Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 193 (2003) 317–348
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

A fast solver for the Stokes equations with distributed forces in complex geometries ☆

George Biros *, Lexing Ying, Denis Zorin

Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA

Received 16 July 2002; received in revised form 5 August 2003; accepted 11 August 2003

Abstract

We present a new method for the solution of the Stokes equations. The main features of our method are: (1) it can be applied to arbitrary geometries in a black-box fashion; (2) it is second-order accurate; and (3) it has optimal algorithmic complexity. Our approach, to which we refer as the embedded boundary integral method (EBI), is based on Anita Mayo's work for the Poisson's equation: ''The Fast Solution of Poisson's and the Biharmonic Equations on Irregular Regions'', SIAM Journal on Numerical Analysis, 21 (1984) 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 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 an $N \log N$ algorithm based on a block representation of the discrete integral operator, combined with (kernel independent) singular value decomposition to sparsify low-rank blocks. 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.

Keywords: 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

☆ This work is supported by the National Science Foundation's Knowledge and Distributed Intelligence (KDI) program through Grant DMS-9980069.
* Corresponding author.
E-mail addresses: biros@cs.nyu.edu (G. Biros), lexing@cs.nyu.edu (L. Ying), dzorin@cs.nyu.edu (D. Zorin).

0021-9991/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.jcp.2003.08.011

1. Introduction

The Stokes equations model very low Reynolds number flows and incompressible linearly elastic solids. They also serve as building blocks for solvers for the velocity–pressure formulation of the Navier–Stokes equations. In this paper we present a method for solving the steady two-dimensional Stokes equations in irregular domains. Our motivation in developing this method is to develop efficient algorithms for flows with moving boundaries. Solving such problems efficiently requires algorithms that do not require expensive preprocessing, like unstructured mesh generation, as the boundary positions change at each time step.

The main features of our method are:

- It can be applied to arbitrary piecewise smooth geometries; the method does not require mesh generation.
- It can solve problems with distributed forces.
- It is second-order accurate and easily generalizes to arbitrary order of accuracy.
- If an optimal boundary integral equation solver is used, the method has $\mathcal{O}(N \log N)$ complexity.[1]

In addition, robust parallel algorithms exist for all the components of this method, both in two and three dimensions. Finally, the method is relatively straightforward to implement.

Our method is based on potential theory for linear second-order elliptic operators. Using an indirect integral formulation, the solution of a Dirichlet problem can be written as the sum of a double layer potential and a *Newton potential* (the domain convolution of Green's function with the distributed force). Under such a scheme the evaluation of the solution must consist of three steps: (1) computation of the Newton potential, (2) solution of a boundary integral equation to compute a double layer potential that satisfies the boundary conditions, and (3) evaluation of a double layer potential everywhere in the domain.

In theory, all steps can be performed with optimal complexity using the fast multipole method (FMM) [21,51]. Only one step requires solving an equation, the other two steps are evaluations of integrals.

However the are several practical problems that have prevented the broad application of approaches of this type. FMM depends on analytic kernel expansions; computing these expansions may be computationally expensive, which results in large constants in method's complexity, which can easily negate the asymptotic advantages of the method even for problems of moderate size. Because work-efficient, highly accurate implementations are far from trivial. This is the reason that such implementations exist only for the Laplace and Helmholtz operators. For example, an efficient scheme for the three-dimensional Laplacian was developed more ten years after the original method was introduced [22]. Furthermore, the complexity constants of the FMM are high, and often, *for problems that do not require adaptive discretization*, regular grid schemes like multigrid or FFT are much faster [19,39,44]. Finally, due to singularities, computing the near field interaction, either for the Newton potential or for evaluations near the boundary can involve increased computational costs and implementation complexity, and is an active research topic [5,12,19,38,54].

For these reasons we have opted to use a different approach in order to compute the Newton potentials and to evaluate the solution everywhere in the domain. Our method, to which we refer to as the embedded boundary integral method (EBI), is an extension of Anita Mayo's work [37], for the Laplace and biharmonic operators. This approach was also used in [39] in combination with fast multipole methods for the boundary integrals only, and was extended in three dimensions [19], again for the Laplacian operator. Instead of using direct integration, we use an efficient regular grid solver for the first and last steps. Note that the asymptotic complexity of the computation does not increase, as the solver is guaranteed to converge in a fixed number of steps for a fixed precision.

[1] In this paper we present an $\mathcal{O}(N \log N)$ variant, however the method can be changed to an $\mathcal{O}(N)$ algorithm – by switching to a classical FMM method for the boundary integral solver.

More specifically, the method handles the three steps enumerated above as follows. The flow domain is is embedded in a larger simple rectangular domain, which can be easily refined to obtain a structured regular grid. *At the first step*, we extend the distributed force to this simple domain and then solve the regular grid problem. *At the second step,* we solve a boundary integral equation for the boundary conditions. This equation yields values for velocity and pressure jumps across the boundary of the original domain, which is now regarded as an interface. *The third step* step is evaluation the velocities and pressures everywhere in the domain. Another regular grid solve is used; however, this time, Taylor expansions are used to express velocity and pressure 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 call Taylor expansion stencil correction (TESC). In greater detail, the steps are discussed in Section 2.

The fact that we use *regular grid* solvers is of crucial importance for making the method more practical and suitable for applications where the boundaries change. First, the domain does not depend on the geometry of the problem, and no mesh generation is required. Second, unlike the case of unstructured meshes, relatively simple and highly efficient multilevel preconditioners with well-understood properties are readily available.

In this paper we apply 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 [6].

An additional benefit of solving the equations on the interface (second step) 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.

1.1. Related work

1.1.1. Finite elements

Undoubtedly finite element methods are one of the most successful discretization methods and allow for the accurate solution of problems in arbitrary geometries. Nevertheless, application of such methods to problems with complex geometries has several difficulties; two main difficulties are mesh generation and design of efficient solvers.

First, for large-scale problems, especially 3D problems and problems with moving boundaries, mesh generation takes a significant fraction of the total computation time. As parallel computation is necessary to solve realistic problems with sufficient accuracy, mesh generation should be also done in parallel. Furthermore, for problems with moving boundaries, the mesh needs to be frequently regenerated, which requires a robust purely automatic mesh generator which guarantees element quality. Although a lot of progress was made in recent years [2,26] at this time, we are not aware of any algorithm that meets all of these requirements: i.e. does not require user intervention, has an efficient and robust parallel implementation and guarantees element quality both in two and three dimensions. In fact, in 3D, even sequential guarantee-quality mesh generation for arbitrary boundaries remains an area of active research. The state of the art is discussed in [33,52,53].

Second, once the discretized operators have been constructed, optimal complexity inversion algorithms require multilevel techniques such as multigrid or domain decomposition [9,10], which extend to unstructured grids. However, for unstructured meshes, the main problem is related to automatically

generating quality coarse meshes from the user-specified mesh. Algebraic multigrid, a method that does not require multilevel meshing, has been successfully applied to scalar positive definite operators, but has not been extended to vector PDEs or indefinite operators like the Stokes equations [55].

1.1.2. Cartesian grid methods

State-of-the-art methods for problems in complex geometries, most often found in applications with dynamic interfaces, are based on regular grid or Cartesian grid discretizations, due to their efficiency, parallel scalability, and implementation simplicity.

Research on this topic dates back to the seventies [8]. 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 [49]. This method solves Neumann and Dirichlet problems 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 not for the Dirichlet problem, since the double-layer approximation results in well-conditioned problems for the unknown interface jumps. One shortcoming of the method is the requirement of a variable number of regular grid solves.

