. With $u(x_j, y_k)$ denoting the true solution, we use the notation

$$E_u = \max\left\{\left|u(x_j, y_k) - U_{j,k}\right| : \phi(x_j, y_k) \geq 0\right\},$$
$$E_{\nabla u} = \max\left\{\left|\nabla^h u(x_j, y_k) - \nabla^h U_{j,k}\right| : \phi(x_j, y_k) \geq 0,\ (x_j, y_k)\notin\mathcal{N}1\right\}.$$

For interior Neumann problems, it is necessary to first adjust the solution by a constant before computing the error. This is due to the fact that Neumann boundary conditions only determine the solution to within an additive constant. Due to the Dirichlet boundary condition on $\partial\Omega$, no correction was needed for exterior Neumann problems.

We measure the observed rate of convergence by repeatedly halving the mesh size. With $E(h)$ denoting the error using mesh size h, the empirical rate of convergence is then

$$\text{Rate} = \log(E(h)/E(h/2))/\log(2).$$

Table 3
Example 1. Interior Dirichlet problem. $\beta(x, y) = 1$. See Fig. 5.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
32×32	16	2.27e−3		2.36e−2	
64×64	18	5.41e−4	2.07	3.84e−3	2.62
128×128	19	1.23e−4	2.14	5.99e−3	2.68
256×256	21	2.90e−5	2.08	1.54e−4	1.96
512×512	23	7.21e−6	2.01	4.13e−5	1.90
1024×1024	25	1.80e−6	2.00	1.13e−5	1.86

Table 4
Example 2. Interior Dirichlet problem. $\beta(x, y) = \exp(xy)$. See Fig. 5.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
32×32	10	8.58e−3		4.00e−2	
64×64	13	1.62e−3	2.40	7.06e−3	2.50
128×128	15	2.47e−4	2.72	1.96e−3	1.85
256×256	18	6.34e−5	1.96	5.48e−4	1.84
512×512	19	1.86e−5	1.77	1.39e−4	1.98
1024×1024	21	4.97e−6	1.90	3.52e−5	1.98

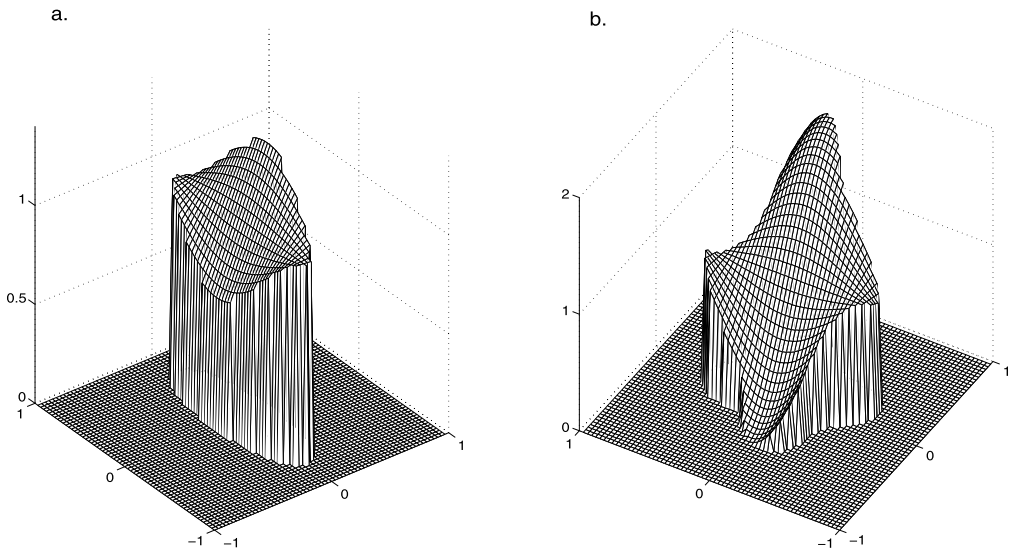

Fig. 5. Solution to Example 1, 5 (left plot), and solution to Example 2, 6, 9 (right plot).

