Product algebras for Galerkin discretisations of boundary integral operators and their applications

TIMO BETCKE, University College London, UK

MATTHEW W. SCROGGS, University College London, UK

WOJCIECH ŚMIGAJ, Simpleware Ltd., UK

Operator products occur naturally in a range of regularized boundary integral equation formulations. However, while a Galerkin discretisation only depends on the domain space and the test (or dual) space of the operator, products require a notion of the range. In the boundary element software package Bempp we have implemented a complete operator algebra that depends on knowledge of the domain, range and test space. The aim was to develop a way of working with Galerkin operators in boundary element software that is as close to working with the strong form on paper as possible while hiding the complexities of Galerkin discretisations. In this paper, we demonstrate the implementation of this operator algebra and show, using various Laplace and Helmholtz example problems, how it significantly simplifies the definition and solution of a wide range of typical boundary integral equation problems.

CCS Concepts: • **Mathematics of computing** → **Numerical analysis**; **Integral equations**;

Additional Key Words and Phrases: Boundary integral equations, operator preconditioning, boundary element software

1 INTRODUCTION

A typical abstract operator problem can be formulated as

$$\mathsf{A}\phi = f,$$

where A is an operator mapping from a Hilbert space $\mathcal{H}_1$ into another Hilbert space $\mathcal{H}_2$ with the unknown $\phi \in \mathcal{H}_1$ and known $f \in \mathcal{H}_2$. Many modern operator preconditioning strategies depend on the idea of having a regulariser $\mathsf{R} : \mathcal{H}_2 \to \mathcal{H}_1$ and solving the equation

$$\mathsf{R}\mathsf{A}\phi = \mathsf{R}f \tag{1}$$

instead. This is particularly common in the area of boundary integral equations, where integral operators can be efficiently preconditioned by operators of opposite order. Now

Authors' addresses: Timo Betcke, Department of Mathematics, University College London, UK, t.betcke@ ucl.ac.uk; Matthew W. Scroggs, Department of Mathematics, University College London, UK, matthew. scroggs.14@ucl.ac.uk; Wojciech Śmigaj, Simpleware Ltd. Exeter, UK, w.smigaj@simpleware.com.

suppose that we want to discretise (1) using a standard Galerkin method. The discretised problem is

$$RM^{-1}A\phi = Rf, \tag{2}$$

where R and A are the Galerkin discretisations of R and A, respectively, and f is the vector of coefficients of the projection of f onto the finite dimensional subspace of $\mathcal{H}_2$. The matrix M is the mass matrix between the basis functions of the finite dimensional subspaces of $\mathcal{H}_2$ and $\mathcal{H}_1$.

In order to solve (2), we have to assemble all involved matrices, form the right-hand side, implement a function that evaluates $RM^{-1}Av$ for a given vector v, and then solve (2) with GMRES or another iterative solver of choice. Ideally, we would not have to deal with these implementational details and just directly write the following code.

```
A = operator(...)
R = operator(...)
f = function(...)
phi = gmres(R * A, R * f)
```

Note that at the end, the solution `phi` is again a function object. In order for this code snippet to work and the mass matrix M to be assembled automatically, either the implementation of the operator product needs to be aware of the test space of A and domain space of R, or the software definition of these operators need to contain information about their ranges. In this paper we will follow this second approach by defining the notion of the strong form of a Galerkin discretisation and demonstrate its benefits.

An implementation of a product algebra based on this idea is contained in the Python/C++ based boundary element library Bempp (www.bempp.com) [13], originally developed by the authors of this paper. Bempp is a comprehensive library for the solution of boundary integral equations for Laplace, Helmholtz and Maxwell problems. The leading design principle of Bempp is to allow a description of BEM problems in Python code that is as close to the mathematical formulation as possible, while hiding implementational details of the underlying Galerkin discretisations. This allows us to formulate complex block operator systems such as those arising in Calderón preconditioned formulations of transmission problems in just a few lines of code. Initial steps towards a Bempp operator algebra were briefly described in [13] as part of a general library overview. The examples in this paper are based on the current version (Bempp 3.3), which has undergone significant development since then and now contains a complete and mature product algebra for operators and grid functions.

As examples for the use of an operator algebra in more complex settings, we discuss: the efficient assembly of the hypersingular operator via a representation using single layer operators; the assembly of Calderón projectors and the computation of their spectral properties and the Calderón preconditioned solution of acoustic transmission problems.

A particular challenge is the design of product algebras for Maxwell problems. The stable discretisation of the electric and magnetic field operators for Maxwell problems requires the use of a non-standard skew symmetric bilinear form. The Maxwell case is discussed in much more detail in [12].

The paper is organised as follows. In Section 2 we review basic definitions of boundary integral operators for Laplace and Helmholtz problems. In Section 3 we introduce the basic concepts of a Galerkin product algebra and discuss some implementational details. Section 4 then gives a first application to the fast assembly of hypersingular operators for Laplace and Helmholtz problems. Then, in Section 5 we discuss block operator systems at the example

of Calderón preconditioned transmission problems. The paper concludes with a summary in Section 6.

While most of the mathematics presented in this paper is well known among specialists, the focus of this paper is on hiding mathematical complexity of Galerkin discretisations. With the wider penetration and acceptance of high-level scripting languages such us Matlab, Python and Julia in the scientific computing community, we now have the tools and structures to make complex computational operations accessible for a wide audience of non-specialist users, making possible the fast dissemination of new algorithms and techniques beyond traditional mathematical communities.

2 BOUNDARY INTEGRAL OPERATORS FOR SCALAR LAPLACE AND HELMHOLTZ PROBLEMS, AND THEIR GALERKIN DISCRETISATION

In this section, we give the basic definitions of boundary integral operators for Laplace and Helmholtz problems and some of their properties needed later. More detailed information can be found in e.g. [11, 14].

We assume that $\Omega \subset \mathbb{R}^3$ is a piecewise smooth bounded Lipschitz domain with boundary Γ. By $\Omega^+ := \mathbb{R}^3 \backslash \overline{\Omega}$ we denote the exterior of Ω. We denote by $\gamma_0^\pm$ the associated interior (-) and exterior (+) trace operators and by $\gamma_1^\pm$ the interior and exterior normal derivative operators. We always assume that the normal direction $\boldsymbol{\nu}$ points outwards into Ω^+.

The average of the interior and exterior trace is defined as $\{\{\gamma_0 f\}\} := \frac{1}{2} \left(\gamma_0^+ f + \gamma_0^- f \right)$. Correspondingly, the average normal derivative is defined as $\{\{\gamma_1 f\}\} := \frac{1}{2} \left(\gamma_1^+ f + \gamma_1^- f \right)$.