One of the most successful techniques is the *immersed boundary method* [41,42] which was designed for a Poisson problem 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* [31] 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 [32] and for the Navier–Stokes problem [35]. 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 matrices with non-standard structure and fast methods are not straightforward to apply. The immersed interface method as presented in [31] was not used on Dirichlet and Neumann problems in general irregular regions, since it requires known jump conditions. In [14], IIM is extended to Neumann problems by modifying interface stencils to account for the unknown jumps. Later versions of IIM (explicit immersed interface method) [56], (fast immersed interface method) [34], addressed non-standard matrices by adding additional equations for the jumps and extended IIM to Dirichlet problems. These approaches however appear to result in considerable additional computational cost since they require tens to hundreds of regular grid solves.

Next, we examine several groups of methods which share some common features, including finite-volume and fictitious domain methods. These methods produce discretizations based on regular grids, with modified stencils and/or right-hand sides to account for the embedded interfaces. Cheng and Fedkiw [11] 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 *Cartesian finite-volume methods*, the stencils modifications are derived from appropriate modification of finite-volume cells to account for the intersections of the Cartesian grids with the interface [1,25].

An algorithm, similar to the IIM and capacitance matrix methods but which first appeared within the finite element community, is the *fictitious domain* method [13,16]. 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.

In our examination of the above methods, 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, interface 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). However, modified stencils make it more difficult to apply fast solvers, especially if the boundaries are moving. If additional unknowns are used, a common approach to solving the resulting system is to invert the Schur complement corresponding to these unknowns. These Schur complement matrices correspond to discretizations of integral equations [49]. A matrix-vector multiplication with such matrix will be expensive since it involves a regular grid solve.

The EBI method computes the jumps directly via boundary integral equations, and circumvents costly computations used by other methods by decoupling the interface from the regular grid. Only one integral solve and two regular grid solves are required independently of the complexity of the interface. The main downsides of this approach is that it is restricted to piecewise constant coefficient problems, and that for a scalable and efficient implementation a fast dense matrix multiplication algorithm is needed. 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, but for optimal asymptotic complexity an FMM-type method is required, with implementation details depending on the choice of the kernel.

1.1.3. Integral formulations for the Stokes problem

Several researchers have used boundary integral formulations to solve the homogeneous Stokes problem. The basic formulation can be found in [29,46,47]. In [17,36,45], the homogeneous Stokes problem is solved using boundary integral representation combined with multipole-like far-field expansions to accelerate the matrix-vector multiplications. In [43,48,57] boundary integral equations have been used for problems with moving and deforming boundaries. In [20] the homogeneous Stokes problem is posed as a biharmonic equation and it is solved for both interior and exterior problems. Inhomogeneous problems however, cannot be reduced to a pure boundary integral formulation. As discussed above, evaluation of Newton potentials is required.

1.2. Overview and notation

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. In Section 6 we conclude with numerical experiments that demonstrate the accuracy of the method.

1.2.1. 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

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

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

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

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

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

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

The 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 hydrodynamic density, $\boldsymbol{\mu}$, on the boundary γ. The solution $\boldsymbol{u}_2$ for an arbitrary point in the interior of ω is the convolution of the double layer kernels with the density. 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 [37], 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

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

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

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 density (Section 4.3).

In summary, the EBI approach uses the following steps:

1. Solve problem (3) on the simpler domain Ω using multigrid or two-level domain decomposition method.
2. Solve the boundary integral problem derived from (4) on γ to obtain the velocity density, using an integral equation formulation with mesh independent condition number, spectrally accurate discretization and a kernel-independent fast multipole acceleration.
3. Compute the right-hand side corrections from the velocity density.
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 integral formulation of a problem of the form (4). We have opted to use a primitive variable formulation instead the formulation used in [20] since it can be extended

without modifications in the three-dimensional case. We assume that the boundary curve γ is curvature-continuous, and the domain ω is bounded. We use the notation

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

to denote the convolution for a kernel $\mathscr{C}$; $\mathscr{C}(\mathbf{x},\mathbf{y})\boldsymbol{w}$ 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

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

$\mathbf{x}$ 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 $\mathscr{D}$ for velocity and the double layer potential for pressure $\mathscr{K}$:

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

where $\boldsymbol{n}(\mathbf{y})$ is the outward surface normal at a boundary point $\mathbf{y}$. For a derivation of [46,47].

We limit our discussion to the interior Dirichlet problem. Its extension to exterior problems is trivial. We have opted to use an indirect double layer formulation for the Dirichlet Stokes problem. This approach results in a Fredholm equation of second kind. Combined with Nÿstrom discretization it results in linear systems with bounded condition number, and it is superalgebraically convergent for analytic geometries. Second kind formulations are also well known for the Neumann problem.

We represent the velocities and pressures as surface potentials convolved with the double layer kernel

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

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

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

The velocity $\boldsymbol{u}$ has to satisfy $\int_\gamma \boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}\gamma = 0$, 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 ([46], p. 159) that the dimension of the null space is exactly one. The null space can be removed by a rank-one modification ([46], p. 615). Let $\mathscr{N}(\mathbf{x},\mathbf{y}) = \boldsymbol{n}(\mathbf{x})\otimes\boldsymbol{n}(\mathbf{y})$. We represent $\boldsymbol{u}$ as

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

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 $\gamma_0,\gamma_1,\ldots,\gamma_n$, where γ_0 encloses all other components, and let $\mathbf{c}_p, p = 1,\ldots,n$ be an interior point of γ_p. Following [46], we represent $\boldsymbol{u}$ as

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

where $\mathbf{r}_p = \mathbf{x} - \mathbf{c}_p$, $\mathscr{R}(\mathbf{r}) = \mathbf{r}^{\perp}/4\pi\rho^2$, 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

$$\begin{aligned} &\int_{\gamma} \boldsymbol{\psi}_p^j(\mathbf{y}) \cdot \boldsymbol{\mu}(\mathbf{y})\, \mathrm{d}\gamma(\mathbf{y}) = \boldsymbol{\alpha}_p, \quad j = 1, 2, \\ &\int_{\gamma} \boldsymbol{\psi}_p^3(\mathbf{y}) \cdot \boldsymbol{\mu}(\mathbf{y})\, \mathrm{d}\gamma(\mathbf{y}) = \beta_p. \end{aligned} \tag{13}$$

Here $\boldsymbol{\psi}_p^i,\ p = 1, \ldots, n,\ i = 1, 2, 3$ 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 [46] is shown that Eqs. (12) and (13) guarantee a unique solution of $\boldsymbol{\mu},\ \boldsymbol{\alpha}$ and β for general admissible boundary condition $\boldsymbol{u}$.

3.1. Jump computation

Once the density $\boldsymbol{\mu}$ is known, we need to compute the jumps at the interface and the velocity to use in Eq. (5) Eq. (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 $\mathscr{K}(\mathbf{x}, \mathbf{y})$ can be also written as $-2\nu\nabla_{\mathbf{x}}(\mathscr{L}(\mathbf{x}, \mathbf{y}))$, where $\mathscr{L}(\mathbf{x}, \mathbf{y}) = (\mathbf{r} \cdot \boldsymbol{n}(\mathbf{y}))/\rho^2$ is the double layer kernel for the Laplace equation.

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

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

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 with 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,

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

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

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

or

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

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 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 [23,24].

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

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

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 [7].