In Tables 3 through 14 "it" gives the number of iterations. In the accompanying figures, the plotted solutions were computed and plotted using a 64×64 grid. For Dirichlet problems the results reflect the use of the convergence forcing term described in Section 2.5.

5. Concluding remarks

We have proposed finite difference algorithms for computing second order accurate solutions to Poisson problems on irregular domains. The method uses a Cartesian grid, and a level set function is used to describe the domain boundary. The boundary conditions are enforced via easily implemented singular source terms. Numerical examples confirm that the method works on both interior and exterior problems, and the domain does not need to be simply connected. The algorithms are iterative, but the number of iterations increases very gradually with increasing mesh refinement, with a growth rate that is far below linear.

The numerical results of Section 4, along with other tests not shown here, indicate second order accuracy (or close to it) of the solution u for all three types of problems. For the gradient ∇u the rate of convergence depends on the type of problem. For Dirichlet problems, we see a convergence rate for the gradient that is better than first order, and close to second order on some problems. For Neumann and Robbin problems the rate of convergence of the gradient is less clear, but seems to be closer to first order.

Table 5
Example 3. Interior Dirichlet problem. $\beta(x, y) = 1$. This example is borrowed from [10], and also appears in [12,18]. See Fig. 6.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
32×32	9	6.94e−2		2.41e−1	
64×64	19	8.11e−3	3.10	5.92e−2	2.03
128×128	19	1.45e−3	2.48	1.15e−2	2.36
256×256	25	4.14e−4	1.81	3.39e−3	1.76
512×512	31	1.28e−4	1.70	1.45e−3	1.23
1024×1024	34	3.39e−5	1.92	4.82e−4	1.59

Table 6
Example 4. Interior Dirichlet problem. $\beta(x, y) = 1$. See Fig. 6.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
32×32	14	2.39e−3		1.01e−2	
64×64	18	4.57e−4	2.39	2.87e−3	1.82
128×128	20	1.55e−4	1.56	8.06e−4	1.83
256×256	21	4.47e−5	1.79	2.31e−4	1.81
512×512	23	1.14e−5	1.97	5.74e−5	2.01
1024×1024	26	2.88e−6	1.99	1.47e−5	1.96

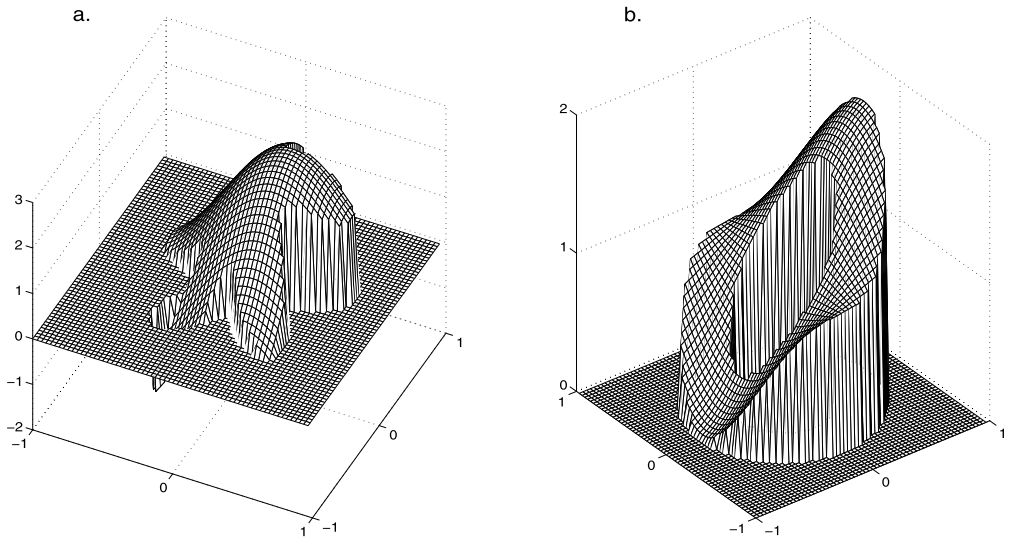

Fig. 6. Solution to Examples 3, 7 (left plot), and solution to Examples 4, 8, 10 (right plot). The example shown in the left plot is borrowed from [10], and also appears in [12,18].

