An Embedded Boundary Integral Solver for the Stokes Equations [1]

George Biros∗, Lexing Ying∗, and Denis Zorin∗
∗ *Courant Institute of Mathematical Sciences, New York University, New York 10012*
Email: {biros,lexing,dzorin}@cs.nyu.edu

Version: June 2002

We present a new method for the solution of the Stokes equations. Our goal is to develop a robust and scalable methodology for two and three dimensional, moving-boundary, flow simulations. Our method is based on Anita Mayo's method for the Poisson's equation: *"The Fast Solution of Poisson's and the Biharmonic Equations on Irregular Regions", SIAM J. Num. Anal., 21 (1984), pp. 285–299.* We embed the domain in a rectangular domain, for which fast solvers are available, and we impose the boundary conditions as interface (jump) conditions on the velocities and tractions. We use an indirect boundary integral formulation for the homogeneous Stokes equations to compute the jumps. The resulting integral equations are discretized by Nyström's method. The rectangular domain problem is discretized by finite elements for a velocity-pressure formulation with equal order interpolation bilinear elements ($Q1$-$Q1$). Stabilization is used to circumvent the $\inf - \sup$ condition for the pressure space. For the integral equations, fast matrix vector multiplications are achieved via a $N \log N$ algorithm based on a block representation of the discrete integral operator, combined with (kernel independent) singular value decomposition to sparsify low-rank blocks. Our code is built on top of PETSc, an MPI based parallel linear algebra library. The regular grid solver is a Krylov method (Conjugate Residuals) combined with an optimal two-level Schwartz-preconditioner. For the integral equation we use GMRES. We have tested our algorithm on several numerical examples and we have observed optimal convergence rates.

Key Words: Stokes equations, fast solvers, integral equations, double-layer potential, fast multipole methods, embedded domain methods, immersed interface methods, fictitious domain methods, Cartesian grid methods, moving boundaries

1. INTRODUCTION

In this paper we propose a boundary integral method for the steady two-dimensional Stokes equations in irregular domains. We have developed this method as a building block for a Navier-Stokes solver for problems with moving boundaries. Most state-of-the-art methods for such problems, with the notable exception of the immersed boundary method, require unstructured meshes for local discretizations. For irregular domains, mesh generation is still a bottleneck—especially for three-dimensional problems, problems with moving boundaries and when parallel implementations are used [2]. This makes methods based on fixed Cartesian grids attractive for such problems.

For certain types of PDEs there is an alternative formulation which is based on integral equations. For example, a constant coefficient elliptic problem can be solved using

[1] This work is supported by the National Science Foundation's Knowledge and Distributed Intelligence (KDI) program through grant DMS-9980069.

a boundary integral formulation. This approach is ideal for exterior problems, it alleviates the need for unstructured meshes, and it dramatically reduces the size of the discrete problem (reduction of dimensionality).

Several researchers have used boundary integral formulations to solve the homogeneous Stokes problem. The basic formulation can be found in [21, 35, 36]. In [12, 27, 34], the homogeneous Stokes problem is solved using boundary integral representation combined with multipole-like far-field expansions to accelerate the matrix-vector multiplications. In [33, 37, 41] boundary integral equations have been used for problems with moving and deforming boundaries. In [14] the Stokes problem is posed as a biharmonic equation and it is solved for both interior and exterior problems.

Despite its effectiveness, a boundary integral formulation becomes less attractive for problems with distributed forces. If a boundary formulation is used then it is necessary to compute domain convolutions of the forcing term with the fundamental solution. These integrals are also known as *Newton potentials*. While fast multipole methods (FMM), [15], can be used to accelerate the integration, the integrals are quite difficult to evaluate accurately for points close to the boundary, as the kernels become nearly singular.

An alternative method which originally appeared in Anita Mayo's work [28], for the Laplace and biharmonic operators, uses finite differences on regular grids to efficiently evaluate the contribution from the distributed forces. This approach was also used in [29] in combination with fast multipole methods for the boundary integral equation. In this paper we term this method as the Embedded Boundary Integral method (EBI). With the EBI method we embed the flow domain inside a larger simple domain, typically a regular domain for which fast and scalable solvers are available. The velocity and pressure are suitably extended to the regular domain; the original boundary becomes an interface across which, depending on the extension, the velocities and tractions are in general discontinuous. An integral formulation is used to compute a suitable extension and the inteface jumps of the velocity and its derivatives. Once this is done, Taylor expansions are used to express these jumps as a body force at regular grid points which are close to the interface. This body force, which appears in the right hand side of a the regular grid problem, we term Taylor Expansion Stencil Correction (TESC). Depending on the details of the implementation the method can be first, second, or higher order accurate. In this paper we extend this approach to the Stokes equation and present fast numerical methods for solving the boundary integral equations and the corrected equations on the regular grid. We have also extended the method to the elastodynamics and to the unsteady Navier-Stokes equations. For the latter preliminary results can be found in [5].

There is a great amount of work on fast solvers for PDE's in irregular geometries. Research on this topic dates back to the seventies [6]. In this paper we focus our attention to methods which are closely related to EBI.

Most of the fundamental ideas that we will discuss below, the connection between immersed interfaces potential theory and integral equations, the interpolation based approximations of jumps, the stencil modification around the boundary, and the utilization of regular grids, go back to the capacitance matrix method [38]. Neumann and Dirichlet problems are solved for the Laplace and Helmholtz problems using domain embedding and finite differences. The stencils that cross the interface are modified and the resulting matrix is written as a sum of the standard five-point Laplace operator and of a low rank modification. This matrix can be inverted by the Sherman-Morrison-Woodbury formula. Instead the authors solve for the jump conditions first (the discrete potential). For the Neumann problem the two approaches are equivalent but for the Dirichlet problem are different since the double-layer approximation results in well conditioned problems for the unknown interface jumps. One shortcoming of the method is the requirement of several regular grid

solves.

One of the most successful techniques is the *Immersed Boundary Method* [31, 32] which was designed for a Poisson problem for a solve for the pressure within a projection algorithm for the unsteady Navier-Stokes equations. The interface is modeled as a set of one-dimensional delta functions whose discretization gives a forcing term. The method is first-order accurate due to smearing of the boundary layers by the discrete delta functions.

The *Immersed Interface Method* [23] is an extension of the immersed boundary method which is second-order accurate. It is designed for problems with discontinuous coefficients and singular forces. It has been successfully applied to moving boundary problems, for example for the Stokes problem with elastic interfaces [24] and for the Navier-Stokes problem [26]. If the singular forces are known then the jumps are known and TESCs can be computed explicitly. For discontinuous coefficients IIM modifies the stencils for points close to the boundary in order to account for the jump conditions. The method results in non-standard matrices and fast methods are not straightforward to apply. The immersed interface method as presented in [23] was not used on Dirichlet on Neumann problems in general irregular regions, since it requires known jump conditions. In [9], IIM is extended to Neumann problems by modifying interface stencils to account for the unknown jumps. Later versions of IIM, (Explicit Immersed Interface Method) [40], (Fast Immersed Interface Method) [25], addressed non-standard matrices by adding additional equations for the jumps and extended IIM to Dirichlet problems; this approach however appears to result in considerable additional computational cost since it requires tens to hundreds of regular grid solves.

Several other methods produce discretizations based on regular grids, with modified stencils and/or right-hand sides to account for the embedded interfaces. Cheng and Fedkiw [7] describe a second order method for the Dirichlet boundary problem. This method results in symmetric positive definite matrices with diagonally-modified stencils and with additional terms on the right hand side. In another class of methods, *Cartesian-finite volume methods*, the stencils modifications stem from appropriate modification of finite volume cells to account for the intersections of the Cartesian grids with the interface [1], [18].

Another algorithm, similar to the IIM and capacitance matrix methods but which first appeared within the finite element community, is the *fictitious domain* method [8, 11]. Based on a finite element variational formulation, Dirichlet boundary conditions are imposed weakly as side constraints. This approach results in a saddle-point problem that includes the primitive variables plus Lagrange multipliers. In fact certain fictitious domain methods are intimately related to the IIM and EBI methods. It can be shown that the Lagrange multipliers correspond to Neumann condition jumps.

The above methods share some common features. Here we restrict our attention to problems with constant coefficients and force singularities which cause interface jumps. When these jumps are *a priori* known, the stencil modifications can be transferred to the right hand side using TESC. However this is almost never the case. In general, inteface discontinuities have to be solved for. One approach is to modify the stencils of the discretization close to the interface (Cartesian grids, immersed interface method, Cheng and Fedkiw method), or to introduce additional equations (fictitious domain, immersed boundary, fast immersed interface, explicit immersed interface). Modified stencils make it more difficult to apply fast solvers, especially if the boundaries are moving. If additional unknowns are used, a common approach is to invert the Schur complement corresponding to these unknowns. These Schur complement matrices correspond to discretizations of integral equations [38]. A matrix-vector multiplication with such matrix will be expensive since it involves a regular grid solve.

Computing the jumps directly via boundary integrals, which is the foundation of the

EBI method, circumvents such costly operations by decoupling the inteface with the background grid. Only one integral solve and two regular grid solves are required independently of the complexity of the immersed interface.

In addition, transferring the equations on the interface results in a natural formulation for coupled problems. For example in fluid-solid interaction problems, the interface conditions are the continuity of the tractions and of the velocities; these conditions can augment boundary integral formulations for the solid and fluid. If an implicit method is used to solve the equations, in this formulation the nonlinear iterations can be restricted on the interface. While time dependent problems require volume computations, a fast solver on a structured grid helps to minimize the computational cost of the volume discretization. These considerations indicate that the EBI-based methods may have advantages for such problems which we plan to explore in the future.

One shortcoming of the EBI method is that it can be used only for problems with a domain that can be partitioned to subdomains in which the fundamental solution is known. The latter, however, does include problems with piecewise constant coefficients, and thus EBI is suitable for a quite large class of problems. Another problem is that EBI is relatively complicated to implemented because, for scalable and efficient implementations, a fast matrix multiplication algorithm for the integral equation must be used. The details of such an implementation (for example FMM) tend to depend on the underlying kernel. In this paper we discuss an efficient $N \log N$ algorithm that can be used with any kernel with rapid decay properties, and only requires kernel evaluations.

In the next section we present the overview of the method. Subsequent sections address the details of the boundary integral formulation (Section 3); discretizations of the boundary integral equations (Section 4.1), regular domain equations (Section 4.2) and Taylor expansion stencil corrections (Section 4.3). Section 5 discusses the implementation of the method, a fast SVD-based solver for the boundary integral equation in particular.

Notation. Scalars will be denoted with lowercase italics, vectors with lowercase boldface letters; tensors and matrices will be denoted with uppercase boldface letters. Infinitely dimensional quantities will be in italics, whereas finite dimensional ones (usually discretizations) will be non-italic fonts. We use $[[\cdot]]$ to denote the jump of a function across an interface (exterior $-$ interior).

2. HIGH LEVEL DESCRIPTION OF THE EBI METHOD

We seek solutions for the interior, possibly multiply-connected, Stokes problem with Dirichlet boundary conditions. We choose a primitive variable formulation (velocities and pressures), for which the momentum and mass conservation laws are given by

$$-\nu\Delta\boldsymbol{u} + \nabla p = \boldsymbol{b} \quad \text{in} \quad \omega, \quad \operatorname{div}\boldsymbol{u} = 0 \quad \text{in} \quad \omega, \quad \boldsymbol{u} = \boldsymbol{g} \quad \text{on} \quad \gamma. \tag{1}$$

Here $\boldsymbol{u}$ is the velocity field, p is the pressure, $\boldsymbol{b}$ is a known forcing term, and $\boldsymbol{g}$ is a given Dirichlet boundary condition for the velocity. The stress tensor $\boldsymbol{S}$ associated with the velocity and pressure is given by

$$\boldsymbol{S} = -p\mathbf{I} + \nu(\nabla u + \nabla u^T). \tag{2}$$

We split the solution of the problem into several steps as follows. We first embed ω in an easy-to-discretize domain Ω, typically a rectangle. By linearity we decompose (1) into two problems: one problem that has an inhomogeneous body force and zero boundary conditions for Ω; the other has no body force, but nontrivial boundary conditions:

$$-\nu\Delta \boldsymbol{u}_1 + \nabla p_1 = \boldsymbol{b} \quad \text{in} \quad \Omega, \qquad \operatorname{div} \boldsymbol{u}_1 = 0 \quad \text{in} \quad \Omega, \qquad \boldsymbol{u}_1 = \boldsymbol{0} \quad \text{on} \quad \Gamma; \tag{3}$$

$$-\nu\Delta \boldsymbol{u}_2 + \nabla p_2 = \boldsymbol{0} \quad \text{in} \quad \omega, \qquad \operatorname{div} \boldsymbol{u}_2 = 0 \quad \text{in} \quad \omega, \qquad \boldsymbol{u}_2 = \boldsymbol{g} - \boldsymbol{u}_1 \quad \text{on} \quad \gamma. \tag{4}$$

Domain Ω is chosen to make the fast solution of (3) possible (Section 4.2). For (4) we use a double layer boundary integral formulation (Section 3) to obtain the velocity potential, $\boldsymbol{\mu}$, on the boundary γ. Solution $\boldsymbol{u}_2$ for an arbitrary point in the interior of ω is the convolution of the double layer kernels with the velocity potential. The solution of the original problem (1) is $\boldsymbol{u} = \boldsymbol{u}_1 + \boldsymbol{u}_2$, $p = p_1 + p_2$.

In practice however, evaluating $\boldsymbol{u}_2$ using convolution presents the same difficulties as the evaluation of a forcing term by convolution. A different approach proposed by Mayo [28], is to use the fact that once problems (3) and (4) are solved, the jumps of the solution $\boldsymbol{u}$ can be very accurately computed on γ. Conceptually, there is a discontinuous extension $\boldsymbol{u}_3$ of $\boldsymbol{u}_2$ on Ω that satisfies

$$-\nu\Delta \boldsymbol{u}_3 + \nabla p_3 = 0 \quad \text{in} \quad \Omega, \quad \operatorname{div} \boldsymbol{u}_3 = 0, \quad \text{in} \quad \Omega, \quad \boldsymbol{u}_3 = \boldsymbol{u}_2, \quad \text{on} \quad \Gamma, \tag{5}$$

$$[[\boldsymbol{u}_3]]_\gamma = \boldsymbol{\mu}, \quad [[\boldsymbol{S}_3 \boldsymbol{n}]]_\gamma = [[-p_3 \boldsymbol{n} + \nu(\nabla \boldsymbol{u}_3 + \nabla \boldsymbol{u}_3^T)\boldsymbol{n}]]_\gamma = 0. \tag{6}$$

Numerically, this problem is solved using the same solver used for problem (3), but with a right-hand side that takes into account the interface jumps computed from the velocity potential (Section 4.3).