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

[2] $P1$–$P1$ could also have been used, but the implementation is somewhat more sensitive on the stabilization parameter [40].

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

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

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 [23]. To ensure convergence, the well-known (`inf–sup` condition) needs to be satisfied, which is not satisfied by the $Q1$–$Q1$ element. The weighted diffusive term in (18) is introduced to circumvent the `inf–sup` condition [23]; parameter β controls the amount of stabilization. In [40] 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

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

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). 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

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

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

If we denote $\| \cdot \|_m$ the usual norm in $\boldsymbol{H}^m(\Omega)$, standard regularity results [15] give

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

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

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

Now let $w_h = \{\boldsymbol{u}_h, p_h\}$ and $b_h = \{\boldsymbol{f}_h, g_h\}$. We can associate a linear operator $\mathscr{A}_h$ to (20), mapping w_h to b_h; since (20) has a unique solution for all b_h, we can also write $w_h = \mathscr{A}_h^{-1} b_h$. The regularity condition (22)

implies that $\|w_h\|_0 \leqslant \|b_h\|_0$ and thus $\|A_h^{-1}\|_0 \leqslant \infty$. Then if e_h is the approximation error and τ_h the truncation error, we get $e_h = \mathscr{A}_h^{-1}\tau_h$, or $\|e_h\|_0 \leqslant \|\mathscr{A}_h^{-1}\|_0\|\tau_h\|_0$. 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. 1). 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 ith 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. 1(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 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. 1(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. 1(b). By using Taylor expansions we can write

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

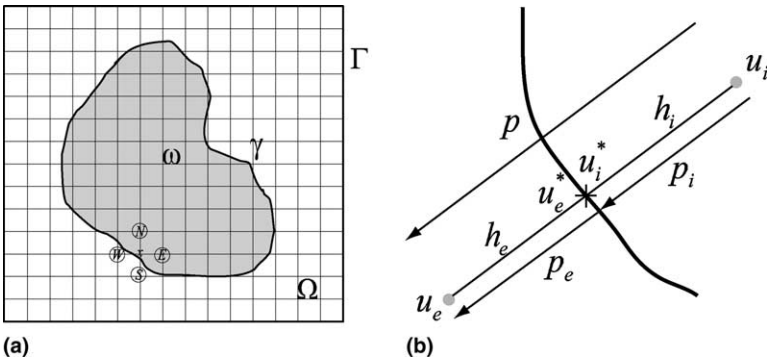

Fig. 1. Stencil corrections. (a) The irregular domain ω is embedded in a simpler domain Ω. For the depicted stencil the truncation error is constant as the discretization step decreases. (b) shows the notation for computing the correction terms.

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

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

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 in an $\mathcal{O}(h^2)$ discretization error for all points [31,37]. 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 are 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. 1) 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 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 [50] 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 density 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 density. First we compute the intersections of γ with the regular grid, using a standard Bezier-clipping algorithm. Then, using the hydrodynamic density we evaluate the correc-

tion 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 density. 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 [18] 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

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

$\mathbf{u}$, $\boldsymbol{\mu}$ and $\mathbf{n}$ 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 $-\frac{1}{2}\mathbf{I} + \mathbf{DJW} + \mathbf{n}\mathbf{n}^{\mathrm{T}}\mathbf{JW}$. 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 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 [27,28] for the single layer formulation of the Laplace equations in triangulated domains. This method uses the 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 Fig. 2.

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 $\mathbf{D}$. The singular value decomposition computes a coordinate transformation, for which $\mathbf{D}$ is diagonal, and eliminates the vectors corresponding to the small

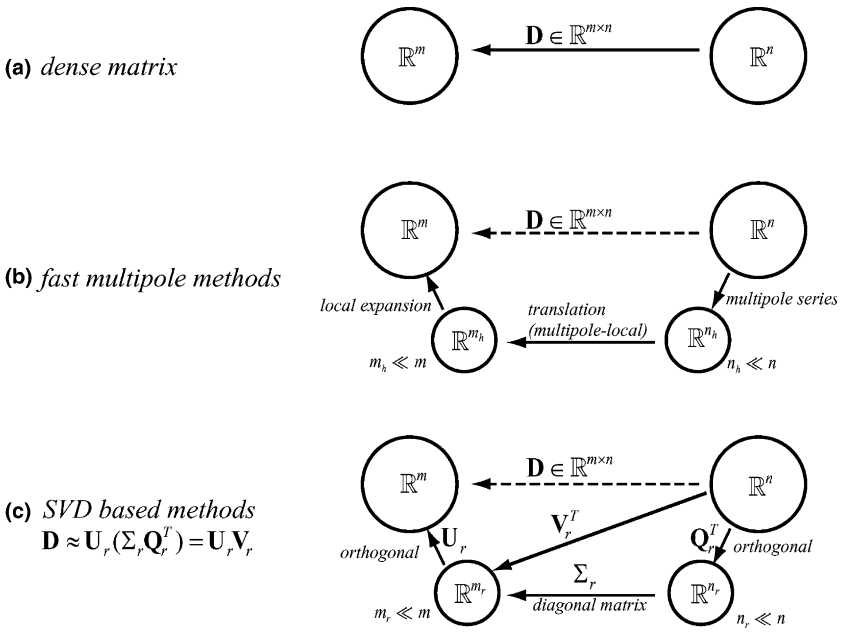

Fig. 2. Low rank approximations of discrete interaction. (a) The dense interaction. (b) Fast multipole method. (c) SVD-based method. Arrows represent linear transformations.

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 [27] 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.

5.1.1. 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, $i = 0, \ldots, n_s - 1$, 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, 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.

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 construction of **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
```

Algorithm 1 is the pseudocode for constructing the matrix **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 sparse 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 *constructSegment-Tree(X,segment_list)* construct a tree representation of the submatrix of **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:

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

(Fig. 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 $\tilde{\mathbf{U}}\mathbf{V}_r = \mathbf{D}_{S_X,Y}$.

5.1.2. 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 $c(left(X))$ with radius $(2\alpha+1)r(left(X))$ because $r(left(X))$ is half the size of $r(X)$, and the same for $left(X)$. We use g to denote a bound on $2\alpha+1$.

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 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

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

Algorithm 2 Matvec of 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
```

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 $\mathcal{O}(N \log N)$.

5.1.3. Matrix-vector multiplication

Algorithm 2 is the pseudocode 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 $\mathbf{D}_{X,Y}$. Fig. 3 shows the sparse structure of a simply connected boundary.

5.1.4. Numerical experiments

All experiments in this section were performed on a Sun 450 MHz Ultra80 workstation. 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 1.5–2. Notice that unlike Barnes-Hut or fast multipole algorithms, α does not have an effect on the accuracy of the algorithm. In our context is just used as an oracle to activate the low-rank approximation. 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. Assuming the SVD was computed exactly (without sampling) ϵ is the relative error in the potential calculation. Once ϵ is chosen, SVD automatically selects the number of required moments to meet the error criterion.