We did a moderate amount of testing of variable coefficient problems, using the modified algorithm of Section 3. The numerical results are shown in Tables 4, 8, and 11 (Examples 2, 6, and 9, where $\beta(x, y) = \exp(xy)$). We also ran those examples with $\beta(x, y) = 1$ (results not shown), and the results are quite similar to those shown in Tables 4, 8, and 11. These three examples, along with a few others not shown, indicate that the numerical results (approximation errors, number of iterations) are not very sensitive to the coefficient β, assuming a fixed solution u.

For problems where the interface Γ has significant curvature, a relatively fine mesh may be required in order to obtain good resolution. One can see this in Table 5 which shows the results for the flower domain of Example 3. The under-resolution of the first few mesh refinement levels is evident in the high observed convergence rates. One possible way to improve the algorithm's performance when the boundary is under-resolved is via local mesh refinement, where a much finer grid is used near the interface. This is left as a potential topic of future investigation.

We have focused on the case where the data is smooth, and all of the numerical examples in Section 4 reflect this. The question arises as to how the algorithm performs with less regular data. For example if the boundary is not smooth the level set function will generally have shocks or kinks and the normal vector (used in the projection (14), for $a_{j,k}$ in the Dirichlet algorithm, and for $\partial_n^h b_{j,k}$ in the Neumann and Robin problems) may be mis-evaluated. A limited amount of testing (results not included in Section 4) on Dirichlet problems where the boundary Γ is Lipschitz continuous, and piecewise smooth with corners, indicates pointwise convergence of both u and ∇u, but with possibly reduced rates, depending on the problem. The only change that we made for these tests was in the level set function reinitialization method: we used

Table 7
Example 5. Interior Neumann problem. $\beta(x, y) = 1$. See Fig. 5.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
32×32	7	1.53e−3		8.80e−3	
64×64	9	3.33e−4	2.19	8.13e−3	0.11
128×128	11	7.18e−5	2.22	2.13e−3	1.93
256×256	12	2.49e−5	1.53	6.72e−4	1.66
512×512	14	6.37e−6	1.97	3.24e−4	1.05
1024×1024	16	1.68e−6	1.92	1.59e−5	1.03

Table 8
Example 6. Interior Neumann problem. $\beta(x, y) = \exp(xy)$. See Fig. 5.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
32×32	22	1.47e−2		6.30e−2	
64×64	27	4.61e−3	1.67	2.09e−2	1.59
128×128	32	1.04e−3	2.15	8.81e−3	1.25
256×256	36	2.58e−4	2.00	2.62e−3	1.75
512×512	42	6.62e−5	1.96	1.03e−3	1.34
1024×1024	47	1.79e−5	1.89	5.59e−4	0.88

Table 9
Example 7. Interior Neumann problem. $\beta(x, y) = 1$. See Fig. 6.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
32×32	11	2.13e−1		6.17e−1	
64×64	30	4.72e−2	2.18	2.80e−1	1.14
128×128	36	9.55e−3	2.31	8.31e−2	1.75
256×256	46	2.63e−3	1.86	4.13e−2	1.01
512×512	57	6.55e−4	2.01	1.45e−2	1.51
1024×1024	62	1.84e−4	1.83	6.25e−3	1.21

Table 10
Example 8. Interior Neumann problem. $\beta(x, y) = 1$. See Fig. 6.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
32×32	36	1.26e−2		3.39e−2	
64×64	46	3.19e−3	1.98	9.17e−3	1.88
128×128	54	8.47e−4	1.91	2.88e−3	1.67
256×256	61	2.10e−4	2.01	1.04e−3	1.48
512×512	69	6.12e−5	1.78	4.22e−4	1.30
1024×1024	77	1.60e−5	1.93	2.71e−4	0.64

Table 11
Example 9. Interior Robin problem, $\sigma(x, y) = 2.5 + .25x\sin(3\theta)$. $\beta(x, y) = \exp(xy)$. See Fig. 5.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
32×32	10	8.39e−3		4.72e−2	
64×64	12	2.16e−3	1.96	1.77e−2	1.41
128×128	13	5.80e−4	1.89	8.87e−3	1.00
256×256	15	1.57e−4	1.89	3.46e−3	1.36
512×512	16	4.17e−5	1.91	1.66e−3	1.06
1024×1024	18	1.14e−5	1.86	9.77e−4	0.76