In summary, the EBI approach uses the following steps:

1. solve the problem (3) on the simpler domain Ω;
2. solve the boundary integral problem derived from (4) on γ to obtain the velocity potential;
3. compute the right-hand side corrections from the velocity potential;
4. solve the second regular problem on Ω with the computed right-hand side;
5. add the solutions obtained at steps 1 and 4 to obtain the complete solution on ω.

3. THE DOUBLE LAYER FORMULATION FOR THE STOKES EQUATIONS

In this section we describe the double layer intergral formulation of a problem of the form (4). We assume that the boundary curve γ is curvature-continuous, and the domain ω is bounded. We use the notation

$$\mathcal{C}[w](x) := \int_\gamma \mathcal{C}(\mathbf{x}, \mathbf{y}) \boldsymbol{w}(\mathbf{y})\, d\gamma(\mathbf{y}),$$

to denote the convolution for a kernel $\mathcal{C}$; $\mathcal{C}(\mathbf{x}, \mathbf{y})\boldsymbol{w}$ is a dot product for vector kernels and matrix-vector product for matrix kernels.

The fundamental solution for the Stokes operator in two dimensions it is given by:

$$\mathcal{U}(\mathbf{x}, \mathbf{y}) = \mathcal{U}(\mathbf{r}) := \frac{1}{4\pi} \left(\ln \frac{1}{\rho} + \frac{\mathbf{r} \otimes \mathbf{r}}{\rho^2} \right), \tag{7}$$

$\mathbf{x}$ is the observation point, $\mathbf{y}$ is the source point, $\mathbf{r} := \mathbf{x} - \mathbf{y}$, $\rho := \|\mathbf{r}\|_2$, and $\otimes$ is the tensor product of two vectors. This kernel is also known as the Stokeslet.

Similar to the potential theory for the Laplace equation we can introduce single and double surface potentials for the velocity and the pressure. We use only the double layer potential $\mathcal{D}$ for velocity and the double layer potential for pressure $\mathcal{K}$:

$$\mathcal{D}(\mathbf{x},\mathbf{y}) := \frac{1}{\pi}\frac{\mathbf{r}\cdot\boldsymbol{n}(\mathbf{y})}{\rho^2}\,\frac{\mathbf{r}\otimes\mathbf{r}}{\rho^2}; \quad \mathcal{K}(\mathbf{x},\mathbf{y}) := -\nu\frac{1}{\pi}\frac{1}{\rho^2}\left(\mathbf{I}-2\frac{\mathbf{r}\otimes\mathbf{r}}{\rho^2}\right)\boldsymbol{n}(\mathbf{y}) \tag{8}$$

where $\boldsymbol{n}(\mathbf{y})$ is the outward surface normal at a boundary point $\mathbf{y}$. For a derivation of [35] and [36].

Green's second identity can be employed to express the solution of the Stokes problem in terms of boundary integrals and thus reduce the problem to a boundary integral equation. While being most general, this approach results in ill-conditioned systems. We use an indirect formulation which yields systems with bounded condition number.

We limit our discussion to the interior Dirichlet problem. The extension to exterior and Neumann problems is straightforward. We represent the velocities and pressures as surface potentials convoluted by the double layer kernel:

$$\boldsymbol{u}(\mathbf{x}) = \mathcal{D}[\boldsymbol{\mu}](\mathbf{x}), \quad p(\mathbf{x}) = \mathcal{K}[\boldsymbol{\mu}](\mathbf{x}), \quad \mathbf{x} \quad \text{in} \quad \omega. \tag{9}$$

Here $\boldsymbol{\mu}$ is the hydrodynamic potential. Taking limits to the boundary from the interior and exterior regions we obtain

$$\boldsymbol{u}(\mathbf{x}) = -\frac{1}{2}\boldsymbol{\mu}(\mathbf{x}) + \mathcal{D}[\boldsymbol{\mu}](\mathbf{x}), \quad \mathbf{x} \quad \text{on} \quad \gamma. \tag{10}$$

The velocity $\boldsymbol{u}$ has to satisfy $\int_\gamma \boldsymbol{u}\cdot\boldsymbol{n}\,d\gamma = 0$, a direct consequence of the conservation of mass. This constraint is an indication that for the simply-connected interior problem the double layer operator has a null space of dimension at least one. In fact, it can be shown ([35], p. 159) that the dimension of the null space is exactly one. The null space can be removed by a rank-one modification ([35], p. 615). Let $\mathcal{N}(\mathbf{x},\mathbf{y}) = \boldsymbol{n}(\mathbf{x})\otimes\boldsymbol{n}(\mathbf{y})$. We represent $\boldsymbol{u}$ as

$$\boldsymbol{u}(\mathbf{x}) = -\frac{1}{2}\boldsymbol{\mu}(\mathbf{x}) + \mathcal{D}[\boldsymbol{\mu}](\mathbf{x}) + \mathcal{N}[\boldsymbol{\mu}](\mathbf{x}), \quad \mathbf{x} \quad \text{on} \quad \gamma. \tag{11}$$

More generally, for the multiply connected interior problem, a direct calculation can verify that the double layer kernel has a larger null space: it is spanned by potentials that correspond to restrictions of rigid body motion velocity fields on the boundary. These fields generate zero boundary tractions and thus belong to the null space of the double layer kernel. Suppose that the boundary γ consists of $n+1$ components $\gamma_0,\gamma_1,\cdots,\gamma_n$, where γ_0 encloses all other components, and let $\mathbf{c}_p, p = 1,\cdots,n$ be an interior point of γ_p. Following [35], we represent $\boldsymbol{u}$ as

$$\boldsymbol{u}(\mathbf{x}) = -\frac{1}{2}\boldsymbol{\mu}(\mathbf{x}) + \mathcal{D}[\boldsymbol{\mu}](\mathbf{x}) + \mathcal{N}[\boldsymbol{\mu}](\mathbf{x}) + \sum_{p=1}^{n}\mathcal{U}(\mathbf{r}_p)\boldsymbol{\alpha}_p[\boldsymbol{\mu}] + \sum_{p=1}^{n}\mathcal{R}(\mathbf{r}_p)\beta_p[\boldsymbol{\mu}] \tag{12}$$

where $\mathbf{r}_p = \mathbf{x}-\mathbf{c}_p$, $\mathcal{R}(\mathbf{r}) = \mathbf{r}^\perp/4\pi\rho^2$, and if $\mathbf{r} = (r_1,r_2)$, $\mathbf{r}^\perp = (r_2,-r_1)$.

The coefficients $\boldsymbol{\alpha}_p$ and β_p are computed by augmenting (12) with

$$\int_\gamma \boldsymbol{\psi}_p^j(\mathbf{y})\cdot\boldsymbol{\mu}(\mathbf{y})\,d\gamma(\mathbf{y}) = \boldsymbol{\alpha}_p,\ j = 1,2,$$
$$\int_\gamma \boldsymbol{\psi}_p^3(\mathbf{y})\cdot\boldsymbol{\mu}(\mathbf{y})\,d\gamma(\mathbf{y}) = \beta_p, \tag{13}$$

where $\boldsymbol{\psi}_p^i, p = 1, \cdots, n, i = 1, 2, 3$ are $3n$ functions spanning the null space of the double layer potential. These functions are explicitly known $\boldsymbol{\psi}_p^i(\mathbf{y}) = (\delta_{1i}, \delta_{2i})$ for $i = 1, 2$ $\boldsymbol{\psi}_p^3(\mathbf{y}) = (y_2, -y_1)$ on γ_p—they are fluid rigid-body motions, restricted on γ_p. In [35] is shown that equations (12) and (13) guarantee a unique solution of $\boldsymbol{\mu}, \boldsymbol{\alpha}$ and β for general admissible boundary condition $\boldsymbol{u}$.

Jump computation. Once the potenital $\boldsymbol{\mu}$ is known, we need to compute the jumps at the interface and the velocity to use in equation (5) Equation (9) is defined for points inside ω. We can use exactly the same relation to extend $\boldsymbol{u}$ in $\mathbb{R}^2/\bar{\omega}$. The resulting field is discontinuous across the interface.

From the properties of the double layer kernel for an interior problem we have the following jump relations for velocity and stress:

$$[[\boldsymbol{u}]] = \boldsymbol{\mu}, \quad [[\boldsymbol{S}\boldsymbol{n}]] = 0. \tag{14}$$

The jump on the pressure can be deduced if we notice that the double layer kernel $\mathcal{K}(\mathbf{x}, \mathbf{y})$ can be also written as $-2\nu\nabla_{\mathbf{x}}(\mathcal{L}(\mathbf{x}, \mathbf{y}))$, where $\mathcal{L}(\mathbf{x}, \mathbf{y}) = (\mathbf{r} \cdot \boldsymbol{n}(\mathbf{y}))/\rho^2$ is the double layer kernel for the Laplace equation.

From (14) we can derive a condition for the pressure:

$$[[p]] = -2\boldsymbol{\mu} \cdot \boldsymbol{t}, \tag{15}$$

where $\boldsymbol{t}$ is the curve tangent.

In addition to jumps in velocity and pressure, we also need the jumps for derivatives of velocity and pressure as well as the jumps in second derivatives of the velocity; these jumps are used to compute corrections to ensure second-order accuracy of the solution of the problem on the domain Ω. The derivation is presented in the appendix.

4. DISCRETIZATION

4.1. Boundary Integral Equation

We discretize (11) by the Nyström method combined the composite trapezoidal rule which achieve superalgebraic convergence for smooth data. Without loss of generality we assume ω to be simply connected. Note that the double layer kernel has no singularities for points on the boundary. Indeed,

$$\lim_{\mathbf{y}\to\mathbf{x}} \mathcal{D}(\mathbf{x}, \mathbf{y}) = -\frac{1}{\pi}(\boldsymbol{t} \otimes \boldsymbol{t})\frac{k}{2}, \quad \mathbf{x}, \mathbf{y} \quad \text{on} \quad \gamma,$$

where $\boldsymbol{t}$ and k are the tangent vector and the curvature at $\mathbf{x}$.

Let $[0, 2\pi]$ be the curve parameterization space and n the number of discretization points with $h = 2\pi/n$. We discretize by:

$$\begin{aligned}\boldsymbol{u}(\mathbf{y}(ih)) = -0.5\boldsymbol{\mu}(ih) + \frac{1}{h}\sum_{j=1}^{n} \mathcal{D}(\mathbf{y}(ih), \mathbf{y}(jh))\boldsymbol{\mu}(\mathbf{y}(jh)) \, \|\nabla\mathbf{y}(jh)\|_2 \\ + \boldsymbol{n}(\mathbf{y}(ih)) \sum_{j=1}^{n} \boldsymbol{\mu}(\mathbf{y}(jh)) \cdot \boldsymbol{n}(\mathbf{y}(jh) \, \|\nabla\mathbf{y}(jh)\|_2, \quad i = 1, \ldots, n,\end{aligned}$$

or

$$\mathbf{u}_i = -0.5\boldsymbol{\mu}_i + \frac{1}{h}\sum_{j=1}^{n} \mathbf{D}_{ij}\boldsymbol{\mu}_j \, \|\nabla \mathbf{y}_j\|_2 + \mathbf{n}_i \sum_{j=1}^{n} \boldsymbol{\mu}_j \cdot \mathbf{n}_j \, \|\nabla \mathbf{y}_j\|_2, \quad i = 1, \ldots, n \quad (16)$$

which results in a dense $2n \times 2n$ linear system. Here $\mathbf{y}(\cdot)$ is the parameterization of γ.

While resulting system has a bounded condition number, it is dense. Fortunately, one can take advantage of the fast decay of the Green's function with distance and use a fast method to solve the system. A number of such methods exist; we use an SVD-based method described in detail in Section 5.1.

4.2. Finite element formulation of the regular region

To solve the equations in the regular domain Ω we use a finite-element discretization of the Stokes operator. It should be noted that we use the finite-element formulation primarily as a convenient mechanism to derive the discretization of the problem. For the regular grid the discrete system obtained by using finite elements is identical to a system obtained by a specific choice of finite difference stencils to which we can apply the right-hand side corrections described in Section 4.3.

We have chosen to solve for the velocity and pressure simultaneously rather than use an Uzawa or pressure correction algorithm using a finite element method with $Q1$-$Q1$ bilinear elements.The advantage of the $Q1$-$Q1$ elements is that they probably result in one of the simplest implementations for the Stokes system since they allow equal order interpolation for the velocity and the pressure on a unstaggered grid [2]. A survey and related references on finite element methods for the Navier-Stokes equations can be found in [16], and [17].

With $L^2(\Omega)$ we denote the space of scalar functions (in Ω) which are square-integrable and with $\boldsymbol{H}^1(\Omega)$ we denote vector functions whose first derivatives are in $\boldsymbol{L}^2(\Omega)$.

We also define

$$\begin{aligned} \boldsymbol{V} &:= \left\{ \boldsymbol{v} \in \boldsymbol{H}^1(\Omega) : \boldsymbol{v}|_\Gamma = \mathbf{0} \right\}, \\ Q &:= \left\{ \phi \in L^2(\Omega) : \int_\Omega \phi \, d\Omega = 0 \right\}. \end{aligned}$$

The domain integral constraint in Q is necessary for pressure uniqueness (for Dirichlet problems pressure is defined up to a constant). It can be implemented by a null space correction within Krylov iterations or by setting the pressure datum at a boundary discretization node. We choose the former since it results in better conditioned linear systems.

In the weak formulation of (1) we seek $\boldsymbol{u} \in \boldsymbol{H}^1(\Omega)$ and $p \in Q$ such that:

$$\int_\Omega \nu \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{v} \, d\Omega - \int_\Omega p \, \operatorname{div} \boldsymbol{v} \, d\Omega - \int_\Omega \boldsymbol{b} \cdot \boldsymbol{v} \, d\Omega = \mathbf{0} \quad \forall \boldsymbol{v} \in \boldsymbol{V}, \quad (17)$$

$$- \int_\Omega q \, \operatorname{div} \boldsymbol{u} \, d\Omega - \beta \, h^2 \int_\Omega \nabla p \cdot \nabla q \, d\Omega = 0 \quad \forall q \in Q. \quad (18)$$

In unconstrained elliptic systems like the Laplace and elasticity equations mere inclusion of the finite element spaces within the continuum spaces suffices for convergence. However, this is not the case for the Stokes equations and the choice of the pressure approximation function space cannot be independent of the choice for the velocities [16].