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

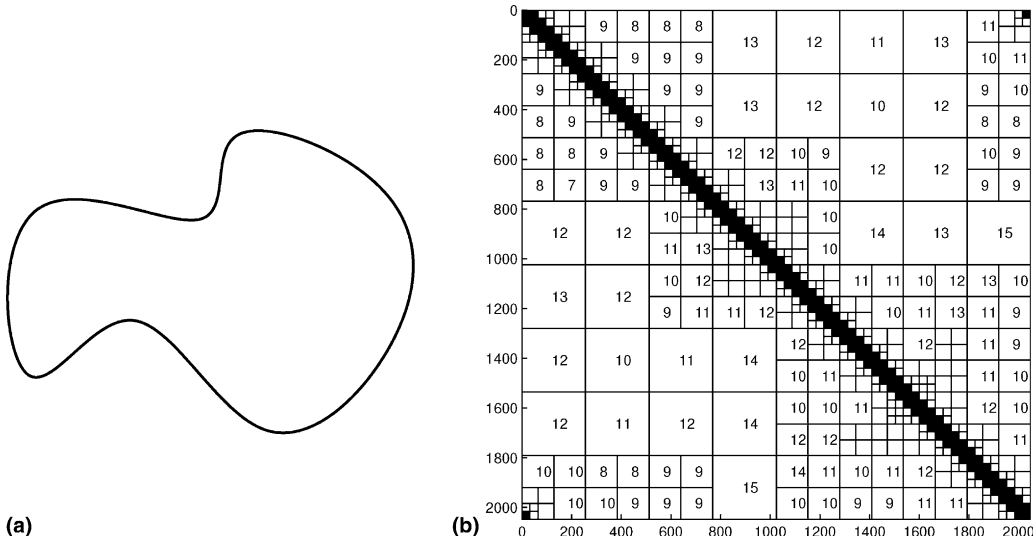

Fig. 3. The sparsity structure of matrix $\mathbf{D}_{X,Y}$ of a boundary curve. The curve is discretized into 1024 quadrature points. The matrix is of size 2048×2048. The number in each block denotes its numerical rank r. The black blocks along the diagonal correspond to close interaction, and thus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

smaller than 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

Table 1
Comparison between the dense matrix matrix-vector multiplication and the SVD-based matrix-vector multiplication for two different flow fields and geometries

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

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.

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 density 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 increase almost linearly with the problem size (cf. Fig. 5).

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 4 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

Table 2
Running times and pointwise errors for the SVD-based sparsification

ϵ	Setup(s)	Solve(s)	$\lvert\mathbf{u}\rvert_{\text{err}}$	p_{err}	Max rank
10^{-2}	3.48	7.13	3.95×10^{-4}	1.84×10^{-4}	8
10^{-3}	4.18	8.09	3.67×10^{-5}	1.43×10^{-5}	10
10^{-4}	5.49	7.95	6.68×10^{-6}	4.63×10^{-6}	12
10^{-5}	6.00	8.59	8.31×10^{-7}	5.82×10^{-7}	14
10^{-6}	6.99	9.71	1.77×10^{-7}	9.49×10^{-8}	16
10^{-7}	7.93	10.8	1.17×10^{-7}	4.65×10^{-8}	18

We report results for the geometry depicted in Fig. 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.

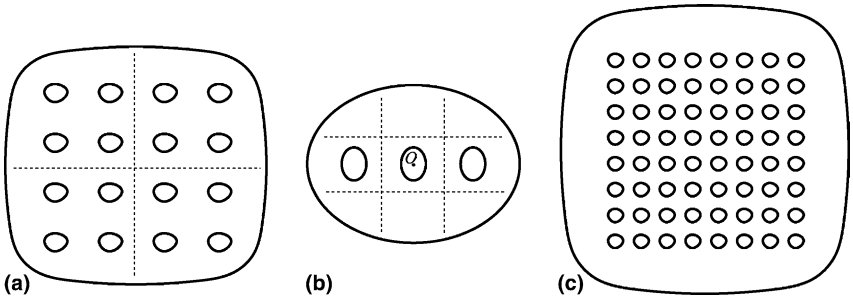

Fig. 4. Test domains. Solid curves represent the boundary of the domain. The dots in the domains are the points used for error estimation.

Table 3
Running times and pointwise errors for the SVD-based sparsification

ϵ	L/N	Setup(s)	Solve(s)	$\|\mathbf{u}\|_{\text{err}}$	p_{err}	Max rank
10^{-4}	4/4544	49.6	111	9.02×10^{-6}	1.63×10^{-5}	14
	5/9088	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^{-6}	4/4544	67.2	137	4.64×10^{-7}	1.11×10^{-6}	19
	5/9088	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	1172	1.23×10^{-7}	1.57×10^{-7}	19

We report results for the geometry depicted in Fig. 4(c) (64 circles) for a Stokeslet flow; for two different values of the numerical rank tolerance ϵ and for an eightfold 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.

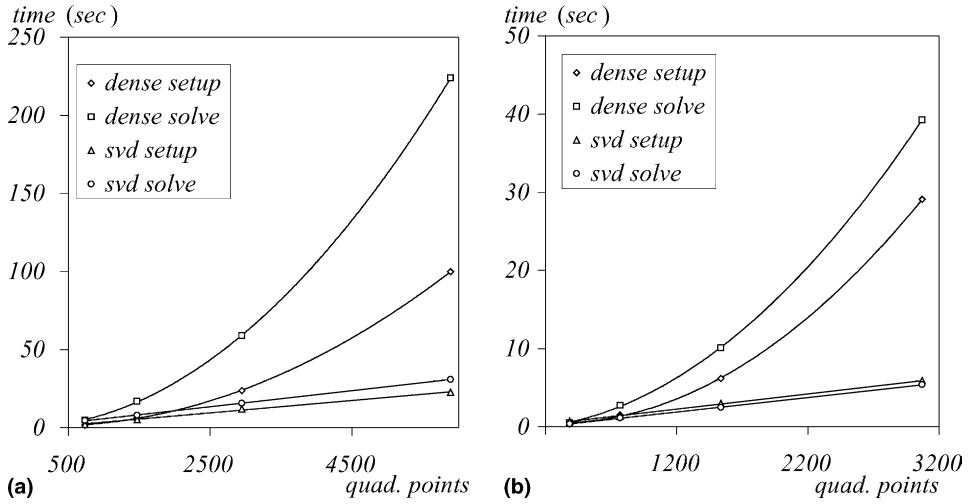

Fig. 5. Plots of the data from Table 1 with linear fit for SVD-based solver and quadratic fit for the dense solver.

Table 4
In this table we compare iteration count and wall-clock time for four different linear solvers for the discretized Stokes problem

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	1240	1450	692	950	18	11	56	98
1024^2	16	2578	6100	1391	3910	19	24	57	260

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 five-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 s 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.

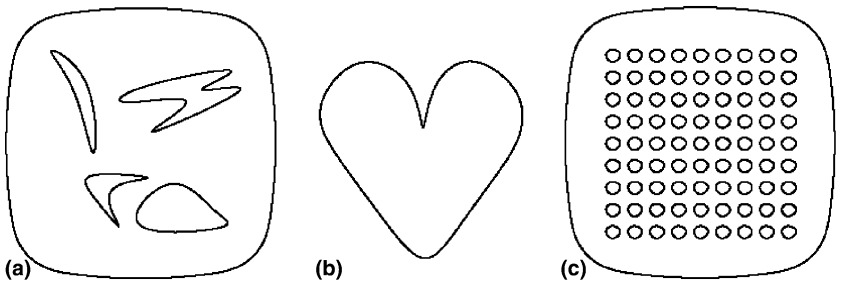

Fig. 6. Domains used in numerical examples.

10 million unknowns on 16 processors. Our intention is not a detailed comparison between the different solution techniques, but to give an numerical evidence of the scalability of the different preconditioners for the $Q1$–$Q1$ discretization.

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 [30].