Table 12
Example 10. Interior Robin problem, $\sigma(x, y) = 2.5 + .25x\sin(3\theta)$. $\beta(x, y) = 1$. See Fig. 6.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
32×32	16	5.05e−3		1.99e−2	
64×64	19	9.97e−4	2.34	5.49e−3	1.86
128×128	21	3.66e−4	1.44	2.89e−3	0.92
256×256	23	9.18e−5	2.00	1.27e−3	1.19
512×512	26	2.64e−5	1.80	9.38e−4	0.44
1024×1024	28	6.66e−6	1.99	5.48e−4	0.78

Table 13
Example 11. Exterior Dirichlet problem. $\beta(x, y) = 1$. See Fig. 7.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
32×32	14	5.48e−3		2.38e−2	
64×64	16	1.55e−3	1.82	7.13e−3	1.74
128×128	18	4.07e−4	1.93	2.07e−3	1.78
256×256	21	1.13e−4	1.85	6.29e−4	1.72
512×512	24	3.04e−5	1.89	1.80e−4	1.80

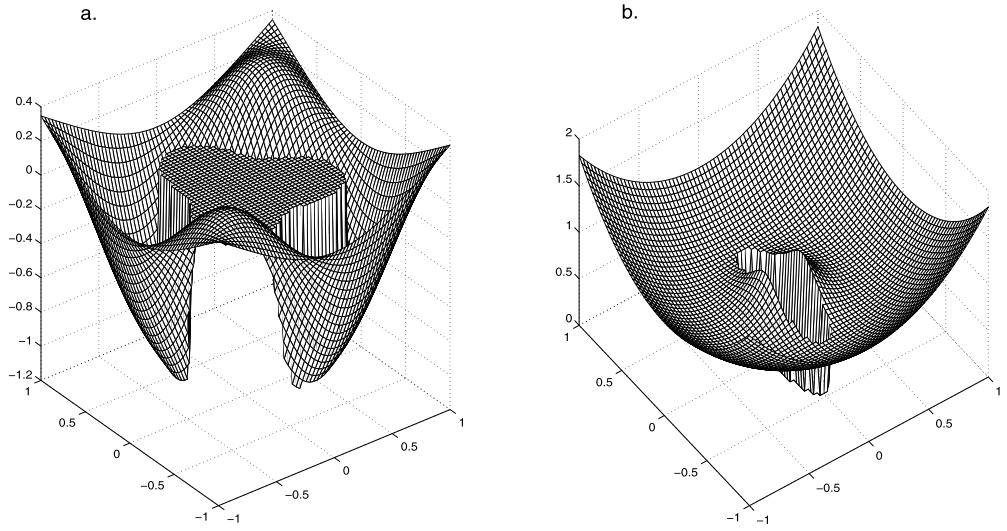

Fig. 7. Solution to Example 11 (left plot, exterior Dirichlet problem), and Example 12 (right plot, exterior Neumann problem). Example 12 is borrowed from [22], specifically Case 1 of Example 6.2, with $a = 0.5$, $b = 0.15$. [22] uses an explicit representation of the interface for this example, which generally gives more accurate results than an implicit representation. Our scheme assumes an implicit representation. Comparing our Table 14 with Table 6.2 of [22], one can see that our results are less accurate but require fewer calls to the Poisson solver (13 versus 20 for a 320×320 grid).

Table 14
Example 12. Exterior Neumann problem. $\beta(x, y) = 1$. See Fig. 7.

Spatial grid	it	E_u	Rate	$E_{\nabla u}$	Rate
40×40	9	1.26e−2		9.62e−2	
80×80	11	2.99e−3	2.07	3.97e−2	1.28
160×160	11	6.72e−4	2.15	1.29e−2	1.62
320×320	13	1.38e−4	2.29	3.84e−3	1.75
640×640	14	3.02e−5	2.19	1.23e−3	1.64

fifth order WENO with third order Runge–Kutta. A signed distance level set function may have shocks or kinks even when the boundary is smooth. The exact level set function is smooth in some neighborhood of the smooth boundary, but if the boundary is not well-resolved by the mesh the singularities will be within a few mesh widths of the boundary, and thus the normal vector can be mis-evaluated in this case also. We have not investigated this error source in isolation, but it is likely to be a major contributor to the overall error in cases where the mesh is not fine enough to adequately resolve the boundary geometry.