2.1 Operator definitions

We consider a function $\phi^- \in H^1(\Omega)$ satisfying the Helmholtz equation $-\Delta \phi^- - k^2 \phi^- = 0$, where $k \in \mathbb{R}$. By Green's representation theorem we have

$$\phi^-(\mathbf{x}) = [\mathcal{V}\gamma_1^- \phi^-](\mathbf{x}) - [\mathcal{K}\gamma_0^- \phi^-](\mathbf{x}), \qquad \mathbf{x} \in \Omega \qquad (3)$$

for the single layer potential operator $\mathcal{V} : H^{-1/2}(\Gamma) \to H^1_{\text{loc}}(\Omega \cup \Omega^+)$ defined by

$$[\mathcal{V}\mu](\mathbf{x}) = \int_\Gamma G(\mathbf{x}, \mathbf{y}) \mu(\mathbf{y}) \, ds(\mathbf{y}), \qquad \mu \in H^{-1/2}(\Gamma)$$

and the double layer potential operator $\mathcal{K} : H^{1/2}(\Gamma) \to H^1_{\text{loc}}(\Omega \cup \Omega^+)$ defined by

$$[\mathcal{K}\xi](\mathbf{x}) = \int_\Gamma \frac{\partial G(\mathbf{x}, \mathbf{y})}{\partial \boldsymbol{\nu}(\mathbf{y})} \xi(\mathbf{y}) \, ds(\mathbf{y}), \qquad \xi \in H^{1/2}(\Gamma).$$

Here, $G(\mathbf{x}, \mathbf{y}) := \frac{e^{ik|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|}$ is the associated Green's function. If $k = 0$, we obtain the special case of the Laplace equation $-\Delta u = 0$.

We now define the following boundary operators as the average of the interior and exterior traces of the single layer and double layer potential operators:

- The *single layer* boundary operator $\mathsf{V} : H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma)$ defined by

$$[\mathsf{V}\mu](\mathbf{x}) = \{\{\gamma_0 \mathcal{V}\mu\}\}(\mathbf{x}), \qquad \mu \in H^{-1/2}(\Gamma), \ \mathbf{x} \in \Gamma.$$

- The *double layer* boundary operator $\mathsf{K} : H^{1/2}(\Gamma) \to H^{1/2}(\Gamma)$ defined by

$$[\mathsf{K}\xi](\mathbf{x}) = \{\{\gamma_0 \mathcal{K}\xi\}\}(\mathbf{x}), \qquad \xi \in H^{1/2}(\Gamma), \ \mathbf{x} \in \Gamma.$$

- The *adjoint double layer* boundary operator $\mathsf{K}' : H^{-1/2}(\Gamma) \to H^{-1/2}(\Gamma)$ defined by

$$[\mathsf{K}'\mu](\mathbf{x}) = \{\{\gamma_1 \mathcal{V}\mu\}\}(\mathbf{x}), \qquad \mu \in H^{-1/2}(\Gamma), \ \mathbf{x} \in \Gamma.$$

- The *hypersingular* boundary operator $\mathsf{W} : H^{1/2}(\Gamma) \to H^{-1/2}(\Gamma)$ defined by

$$[\mathsf{W}\xi]\,(\mathbf{x}) = -\{\{\gamma_1 \mathcal{K}\xi\}\}(\mathbf{x}), \qquad\qquad \xi \in H^{1/2}(\Gamma),\ \mathbf{x} \in \Gamma.$$

Applying the interior traces γ_0^- and γ_1^- to the Green's representation formula (3), and taking into account the jump relations of the double layer and adjoint double layer boundary operators on the boundary Γ [14, Section 6.3 and 6.4] we arrive at

$$\begin{bmatrix} \gamma_0^- \phi^- \\ \gamma_1^- \phi^- \end{bmatrix} = \left(\tfrac{1}{2}\mathsf{Id} + \mathsf{A}\right) \begin{bmatrix} \gamma_0^- \phi^- \\ \gamma_1^- \phi^- \end{bmatrix} \tag{4}$$

with

$$\mathsf{A} := \begin{bmatrix} -\mathsf{K} & \mathsf{V} \\ \mathsf{W} & \mathsf{K}' \end{bmatrix}, \tag{5}$$

which holds almost everywhere on Γ. The operator $\mathcal{C}^- := \tfrac{1}{2}\mathsf{Id} + \mathsf{A}$ is also called the interior Calderón projector. If ϕ^+ is a solution of the exterior Helmholtz equation $-\Delta\phi^+ - k^2\phi^+ = 0$ in Ω^+ with boundary condition at infinity

$$\lim_{|\mathbf{x}|\to\infty} |\mathbf{x}| \left(\frac{\partial}{\partial|\mathbf{x}|} \phi^+ - \mathrm{i}k\phi^+ \right) = 0$$

for $k \neq 0$ and

$$\lim_{|\mathbf{x}|\to\infty} |\phi^+(\mathbf{x})| = \mathcal{O}\left(\frac{1}{|\mathbf{x}|} \right)$$

for $k = 0$, Green's representation formula is given as

$$\phi^+(\mathbf{x}) = \left[\mathcal{K}\gamma_0^+ \phi^+\right](\mathbf{x}) - \left[\mathcal{V}\gamma_1^+ \phi^+\right](\mathbf{x}),\ \mathbf{x} \in \Omega^+. \tag{6}$$

Taking the exterior traces γ_0^+ and γ_1^+ now gives the system of equations

$$\begin{bmatrix} \gamma_0^+ \phi^+ \\ \gamma_1^+ \phi^+ \end{bmatrix} = \left(\tfrac{1}{2}\mathsf{Id} - \mathsf{A}\right) \begin{bmatrix} \gamma_0^+ \phi^+ \\ \gamma_1^+ \phi^+. \end{bmatrix} \tag{7}$$

with associated exterior Calderón projector $\mathcal{C}^+ := \tfrac{1}{2}\mathsf{Id} - \mathsf{A}$.

2.2 Galerkin discretisation of integral operators

Let $\mathcal{T}_h$ be a triangulation of Γ with N piecewise flat triangular elements τ_j and M associated vertices $\mathbf{p}_i$. We define the function space S_h^0 of elementwise constant functions ϕ_j such that

$$\phi_j(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in \tau_j \\ 0, & \text{otherwise,} \end{cases}$$

and the space S_h^1 of globally continuous, piecewise linear hat functions ρ_i such that

$$\rho_i(\mathbf{p}_\ell) = \begin{cases} 1, & i = \ell \\ 0, & \text{otherwise.} \end{cases}$$

Denote by $\langle u, v \rangle_\Gamma$ the standard surface dual form $\int_\Gamma u(\mathbf{x})\overline{v(\mathbf{x})}\,\mathrm{d}s(\mathbf{x})$ of two functions u and v. By restricting $H^{1/2}(\Gamma)$ onto S_h^1 and $H^{-1/2}(\Gamma)$ onto S_h^0, we obtain the Galerkin discretizations V, K, K', W defined as

$$\begin{aligned} [V]_{ij} &:= \langle \mathsf{V}\phi_j, \phi_i \rangle_\Gamma, & [K]_{ij} &:= \langle \mathsf{K}\rho_j, \phi_i \rangle_\Gamma \\ [K']_{ij} &:= \langle \mathsf{K}'\phi_j, \rho_i \rangle_\Gamma, & [W]_{ij} &:= \langle \mathsf{W}\rho_j, \rho_i \rangle_\Gamma \end{aligned}$$

From this definition it follows that $K' = K^T$. A computable expression of W using weakly singular integrals is given in Section 4.