Finally by comparing Table 4 with Tables 1 and 2, we can observe that for simple geometries the regular grid solver takes similar time with the boundary integral solver. For more complicated geometries the cost of the regular grid solve becomes quite small compared to the boundary integral solver. Indeed for the 64 spheres problem, the boundary integral setup and solve requires 160 s for the lowest accuracy (Table 3), whereas the regular grid solver for the 128^2 problem, takes 3 s on two processors on the Alpha server which correspond to less than 12 s per solve on the Sun workstation. Since our method requires two regular grid solves, the time spent to calculate the distributed terms and the solution everywhere is 10% of the time spent on the boundary integral solver. [3]

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.

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

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

a "cubic flow"

[3] The time required on computing the jumps, the intersections and the derivatives of the hydrodynamic densities is negligible compared to the domain an boundary solvers.

Table 5
Convergence results for the cubic flow inside a circle

grid	*dense-1*		*dense-2*		*svd*	
	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 (*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.

$$u = \{y^3, x^3\}, \quad p = 6xy,$$

a "body force flow"

$$u = 2\{-x^2y, y^2x\}, \quad p = \sin(xy), \quad b = 4v\{y(1+\cos(xy)), -x(1+\cos(xy))\}.$$

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, 450 MHz 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 was 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.

In the second example we repeat the same test, but for the geometry depicted in Fig. 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 3072 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 Fig. 7 (for the Poiseuille flow) (cf. Table 6).

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, Fig. 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 Fig. 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

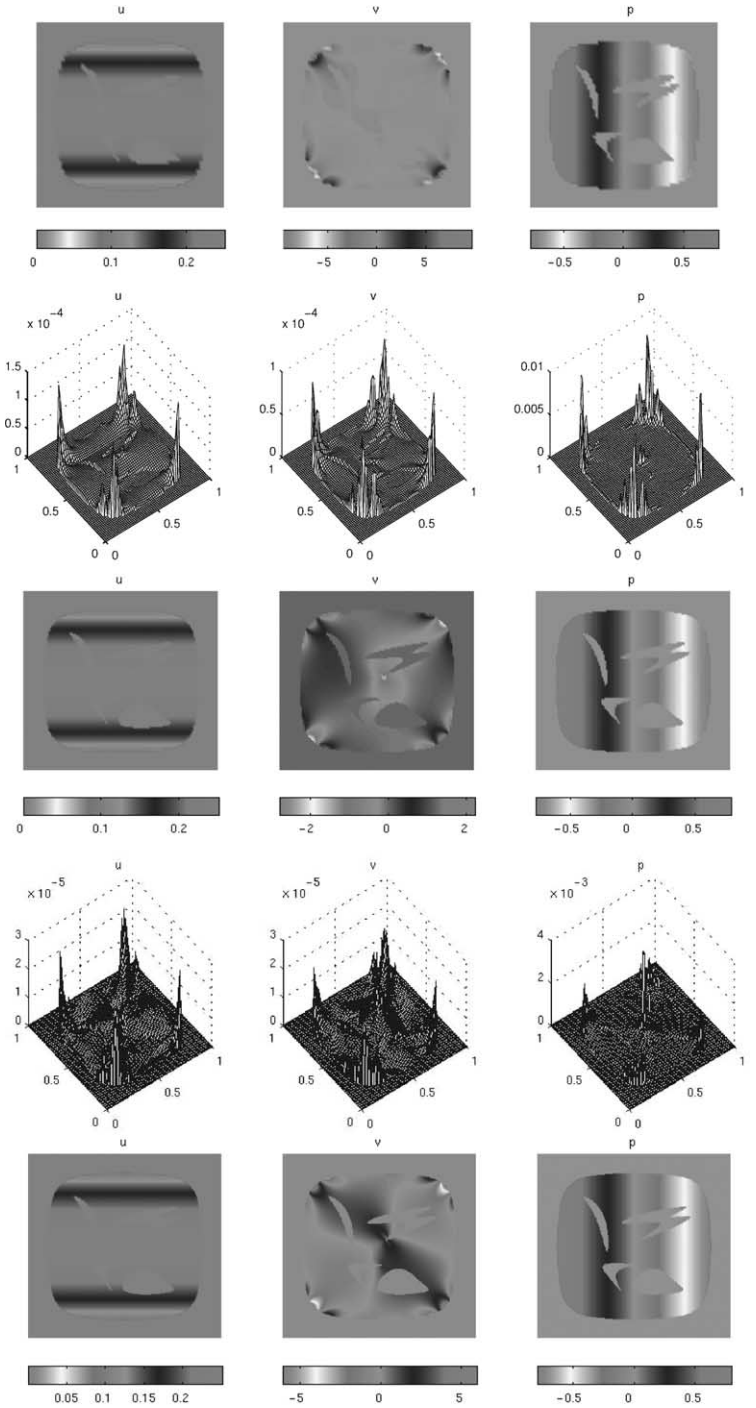

Fig. 7. Exact solution and error distribution (top to bottom), for 64^2, 128^2, and 256^2; the error plot for the 256^2 grid is omitted. The solution is a Poiseuille flow.

Table 6
Convergence results for two flows in the domain Fig. 6(a)

grid	*dense-1*		*dense-2*		*svd*	
	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}

Here (*dense-1*), (*dense-2*), (*svd*) and (**u**), (**p**) are as in Table 5.

Table 7
Convergence results for a stokeslet flow generated by a pole just outside the domain

grid	*dense-1*		*svd-1*		*svd-2*	
	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}

Here the jumps are second-order accurate. All problems use 1664 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.

experimentally based on dense solves; nearly 800 points are enough to resolve the problem; in this test we took 1664 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 Fig. 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 9088 Nyström points and for the two finer grids we use 18,176 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

Table 8
Convergence rates and pointwise accuracy for the 81-circle geometry and for the "body-force" flow

grid	*svd-1*		*svd-2*		*svd-3*	
	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}

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*.