[2] $P1 - P1$ could also have been used, but the implementation is somewhat more sensitive on the stabilization parameter [30].

To ensure convergence, the well-known (inf-sup condition) needs to be satisfied, which is not satisfied by he $Q1$-$Q1$ element. The weighted diffusive term in (18) is introduced to circumvent the inf-sup condition [16]; parameter β controls the amount of stabilization. In [30] it is shown how to choose an optimal value for β; for regular domains and periodic boundary conditions $\beta = 1/24$. The resulting approximation is second-order accurate for the velocities and first order accurate for the pressures in the L^2 norm.

The resulting discrete system is where $\mathbf{U}$ is the Laplacian with Dirichlet boundary conditions, $\mathbf{V}$ is the Laplacian with homogeneous Neumann boundary conditions (since the pressure is unknown on the boundary).

$$\begin{bmatrix} \mathbf{U} & \mathbf{0} & \mathbf{B}_{1p} \\ \mathbf{0} & \mathbf{U} & \mathbf{B}_{2p}^T \\ \mathbf{B}_{p1} & \mathbf{B}_{p2} & -\beta h^2 \mathbf{V} \end{bmatrix} \begin{Bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{p} \end{Bmatrix} = \begin{Bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{0} \end{Bmatrix}. \tag{19}$$

The corresponding finite difference stencils are provided in the appendix.

We use this discretization to solve all equations on the rectangular domain Ω.When solving the system (5) we apply corrections computed as described in Section 4.3 to the right-hand side of the system, which ensures second-order convergence. The derivation of these corrections is based on the standard finite difference analysis, assuming sufficient smoothness of the solution. Although the discretization we use is derived using finite elements, truncation error can be easily shown to be second order accurate for (3). However standard maximum principle techniques cannot be used for the Stokes equations, because they correspond to an indefinite and thus not coercive operator. For this reason we use FEM theory to obtain an error estimate in the L^2 norm.

Given $\boldsymbol{f}$ in $\boldsymbol{H}^{-1}(\Omega)$, and g in $L^2(\Omega)$ for the stabilized Q1-Q1 formulation we know that the following problem has a unique solution. Find $\boldsymbol{u}_h \in \boldsymbol{V_h}$, $p \in Q_h$ such that:

$$\int_\Omega \nu \nabla \boldsymbol{u}_h \cdot \nabla \boldsymbol{v}_h \, d\Omega - \int_\Omega p_h \, \operatorname{div} \boldsymbol{v}_h \, d\Omega = \int_\Omega \boldsymbol{f}_h \cdot \boldsymbol{v}_h \, d\Omega \quad \forall \boldsymbol{v}_h \in \boldsymbol{V_h}, \tag{20}$$

$$-\int_\Omega q_h \, \operatorname{div} \boldsymbol{u}_h \, d\Omega - \beta \, h^2 \int_\Omega \nabla p_h \cdot \nabla q_h \, d\Omega = \int_\Omega g_h \, q_h \, d\Omega \quad \forall q_h \in Q_h. \tag{21}$$

If we denote $\| \cdot \|_m$ the usual norm in $\boldsymbol{H}^m(\Omega)$, standard regularity results [10] give

$$\|\boldsymbol{u}_h\|_1 + \|p_h\|_0 \leq c(\|\boldsymbol{f}_h\|_{-1} + \|g_h\|_0),$$

or (since $\| \cdot \|_{-1} \leq \| \cdot \|_0 \leq \| \cdot \|_1$)

$$\|\boldsymbol{u}_h\|_0 + \|p_h\|_0 \leq c(\|\boldsymbol{f}_h\|_0 + \|g_h\|_0). \tag{22}$$

Now let $w_h = \{\boldsymbol{u}_h, p_h\}$ and $b_h = \{\boldsymbol{f}_h, g_h\}$. We can associate a linear operator $\mathcal{A}_h$ to (20), mapping w_h to b_h; since (20) has a unique solution for all b_h, we can also write $w_h = \mathcal{A}_h^{-1} b_h$. The regularity condition (22) implies that $\|w_h\|_0 \leq \|b_h\|_0$ and thus $\|A_h^{-1}\|_0 \leq \infty$. Then if e_h is the approximation error and τ_h the truncation error, we get $e_h = \mathcal{A}_h^{-1} \tau_h$, or $\|e_h\|_0 \leq \|\mathcal{A}_h^{-1}\|_0 \|\tau_h\|_0$. If we assume that $\|\tau_h\|_0$ is $\mathcal{O}(h^2)$ we obtain $\|e_h\|_0 = \mathcal{O}(h^2)$. In our numerical experiments we have observed a similar convergence rate in the infinity norm.

4.3. Taylor Expansion Stencil Corrections

In this section we show how discontinuities across the interface (jumps) can be used as a correction term for a discretization obtained using a simpler domain in which the interface is embedded.

The derivation of the basic formulas for the corrections does not depend on the problem, as long as the jumps of the variables across the interface can be computed.

To illustrate the basic idea, suppose we solve Poisson's equation $\Delta u = b$, in ω with given Dirichlet boundary conditions on γ (Fig. 4.3). Assume further that we use a discontinuous extension of u in Ω which satisfies the same equation outside ω. We assume that the discontinuities are known up to second derivatives. Typical discretizations of elliptic PDE's (finite elements, finite differences or finite volumes) produce a linear system with i-th equation of the form $\alpha u_i + \sum_j \beta_j u_j = \zeta b_i$, where j runs through the neighbors of u_i. The coefficients of the equations for regular grids are the same for all interior points, and depend only on the relative position of u_i and u_j. These coefficients together with corresponding relative displacements are usually referred to as stencils. For the standard 2D five-point discrete Laplacian (Fig. 4.3(a)) the equations have the form $(1/4)u_i - \sum_{j=1}^{4} u_j = h^2 b_i$, where h is the mesh size. In the absence of an interface

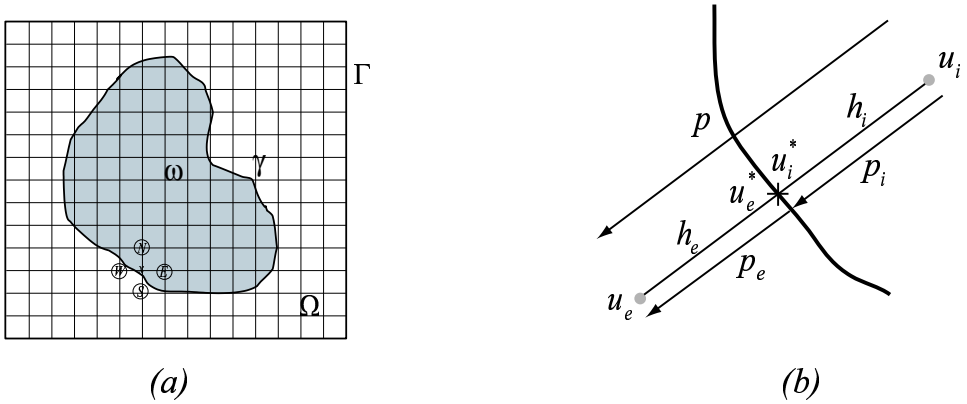


FIG. 1 Stencil corrections. (a) The irregular domain ω is embedded in a simpler domain Ω. For the depicted stencil the truncation error is constant as the discretization step decreases. Figure (b) shows the notation for computing the correction terms.

this stencil is well-defined and second order accurate. For stencils that intersect with the interface, however, this is not true, as the solution is discontinuous across the interface. In Fig. 4.3(b), we show an example for which two unknowns u_i and u_e are related in a discretization stencil that "crosses" the interface at point X. The limit of the solution from the interior is denoted as u_i^* and the limit from the exterior is denoted as u_e^*. The key idea is that the truncation error of the stencil can be corrected to be second (or higher-order) accurate if we know the difference between the interface limits, and not their exact values. Define $\boldsymbol{n} = \boldsymbol{p}/h$ to be the unit-length direction vector oriented from u_i to u_e, $p_e = h_e \boldsymbol{n}$ and $p_i = h_i \boldsymbol{n}$, Fig. 4.3(d). By using Taylor expansions we can write

$$\begin{aligned} u_e &= u_e^* + h_e \boldsymbol{D} u_e^* \cdot \boldsymbol{n} + \frac{h_e^2}{2} \boldsymbol{n} \cdot (\boldsymbol{D}^2 u_e^*) \boldsymbol{n} + \mathcal{O}(h^3) \\ &= ([[u]] + u_i^*) + h_e([[\boldsymbol{D}u]] + \boldsymbol{D}u_i^*) \cdot \boldsymbol{n} + \frac{h_e^2}{2} \boldsymbol{n} \cdot ([[\boldsymbol{D}^2 u]] + \boldsymbol{D}^2 u_i^*) \boldsymbol{n} + \mathcal{O}(h^3). \end{aligned} \tag{23}$$

Defining

$$s_i = [[u]] + h_e [[\boldsymbol{D}u]] \cdot \boldsymbol{n} + \frac{h_e^2}{2} \boldsymbol{n} \cdot [[\boldsymbol{D}^2 u]] \boldsymbol{n} \tag{24}$$

and expanding u_i^* using Taylor series at u_i we obtain

$$u_e = u_i + h\boldsymbol{D}u_i \cdot \boldsymbol{p} + \frac{h}{2}\boldsymbol{p} \cdot (\boldsymbol{D}^2 u_i)\boldsymbol{p} + s_i + \mathcal{O}(h^3).$$

Similarly we can write

$$u_i = u_e - h\boldsymbol{D}u_e \cdot \boldsymbol{p} + \frac{h^2}{2}\boldsymbol{p} \cdot (\boldsymbol{D}^2 u_e)\boldsymbol{p} + s_e + \mathcal{O}(h^3),$$

where s_e is given by:

$$s_e = -([[u]] - h_i[[\boldsymbol{D}u]] \cdot \boldsymbol{n} + \frac{h_i^2}{2}\boldsymbol{n} \cdot [[\boldsymbol{D}^2 u]]\boldsymbol{n}). \tag{25}$$

For the stencil centered at u_e we use (25) and for the stencil centered at u_i we use (24). More specifically, in the equation $\alpha u_i + \beta_e u_e + \cdots = \zeta b_i$, we replace u_e with $u_e - s_i$, which results in the correction to the right-hand side $\beta_e s_i$, and yields the desired accuracy.

By using the correction term we achieve $\mathcal{O}(h^{3/2})$ truncation error for a second order discretization of the Laplacian for the points immediate to the boundary and $\mathcal{O}(h^2)$ for the remaining set of points. This results to an $\mathcal{O}(h^2)$ discretization error for all points [23, 28]. It also implies a second order truncation error in the L^2 norm. Therefore second order convergence can be achieved using jump information up to second derivatives.

5. THE IMPLEMENTATION OF THE EBI METHOD

In this section we summarize the algorithmic components of the EBI method and we provide some implementation details.

The input data is the boundary geometry γ, the body force, and the boundary conditions. The boundary is represented as a collection of cubic B-spline curves. Solution includes the following steps.

1. Define the regular domain Ω. Its boundary Γ (Fig. 4.3) should not be too close to the boundary of the target domain, since we use (9) to evaluate the velocity; the integrals are nearly singular as we approach γ.

2. Solve the problem (3) on the rectangular domain Ω. We use standard numerical methods to solve the discretized system as discussed in Section 6. tests the forcing term is analytically known everywhere; in the general case it will be known only in the domain. We have used Shepard cubic extrapolation ([39]) to compute a smooth extension of the body force.

3. Solve of the boundary integral equation corresponding to (4) using SVD acceleration discussed in Section 5.1. This step requires the trace of a particular solution to correct the boundary conditions for $\boldsymbol{u}_2$ by setting $\boldsymbol{u}_2|_\gamma = \boldsymbol{g} - \boldsymbol{u}_1|_\gamma$. We use cubic Lagrange interpolation to compute $\boldsymbol{u}_1$ on γ. Provided that the trace is interpolated consistently to the accuracy of the FEM solution, and provided the potential calculation is higher-order accurate, the error in the boundary integral equation data ($\boldsymbol{u}_2|_\gamma$ includes the approximation error from $\boldsymbol{u}_1$) does not decrease the overall accuracy of the method.

4. Compute corrections using the potential. First we compute the intersections of γ with the regular grid, using a standard Bezier-clipping algorithm. Then, using the

velocity potential we evaluate the correction terms for the regular grid neighbors of every intersection point. Furthermore in order to compute the jumps we need to compute first and second derivatives of the double layer potential. For this purpose we use cubic spline interpolation for every curve on the boundary.

5. Solve (5) to evaluate the homogeneous solution. For this step we need to set appropriate boundary conditions on Γ. We use (9) to evaluate the boundary condition. For this step we use a dense evaluation of the boundary integral. This approach is not scalable but the constant is very small. SVD acceleration can be used.

The overall solution is given by the restriction in ω of the sum of the particular and the homogeneous solutions.

To compute the solution we need two numerical methods: one to solve the boundary integral equations and the other to solve the linear systems obtained by discretization of the Stokes equation on Ω.

5.1. Fast BIE solver using SVD

The linear systems resulting from the Nyström discretization of a double layer potential have bounded condition number. The double layer kernel is weakly singular, and thus compact for domains with C^1-boundary. Compact perturbations of the identity have bounded condition number; for such system the expected number of iterations for a Krylov method (like GMRES) will be independent of the mesh size. For example, for the unit circle the condition number is exactly 2, and it is independent of the number of discretization points. In addition, for the interior problem, there are only two eigenvalues—therefore GMRES converges in two iterations. For multiply-connected domains the condition number scales with the number of simply-connected components. In [13] an effective preconditioner is proposed; our implementation includes this preconditioner. However, as the matrix of the system is dense, each iteration is expensive and further acceleration is required for large problems.