A problem with this definition of discretisation spaces is that S_h^0 and S_h^1 have a different number of basis functions, leading to non-square matrices K and K'. Hence, it is only suitable for discretisations of integral equations of the first-kind involving only V or W on the left-hand side. There are two solutions to this.

(1) Discretise both spaces $H^{1/2}(\Gamma)$ and $H^{-1/2}(\Gamma)$ with the continuous space S_h^1. This works well if Γ is sufficiently smooth. However, if Γ has corners then Neumann data in $H^{-1/2}(\Gamma)$ is not well represented by continuous functions.

(2) Instead of the space S_h^0 use the space of piecewise constant functions ϕ_D on the dual grid which is obtained by associating each element of the dual grid with one vertex of the original grid. We denote this piecewise constant space by $S_{D,h}^0$. With this definition of piecewise constant functions also the matrix K is square. Moreover, the mass matrix between the basis functions in S_h^0 and $S_{D,h}^0$ is inf-sup stable [2, 8].

3 GALERKIN PRODUCT ALGEBRAS AND THEIR IMPLEMENTATION

In this section we discuss the product of Galerkin discretisations of abstract Hilbert space operators and how a corresponding product algebra can be implemented in software. While the mathematical basis is well known, most software libraries do not support a product algebra, making implementations of operator based preconditioners and many other operations more cumbersome than necessary. This section proposes a framework to elegantly support operator product algebras in general application settings. The formalism introduced here is based on Riesz mappings between dual spaces. A nice introduction in the context of Galerkin discretizations is given in [9].

3.1 Abstract formulation

Let $A : \mathcal{H}_A^{\mathrm{dom}} \to \mathcal{H}_A^{\mathrm{ran}}$ and $B : \mathcal{H}_B^{\mathrm{dom}} \to \mathcal{H}_B^{\mathrm{ran}}$ be operators mapping between Hilbert spaces. If $\mathcal{H}_A^{\mathrm{ran}} \subset \mathcal{H}_B^{\mathrm{dom}}$ the product

$$g = BAf \tag{8}$$

is well defined in $\mathcal{H}_B^{\mathrm{ran}}$. We now want to evaluate this product using Galerkin discretisations of the operators A and B.

Let $\mathcal{H}_A^{\mathrm{dual}}$ be dual to $\mathcal{H}_A^{\mathrm{ran}}$ with respect to a given dual pairing $\langle \cdot, \cdot \rangle_A : \mathcal{H}_A^{\mathrm{ran}} \times \mathcal{H}_A^{\mathrm{dual}} \to \mathbb{C}$. Correspondingly, we define the space $\mathcal{H}_B^{\mathrm{dual}}$ as dual space to $\mathcal{H}_B^{\mathrm{ran}}$ with respect to a dual pairing $\langle \cdot, \cdot \rangle_B$.

Defining the function $q = Af$, the operator product (8) can equivalently be written as

$$q = Af$$
$$g = Bq.$$

Rewriting this system in its variational form leads to the problem of finding $(q, g) \in \mathcal{H}_A^{\mathrm{ran}} \times \mathcal{H}_B^{\mathrm{ran}}$ such that

$$\langle q, \mu \rangle_A = \langle Af, \mu \rangle_A$$
$$\langle g, \tau \rangle_B = \langle Bq, \tau \rangle_B \tag{9}$$

for all $(\mu, \tau) \in \mathcal{H}_A^{\mathrm{dual}} \times \mathcal{H}_B^{\mathrm{dual}}$. We now introduce the finite dimensional subspaces $\mathcal{V}_{h,X}^{\mathrm{dom}} \subset \mathcal{H}_X^{\mathrm{dom}}$, $\mathcal{V}_{h,X}^{\mathrm{ran}} \subset \mathcal{H}_X^{\mathrm{ran}}$ and $\mathcal{V}_{h,X}^{\mathrm{dom}} \subset \mathcal{H}_X^{\mathrm{dual}}$ with basis functions $\zeta_{X,j}^{\mathrm{dom}}$, $\zeta_{X,i}^{\mathrm{ran}}$, $\zeta_{X,\ell}^{\mathrm{dual}}$ for $X = A, B$. In what follows we assume that the dimension of $\mathcal{V}_{h,X}^{\mathrm{dual}}$ is identical to the dimension of $\mathcal{V}_{h,X}^{\mathrm{ran}}$ and that the associated dual-pairing is inf-sup stable in the sense that

$$\sup_{\xi_X^{\mathrm{dual}} \in \mathcal{V}_{h,X}^{\mathrm{dual}}} \frac{\langle \xi_X^{\mathrm{ran}}, \xi_X^{\mathrm{dual}} \rangle_X}{\|\xi_X^{\mathrm{dual}}\|_{\mathcal{H}_X^{\mathrm{dual}}}} \geq c_X \|\xi_X^{\mathrm{ran}}\|_{\mathcal{H}_X^{\mathrm{ran}}}, \quad \forall \xi_X^{\mathrm{ran}} \in \mathcal{V}_{h,X}^{\mathrm{ran}}$$

for some $c_X > 0$, implying that the associated mass matrix is invertible. The discrete version of (9) is now given as

$$M_\mathsf{A} \boldsymbol{q} = A \boldsymbol{f},$$
$$M_\mathsf{B} \boldsymbol{g} = B \boldsymbol{q},$$

where $[M_\mathsf{X}]_{\ell,i} = \langle \phi^{\mathrm{ran}}_{\mathsf{X},i}, \phi^{\mathrm{dual}}_{\mathsf{X},\ell} \rangle_\mathsf{X}$, $\mathsf{X} = \mathsf{A}, \mathsf{B}$. are the mass matrices of the dual pairings. The vectors $\boldsymbol{f}$, $\boldsymbol{q}$ and $\boldsymbol{g}$ are the vectors of coefficients of the corresponding functions. Combining both equations we obtain

$$\boldsymbol{q} = M_\mathsf{B}^{-1} B M_\mathsf{A}^{-1} A \boldsymbol{f}.$$

The matrix A is also called the discrete *weak form* of the operator A. This motivates the following definition.