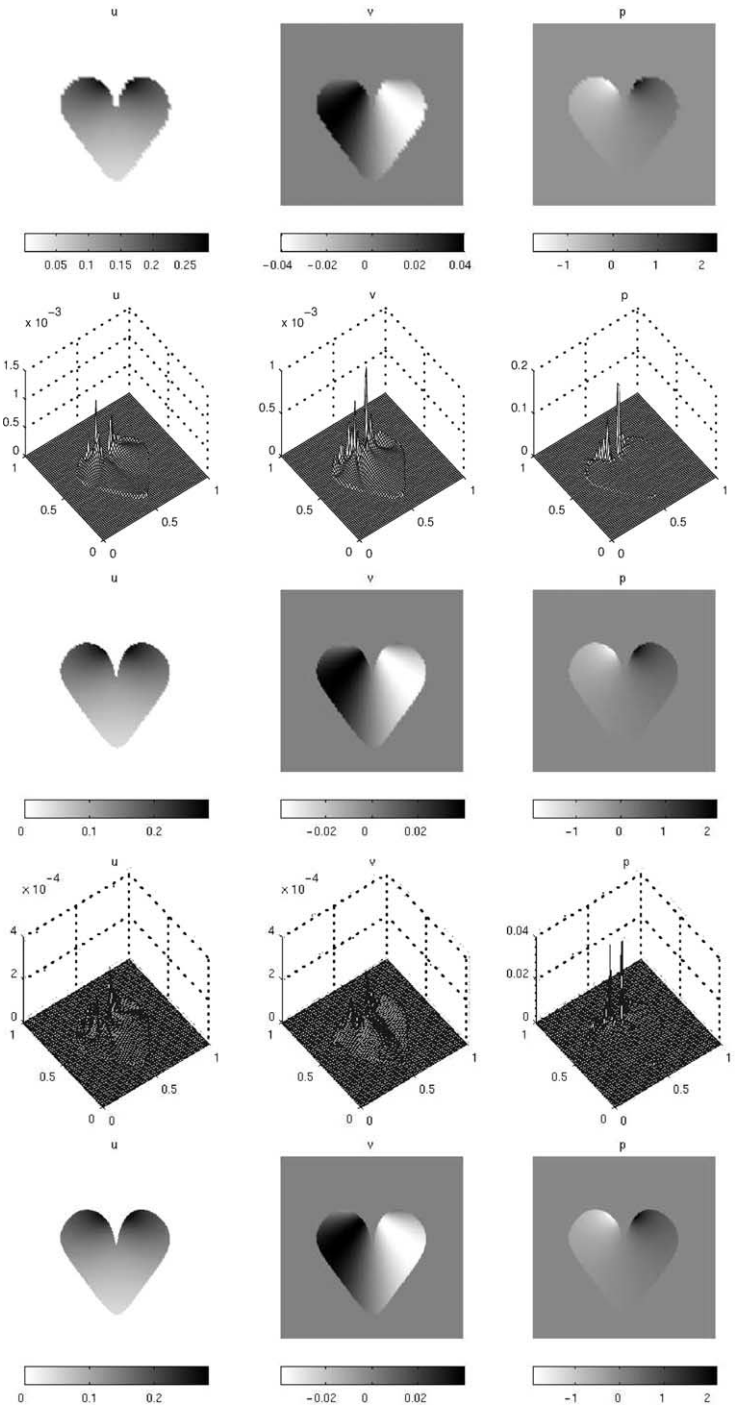

Fig. 8. Exact solution and error distribution (top to bottom), for 64^2, 128^2, and 256^2; the error plot for the 256^2 grid is omitted. The solution is the Stokeslet located at (0.5,0.7).

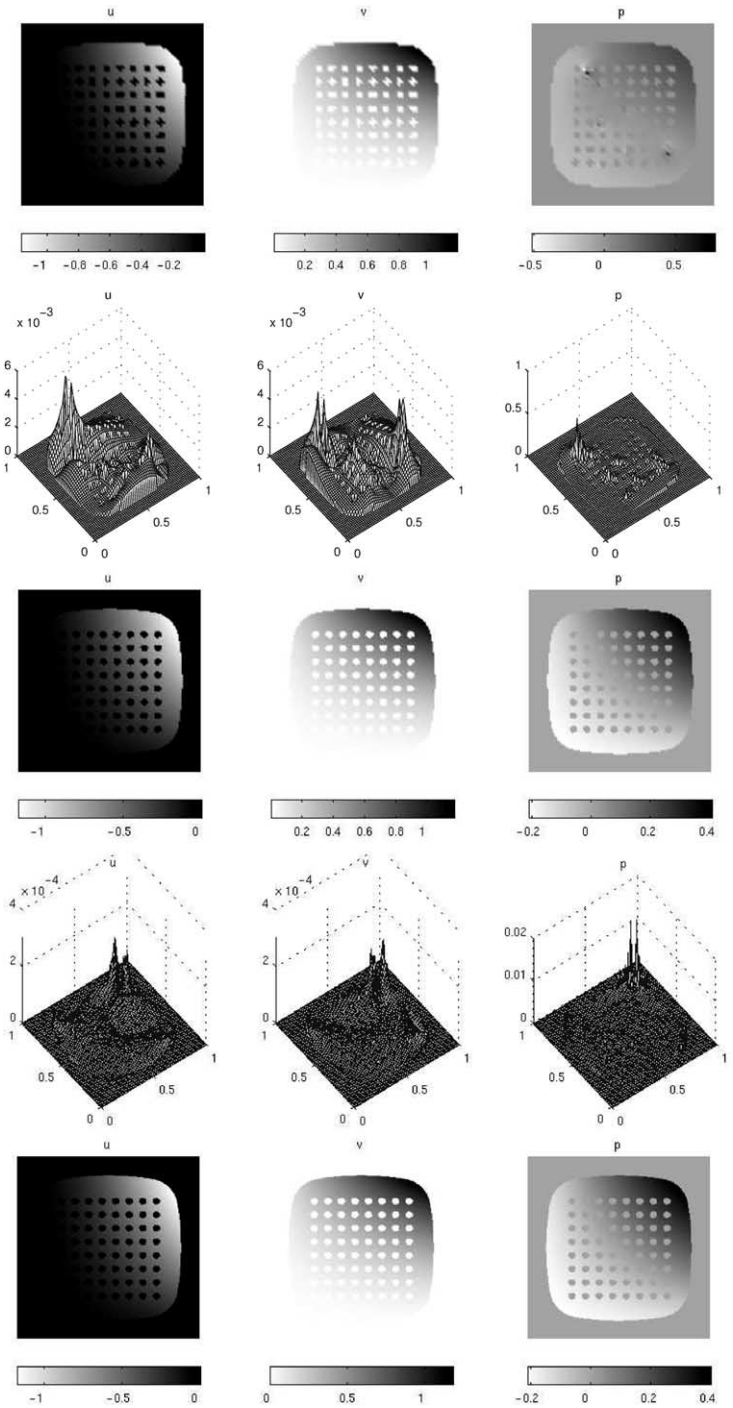

Fig. 9. Exact solution and error distribution (top to bottom), for 64^2, 128^2, and 256^2; the error plot for the 256^2 grid is omitted. The exact solution is a the body force flow.

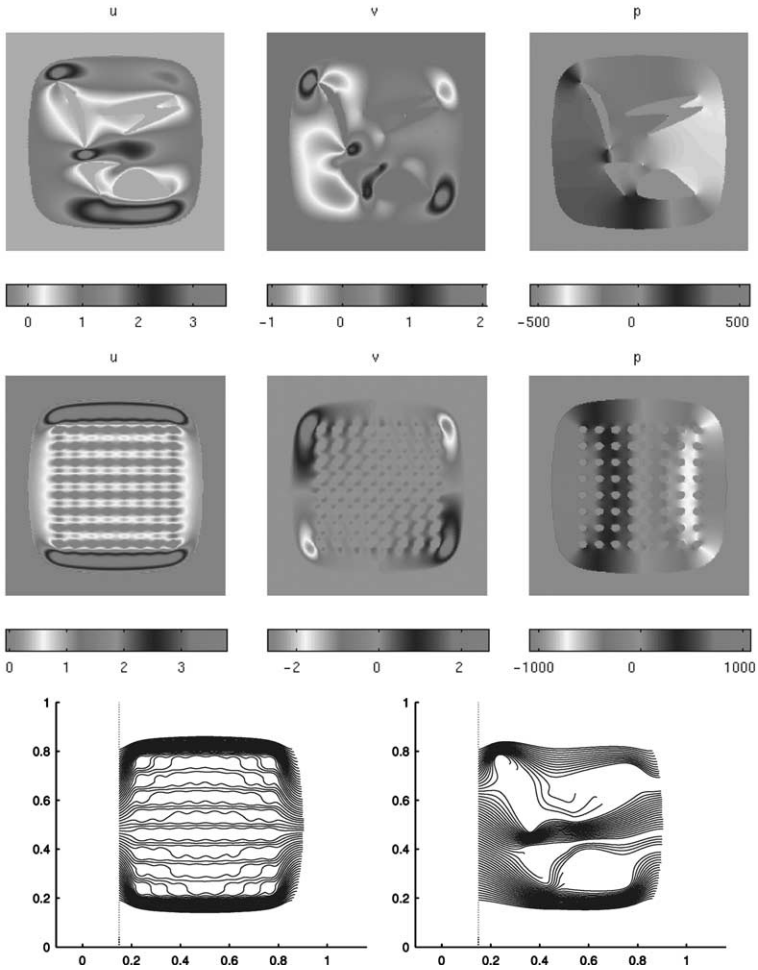

Fig. 10. Solution for a problem with Dirichlet conditions corresponding to a unit wind flow, presented for two different geometries. The two bottom pictures depict the resulting streamlines.