The discretized equation(16) can be written in the vector form as:

$$\mathbf{u} = -\frac{1}{2}\mathbf{I}\boldsymbol{\mu} + \mathbf{DJW}\boldsymbol{\mu} + \mathbf{n}\mathbf{n}^T(\mathbf{JW}\boldsymbol{\mu}). \tag{26}$$

$\mathbf{u}, \boldsymbol{\mu}, \mathbf{n}$ are the vectors of boundary velocity, density and normal respectively; $\mathbf{D}$ is the matrix of the double layer kernel; $\mathbf{J}$ is the diagonal Jacobian matrix of the curve parameterization; and $\mathbf{W}$ is the diagonal matrix of quadrature weights. The essential step of the iterative solver is the multiplication of matrix $-\frac{1}{2}\mathbf{I} + \mathbf{DJW} + \mathbf{n}\mathbf{n}^T\mathbf{JW}$. Since $\mathbf{J}$ and $\mathbf{W}$ are all diagonal matrices, the only expensive step is the multiplication with $\mathbf{D}$.

This matrix-vector multiplication operation costs $\mathcal{O}(N^2)$ where N is the number of Nyström points. To accelerate the method we should take advantage of the fact that the Green's function rapidly decays with distance, and thus the double and single layer kernels become nearly degenerate. Several techniques exist to accelerate this matrix-vector multiplication, for example the Barnes-Hut algorithm (to $\mathcal{O}(N \log N)$) and the Fast Multipole Method (to $\mathcal{O}(N)$).

We use a fast matrix-vector multiplication algorithm, which was first proposed in [20] and [19] for the single layer formulation of the Laplace equations in triangulated domains. This method uses singular value decomposition (SVD) to sparsify large low-rank blocks of the discretized double layer operator. The basic ideas of the Fast Multipole and the SVD-based method are illustrated in Figure 2.

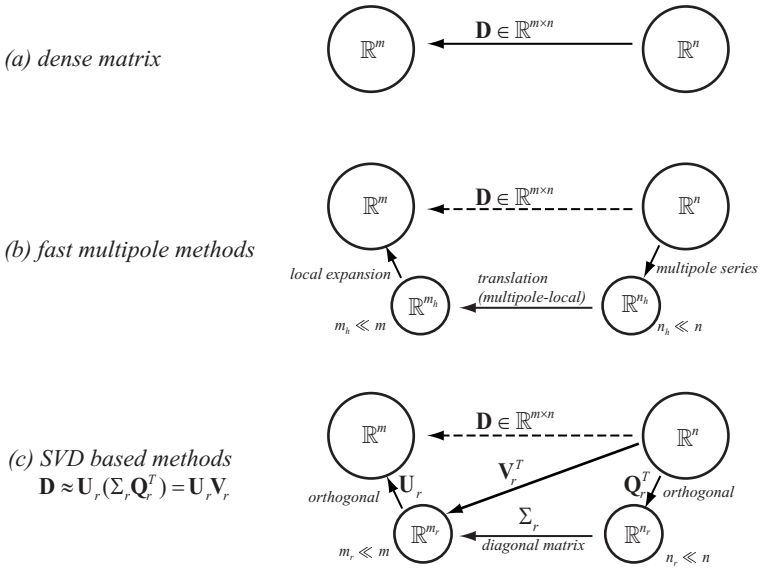


FIG. 2 Low rank approximations of discrete interaction. (*a) The dense interaction. (*b) Fast Multipole method. (*c) SVD based method. Arrows represent linear transformations.*

The dense linear map $\mathbf{D}$ represents the hydrodynamic forces of n source points to m target points. If we assume that these two groups are geometrically well separated the $\mathbf{D}$ is expected to be numerically low rank, i.e. the ratio $s/s_1 < \epsilon$ for all but $r \ll m$ singular values s, where s_1 is the largest singular value and ϵ is a constant determining the accuracy of the computations. Fast multipole methods use truncated analytic expansions and translation operators to sparsify $\mathbf{D}$. Singular value decomposition computes a coordinate transformation, for which $\mathbf{D}$ is diagonal, and eliminates the vectors corresponding to the small singular values. Compared with Fast Multipole Method, SVD-based compression is kernel independent and easy to implement. However, its main disadvantage is the higher algorithmic complexity, $\mathcal{O}(N \log N)$ instead of $\mathcal{O}(N)$. In [19] an orthogonal recursive bisection to create the partition into low-rank blocks. Here we give a version of the algorithm using a hierarchical structure based on curve subdivision. There are two algorithms: the algorithm that sets up the hierarchical matrix representation and the algorithm implementing matrix-vector multiplication.

The setup algorithm. The input to the algorithm is the collection of boundary curves and quadrature points, and three parameters: p, α and ϵ. Parameters p and ϵ are used in the computation of the low-rank representation for blocks and α is used to determine when sets of quadrature points are well-separated. The precise meaning of the parameters is described below. The output is a hierarchical representation of the matrix. To define the matrix representation, we partition quadrature points into a geometry-based hierarchy. First, we partition the boundary curves into several top level segments $E_i^0, i = 0, \cdots, n_s - 1$, each containing roughly the same number of quadrature points. Second, we subdivide every E_i^0 into two segments: E_{2i}^1 and E_{2i+1}^1. We repeat this procedure at each level and we stop when the finest level segment has less than n_p quadrature points in it. We take L to be the number of levels with levels numbered $0 \ldots L-1$. For each segment at each level,

Algorithm 1 construction of $\mathbf{D}$

```
function constructMatrix(top_segment_list)
  matrix_trees := ∅
  for X ∈ top_segment_list do
    matrix_trees := matrix_trees ∪ constructSegmentTree(X,top_segment_list)
  end for
  return matrix_trees
end constructMatrix

function constructSegmentTree(X, segment_list)
  node.submatrices, node.leftchild, node.rightchild := ∅
  node.segment :=X
  near_list := ∅
  for Y ∈ segment_list do
    if separated(B(X), B(Y) ) then
      node.submatrices := node.submatices ∪ {(Y, SPARSE, constructSparse(X, Y)}
    else
      near_list := near_list ∪ Y
    end if
  end for
  if level(X) = L-1 then
    for Y ∈ near_list do
      node.submatrices := node.submatices ∪ {(Y, DENSE, constructDense(X, Y) )}
    end for
  else
    new_list := ∅
    for Y ∈ near_list do
      new_list := new_list ∪ { left(Y),right(Y) }
    end for
    node.leftchild := constructSegmentTree( left(X), new_list)
    node.rightchild := constructSegmentTree( right(X), new_list)
  end if
  return node
end constructSegmentTree
```

we calculate a bounding box of its quadrature points. For a segment X, we use $B(X)$ to denote its bounding box, $I(X)$ to denote the set of indices of its quadrature points, $c(X)$ center of $B(X)$, $r(X)$ the radius of $B(X)$, and *left(X)* and *right(X)* the left and right subsegments of X.

$\mathbf{D}$ is represented as a collection of blocks organized into a hierarchy; each block corresponds to the interaction between two segments. Similarly to FMM methods, we use a low rank representation if two segments are well-separated; otherwise we compute a dense block.

Algorithm (1) is the pseudocode for constructing the matrix $\mathbf{D}$. The matrix is represented as a set of trees, one tree per each top-level segment. Each node of the tree on level l corresponds to a segment E_j^l. Each non-leaf node corresponding to a segment X contains a list of matrices in a low-rank sparce representation described below; each matrix corresponds to a segment on the same level as Y, for which a separation criterion is satisfied. In addition, a non-leaf node contains pointers to two nodes corresponding to the subsegments

of X. The leaf nodes contain only a list of matrices; for segments Y which do not satisfy the separation criterion a dense matrix is stored.

The main function *constructMatrix* simply calls *constructSegmentTree* on each top-level segment X to compute the interaction between X and all other top-level segments. Function *constructSegmentTree(X,segment_list)* construct a tree representation of the submatrix of $\mathbf{D}$ corresponding to interactions of X with segments from *segment_ list*. Function *separated(B_1,B_2)* is used to test whether two bounding boxes B_1 and B_2 are well-separated. If the ratio of the distance between centers $c(B_1)$ and $c(B_2)$ to the sum of their radii is less than a constant α, they are regarded to be not well-separated and either further refinement is necessary, or a dense matrix has to be built.

When two segments X and Y are well-separated, *constructSparse* is called to construct a low-rank representation of the interaction matrix $\mathbf{D}_{X,Y}$ between the sets of quadrature points of X and Y, i.e. to find a column basis of matrix $\mathbf{D}_{X,Y}$ and represent the whole matrix $\mathbf{D}_{X,Y}$ as a linear combination of this basis:

$$\mathbf{D}_{X,Y} \approx \mathbf{U}_r \mathbf{V}_r,$$

(Figure 2).

Let S_X and S_Y be the set of p sampling points from the sets X and Y respectively. For the time being, we assume p to be significantly greater than the numerical rank r of the interaction matrix $\mathbf{D}_{X,Y}$. We explain the estimation of r and p, and the selection of sampling points along with the numerical experiments.

First we construct $\mathbf{D}_{X,S_Y}$ and use SVD or modified Gram-Schmidt to get $\mathbf{U}_r$ which is of size $n \times r$, where r is the numerical rank of matrix $\mathbf{D}_{X,S_Y}$. The Modified Gram-Schmidt algorithm is faster, with small loss of compression effectiveness. In our implementation we use a column pivoted Modified Gram-Schmidt method; the pivoting is used to detect the maximum 2-norm of the remaining vectors and we stop the process whenever that maximum is less than the prescribed constant ϵ.

The matrix $\mathbf{U}_r$ is used to compute $\mathbf{V}_r$. First we evaluate $\mathbf{D}_{S_X,Y}$, and then we subsample $\mathbf{U}_r$ by choosing $\tilde{\mathbf{U}}$ whose rows are the rows of $\mathbf{U}_r$ corresponding to the set of points S_X. We compute $\mathbf{V}_r$ from the least square system $\tilde{\mathbf{U}}\mathbf{V}_r = \mathbf{D}_{S_X,Y}$.

Complexity analysis. There are three important observations on which the complexity analysis of the construction algorithm are based. First, as we pointed out, the time complexity of *constructSparse*(X, Y) can be bounded by $C \cdot n$, where n is the number of points in the larger of X and Y. Second, the complexity of *constructDense*(X, Y) is $\mathcal{O}(n^2)$. Lastly, except for the segments at the top level, every segment gets an $\mathcal{O}(1)$ number of segments in the *segment_ list* from its parent segment, and passes also an $\mathcal{O}(1)$ number of segments in the *new_ list* to its children, under the assumption that the boundary curve is smooth and the distribution of quadrature points is uniform. Consider a segment X, the segments in the *near_ list* of X have centers in a circle centered at $c(X)$ with radius $(2\alpha + 1)r(X)$. There are about $2\alpha + 1$ segments in this *near_ list* due to the assumption about uniformity. Therefore, the *new_ list* contains about $4\alpha + 2$ segments because each segment in the *new_ list* is a child of some segment in *near_ list* . However, among them, there would be roughly only half, about $2\alpha + 1$, of them falling in the the circle centered at $c(\mathit{left}(X))$ with radius $(2\alpha + 1)r(\mathit{left}(X))$ because $r(\mathit{left}(X))$ is half the size of $r(X)$, and same for $\mathit{left}(X)$. We use g to bound this $2\alpha + 1$ number.

At the coarsest level, each segment computes its interaction with the remaining $max(n_s - g, 0)$ top level segments using the SVD method. The work can be bounded by $n_s \cdot n_s \cdot (Cn_p 2^{L-1})$. For any other level i, we have $\mathcal{O}(2^i n_s)$ segments, each of which computes

Algorithm 2 Matvec of $\mathbf{D}$

```
function matVec(matrix_trees, x)
  b := 0
  for tree ∈ matrix_trees do
    b :=matVecSegment(tree, x, b)
  end for
  return b
end matVec

function matVecSegment(node, x, b_old)
  b :=b_old
  for (type, matrix, src) ∈ node.submatrices do
    if type = DENSE then
      b(I(node.segment)) := b(I(node.segment)) + matVecDense(mat, x(I(src)))
    else
      b(I(node.segment)) := b(I(node.segment)) + matVecSparse(mat, x(I(src)))
    end if
  end for
  b :=matVecSegment(leftchild, x, b)
  b :=matVecSegment(rightchild, x, b)
  return b
end matVecSegment
```

the interaction with g other segments at the same level. This work is proportional to $2^i n_s \cdot g \cdot (Cn_p 2^{L-1-i})$. For the finest level each segment also must compute the dense interaction between itself and its neighbors, at most g of them. This costs $2^{L-1} n_s \cdot g \cdot n_p^2$. The total cost can be bounded by

$$\begin{aligned} & n_s \cdot n_s \cdot (Cn_p 2^{L-1}) + \sum_{i=1}^{L-1} 2^i n_s \cdot g \cdot (Cn_p 2^{L-1-i}) + 2^{L-1} n_s \cdot g \cdot n_p^2 \\ \leq \; & Cn_s \cdot n_s n_p 2^{L-1} + Cg \cdot L \cdot n_s n_p 2^{L-1} + gn_p \cdot n_s n_p 2^{L-1} \\ = \; & (Cn_s + Cg \cdot L + gn_p) \cdot n_s n_p 2^{L-1} \end{aligned}$$

Cn_s, Cg and gn_p are all constants. $n_s n_p 2^{L-1}$ is the total number of quadrature points N. L is the depth of the hierarchical structure, so it is $\mathcal{O}(\log N)$. Therefore the total complexity $(Cn_s + Cg \cdot L + gn_p) \cdot n_s n_p 2^{L-1}$ is bounded by $\mathcal{O}(N \log N)$.

Matrix-vector multiplication. Algorithm (2) is the pseudo code for matrix-vector multiplication using the SVD-based representation of $\mathbf{D}$. Function *matVecDense* simply multiplies the densely stored matrix with a vector. On the other hand, *matVecSparse* of two segment X and Y uses the sparse representation: $\mathbf{D}_{X,Y} = \mathbf{U}_r \mathbf{V}_r$. Since $\mathbf{U}_r$ and $\mathbf{V}_r$ are both of size $n \times r$, assuming n is the size of X and Y, multiplication with $\mathbf{V}_r$ and $\mathbf{U}_r$ is much cheaper than multiplication with $\mathbf{D}_{X,Y}$. Figure 3 shows the sparse structure of a simply-connected boundary.