Definition 3.1. Given the discrete weak form A defined as above. We define the associated *discrete strong form* as the matrix

$$A^S := M_\mathsf{A}^{-1} A.$$

Note that M_A^{-1} is the discrete Riesz map from the dual space into the range space of A [9]. The notation of the discrete strong form allows us to define a Galerkin product algebra as follows.

Definition 3.2. Given the operator product $\mathsf{C} := \mathsf{BA}$. We define the associated discrete operator product weak form as

$$C := B \odot A := B \cdot A^S = B M_\mathsf{A}^{-1} A$$

and the associated discrete strong form as

$$C^S := M_\mathsf{B}^{-1} (B \odot A).$$

We note that the a direct discretisation $\langle \mathsf{BA} \phi^{\mathrm{dom}}_{\mathsf{A},j}, \phi^{\mathrm{dual}}_{\mathsf{B},\ell} \rangle$ is usually not identical to C as the latter is computed as the solution of the operator system (9) whose discretisation error also depends on the space $\mathcal{V}^{\mathrm{ran}}_{h,\mathsf{A}}$ and the corresponding discrete dual. However, the discretisation of the operator product BA can rarely be computed directly and solving (9) is usually the only possibility to evaluate this product.

This discrete operator algebra is associative since

$$(C \odot B) \odot A = C M_\mathsf{B}^{-1} B M_\mathsf{A}^{-1} A = C \odot (B \odot A).$$

Moreover, since a Hilbert space is self-dual in its natural inner product $(\cdot, \cdot)$ the discretization

$$\left[M_\mathsf{A}^{\mathrm{dom}} \right]_{ij} = \left(\zeta^{\mathrm{dom}}_{A,i}, \zeta^{\mathrm{dom}}_{A,j} \right)$$

of the identity operator $\mathsf{Id}^{\mathrm{dom}}_\mathsf{A}$ is the right unit element with respect to this discrete operator algebra. Correspondingly, the matrix $M^{\mathrm{ran}}_\mathsf{A}$ is the left unit element.

We have so far considered the approximation of the weak form $\langle \mathsf{BA} \phi^{\mathrm{dom}}_{\mathsf{A},j}, \phi^{\mathrm{dual}}_{\mathsf{B},\ell} \rangle$, where the operator B acts on $\mathsf{A} \phi^{\mathrm{dom}}_{\mathsf{A},j}$. However, there are situations where we want a discrete approximation of the product $\langle \mathsf{A} \phi^{\mathrm{dom}}_{\mathsf{A},j}, \mathsf{B} \phi^{\mathrm{dom}}_{\mathsf{B},\ell} \rangle$ for $\mathsf{B} : \mathcal{H}^{\mathrm{dom}}_\mathsf{B} \to \mathcal{H}^{\mathrm{dual}}_\mathsf{A}$. An example for the assembly of hypersingular operators will be given later. Note that if $\mathbf{y}$ is a coefficient vector of a function $\phi \in \mathcal{H}^{\mathrm{dom}}_\mathsf{B}$ then $\tilde{\mathbf{y}} = M_B^{-1} B \mathbf{y}$ is the coefficient vector to the Galerkin approximation of $\tilde{\phi} = \mathsf{B}\phi$. Hence, a discrete approximation of the weak form $\langle \mathsf{A} \phi^{\mathrm{dom}}_{\mathsf{A},j}, \mathsf{B} \phi^{\mathrm{dom}}_{\mathsf{B},\ell} \rangle$ is given by

$$B^H \cdot M_\mathsf{B}^{-H} \cdot A = \left[B^S \right]^H A.$$

This motivates the following definition.

Definition 3.3. We define the dual discrete product weak form associated with the operators A and B as

$$B \odot_D A := B^H \cdot M_{\mathsf{B}}^{-H} \cdot A \tag{10}$$

and the associated discrete strong form as

$$C := M_{\mathsf{B},\mathsf{A}}^{-1} \left(B \odot_D A \right).$$

where $M_{\mathsf{B},\mathsf{A}}$ is the mass matrix between the domain space of B and the range space of A.

3.2 Example: Operator preconditioned Dirichlet problems

As a first example, we describe the formulation of an operator preconditioned interior Dirichlet problem using the above operator algebra. We want to solve

$$-\Delta \phi^- - k^2 \phi^- = 0 \qquad \text{in } \Omega$$
$$\gamma_0 \phi^- = g \qquad \text{on } \Gamma$$

for a given function $g \in H^{1/2}(\Gamma)$. From the first line of (4) we obtain that

$$\gamma_0^- \phi^- = \left(\tfrac{1}{2}\mathsf{Id} - \mathsf{K} \right) \gamma_0^- \phi^- + \mathsf{V}\gamma_1^- \phi^-.$$

Substituting the boundary condition, we obtain the integral equation of the first kind

$$\mathsf{V}\gamma_1^- \phi^- = \left(\tfrac{1}{2}\mathsf{Id} + \mathsf{K} \right) g. \tag{11}$$

The operator $\mathsf{V} : H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma)$ is a pseudodifferential operator of order -1 and can be preconditioned by the hypersingular operator $\mathsf{W} : H^{1/2}(\Gamma) \to H^{-1/2}(\Gamma)$, which is a pseudodifferential operator of order 1 [8, 15]. We arrive at the preconditioned problem

$$\mathsf{W}\mathsf{V}\gamma_1^- \phi^- = \mathsf{W} \left(\tfrac{1}{2}\mathsf{Id} + \mathsf{K} \right) g. \tag{12}$$

Note that the operator W is singular if $k = 0$. In that case a rank-one modification of the hypersingular operator can be used [15]. For the Galerkin discretisation of (12) we use the standard L^2 based dual pairing $\langle \cdot, \cdot \rangle$ defined by

$$\langle u, v \rangle = \int_\Gamma u(x)\overline{v}(x)dx, \quad u, v \in L^2(\Gamma)$$

and note that the spaces $H^{1/2}(\Gamma)$ and $H^{-1/2}(\Gamma)$ are dual with respect to this dual pairing.

For the discretisation of the operators we use the spaces $S_{D,h}^0$ and S_h^1 as described in Section 2.2. Using the notation introduced in Section 3.1 we obtain the discrete system

$$W \odot V \boldsymbol{x} = W \odot \left(\tfrac{1}{2}M + K \right) \boldsymbol{g}, \tag{13}$$

where $\boldsymbol{x}$ is the vector of coefficients of the unknown function ϕ_h in the basis $S_{D,h}^0$. The matrix M is the discretisation of the identity operator on $H^{1/2}(\Gamma)$. If in addition we want to use Riesz (or mass matrix) preconditioning we can simply take the discrete strong forms of the product operators on the left and right hand side of (13).