The linear algebra processing required by the algorithms is elementary. For interior problems the FFT-based Poisson solver is simplified by the fact that the (zero-extended) solution is periodic. For example, on a 2.26 GHz dual core processor, a run of the interior Dirichlet algorithm on a 256×256 grid requiring 20 iterations takes approximately 2.5 seconds (wall clock time), including all of the problem setup. For a 512×512 grid with 20 iterations, it takes less than 10 seconds. Exterior problems are somewhat more complicated, due to the generally nonperiodic boundary conditions on $\partial\Omega$. But the processing for the boundary condition (on $\partial\Omega$) only needs to be performed on the first iteration, and one still gets the benefit of the FFT.

Acknowledgements

I thank the anonymous referees for their careful reading of the manuscript, and their thoughtful comments and suggestions.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] T. Askham, A. Cerfon, An adaptive fast multipole accelerated Poisson solver for complex geometries, J. Comput. Phys. 344 (2017) 1–22.
[2] T. Aslam, A partial differential equation approach to multidimensional extrapolation, J. Comput. Phys. 193 (2003) 349–355.
[3] T. Aslam, S. Luo, H. Zhao, A static PDE approach to multi-dimensional extrapolations using fast sweeping methods, SIAM J. Sci. Comput. 36 (2014) A2907–A2928.
[4] J. Beale, M. Lai, A method for computing nearly singular integrals, SIAM J. Numer. Anal. 38 (2001) 1902–1925.
[5] J. Beale, A. Layton, On the accuracy of finite difference methods for elliptic problems with interfaces, Commun. Appl. Math. Comput. Sci. 1 (2006) 91–119.
[6] J. Bedrosian, J.J. von Brecht, S. Zhu, E. Sifakis, J. Teran, A second order virtual node method for elliptic problems with interfaces and irregular domains, J. Comput. Phys. 229 (2010) 6405–6426.
[7] D. Calhoun, A Cartesian grid method for solving the streamfunction-vorticity equation in irregular regions, J. Comput. Phys. 176 (2002) 231–275.
[8] D. Chopp, Some improvements of the fast marching method, SIAM J. Sci. Comput. 23 (2001) 230–244.
[9] B. Buzbee, F. Dorr, J. George, G. Golub, The direct solution of the discrete Poisson equation on irregular domains, SIAM J. Numer. Anal. 8 (1971) 722–736.
[10] F. Gibou, R. Fedkiw, A fourth order accurate discretization for the Laplace and heat equations on arbitrary domains, with applications to Stefan problems, J. Comput. Phys. 202 (2002) 577–601.
[11] F. Gibou, R. Fedkiw, K. Cheng, M. Kang, A second order accurate symmetric discretization of the Poisson equation on irregular domains, J. Comput. Phys. 176 (2003) 1–23.
[12] F. Gibou, C. Min, R. Fedkiw, High resolution sharp computational methods for elliptic and parabolic problems in complex geometries, J. Sci. Comput. 54 (2013) 369–413.
[13] A. Guittet, M. Lepilliez, S. Tanguy, F. Gibou, Solving elliptic problems with discontinuities on irregular domains — The Voronoi interface method, J. Comput. Phys. 298 (2015) 747–765.
[14] K. Ito, K. Kunisch, Z. Li, Level-set function approach to an inverse interface problem, Inverse Probl. 17 (2001) 1225–1242.
[15] H. Johansen, P. Colella, A Cartesian grid embedded boundary method for Poisson's equation on irregular domains, J. Comput. Phys. 147 (1998) 60–85.
[16] Z. Jomaa, C. Macaskill, The embedded finite difference method for the Poisson equation in a domain with an irregular boundary and Dirichlet boundary conditions, J. Comput. Phys. 202 (2005) 488–506.