Numerical Experiments. All experiments in this section were performed on a SUN Ultra80, 450MHz workstation (single processor). We use three parameters in the matrix construction algorithm: α for separation detection, ϵ for modified Gram-Schmidt algorithm and p for sampling matrix columns and rows. The value of α is usually chosen to be 1.5

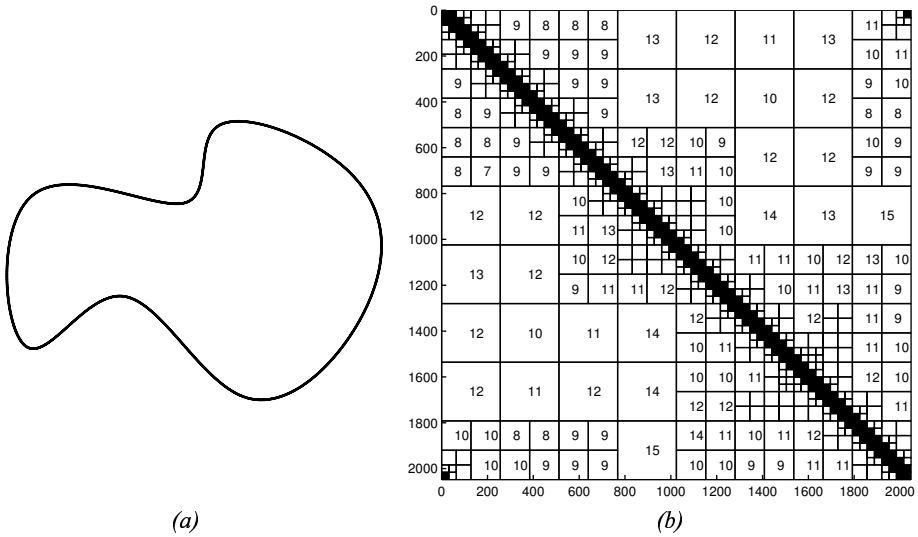


FIG. 3 The sparsity structure of matrix $\mathbf{D}_{X,Y}$ of a boundary curve. The curve is discretized into 1024 quadrature points. The matrix is of size 2048×2048. The number in each block denotes its numerical rank r. The black blocks along the diagonal correspond to close interaction, and thus are stored densely.

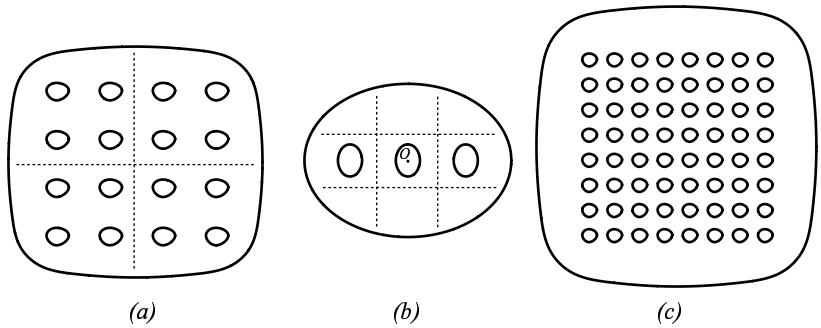


FIG. 4 Test domains. Solid curves represent the boundary of the domain. The dots in the domains are the points used for error estimation.

to 2. Numerical experiments indicate that increasing the value α reduces unacceptably the accuracy without significant savings in speed.

The tolerance ϵ is the most important parameter; it determines the speed and accuracy of the SVD-approximation and in some sense corresponds to the truncation of the analytic expansions in the Fast Multipole Method.

The estimation of r and p can be obtained by the following incremental procedure. For two segments X and Y, we first choose a small number for p, and use these p sampling points to construct the SVD representation of $\mathbf{D}_{X,Y}$. If the numerical rank r of $\mathbf{D}_{X,Y}$ is close to p, which means that the number of sampling points p is not enough, then we double p and compute the SVD representation of $\mathbf{D}_{X,Y}$ again until r is much smaller than

TABLE 1
Comparison between the dense matrix matrix-vector multiplication and the SVD-based matrix-vector multiplication for two different flow fields and geometries. Setup time includes the construction of the matrix and the preconditioner. Solve time is the time used by GMRES solver. We see that as the problem scales, the dense approach grows up quadratically, while SVD based approaches scales almost linearly.

domain, solution	N	matrix	setup	solve	$\|\mathbf{u}\|_{err}$	p_{err}
Fig.4(a)	736	dense	1.79	4.89	2.34×10^{-06}	2.67×10^{-7}
		svd	2.11	4.56	7.28×10^{-06}	3.18×10^{-6}
cubicflow	1472	dense	6.37	17.0	8.82×10^{-08}	2.30×10^{-8}
		svd	5.25	7.98	8.12×10^{-06}	4.63×10^{-6}
	2944	dense	23.9	58.9	1.04×10^{-08}	2.65×10^{-9}
		svd	12.2	15.8	5.65×10^{-07}	2.91×10^{-7}
	5888	dense	100	224	1.30×10^{-09}	3.10×10^{-10}
		svd	22.7	30.9	5.39×10^{-07}	2.31×10^{-7}
Fig.4(b)	384	dense	0.44	0.52	1.80×10^{-06}	1.07×10^{-6}
		svd	0.74	0.37	5.08×10^{-06}	3.31×10^{-6}
Stokeslet	768	dense	1.53	2.75	2.47×10^{-07}	1.40×10^{-7}
		svd	1.35	1.14	1.95×10^{-06}	2.89×10^{-6}
	1536	dense	6.18	10.1	3.44×10^{-08}	1.80×10^{-8}
		svd	3.01	2.50	1.07×10^{-06}	9.46×10^{-7}
	3072	dense	29.1	39.3	4.63×10^{-09}	2.29×10^{-9}
		svd	5.91	5.40	9.85×10^{-07}	4.84×10^{-7}

p. In practice, we stop when r is less than $p/3$, which ensures that the algorithm can find a good basis $\mathbf{U}$ of matrix $\mathbf{D}_{X,Y}$ with very high probability. The position of these p sampling points are chosen to be evenly spaced on the boundary..

In Table 1 we report wall-clock time and accuracy comparisons between the dense and the SVD-sparsified double layer operators. We solve two different problems, a cubic flow, and a flow that corresponds to a Stokeslet. We use pointwise error on a fixed number of points to evaluate the accuracy. We first solve the integral equation for the hydrodynamic potential and then we evaluate the velocities and pressures with (9).

The sparsification is divided to setup a phase and an iterative solution phase. As expected, the setup time for the dense matrix scales with the square of the number of unknowns. The fast methods scales almost linearly since the $\log(N)$ is quite small. In this example we have used a fixed tolerance $\epsilon = 10^{-4}$—that is why there is no improvement in the error for the larger problem. Table 2 compares running time and accuracy for different choices of ϵ, for the geometry depicted in Fig. 4(b) with a Stokeslet flow. As expected the accuracy improves without significant increase in running time.

Perhaps a more representative example for the scalability of the method is depicted in Table 3. The geometry is that of Fig. 4(c) for a Stokeslet flow. We solve for two different values of ϵ and for a eight-fold increase in the problem size. It is apparent that about 10,000 quadrature points are enough to get single precision accuracy. The running times are increase almost linearly with the problem size.

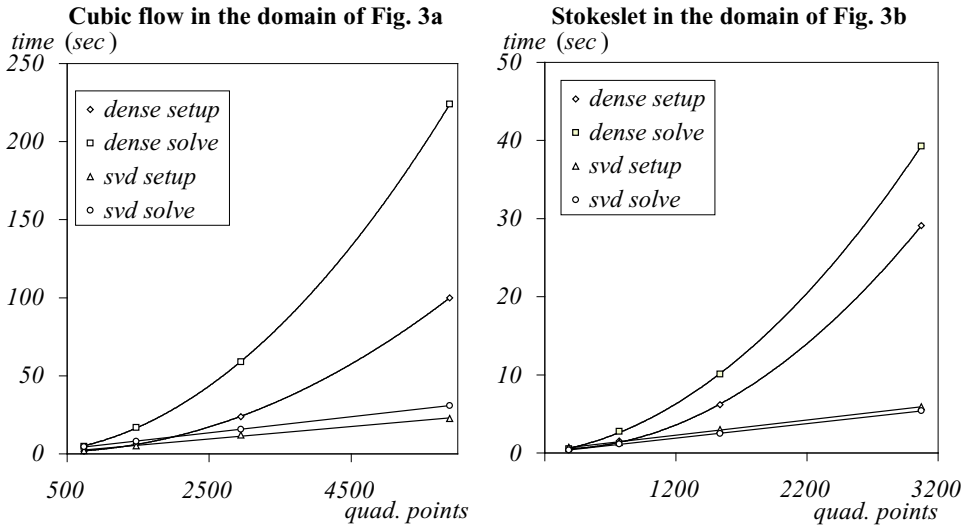


FIG. 5 Plots of the data from Table 1
with linear fit for SVD-based solver and quadratic fit for the dense solver.

TABLE 2
Running times and pointwise errors for the SVD-based sparsification. We report results for the geometry depicted in Figure 4(b) for a Stokeslet flow. We vary the numerical rank tolerance ϵ and we hold the number of quadrature points fixed (768); here *max rank* indicates the maximum numerical rank for a SVD-approximated block.

ϵ	setup(s)	solve(s)	$\lvert\mathbf{u}\rvert_{err}$	p_{err}	max rank
10^{-02}	3.48	7.13	3.95×10^{-4}	1.84×10^{-4}	8
10^{-03}	4.18	8.09	3.67×10^{-5}	1.43×10^{-5}	10
10^{-04}	5.49	7.95	6.68×10^{-6}	4.63×10^{-6}	12
10^{-05}	6.00	8.59	8.31×10^{-7}	5.82×10^{-7}	14
10^{-06}	6.99	9.71	1.77×10^{-7}	9.49×10^{-8}	16
10^{-07}	7.93	10.8	1.17×10^{-7}	4.65×10^{-8}	18

5.2. Regular grid solver

There exist several methods for the efficient solution of linear systems representing discretizations of elliptic PDEs. Examples are FFTs, multigrid and two-level domain decomposition algorithms which are asymptotically optimal. However for medium size problems it turns out the domain decomposition methods are faster. We have developed our code on top of the PETSc library [3, 4]. PETSc includes several methods for regular grids such as domain-decomposition preconditioners and multigrid. In Table 5.2 we report timings for four different preconditioners: block-Jacobi, single-grid additive Schwarz, two-grid additive Schwarz, and a V-cycle multigrid. We report isogranular scalability results for problems up to 10 million unknowns on 16 processors[3]. Our intention is not a detailed

[3]Let us note here that only the regular grid solver is parallelized in our implementation. The parallelization of the boundary integral solver is in progress.

TABLE 3
Running times and pointwise errors for the SVD-based sparsification. We report results for the geometry depicted in Figure 4(c) (64 circles) for a Stokeslet flow; for two different values of the numerical rank tolerance ϵ and for an eight-fold increase in problem size. Observe the almost linear scaling in *setup* and *solve* running times with the problem size. For this example about 10,000 Nyström points give single precision machine accuracy.

ϵ	L/N	setup(s)	solve(s)	$\|\mathbf{u}\|_{err}$	p_{err}	max rank
10^{-04}	4/4,544	49.6	111	9.02×10^{-6}	1.63×10^{-5}	14
	5/9,088	118	226	1.52×10^{-6}	1.59×10^{-6}	14
	6/18,176	217	435	1.35×10^{-6}	1.02×10^{-6}	14
	7/36,352	487	904	1.07×10^{-6}	9.24×10^{-7}	14
10^{-06}	4/4,544	67.2	137	4.64×10^{-7}	1.11×10^{-6}	19
	5/9,088	164	287	1.85×10^{-7}	2.60×10^{-7}	19
	6/18,176	294	559	1.09×10^{-7}	1.23×10^{-7}	19
	7/36,352	682	1,172	1.23×10^{-7}	1.57×10^{-7}	19

comparison between the different solution techniques, but to give an numerical evidence of the scalability of the different preconditioners for the $Q1 - Q1$ discretization.

TABLE 4
In this table we compare iteration count and wall-clock time for 4 different linear solvers for the discretized Stokes problem. All use the same Krylov solver (Conjugate Residuals). What differs is the preconditioner. Here *grid* is the number of grid points (3 degrees of freedom per grid point); *p* is the number of processors; *BJ* denotes a block-Jacobi domain-decomposition with ILU(1) preconditioning in each subdomain; *ASM* is an additive Schwarz preconditioner with fixed overlap; *2L-ASM* is a two level additive Schwarz preconditioner in which the fine grid uses the ASM method described above and the coarse grid is solved redundantly on every processor using a sparse LU factorization. The coarse grid is 10 times smaller; *MG* is a 5-level single V-cycle multigrid preconditioner with sparse LUs for the coarsest level and the BJ preconditioner for the rest. For each different preconditioner we report wall-clock time in seconds (*sec*) and iteration counts (*it*) for a relative residual reduction of 1×10^{-7} . The largest problem has 10 million unknowns and it took 15 seconds to solve. The preconditioners are parts of the PETSc library. The runs were performed on a 900 MHz Compaq server at the Pittsburgh Supercomputing Center.

grid	*p*	*BJ*		*ASM*		*2L-ASM*		*MG*	
		it	*sec*	*it*	*sec*	*it*	*sec*	*it*	*sec*
128^2	2	296	78	161	45	26	3	34	11
256^2	4	602	350	330	220	21	6	47	36
512^2	8	1,240	1,450	692	950	18	11	56	98
1024^2	16	2,578	6,100	1,391	3,910	19	24	57	260

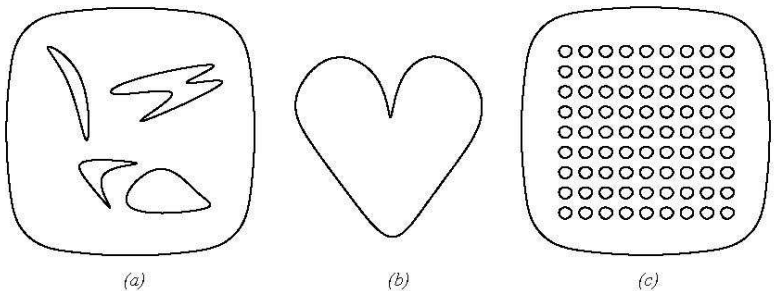
As expected the single-grid preconditioners perform quite poorly compared to multilevel methods. For the latter we can observe mesh size-independence on the number

of Krylov steps. Notice that we quadruple the problem size and we double the number of processors. Thus, for an optimal algorithm, wall-clock time should double with the problem size. Indeed, this is the case for the *2L-ASM* preconditioner which outperforms the other methods. Multigrid is optimal in the number of iterations, but (for the specific implementation) it is significantly slower, probably due to interprocessor communication overhead. We have not attempted to fine-tune the multigrid preconditioner and thus we do not advocate one method over the other. We have chosen the two-level method because is somewhat simpler to combine with the boundary integral solver. For details on the the theory of two-level preconditioners for indefinite elliptic systems see [22].