In terms of mathematics the definition of the discrete strong form is simply a notational convenience. We could equally write (13) by directly inserting the mass matrix inverses. The main advantage of an operator product algebra first becomes visible in a software implementation that directly supports the notions of discrete strong forms and operator products. This is described below.

3.3 Basic software implementation of an operator algebra

Based on the definition of a discrete product algebra for Galerkin discretisations, we can now discuss the software implementation. Two concepts are crucial: namely that of a *grid function*, which represents functions defined on a grid; and that of an *operator*, which maps grid functions from a discrete domain space into a discrete range space.

3.3.1 Grid functions. We start with the description of a grid function. A basic grid function object is defined by a discrete function space and a vector of coefficients on the space. However, for practical purposes this is not always sufficient. Consider the following situation of multiplying the discrete single layer operator V, discretised with the space of piecewise constant functions S_h^0 and a vector of coefficients $\boldsymbol{f}$. The result $\boldsymbol{y} = V\boldsymbol{f}$ is defined as $y_i = \sum_{j=1}^n f_j \langle \mathsf{V}\phi_j, \phi_i \rangle$. Since the single layer operator maps onto $H^{1/2}(\Gamma)$ we would like to obtain a suitable vector of coefficients $\tilde{y}$ of piecewise linear functions in S_h^1 such that

$$\boldsymbol{y} = M\tilde{\boldsymbol{y}},$$

where M is the rectangular mass-matrix between the spaces S_h^0 and S_h^1. Solving for $\tilde{\boldsymbol{y}}$ is only possible in a least-squares sense. Moreover, for these two spaces the matrix M may even be ill-conditioined or singular in the least-squares sense, making it difficult to obtain a good approximation in the range space. Hence, we also allow the definition of a grid function purely through the vector of coefficients into the dual space.

The constructors to define a grid function either through coefficients in a given space or through projections into a dual space are defined as follows.

```
fun = GridFunction(space, coefficients=...)
fun = GridFunction(space, dual_space=..., projections=...)
```

Associated with these two constructors are two methods that extract the vectors of coefficients or projections.

```
coeffs = fun.coefficients()
proj = fun.projections(dual_space)
```

If the grid function is initialised with a coefficient vector, then the first operation just returns this vector. The second operation sets up the corresponding mass matrix `M` and returns the vector `M * coeffs` . If the grid function is initialised with a vector of projections and a corresponding dual space then access to the coefficients results in a solution of a linear system if the space and dual space have the same number of degrees of freedom. Otherwise, an exception is thrown. If the `projections` method is called and the given dual space is identical to the original dual space on initialisation the vector `projections` is returned. Otherwise, first a conversion to coefficient form via a call to `coefficients()` is attempted.

This dual representation of a grid function via either a vector of coefficients or a vector of projections makes it possible to represent functions in many standard situations, where a conversion between coefficients and projections is mathematically not possible and not necessary for the formulation of a problem.

3.3.2 Operators. Typically, in finite element discretisation libraries the definition of an operator requires an underlying weak form, a domain space and a test space. However, to support the operator algebra introduced in Section 3.1 the range space is also required. Hence, we represent a constructor for a boundary operator in the following form.

```
op = operator(domain, range_, dual_to_range, ...)
```

Here, the objects `domain`, `range_` and `dual_to_range` describe the finite dimensional domain, range and dual spaces. Each operator provides the following two methods.

```
discrete_weak_form = op.weak_form()
discrete_strong_form = op.strong_form()
```

The first one returns the standard discrete weak form while the second one returns the discrete strong form. The `discrete_weak_form` and `discrete_strong_form` are objects that implement at least a matrix-vector routine to multiply a vector with the corresponding discrete operator. The multiplication with the inverse of the mass matrix in the strong form is implemented via computing an LU decomposition and solving the associated linear system.

Important for the performance is caching. The weak form is computed in the first call to the `weak_form()` method and then cached. Correspondingly, the LU decomposition necessary for the strong form is computed only once and then cached.

3.3.3 Operations on operators and grid functions. With this framework the multiplication `res_fun = op * fun` of a boundary operator `op` with a grid function `fun` can be elegantly described in the following way:

```
result_fun = GridFunction(
    space=op.range_,
    dual_space=op.dual_to_range,
    projections=op.weak_form() * fun.coefficients)
```

Alternatively, we could have more simply presented the result as

```
result_fun = GridFunction(
    space=op.range_,
    coefficients=op.strong_form() * fun.coefficients)
```

However, the latter ignores that there may be no mass matrix transformation available that could map from the discrete dual space to the discrete range space.

As an example, we present a small code snippet from Bempp that maps the constant function $f(\mathbf{x}) = 1$ on the boundary of the cube to the function $g = \mathsf{V}f$, where V is the Laplace single layer boundary operator. f is represented in a space of piecewise constant functions on the dual grid and g is represented in a space of continuous, piecewise linear functions, reflecting the smoothing properties of the Laplace single layer boundary operator. The following lines define the cube grid with an element size of $h = 0.1$ and the spaces of piecewise constant functions on the dual grid, and continuous, piecewise linear continuous functions on the primal grid.

```
grid = bempp.api.shapes.cube(h=0.1)
const_space = bempp.api.function_space(grid, "DUAL", 0)
lin_space = bempp.api.function_space(grid, "B-P", 1)
```

We would like to remark on the parameter B-P (*barycentric-polynomial*) in the code given above for the function space definitions. Since the piecewise constant functions are defined on the dual grid, we are working with the barycentric refinement of the original grid [2]. Hence, the piecewise linear functions on the primal grid also need to be defined over the
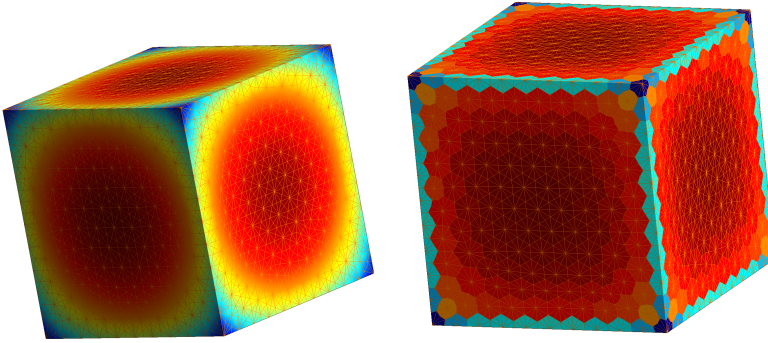
Fig. 1. Left: The Laplace single layer operator applied to a constant function on the boundary of a cube. Right: The Laplace hypersingular operator applied to the function on the left.

barycentric refinement (denoted by the parameter `B-P`) as the discretisation routines require the same refinement level for the domain and dual to range space. Mathematically, the standard space of continuous, piecewise linear functions over the primal grid and the space `B-P` over the barycentric refinement are identical.