convergence study. Optimal rates are obtained for the most accurate representation of the double layer. Fig. 9 depicts the exact solution and the error distribution. For the last example we do not have an analytic solution, and we just plot the solution in Fig. 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 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. In the case of geometries with corners the singularities can be resolved analytically and their contribution to the jump terms can be directly evaluated at the stencil points. In 3D this is no longer possible. However this can be circumvented by replacing the jump computation by direct evaluation. For example the jump terms can be computed to very high 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.

Another limitation 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.

Higher order accurate extensions are possible with further differentiation of the hydrodynamic density and use of high-order or spectral regular grid discretizations.

Acknowledgements

We thank L. Greengard, M. Shelley and C. Peskin 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

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

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:

$$[[q]] = \phi, \tag{A.1}$$

$$[[Dq \cdot n]] = 0. \tag{A.2}$$

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

$$[[Dq]] \cdot \dot{\mathbf{y}} = \dot{\phi} \tag{A.3}$$

for the tangential derivative. Eqs. (A.2) and (A.3) 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

$$[[\Delta q]] = 0, \tag{A.4}$$

$$[[D^2 q]]\dot{\mathbf{y}} \cdot \dot{\mathbf{y}} + [[Dq]] \cdot \ddot{\mathbf{y}} = \ddot{\boldsymbol{\phi}}, \tag{A.5}$$

$$[[D^2 q]]\dot{\mathbf{y}} \cdot n + [[Dq]] \cdot \dot{\boldsymbol{n}} = 0. \tag{A.6}$$

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

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

Let q_i be given by

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

with

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

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 (A.1) and (A.2). These relations can be derived by taking appropriate limits across the interface [46]. In fact, if the velocity is given by

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

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

$$[[u]] = \boldsymbol{\mu}, \tag{A.7}$$

$$[[Sn]] = [[-p\mathbf{I} + v(Du + Du^{\mathrm{T}})n]] = 0. \tag{A.8}$$

In order to construct TESCs for the momentum and incompressibility equations we need to compute $[[Du]]$ and $[[D^2u]]$. (We already have $[[p]]$ and $[[Dp]]$.) If we differentiate (A.7) (with respect the curve parameterization t), we obtain

$$[[Du]]\dot{\mathbf{y}} = \dot{\boldsymbol{\mu}}. \tag{A.9}$$

Eqs. (A.9) and (A.8) give four equations with four unknowns $[[Du]]$. If we differentiate once more and use the momentum equation balance we obtain ($u = \{u_x, u_y\}$, $n = \{n_x, n_y\}$)

$$\begin{aligned}
&v[[\Delta u]] = -[[Dp]],\\
&\left\{\begin{array}{l}[[D^2u_x]]\dot{\mathbf{y}}\cdot\dot{\mathbf{y}}\\ [[D^2u_y]]\dot{\mathbf{y}}\cdot\dot{\mathbf{y}}\end{array}\right\} = \ddot{\boldsymbol{\mu}} - [[Du]]\ddot{\mathbf{y}},\\
&\left\{\begin{array}{l}[[D^2u_x]]\dot{\mathbf{y}}\cdot n\\ [[D^2u_y]]\dot{\mathbf{y}}\cdot n\end{array}\right\} + [[D^2u_x]]\dot{\mathbf{y}}n_x + [[D^2u_y]]\dot{\mathbf{y}}n_y = [[Dp]]\cdot\dot{\mathbf{y}} - [[v(Du + Du^{\mathrm{T}})]]\dot{\boldsymbol{n}}.
\end{aligned}$$

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 Fig. 11.

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.

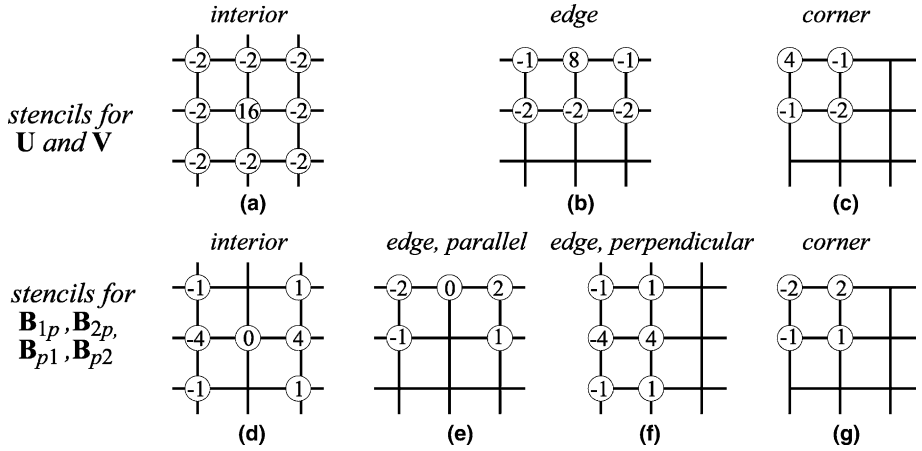

Fig. 11. Essentially different stencils of the $Q1$–$Q1$ finite element discretization. All other stencils are obtained by reflections of these stencils about vertical, horizontal and diagonal directions. The coefficients stencils in the upper row are computed as $\int_\Omega \nabla\phi_i\nabla\phi_j\,d\Omega$ 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 $1/12h$.

References

[1] M.J. Aftosmis, M.J. Berger, G. 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] J.F. Antaki, G.E. Blelloch, O. Ghattas, I. Malčević, G.L. Miller, N.J. Walkington, A parallel dynamic-mesh Lagrangian method for simulation of flows with dynamic interfaces, in: Proceedings of Supercomputing 2000, ACM/IEEE, Dallas, TX, 2000.

[3] S. Balay, K. Buschelman, W.D. Gropp, D. Kaushik, L. Curfman McInnes, B.F. Smith, PETSc home page, 2001. Available from http://www.mcs.anl.gov/petsc.

[4] S. Balay, W.D. Gropp, L. Curfman McInnes, B.F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A.M. Bruaset, H.P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhäuser Press, Basel, 1997, pp. 163–202.

[5] J. Thomas Beale, Ming-Chih Lai, A method for computing nearly singular integrals, SIAM Journal on Numerical Analysis 38 (6) (2001) 1902–1925.

[6] G. Biros, Lexing Ying, D. 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, 2002.

[7] P. Bochev, R.B. Lehoucq, On finite element discretizations of the pure Neumann problem, Technical Report SAND2001-0733J, Sandia National Laboratories, Albuquerque, NM, 20001.

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

[9] Xiao-Chuan Chai, O.B. Widlund, Domain decomposition algorithms for indefinite elliptic problems, SIAM Journal on Scientific and Statistical Computing 13 (1) (1992) 243–258.