6. NUMERICAL EXPERIMENTS

In this section we test EBI on problems with exact analytic solutions. We assess the pointwise accuracy of the solver and we investigate the effects of the accuracy of the boundary integral solver on the overall accuracy of the method.

FIG. 6 Domains used in numerical examples



We present results for four different problems. The solutions are restricted to the target domains (Figure 6), which are embedded in the unit square. We have chosen the following analytic solutions: a Poiseuille flow

$$\boldsymbol{u} = \{y(1-y), 0\}\,,\ p = -2x,$$

a "cubic flow"

$$\boldsymbol{u} = \left\{y^3, x^3\right\}, p = 6xy,$$

a "body force flow"

$$\boldsymbol{u} = 2\left\{-x^2y, y^2x\right\}\,,\ p = \sin(xy),\ \boldsymbol{b} = 4\nu\left\{y(1+\cos(xy), -x(1+\cos(xy)\right\}.$$

We also use a Stokeslet (7) centered at $(0.5, 0.7)$ and oriented along $\mathbf{e} = \{1, 1\}$. The corresponding pressure is given by

$$p = \frac{1}{2\pi}\frac{\mathbf{r}}{\rho^2}\cdot\mathbf{e}.$$

All experiments in this section were performed on a SUN Ultra80, 450MHz workstation (single processor).

In the first example we use the cubic flow solution for the interior problem in a circle of radius 0.3. Convergence results are presented in Table 5. We report and compare convergence rates for first-order accurate (*dense-1*) and second-order accurate TESCs (*dense-2*);

for the latter we also report results for the SVD acceleration (*svd*). The integral equation has been discretized by 320 quadrature points. Increasing this number did not affect the accuracy significantly. For the first order TESCs the convergence rate for the velocity is superlinear and hence suboptimal; with second order TESCs both dense and sparse computations result in optimal convergence rates for the velocities and pressures.

TABLE 5
Convergence results for the cubic flow inside a circle. In (*dense-1*) TESCs were first order accurate; in (*dense-2*) TESCs include second order derivatives. In (*svd*) we use second order TESCs combined with the svd acceleration. The rank tolerance ϵ is 10^{-7}; (**u**) and (**p**) denote error in the infinity norm for the velocities and pressures.

	dense-1		*dense-2*		*svd*	
grid	**u**	**p**	**u**	**p**	**u**	**p**
32^2	2.43×10^{-3}	1.19×10^{-1}	8.35×10^{-4}	2.31×10^{-2}	8.84×10^{-4}	2.43×10^{-2}
64^2	8.06×10^{-4}	1.07×10^{-1}	1.81×10^{-4}	1.33×10^{-2}	1.90×10^{-4}	1.37×10^{-2}
128^2	3.06×10^{-4}	8.44×10^{-2}	4.95×10^{-5}	1.83×10^{-3}	5.23×10^{-5}	2.16×10^{-3}
256^2	1.20×10^{-4}	4.21×10^{-2}	1.12×10^{-5}	4.79×10^{-4}	1.20×10^{-5}	7.54×10^{-4}

In the second example we repeat the same test, but for the geometry depicted in Figure 6(b) and for two different analytic solutions, the Poiseuille and the body-force flow. In this example the number of quadrature points for the integral equation varies. For *dense-1* (first-order) we used 768 points for the 32^2 grid, 1546 for the other two grids, and 3,072 points for the 256^2 grid. For the *dense-2* (second-order) we used 768 points for all background grid sizes. In *svd* we used 1536 points. The increased number of quadrature points did not improve the convergence rate for the first-order TESCs. Optimal pointwise convergence rates are observed for the velocities and pressures for both the dense and SVD versions. The exact solution along with the error distribution for three different grids are shown in Figure 7 (for the Poiseuille flow).

In our previous examples the approximation tolerance for the SVDs was kept constant to 10^{-7}. For the following test we have chosen an example for which both the geometry and the solution vary rapidly close to a specific location. We look at the 2D-heart-shaped domain, Figure 6(b), for which the exact solution is given by the stokeslet solution from a pole located at (0.5,0.7). This location is very close to the rapidly changing geometry at the top of the 2D-heart. As a result we expect that a large number of quadrature points is required to obtain sufficient accuracy. Table 7 summarizes the results for this experiment and Figure 8 depicts the exact solution and the error distribution. The number of necessary quadrature points to obtain optimal pointwise convergence in the background grid was determined experimentally based on dense solves; nearly 800 points are enough to resolve the problem; in this test we took 1,664 quadrature points; we found that this extra discretization does not help as we can see by comparing the columns of Table 7. We use *dense-1* as the reference calculation. In *svd-1* the truncation tolerance for the modified Gram-Schmidt is 10^{-3}; it results in suboptimal convergence rates. By using a tighter tolerance, 10^{-5}, we recover optimal rates. In Figure 8 we show the exact solution and the pointwise error distribution.

In the next example we look at an interior flow (body force flow) around 81 circles. For the 64^2 grid we use 9,088 Nyström points and for the two finer grids we use 18,176

TABLE 6
Convergence results for two flows in the domain Figure 6(a). Here (*dense-1*), (*dense-2*), (*svd*) and (**u**), (**p**) are as in Table 5.

	dense-1		*dense-2*		*svd*	
grid	**u**	**p**	**u**	**p**	**u**	**p**
			Poiseuille			
32^2	8.84×10^{-3}	1.01×10^{-1}	4.30×10^{-4}	3.70×10^{-2}	4.83×10^{-4}	3.79×10^{-2}
64^2	2.71×10^{-4}	9.33×10^{-2}	1.31×10^{-4}	9.84×10^{-3}	1.43×10^{-4}	1.09×10^{-2}
128^2	1.39×10^{-4}	4.07×10^{-2}	2.76×10^{-5}	3.67×10^{-3}	2.98×10^{-5}	2.86×10^{-3}
256^2	2.93×10^{-5}	1.57×10^{-2}	7.45×10^{-6}	2.22×10^{-3}	7.51×10^{-6}	8.14×10^{-4}
			Body force			
32^2	3.47×10^{-2}	8.99×10^{-1}	2.25×10^{-2}	6.31×10^{-2}	1.36×10^{-3}	7.28×10^{-2}
64^2	2.33×10^{-3}	2.68×10^{-1}	5.62×10^{-4}	5.03×10^{-2}	6.67×10^{-4}	4.19×10^{-2}
128^2	6.23×10^{-4}	1.45×10^{-1}	1.46×10^{-4}	3.87×10^{-2}	1.47×10^{-4}	2.67×10^{-2}
256^2	2.43×10^{-4}	1.15×10^{-1}	3.58×10^{-5}	1.06×10^{-2}	4.31×10^{-5}	1.14×10^{-2}

TABLE 7
Convergence results for a stokeslet flow generated by a pole just outside the domain. Here the jumps are second-order accurate. All problems use 1,664 quadrature points.In (*dense-1*) we evaluated a dense double layer matrix. In (*svd-1*) and (*svd-2*) we sparsify using variable rank tolerance; 10^{-3} for the former and 10^{-5} for the latter.

	dense-1		*svd-1*		*svd-2*	
grid	**u**	**p**	**u**	**p**	**u**	**p**
32^2	7.01×10^{-3}	3.36×10^{-1}	7.05×10^{-3}	2.97×10^{-1}	7.05×10^{-3}	2.46×10^{-1}
64^2	1.01×10^{-3}	1.55×10^{-1}	1.08×10^{-3}	2.27×10^{-1}	9.96×10^{-4}	1.57×10^{-1}
128^2	2.10×10^{-4}	9.70×10^{-3}	4.12×10^{-4}	1.21×10^{-1}	2.13×10^{-4}	1.48×10^{-2}
256^2	4.61×10^{-5}	4.16×10^{-3}	9.55×10^{-5}	4.69×10^{-2}	4.80×10^{-5}	1.04×10^{-2}

points. We vary the accuracy of the SVD approximations by truncating at 10^{-3} (*svd-1*), 10^{-5} (*svd-2*), and 10^{-7} (*svd-3*). Table 8 summarizes the convergence study. Optimal rates are obtained for the most accurate representation of the double layer. Figure 9 depicts the exact solution and the error distribution. For the last example we do not have an analytic solution, and we just solution in figure 10. The boundary conditions are $\{1, 0\}$ on the enclosing curve, and zero on the internal domains.

7. CONCLUSIONS AND EXTENSIONS

We have presented a second-order accurate solver for the Stokes operator defined on arbitrary geometry domains. We use a hybrid boundary integral, finite element formulation to circumvent the need for mesh generation. We employ an efficient double layer formulation for the integral equations. The method requires two regular grid solves and

FIG. 7 Exact solution (color maps) and error distribution (top to bottom), for 64^2,128^2, and 256^2; the error plot for the 256^2 grid is omitted. The solution is a Poiseuille flow.

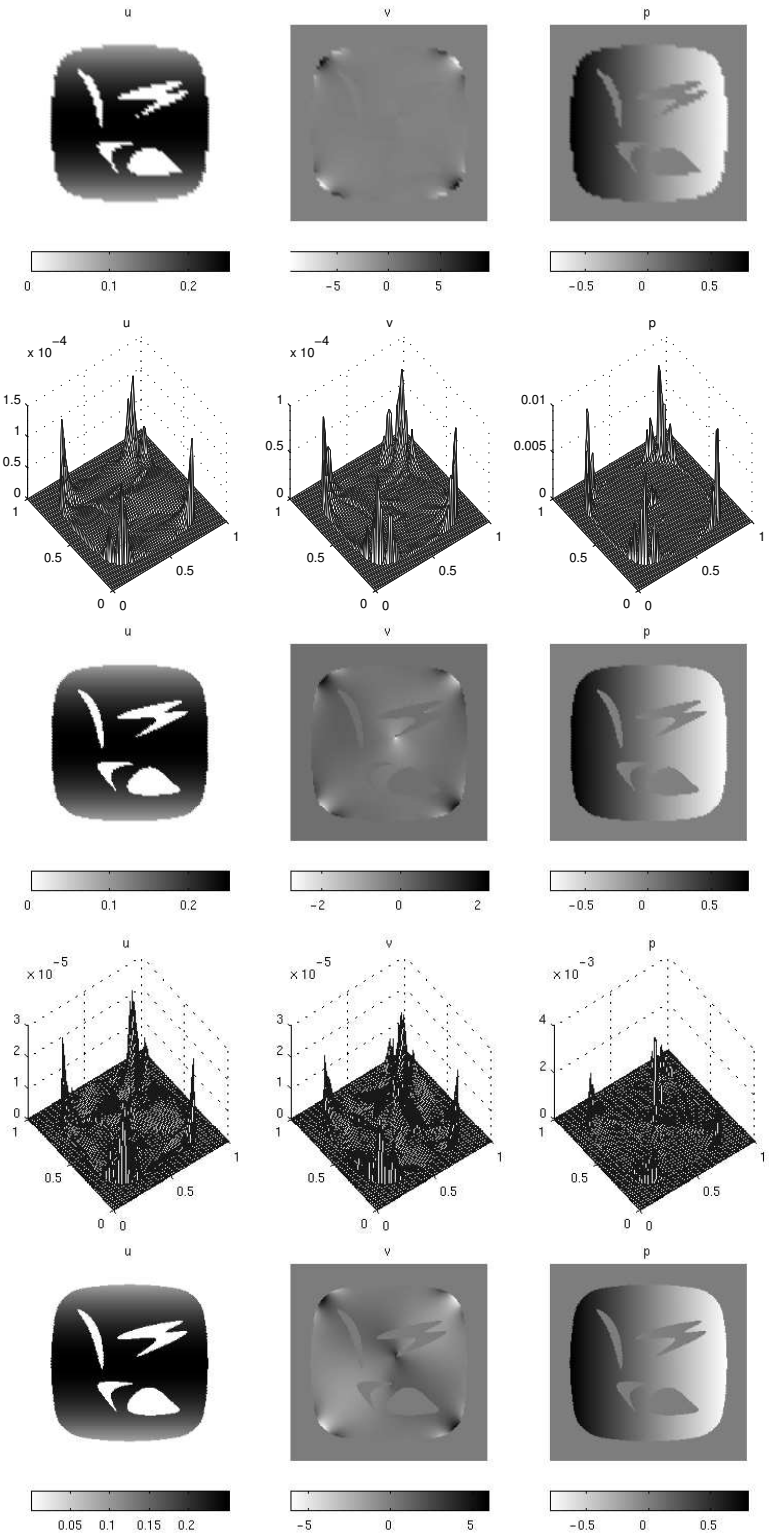


FIG. 8 Exact solution and error distribution (top to bottom), for 64^2,128^2, and 256^2; the error plot for the 256^2 grid is omitted. The solution is the Stokeslet located at (0.5,0.7).