We now define the operator and the constant grid function. For the grid function the coefficient vector is created via the NumPy routine `ones`, taking as input the number of degrees of freedom in the space.

```
op = bempp.api.operators.boundary.laplace.single_layer(
    const_space, lin_space, const_space)
fun = bempp.api.GridFunction(
    const_space,
    coefficients=np.ones(const_space.global_dof_count))
```

We can now multiply the operator with the function and plot the result.

```
result = op * fun
result.plot()
```

The output is the left cube shown in Figure 1. It is a continuous function in $H^{1/2}(\Gamma)$. The right cube in Figure 1 shows the result of multiplying the Laplace hypersingular operator defined by

```
op = bempp.api.operators.boundary.laplace.hypersingular(
    lin_space, const_space, lin_space)
```

with the function on the left. Since the hypersingular operator maps into $H^{-1/2}(\Gamma)$, the appropriate range space consists of piecewise constant functions, and the result of the discrete operation correspondingly uses a space of piecewise constant functions.

Under the condition that the operations mathematically make sense and operators and functions are correctly defined this mechanism always maps grid function objects into the right spaces under the action of a boundary operator while hiding all the technicalities of Galerkin discretisations.

The internal implementation of the product of two operators is equally simple in this framework. Given two operators `op1` and `op2`. Internally, the `weak_form()` method of the product `op1 * op2` is defined as follows.

```python
def weak_form():
    return op1.weak_form() * op2.strong_form()
```

Correspondingly, the strong form of the product is implemented as:

```python
def strong_form():
    return op1.strong_form() * op2.strong_form()
```

Internally, the product of two discrete operators provides a matrix-vector routine that successively applies the two operators to a given vector. If `op1` and `op2` implement caching then an actual discretisation of a weak form is only performed once, and the product of the two operators is performed with almost no overhead.

It is very easy to wrap standard iterative solvers to support this operator algebra. Suppose we want to solve the product system (13). Using an operator algebra wrapper to any standard GMRES (such as the one in SciPy [1]) the solution to the system (13) now takes the form

```python
solution, info = gmres(W * V, W * (.5 * ident + K) * g)
```

with `solution` being a grid function that lives in the correct space of piecewise constant functions. The definition of such a GMRES routine is as follows:

```python
def gmres(A, b, ...):
    from scipy.sparse import linalg
    x, info = linalg.gmres(
        A.weak_form(),
        b.projections(A.dual_to_range),
        ...)
    return GridFunction(A.domain, coefficients=x), info
```

The product algebra automatically converts `W * V` into a new object that provides the correct space attributes and a `weak_form` method as defined above. Similarly, the right-hand side `b` is evaluated into a vector with the `projections` method. The full Bempp implementation provides among other options also a keyword attribute `use_strong_form`. If this is set to true then inside the GMRES routine the solution is computed as

```python
x, info = linalg.gmres(A.strong_form(), b.coefficients)
```

This corresponds to standard Riesz (or mass matrix) preconditioning and comes naturally as part of this algebra. Note that we have left out of the description checks that the spaces of the left and right hand side are compatible. In practice, this should be done by the code as sanity check.

Finally, the weak form of the dual product $B \odot_D A$ can be be implemented as

```python
def weak_form():
    return B.strong_form().adjoint() * A.strong_form()
```

The range space and domain space of the dual product are the same as that of A while the dual space is the same as the domain space of B.

3.4 A note on the performance of the operator algebra

The operator algebra described above relies on being able to perform fast mass matrix LU decompositions and solves. In finite element methods LU decompositions with a mass matrix can be as expensive as solves with a stiffness matrix. In BEM the situation is quite different. Even with the utilisation of fast methods such as FMM (fast multipole method [5]) or hierarchical matrices [6], the assembly and matrix-vector product of a boundary operator is typically much more expensive than assembling a mass matrix and performing an LU decomposition of it. Therefore, mass matrix operations can be essentially treated as on-the-fly operations compared to the rest. One potential problem is the complexity of the LU decomposition of a mass matrix over a surface function space on Γ. For banded systems the complexity of Gaussian elimination scales like $\mathcal{O}(n)$. However, a closed surface has a higher element connectivity than a standard plane in 2d and we cannot expect a simple $\mathcal{O}(n)$ scaling even with reordering. In practice though, this has made so far little difference and we have used the SuperLU code provided by SciPy for the LU decomposition and surface linear system solves on medium size BEM problems with hundreds of thousands of surface elements without any noticeable performance issues, and we expect little performance overhead even for very large problems with millions of unknowns as the FMM or hierarchical matrix operations on the operators have much larger effective costs and significantly more complex data structures to operate on.

4 THE FAST ASSEMBLY OF HYPERSINGULAR BOUNDARY OPERATORS

The weak form of the hypersingular boundary operator can, after integration by parts, be represented as [7, 10]

$$
\begin{aligned}
W_{ij} = {} & \frac{1}{4\pi} \int_{\Gamma} \int_{\Gamma} \frac{\mathrm{e}^{\mathrm{i}k|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|} \langle \mathbf{curl}_{\Gamma}\rho_i(\mathbf{x}), \mathbf{curl}_{\Gamma}\rho_j(\mathbf{y}) \rangle_2 \, \mathrm{d}s(\mathbf{y}) \, \mathrm{d}s(\mathbf{x}) \\
& - \frac{k^2}{4\pi} \int_{\Gamma} \int_{\Gamma} \frac{\mathrm{e}^{\mathrm{i}k|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|} \rho_i(\mathbf{x})\rho_j(\mathbf{y}) \langle \boldsymbol{\nu}(\mathbf{x}), \boldsymbol{\nu}(\mathbf{y}) \rangle_2 \, \mathrm{d}s(\mathbf{y}) \, \mathrm{d}s(\mathbf{x}),
\end{aligned}
\tag{14}
$$

where the basis and test function ρ_j and ρ_i are basis functions in S_h^1. Both terms in (14) are now weakly singular and can be numerically evaluated.

However, (14) motivates another way of assembling the hypersingular operator, which turns out to be significantly more efficient in many cases. In both terms of (14), a single layer kernel is appearing. We can use this and represent W in the form

$$
W = \sum_{j=1}^{3} P_j^T \cdot V \cdot P_j - k^2 \sum_{j=1}^{3} Q_j^T \cdot V \cdot Q_j,
\tag{15}
$$

where we now only need to assemble a single layer boundary operator V with smooth kernel in a space of discontinuous elementwise linear functions, and the P_j and Q_j are sparse matrices. P_j maps a continuous piecewise linear function to the jth component of its surface curl and Q_j scales the basis functions with the contributions of $\boldsymbol{\nu}$ in the jth component in each element. If $k = 0$ (Laplace case) the second term in (15) becomes zero and it would even be sufficient to use a space of piecewise constant functions to represent V.