[17] Z. Jomaa, C. Macaskill, Numerical solution of the 2d Poisson equation on an irregular domain with Robin boundary conditions, ANZIAM J. 50 (2008) C413–C428.
[18] C. Kublik, N.M. Tanushev, R. Tsai, An implicit interface boundary integral method for Poisson's equation on arbitrary domains, J. Comput. Phys. 247 (2013) 269–311.
[19] R. Leveque, Z. Li, The immersed interface method for elliptic equations with discontinuous coefficients and singular sources, SIAM J. Numer. Anal. 31 (1994) 1019–1044.
[20] R. LeVeque, Z. Li, Immersed interface methods for Stokes flow with elastic boundaries or surface tension, SIAM J. Sci. Comput. 18 (1997) 709–735.
[21] Z. Li, A fast iterative algorithm for elliptic interface problems, SIAM J. Numer. Anal. 35 (1998) 230–254.
[22] Z. Li, K. Ito, The Immersed Interface Method, Frontiers in Applied Mathematics, SIAM, Philadelphia, PA, 2006.
[23] Z. Li, W. Wang, A fast finite difference method for solving Navier–Stokes equations on irregular domains, Commun. Math. Sci. 1 (2003) 180–196.
[24] Z. Li, H. Zhao, H. Gao, A numerical study of electro-migration voiding by evolving level set functions on a fixed cartesian grid, J. Comput. Phys. 152 (1999) 281–304.
[25] X. Liu, R. Fedkiw, M. Kang, A boundary condition capturing method for Poisson's equation on irregular domains, J. Comput. Phys. 160 (2000) 151–178.
[26] A. Marques, J. Nave, R. Rosales, A correction function method for Poisson problems with interface jump conditions, J. Comput. Phys. 230 (2011) 7567–7597.
[27] A. Mayo, The fast solution of Poisson's and the biharmonic equations on irregular regions, SIAM J. Numer. Anal. 21 (1984) 285–299.
[28] A. McKenney, L. Greengard, A. Mayo, A fast Poisson solver for complex geometries, J. Comput. Phys. 118 (1995) 348–355.
[29] C. Min, F. Gibou, A second order accurate level set method on non-graded adaptive Cartesian grids, J. Comput. Phys. 225 (2007) 300–321.
[30] S. Osher, R. Fedkiw, Level Set Methods and Dynamic Implicit Surfaces, Springer-Verlag, New York, NY, 2003.
[31] S. Osher, J. Sethian, Fronts propagating with curvature dependent speed: algorithms based on Hamilton–Jacobi formulations, J. Comput. Phys. 79 (1988) 12–49.
[32] J. Papac, F. Gibou, C. Ratsch, Efficient symmetric discretization for the Poisson, heat and Stefan-type problems with Robin boundary conditions, J. Comput. Phys. 229 (2010) 875–889.
[33] D. Russell, Z. Wang, A Cartesian grid method for modeling multiple moving objects in 2D incompressible viscous flow, J. Comput. Phys. 191 (2003) 177–205.
[34] J. Sethian, Level Set Methods and Fast Marching Methods, Cambridge University Press, Cambridge, 1999.
[35] D. Stein, R. Guy, B. Thomases, Immersed Boundary Smooth Extension (IBSE): a high-order method for solving incompressible flows in arbitrary smooth domains, J. Comput. Phys. 335 (2017) 155–178.
[36] A. Tornberg, B. Engquist, Numerical approximations of singular source terms in differential equations, J. Comput. Phys. 200 (2004) 462–488.
[37] J. Towers, Two methods for discretizing a delta function supported on a level set, J. Comput. Phys. 220 (2007) 915–931.
[38] J. Towers, Finite difference methods for approximating Heaviside functions, J. Comput. Phys. 228 (2009) 3478–3489.
[39] J. Towers, Finite difference methods for discretizing singular source terms in a Poisson interface problem, Contemp. Math. 526 (2010) 359–389.
[40] A. Wiegmann, K. Bube, The explicit-jump immersed interface method: finite difference methods for PDEs with piecewise smooth solutions, SIAM J. Numer. Anal. 37 (2000) 827–862.