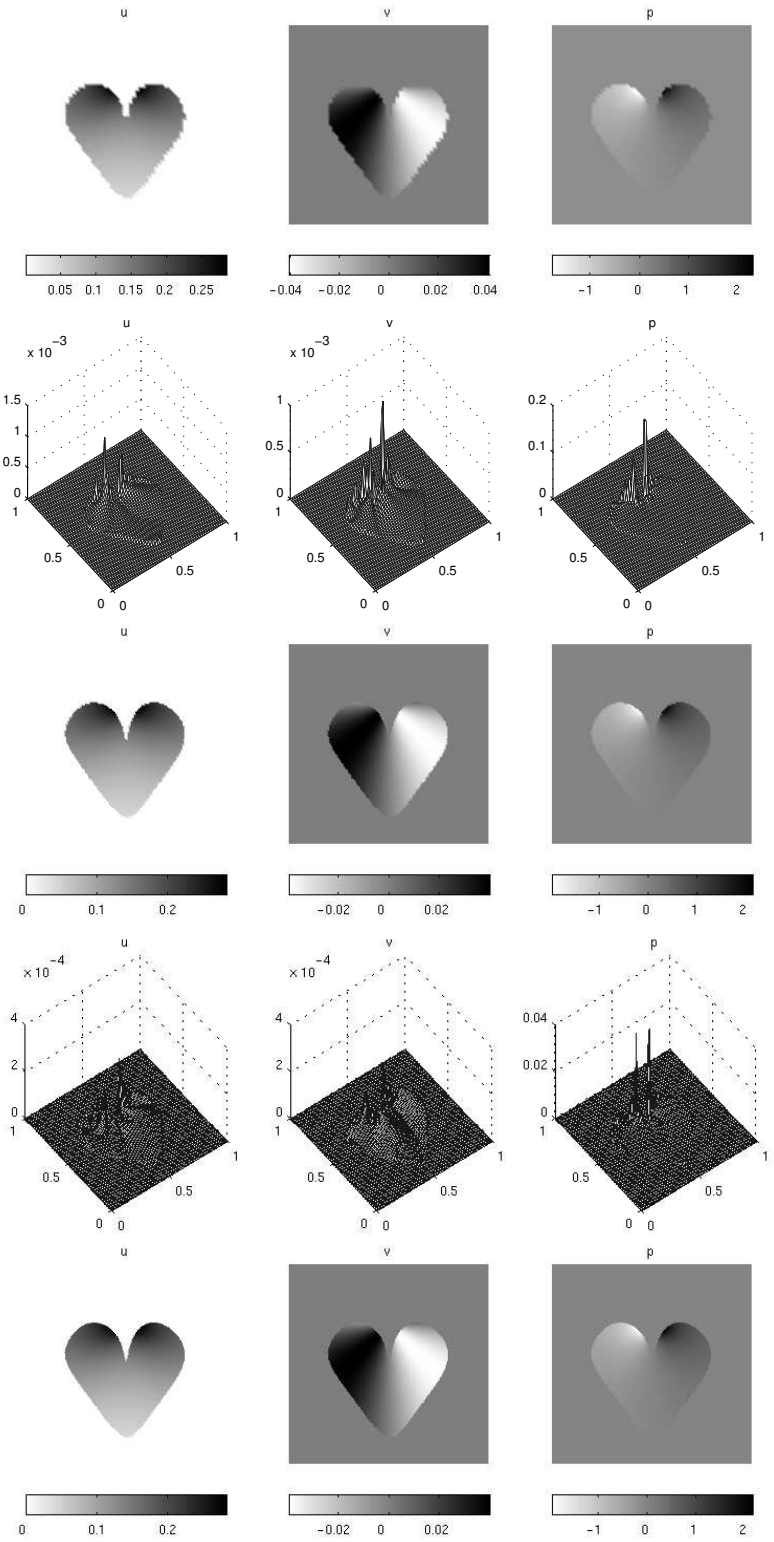


FIG. 9 Exact solution and error distribution (top to bottom), for 64^2,128^2, and 256^2; the error plot for the 256^2 grid is omitted. The exact solution is a the body force flow.

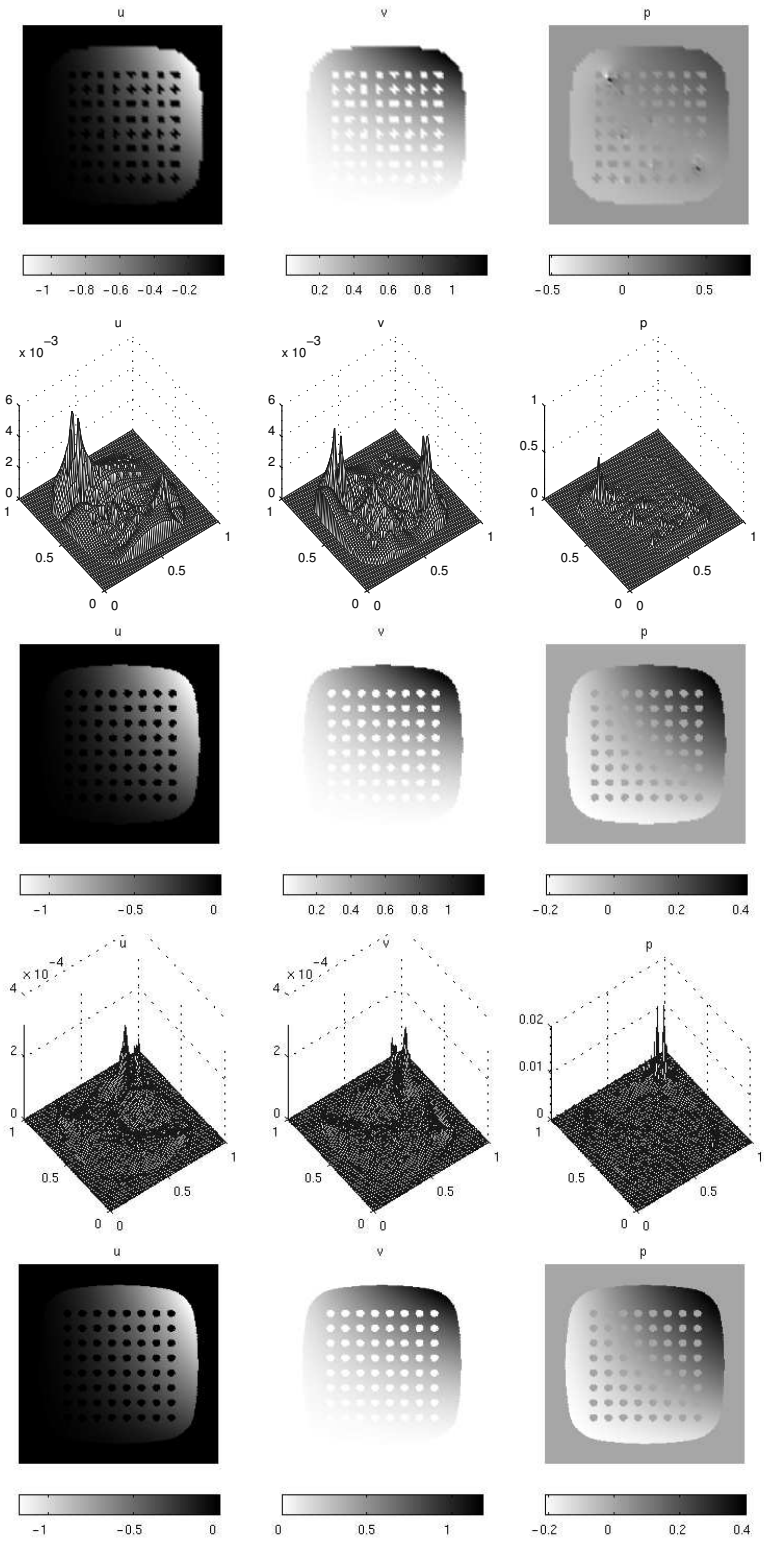


FIG. 10 Solution for a problem with Dirichlet conditions corresponding to a unit wind flow, presented for two different geometries. The two bottom pictures depict the resulting streamlines.

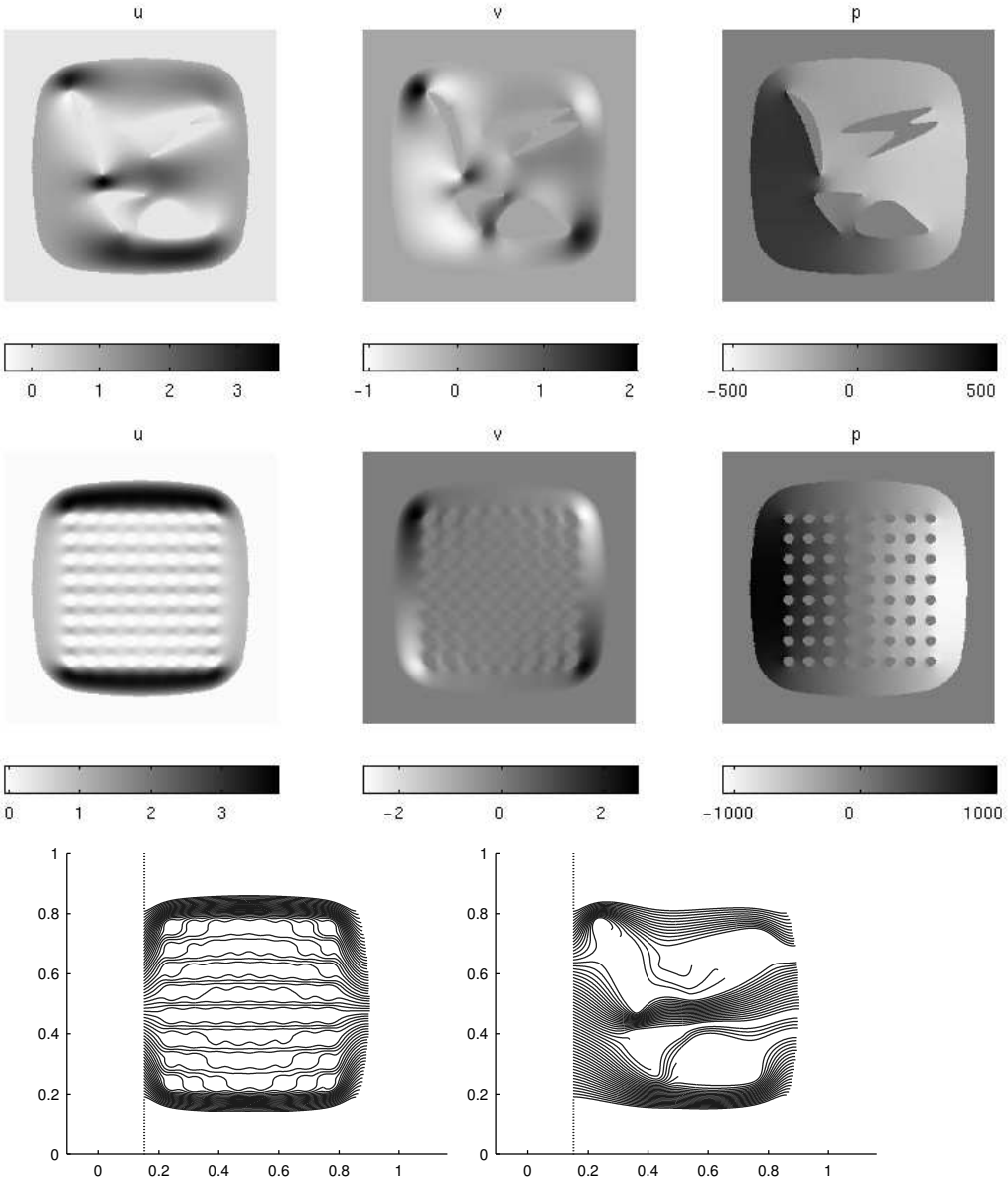


TABLE 8
Convergence rates and pointwise accuracy for the 81-circle geometry and for the "body-force" flow. Here (*svd-1*) is computed with $\epsilon = 10^{-3}$, (*svd-2*) with $\epsilon = 10^{-5}$, and (*svd-3*) with $\epsilon = 10^{-7}$. Optimal convergence rates can be verified for *svd-3*.

	svd-1		*svd-2*		*svd-3*	
grid	**u**	**p**	**u**	**p**	**u**	**p**
64^2	5.89×10^{-2}	7.72×10^{-0}	5.65×10^{-3}	4.76×10^{-1}	5.79×10^{-3}	4.89×10^{-1}
128^2	2.65×10^{-2}	5.53×10^{-0}	2.38×10^{-4}	5.54×10^{-2}	1.68×10^{-4}	1.57×10^{-2}
256^2	6.61×10^{-3}	2.99×10^{-0}	7.75×10^{-5}	2.33×10^{-2}	3.45×10^{-5}	6.95×10^{-3}

one integral solve.

We looked in detail the problem for which the boundary conditions for the velocities given. The method extends to Neumann and mixed boundary value problems. The latter case however, the integral equations require preconditioning.

We also presented scalability and convergence studies for both the regular and boundary solvers. We have implemented an easy way to to accelerate the matrix-vector multiplications required in the solution of the integral equation.

One restriction of the method as we presented it, is the stringent requirements on the regularity of the boundary geometry. However this can be circumvented by replacing the jump computation by direct evaluation. For example the jump terms can be computed to machine accuracy by plugging in the exact solution in the stencils that cross the boundary. The exact solution can be obtain by direct evaluation of the velocity. This will require adaptive quadratures—but only for the points close to a corner.

Currently only the background grid computations are parallelized, but we work on parallelizing the boundary integral solver as well.

Acknowledgments. We thank L. Greengard for valuable discussions leading to the formulation of the approach.

APPENDIX A: COMPUTATION OF JUMPS FOR THE STOKES OPERATOR

Here we show how the jumps on the velocities and pressures can be computed. We use $[[\cdot]]$ to denote the jump of a function across the interface (exterior $-$ interior). We use D to denote Gateaux differentiation. We also assume that the curve parameterization $t \mapsto \mathbf{y}(t)$ is smooth enough (at least in C^2). We write $\dot{\mathbf{y}}$ and $\ddot{\mathbf{y}}$ to denote the first and second derivative with respect t. In order to derive the jumps for the pressure we first define a potential q corresponding to a solution of the Laplace operator:

$$q(\mathbf{x}) = \int_\gamma \frac{\mathbf{r}\cdot\boldsymbol{n}(\mathbf{y})}{\rho^2}\phi(\mathbf{y})\,d\gamma(\mathbf{y}),\ \mathbf{x}\in\omega$$

Then $q(\mathbf{x})$ satisfies $-\Delta q = 0$, in $\mathbb{R}^2/\gamma$ with appropriate Dirichlet boundary conditions. From potential theory we know that the extension of q outside ω is discontinuous. More precisely the following relations hold true:

$$[[q]] = \phi, \tag{27}$$

$$[[Dq\cdot\boldsymbol{n}]] = 0. \tag{28}$$

The first equation gives the zeroth order jump. To compute the first order jumps we differentiate the first equation (with respect to t) and by the chain rule we obtain

$$[[Dq]] \cdot \dot{\mathbf{y}} = \dot{\phi} \tag{29}$$

for the tangential derivative. Equations (28) and (29) define a system with two equations and two unknowns, $[[\partial_x q]], [[\partial_y q]]$. Second order derivatives can be computed by taking tangential derivatives, and the jumps in the Laplacian. Thus we obtain

$$[[\Delta q]] = 0, \tag{30}$$

$$[[D^2 q]]\dot{\mathbf{y}} \cdot \dot{\mathbf{y}} + [[Dq]] \cdot \ddot{\mathbf{y}} = \ddot{\phi}, \tag{31}$$

$$[[D^2 q]]\dot{\mathbf{y}} \cdot \boldsymbol{n} + [[Dq]] \cdot \dot{\boldsymbol{n}} = 0. \tag{32}$$

Now we have three equations with three unknowns: $[[\partial_{xx} q]]$, $[[\partial_{yy} q]]$, $[[\partial_{xy} q]]$.