This evaluation trick is well known and is suitable for discretising the hypersingular operator with continuous, piecewise linear basis functions on flat triangles. The disadvantage is that an explicit representation of the sparse matrices P_j and Q_j is necessary. This representation depends on the polynomial order and dof numbering of the space implementation.

In the following we use the product algebra concepts to write the representation (14) in a form that generalises to function spaces of arbitrary order on curved triangular elements without requiring details of the dof ordering in the implementation. Given a finite dimensional trial space V_h^{trial} with basis $\theta_1, \ldots, \theta_L$ and a corresponding test space V_h^{test} with basis $\xi_1, \ldots, \xi_{L'}$ we define the discrete sparse surface operators

$$\left[C^\ell\right]_{ij} = \langle [\mathbf{curl}_\Gamma \theta_j]_\ell, \xi_i \rangle_\Gamma,$$
$$\left[N^\ell\right]_{ij} = \langle \theta_j [\boldsymbol{\nu}]_\ell, \xi_i \rangle_\Gamma.$$

The operator C^ℓ weakly maps a function f to its elementwise ℓth surface curl component, and the operator N^ℓ weakly multiplies a function f with the ℓth component of the surface normal direction.

We can now represent the hypersingular operator as

$$W = \sum_{j=1}^{3} C^j \odot_D V \odot C^j - k^2 \sum_{j=1}^{3} N^j \odot_D V \odot N^j. \tag{16}$$

The dual multiplication $\odot_D$ in (16) acts on the test functions and the right multiplication $\odot$ acts on the trial functions. Let $V_h^{m,\text{cont}}$ be a globally continuous, elementwise polynomial function space of order m and denote by $V_h^{m,\text{disc}}$ the corresponding space of discontinuous elementwise polynomial functions of order m. Then the operators in (14) have the following domain, range and dual spaces.

Operator	domain	range	dual
W	$V_h^{m,\text{cont}}$	$V_h^{m,\text{disc}}$	$V_h^{m,\text{cont}}$
V	$V_h^{m,\text{disc}}$	$V_h^{m,\text{disc}}$	$V_h^{m,\text{disc}}$
N^j	$V_h^{m,\text{cont}}$	$V_h^{m,\text{disc}}$	$V_h^{m,\text{disc}}$
C^j	$V_h^{m,\text{cont}}$	$V_h^{m,\text{disc}}$	$V_h^{m,\text{disc}}$

We note that (14) only requires inverses of dual parings on $V_h^{m,\text{disc}}$ with itself as dual space and not dual pairings between $V_h^{m,\text{disc}}$ and $V_h^{m,\text{cont}}$ which are not invertible. If $k = 0$ we can use spaces of order $m - 1$ for V and the dual and range space of C since then the second sum in (15) vanishes and the first sum only contains products of derivatives of the basis and trial functions. Also, we have chosen the discontinuous function space $V_h^{m,\text{disc}}$ as range space of V. This guarantees that the result in (16) has the correct range space.

In terms of standard matrix products (16) has the form

$$W = \sum_{j=1}^{3} \left[C^j\right]^T \cdot M^{-T} \cdot V \cdot M^{-1} \cdot C^j - k^2 \sum_{j=1}^{3} \left[N^j\right]^T \cdot M^{-T} \cdot V \cdot M^{-1} \cdot N^j,$$

where M is the mass matrix associated with the space $V_h^{m,\text{disc}}$ of discontinuous basis functions. Hence, M is elementwise block-diagonal and therefore M^{-1} is too, and we can efficiently directly compute M^{-1} as a sparse matrix. We can then accumulate the sparse matrix products in the sum above to obtain (15) with $P_j = M^{-1} \cdot C^j$ and $Q_j = M^{-1} \cdot N^j$. In Bempp the whole implementation of the hypersingular operator can be written as follows.

```
D = ZeroBoundaryOperator(...)
for i in range(3):
```

N (cont/discont)	Standard		projection		via single layer	
	time	mem	time	mem	time	mem
258 / 1536	0.7 s	1.0 MiB	1.0 s	31 MiB	0.4 s	15.5 MiB
1026 / 6144	3.9 s	9.0 MiB	3.5 s	177 MiB	1.6 s	88 MiB
4098 / 24576	19.6 s	60.1 MiB	15.2 s	907 MiB	7.7 s	467 MiB
16386 / 98304	1.6 m	345 MiB	1.4 m	4.4 GiB	39.4 s	2.2 GiB
65538 / 393216	7.6 m	1.79 GiB	8.6 m	21.5 GiB	3.9 m	11.0 GiB

Fig. 2. Time and memory for the assembly of the hypersingular operator using the standard weak form on the continuous space, discontinuous assembly with projection spaces or a single layer formulation. In the latter two cases only the assembly time and memory of the boundary operator is given. Assembly time and memory requirements for the sparse operators are negligble.

```
D += C[i].dual_product(V) * C[i]
D += -k**2 * N[i].dual_product(V) * N[i]
```

Due to efficient caching strategies, all operators, including the mass matrices and their inverses, are computed only once. Hence, there is minimal overhead from using a high-level expressive formulation.

In Figure 2, we compare times and memory requirements for the hierarchical matrix assembly of the hypersingular boundary operator on the unit sphere with wavenumber $k = 1$ using basis functions in S_h^1. The left column shows the standard assembly based on (14) and S_h^1 basis functions. The middle column shows results for assembling the operator directly on a larger space of piecewise linear discontinuous functions using the weak form (14) and then projecting down to basis functions in S_h^1, that is $W = P^T W_{\mathrm{disc}} P$ for a sparse matrix P that maps from S_h^1 to a space of piecewise linear discontinuous functions. This assembly allows matrix compression directly on the elementwise basis functions instead of only compressing on nodal basis functions after summing up the elementwise contributions. However, in the case of the hypersingular operator, this leads to larger memory consumption through the larger matrix size on the discontinuous space, but not faster assembly times. The interesting case is the single layer formulation in (16). Even though the single layer operator is assembled on the larger discontinuous space it compresses better since it is a smoothing operator and therefore leads to around twice as fast assembly times. The price is a larger memory size compared to the standard assembly. If this is not of concern then the single layer based assembly is preferable. Note that the evaluation of the matrix-vector product using (16) requires six multiplications with the single layer operator. So if a large number of matrix-vector products is needed this can become a bottleneck.

5 BLOCK OPERATOR SYSTEMS

Block operator systems occur naturally in boundary element computations since we are typically dealing with pairs of corresponding Dirichlet and Neumann data whose relationship is given by the Calderón projector shown in (4) for the interior problem and (7) for the exterior problem. In this section we want to demonstrate some interesting computations with the Calderón projector which can be very intuitively performed in the framework of block operator extensions of the product algebra.