[10] T.F. Chan, T.P. Mathew, Domain decomposition algorithms, Acta Numerica (1994).

[11] Li-Tien Cheng, R.P. Fedkiw, F. Gibou, Myungjoo Kang, A second-order accurate symmetric discretization of the Poisson equation on irregular domains, Journal of Computational Physics 171 (2001) 205–227.

[12] L. Farina, Evaluation of single layer potentials over curved surfaces, SIAM Journal on Scientific Computing 23 (1) (2001) 81–91.

[13] G. Fix, Hybrid finite element methods, SIAM Review 18 (3) (1976) 460–484.

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

[15] V. Girault, P.-A. Raviart, Finite Element Methods for Navier–Stokes Equations, Springer, Berlin, 1986.

[16] R. Glowinski, T.W. Pan, 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) (1994) 133–148.

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

[18] A. Greenbaum, L. Greengard, G.B. McFadden, Laplace's equation and the Dirichlet–Neumann map in multiply connected domains, Journal of Computational Physics 105 (1993) 267–278.

[19] A. Greenbaum, A. Mayo, Rapid parallel evaluation of integrals in potential theory on general three-dimensional regions, Journal of Computational Physics 145 (2) (1998) 731–742.

[20] L. Greengard, M. Catherine Kropinski, A. Mayo, Integral equation methods for Stokes flow and isotropic elasticity in the plane, Journal of Computational Physics 125 (1996) 403–414.

[21] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, Journal of Computational Physics 73 (1987) 325–348.

[22] L. Greengard, V. Rokhlin, A new version of the fast multipole method for the Laplace equation in three dimensions, Acta Numerica (1997) 229–269.

[23] M.D. Gunzburger, Finite Element for Viscous Incompressible Flows, Academic Press, New York, 1989.

[24] M.D. Gunzburger, R.A. Nicolaides (Eds.), Incompressible Computational Fluid Dynamics, Cambridge University Press, Cambridge, MA, 1993.

[25] H. Johansen, P. Colella, A Cartesian grid embedded boundary method for Poisson's equation on irregular domains, Journal of Computational Physics 147 (1998) 60–85.

[26] C. Kadow, N. Walkington, Design of a projection-based parallel Delaunay mesh generation and refinement algorithm, in: Proceedings of the 7th US National Congress in Computational Mechanics, 2003.

[27] S. Kapur, D.E. Long, IES_3: efficient electrostatic and electromagnetic simulation, IEEE Computational Science and Engineering 5 (4) (1998) 60–67.

[28] S. Kapur, Jinsong Zhao, A fast method of moments solver for efficient parameter extraction of MCMs, in: Design Automation Conference, 1997, pp. 141–146.

[29] S. Kim, S.J. Karrila, Microhydrodynamics: Principles and Selected Applications, Butterworth–Heinemann, London, 1991.

[30] A. Klawonn, Preconditioners for Indefinite Systems, Ph.D. Thesis, Courant Institute, New York University, New York, NY 10021, 1996.

[31] R.J. LeVeque, Zhilin Li, The immersed interface method for elliptic equations with discontinuous coefficients and singular sources, SIAM Journal on Numerical Analysis 31 (1994) 1019–1044.
[32] R.J. LeVeque, Zhilin Li, Immersed interface methods for Stokes flow with elastic boundaries or surface tension, SIAM Journal on Scientific Computing 18 (1997) 709–735.
[33] Xiang-Yang Li, Shang-Hua Teng, Generating well-shaped Delaunay meshes in 3D, in: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, ACM, Washington, DC, 2001.
[34] Zhilin Li, A fast iterative algorithm for elliptic interface problems, SIAM Journal on Numerical Analysis 35 (1998) 23–254.
[35] Zhilin Li, Ming-Chih Lai, The immersed interface method for the Navier–Stokes equations with singular forces, Journal of Computational Physics 171 (2001) 822–842.
[36] A.A. Mammoli, M.S. Ingber, Parallel multipole BEM simulation of two-dimensional suspension flows, Engineering Analysis with Boundary Elements 24 (2000) 65–73.
[37] A. Mayo, The fast solution of Poisson's and the biharmonic equations on irregular regions, SIAM Journal on Numerical Analysis 21 (2) (1984) 285–299.
[38] A. McKenney, An adaptation of the Fast Multipole Method for evaluating layer potentials in two dimensions, Computers and Mathematics with Applications 32 (1) (1996) 33–57.
[39] A. McKenney, L. Greengard, A. Mayo, A fast Poisson solver for complex geometries, Journal of Computatational Physics 118 (1994) 348–355.
[40] S. Norburn, D. Silvester, Fourier Analysis of Stabilized $Q1$–$Q1$ Mixed Finite Element Approximation, Numerical Analysis Report 348, The University of Manchester, 1999.
[41] C.S. Peskin, D.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 (1989) 372–405.
[42] C.S. Peskin, B.F. Prinz, Improved volume conservation in the computation of flows with immersed elastic boundaries, Journal of Computational Physics 105 (1993) 113–132.
[43] Nhan Phan-Thien, Ka Yan Lee, D. Tullock, Large scale simulation of suspensions with PVM, in: Proceedings of SC97, The SCxy Conference Series, ACM/IEEE, San Jose, CA, 1997.
[44] J.R. Phillips, J.K. White, A precorrected-FFT method for electorstatic analysis of complicated 3-D structures, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 16 (10) (1997) 1059–1072.
[45] H. Power, The completed double layer integral equation method for two-dimensional stokes flow, IMA Journal of Applied Mathematics 51 (1993) 123–145.
[46] H. Power, L. Wrobel, Boundary Integral Methods in Fluid Mechanics, Computational Mechanics Publications, 1995.
[47] C. Pozrikidis, Boundary Integral and Singularity Methods for Linearized Viscous Flow, Cambridge University Press, Cambridge, MA, 1992.
[48] C. Pozrikidis, Interfacial dynamics for Stokes flow, Journal of Computational Physics 169 (2001) 250–301.
[49] W. Proskurowski, O.B. Widlund, On the numerical solution of Helmholtz's equation by the capacitance matrix method, Mathematics of Computation 30 (135) (1976) 433–468.
[50] R.J. Renka, Algorithm 790: CSHEP2D: Cubic Shepard method for bivariate interpolation of scattered data, ACM Transactions on Mathematical Software 25 (1) (1999) 70–73.
[51] V. Rokhlin, Rapid solution of integral equations of classical potential theory, Journal of Computational Physics 60 (1983) 187–207.
[52] J. Ruppert, R. Seidel, On the difficulty of triangulating three-dimensional nonconvex polyedra, Discrete and Computational Geometry 7 (3) (1992).
[53] J.R. Shewchuk, Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations, in: Proceedings of the sixteenth annual symposium on Computational geometry, ACM, Kowloon, Hong Kong, 2000.
[54] J. Strain, Locally-corrected multidimensional quadrature rules for singular functions, SIAM Journal on Scientific Computing 6 (4) (1995) 992–1017.
[55] K. Stüben, A review of algebraic multigrid, Journal of Computational and Applied Mathematics 128 (1–2) (2001) 281–309.
[56] A. Wiegmann, K.P. Bube, The explicit-jump immersed interface method: finite difference methods for PDEs with piecewise smooth solutions, SIAM Journal on Numerical Analysis 37 (3) (2000) 827–862.
[57] A.Z. Zinchenko, R.H. Davis, An efficient algorithm for hydrodynamical interaction of many deformable drops, Journal of Computational Physics 157 (1999) 539–587.