The pressure jumps can be derived from the above relations. Since the discretization is only first order accurate for the pressure, we only need zero and first order jumps. For the double layer potential we have

$$p(\mathbf{x}) = \mathcal{K}[\boldsymbol{\mu}](\mathbf{x}) = \frac{1}{2\pi} \int_\gamma \nabla_x \frac{\mathbf{r} \cdot \boldsymbol{n}(\mathbf{y})}{\rho^2} \cdot (-2\nu \boldsymbol{\mu}(\mathbf{y}))\, d\gamma(\mathbf{y}).$$

Let q_i be given by

$$q_i = \frac{1}{2\pi} \int_\gamma \frac{\mathbf{r} \cdot \mathbf{n}}{\rho^2} \phi_i \, d\gamma, \; i = 1, 2,$$

with

$$\phi_i = -2\nu \boldsymbol{\mu}_i.$$

Then

$$p = \sum_{i=1,2} \partial_i q_i,$$

and hence

$$[[p]] = \sum_{i=1,2} [[\partial_i q_i]];$$

that is the zeroth- and first-order jumps in the pressure correspond the sum of the first- and second-order jumps of q_i.

For the double layer formulation of the velocity we use similar relations with (27) and (28). These relations can be derived by taking appropriate limits across the interface [35]. In fact, if the velocity is given by

$$\boldsymbol{u}(\mathbf{x}) = \frac{1}{\pi} \int_\gamma \frac{\mathbf{r} \otimes \mathbf{r}}{\rho^2} \, \frac{\mathbf{r} \cdot \boldsymbol{n}(\mathbf{y})}{\rho^2} \boldsymbol{w}(\mathbf{y})\, d\gamma(\mathbf{y}),$$

then the following interface conditions hold for the jumps across the interface:

$$[[\boldsymbol{u}]] = \boldsymbol{\mu}, \tag{33}$$

$$[[\boldsymbol{S}\boldsymbol{n}]] = [[-p\mathbf{I} + \nu(D\boldsymbol{u} + D\boldsymbol{u}^T)\boldsymbol{n}]] = 0. \tag{34}$$

In order to construct TESCs for the momentum and incompressibility equations we need to compute $[[D\boldsymbol{u}]]$ and $[[D^2\boldsymbol{u}]]$. (We already have $[[p]]$ and $[[Dp]]$). If we differentiate (33) (with respect the curve parameterization t), we obtain:

$$[[D\boldsymbol{u}]]\dot{\mathbf{y}} = \dot{\boldsymbol{\mu}}. \tag{35}$$

Equations (35) and (34) give four equations with four unknowns $[[D\boldsymbol{u}]]$. If we differentiate once more and use the momentum equation balance we obtain ($\boldsymbol{u} = \{u_x, u_y\}$, $\boldsymbol{n} = \{n_x, n_y\}$)

$$\nu[[\Delta \boldsymbol{u}]] = -[[Dp]],$$

$$\left\{ \begin{array}{c} [[D^2 u_x]]\dot{\mathbf{y}} \cdot \dot{\mathbf{y}} \\ [[D^2 u_y]]\dot{\mathbf{y}} \cdot \dot{\mathbf{y}} \end{array} \right\} = \ddot{\boldsymbol{\mu}} - [[D\boldsymbol{u}]]\ddot{\mathbf{y}},$$

$$\left\{ \begin{array}{c} [[D^2 u_x]]\dot{\mathbf{y}} \cdot \boldsymbol{n} \\ [[D^2 u_y]]\dot{\mathbf{y}} \cdot \boldsymbol{n} \end{array} \right\} + [[D^2 u_x]]\dot{\mathbf{y}} n_x + [[D^2 u_y]]\dot{\mathbf{y}} n_y = [[Dp]] \cdot \dot{\mathbf{y}} - [[\nu(D\boldsymbol{u} + D\boldsymbol{u}^T)]]\dot{\boldsymbol{n}}.$$

This system has six equations with six unknowns.

APPENDIX B: STENCILS FOR THE FEM DISCRETIZATION OF STOKES EQUATIONS ON A REGULAR GRID

As discussed in Section 4.2 the finite element discretization on a regular grid is equivalent to a finite difference discretization for a certain choice of stencils.

The stencils are shown explicitly in Figure 11.

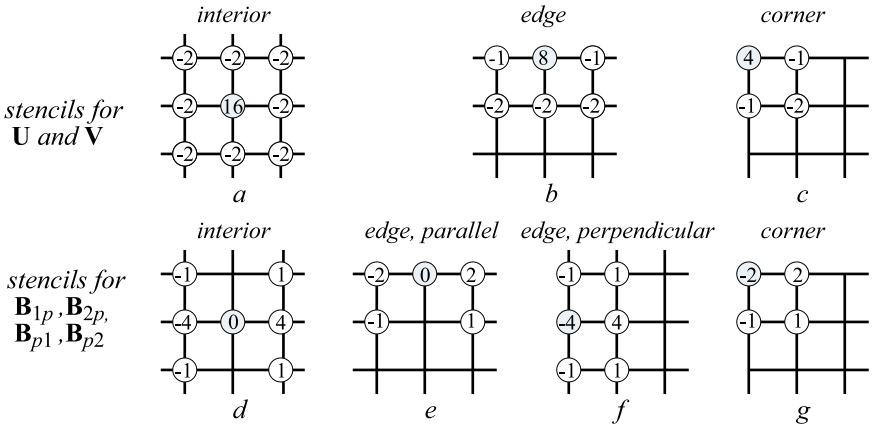


FIG. 11 Essentially different stencils of the $Q1$-$Q1$ finite element discretization. All other stencils are obtained by reflections of these stencils about vertical, horizontal and diagonal directions.
The coefficients stencils in the upper row are computed as $\int_\Omega \nabla\phi_i \nabla\phi_j \, d\Omega$ for a fixed grid point i and varying j, where ϕ_i is the $Q1$-$Q1$ node functions centered at i. The stencils in the lower row result from computing $\int_\Omega \nabla\phi_i \phi_j \, d\Omega$. The omitted scaling factor for the stencils in the upper row is $1/6h^2$, and for the lower row $1/12h$.

Interior stencils *a* and *d* are second-order accurate. Stencils *b* and *c* are used only in discretization of the stabilization term which has an extra scaling factor h^2 in front of it. Although these stencils do not approximate Laplacian, because of the scaling factor the terms in the equation corresponding to these stencils vanish as $O(h)$. Edge and corner stencils for first derivatives *e,f,g* are only first-order accurate; however these stencils are used only at the boundary in equations for pressure, hence do not affect the L^2 norm of the truncation error.

REFERENCES

[1] Michael J. Aftosmis, Marsha J. Berger, and Gediminas Adomavicius. A parallel multilevel method for adaptively refined cartesian grids with embedded boundaries. In *38th Aerospace Sciences Meeting and Exhibit*, Reno, NV, January 2000. AIAA.

[2] James F. Antaki, Guy E. Blelloch, Omar Ghattas, Ivan Malčević, Gary L. Miller, and Noel J. Walkington. A parallel dynamic-mesh Lagrangian method fro simulation of flows with dynamic interfaces. In *Proceedings of Supercomputing 2000*, Dallas, TX, November 2000. IEEE.

[3] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Lois Curfman McInnes, and Barry F. Smith. PETSc home page. http://www.mcs.anl.gov/petsc, 2001.

[4] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.

[5] George Biros, Lexing Ying, and Denis Zorin. The embedded boundary integral method for the incompressible Navier-Stokes equations. In *Proceedings of the International Association for Boundary Element Methods 2002 Symposium, CDROM*, University of Texas at Austin, Austin TX, May 28–30 2002.

[6] B.L. Buzbee, F.W. Dorr, J.A. George, and G.H. Golub. The direct solution of the discrete Poisson equation on irregular grids. *SIAM Journal on Numerical Analysis*, 8(722–736), 1971.

[7] Li-Tien Cheng, Ronald P. Fedkiw, Frederic Gibou, and Myungjoo Kang. A second order accurate symmetric discretization of the poisson equation on irregular domains. *Journal of Computational Physics*, 171:205–227, 2001.

[8] George Fix. Hybrid finite element methods. *SIAM Review*, 18(3):460–484, 1976.

[9] Aaron L. Fogelson and James P. Keener. Immersed interface methods for Neumann and related problems in two and three dimensions. *SIAM Journal on Scientific Computing*, 22(5):1630–1654, 2000.

[10] Vivette Girault and Pierre-Arnaud Raviart. *Finite Element Methods for Navier-Stokes Equations*. Springer Verlag, 1986.

[11] R. Glowinski, T.W. Pan, and J. Periaux. A fictitious domain method for external incompressible viscous flow modeled by Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 112(1–4):133–148, February 1994.

[12] J.E. Gómez and Henry Power. A multipole direct and indirect BEM for 2D cavity flow at low Reynolds number. *Engineering Analysis with Boundary Elements*, 19:17–31, 1997.

[13] Anne Greenbaum, Leslie Greengard, and G. B. McFadden. Laplace's equation and the Dirichlet-Neumann map in multiply connected domains. *Journal of Computational Physics*, 105:267–278, 1993.

[14] Leslie Greengard, Mary Catherine Kropinski, and Anita Mayo. Integral equation methods for Stokes flow and isotropic Elasticity in the plane. *Journal of Computational Physics*, 125:403–414, 1996.

[15] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[16] Max D. Gunzburger. *Finite Element for Viscous Incompressible Flows*. Academic Press, 1989.

[17] Max D. Gunzburger and Roy A. Nicolaides, editors. *Incompressible Computational Fluid Dynamics*. Cambridge University Press, 1993.

[18] Hans Johansen and Philip Colella. A Cartesian grid embedded boundary method for Poisson's equation on irregular domains. *Journal of Computational Physics*, 147:60–85, 1998.

[19] Sharad Kapur and David E. Long. IES_3: Efficient electrostatic and electromagnetic simulation. *IEEE Computational Science and Engineering*, 5(4):60–67, 1998.

[20] Sharad Kapur and Jinsong Zhao. A fast method of moments solver for efficient parameter extraction of MCMs. In *Design Automation Conference*, pages 141–146, 1997.

[21] Sangtae Kim and Sappo J. Karrila. *Microhydrodynamics: Principles and Selected Applications*. Butterworth-Heinemann, 1991.

[22] Alex Klawonn. *Preconditioners for Indefinite Systems*. PhD thesis, Courant Institute, New York University, New York, NY 10021, 1996.

[23] Randall J. LeVeque and Zhilin Li. The immersed interface method for elliptic equations with discontinuous coefficients and singular sources. *SIAM Journal on Numerical Analysis*, 31:1019–1044, 1994.

[24] Randall J. LeVeque and Zhilin Li. Immersed interface methods for Stokes flow with elastic boundaries or surface tension. *SIAM Journal on Scientific Computing*, 18:709–735, 1997.

[25] Zhilin Li. A fast iterative algorithm for elliptic interface problems. *SIAM Journal on Numerical Analysis*, 35:23–254, 1998.

[26] Zhilin Li and Ming-Chih Lai. The immersed interface method for the Navier-Stokes equations with singular forces. *Journal of Computational Physics*, 171:822–842, 2001.

[27] A.A. Mammoli and M.S. Ingber. Parallel multipole BEM simulation of two-dimensional suspension flows. *Engineering Analysis with Boundary Elements*, 24:65–73, 2000.

[28] Anita Mayo. The fast solution of Poisson's and the biharmonic equations on irregular regions. *SIAM Journal on Numerical Analysis*, 21(2):285–299, 1984.

[29] Alan McKenney, Leslie Greengard, and Anita Mayo. A fast Poisson solver for complex geometries. *Journal of Computatational Physics*, 118:348–355, 1994.

[30] Sean Norburn and David Silvester. Fourier analysis of stabilized Q1-Q1 mixed finite element approximation. Numerical Analysis Report 348, The University of Manchester, 1999.

[31] Charles S. Peskin and David M. McQueen. A three-dimensional computational method for blood flow in the heart i. immersed elastic fibers in a viscous incompressible fluid. *Journal of Computational Physics*, 81:372–405, 1989.

[32] Charles S. Peskin and B.F. Prinz. Improved volume conservation in the computation of flows with immersed elastic boundaries. *Journal of Computational Physics*, 105:113–132, 1993.

[33] Nhan Phan-Thien, Ka Yan Lee, and David Tullock. Large scale simulation of suspensions with PVM. In *Proceedings of SC97*, The SCxy Conference series, San Jose, CA, November 1997. ACM/IEEE.

[34] Henry Power. The completed double layer integral equation method for two-dimensional stokes flow. *IMA Journal of Applied Mathematics*, 51:123–145, 1993.

[35] Henry Power and L. Wrobel. *Boundary Integral Methods in Fluid Mechanics*. Computational Mechanics Publications, 1995.

[36] Costas Pozrikidis. *Boundary Integral and Singularity Methods for Linearized Viscous Flow*. Cambridge University Press, 1992.

[37] Costas Pozrikidis. Interfacial dynamics for Stokes flow. *Journal of Computational Physics*, 169:250–301, 2001.

[38] Wlodzimierz Proskurowski and Olof B. Widlund. On the numerical solution of Helmholtz's equation by the capacitance matrix method. *Mathematics of Computation*, 30(135):433–468, 1976.

[39] Robert J. Renka. Algorithm 790: CSHEP2D : Cubic Shepard method for bivariate interpolation of scattered data. *ACM Transactions on Mathematical Software*, 25(1):70–73, 1999.

[40] Andreas Wiegmann and Kenneth P. Bube. The explicit-jump immersed interface method: Finite difference methods for PDEs with piecewise smooth solutions. *SIAM Journal on Numerical Analysis*, 37(3):827–862, 2000.

[41] Alexander Z. Zinchenko and Robert H. Davis. An efficient algorithm for hydrodynamical interaction of many deformable drops. *Journal of Computational Physics*, 157:539–587, 1999.