Within the Bempp framework, a blocked operator of given block dimension (m, n) is defined as

```
blocked_operator = bempp.api.BlockedOperator(m, n)
```

We can now assign individual operators to the blocked operator by e.g.

```
blocked_operator[0, 1] = laplace.single_layer(...)
```

Not every entry of a blocked operator needs to be assigned a boundary operator. Empty positions are automatically treated as zero operators. However, we require the following conditions before computations with blocked operators can be performed:

- There can be no empty rows or columns of the blocked operator.
- All operators in a given row must have the same `range` and `dual_to_range` space.
- All operators in a given column must have the same `domain` space.

These conditions are easily checked while assigning components to a blocked operator. The weak form of a blocked operator is obtained as

```
discrete_blocked_operator = blocked_operator.weak_form()
```

This returns an operator which performs a matrix-vector product by splitting up the input vector into its components with respect to the columns of the blocked operator, performs multiplications with the weak forms of the individual components, and then assembles the result vector back together again.

The interesting case is the definition of a strong form. Naively, we could just take the strong forms of the individual component operators. However, since each strong form involves the solution of a linear system with a mass matrix we want to avoid this. Instead, we multiply the discrete weak form of the operator from the left with a block diagonal matrix whose block diagonal components contain the inverse mass matrices that map from the dual space in the corresponding row to the range space. This works due to the compatibility condition that all test and range spaces within a row must be identical.

5.1 Stable discretisations of Calderón projectors

With the concept of a block operator we now have a simple framework to work with Calderón projectors $\mathcal{C}^\pm = \left(\frac{1}{2}\mathsf{Id} \mp \mathsf{A}\right)$ with A defined as in (5). For the sake of simplicity in the following we use the Calderón projector $\mathcal{C}^+$ for the exterior problem. The interior Calderón projector $\mathcal{C}^-$ is treated in the same way. Remember that both operators are defined on the product space $H^{1/2}(\Gamma) \times H^{-1/2}(\Gamma)$

Two properties are fundamental to Calderón projectors. First, $(\mathcal{C}^+)^2 = \mathcal{C}^+$; and second, if $U = \left[\gamma_0^+ u, \; \gamma_1^+ u\right]^T$ is the Cauchy data of an exterior Helmholtz solution u satisfying the Sommerfeld radiation condition, it holds that $U = \mathcal{C}^+ U$, or equivalently $\mathcal{C}^- U = 0$.

Based on the product algebra framework introduced in this paper we can easily represent these properties on a discrete level to obtain a numerical Calderón projector up to the discretisation error.

As an example, we consider the Calderón projector on the unit cube with wavenumber $k = 2$. Assembling the projector within the Bempp product operator framework is simple, and corresponding functions are already provided.

```
k = 2
from bempp.api.operators.boundary.sparse \
    import multitrace_identity
from bempp.api.operators.boundary.helmholtz \
```

```
    import multitrace_operator
calderon = .5 * multitrace_identity(grid, spaces="dual") \
          - multitrace_operator(grid, k, spaces="dual")
```

In this code snippet, the option `spaces="dual"` automatically discretises the Calderón projector using stable dual pairings of continuous, piecewise linear spaces on the primal grid, and piecewise constant functions on the dual grid.

To demonstrate the action of the Calderón projector to a pair of non-compatible Cauchy data we define two grid functions, both of which are constant one on the boundary.

```
f1 = bempp.api.GridFunction.from_ones(
    calderon.domain_spaces[0])
f2 = bempp.api.GridFunction.from_ones(
    calderon.domain_spaces[1])
```

The two functions are defined on the pair of domain spaces discretising the product space $H^{1/2}(\Gamma) \times H^{-1/2}(\Gamma)$. We can now apply the Calderón projector to this pair of spaces to compute new grid functions which form a numerically compatible pair of Cauchy data for an exterior Helmholtz solution. The code snippet for this operation is given by

```
[u1, v1] = calderon * [f1, f2]
```

The grid functions `u1` and `v1` again live in the spaces of piecewise continuous and piecewise constant functions, respectively. We now apply the Calderón projector again to obtain

```
[u2, v2] = calderon * [u1, v1]
```

The grid functions `u1` and `u2`, respectively `v1` and `v2` should only differ in the order of the discretisation error. We can easily check this.

```
error_dirichlet = (u2-u1).l2_norm() / u2.l2_norm()
error_neumann = (v2-v1).l2_norm() / v2.l2_norm()
```

For the corresponding values we obtain 1.2×10^{-4} and 8.0×10^{-4}. It is interesting to consider the singular values and eigenvalues of the discrete strong form of the Calderón projector. We can compute them easily as follows.

```
from scipy.linalg import svdvals, eigvals
calderon_dense = bempp.api.as_matrix(calderon.strong_form())
sing_vals = svdvals(calderon_dense)
eig_vals = eigvals(calderon_dense)
```

The grid has 736 nodes. This means that the discrete basis for the possible Dirichlet data has dimension 736. For each Dirichlet basis function there is a unique associated Neumann function via the Dirichlet-to-Neumann map. Hence, we expect the range of the Calderón projector to be of dimension 736 with all other singular values being close to the discretisation error. Correspondingly, for the eigenvalues we expect 736 eigenvalues close to 1 with all other eigenvalues being close to 0. This is indeed what happens as shown in Figure 3. In the top plot we show the singular values of the discrete Calderón projector and in the right plot the eigenvalues. While the eigenvalues cluster around 1 and 0 the singular values show a significant drop-off between $\sigma_{736} \approx 1.04$ and $\sigma_{737} \approx 4.9 \times 10^{-3}$, which corresponds to the

approximation error as the accuracy of the hierarchical matrix approximation was chosen to be 10^{-3}.

Finally, we would like to stress that while the eigenvalues of the discrete strong form are approximations to the eigenvalues of the continuous operator, the singular values of the discrete strong form are generally not. Given any operator A acting on a Hilbert space $\mathcal{H}$ the Galerkin approximation of the continuous eigenvalue problem $\mathsf{A}\phi = \lambda\phi$ is given as $Ax = \lambda M_{\mathsf{A}}x$, where M_{A} is the mass matrix between the dual space and $\mathcal{H}$ with respect to the chosen dual form. If M_{A} is invertible this is equivalent to $M_{\mathsf{A}}^{-1}Ax = \lambda x$ or $A^S x = \lambda x$. The situation is more complicated for the singular values. For simplicity consider a compact operator (e.g. the single layer boundary operator) acting on $L^2(\Gamma)$. We have that

$$\|A\|_{L^2(\Gamma)} = \sup_{\phi \in L^2(\Gamma)} \frac{\|A\phi\|_{L^2(\Gamma)}}{\|\phi\|_{L^2(\Gamma)}}.$$

Let $M = C^T C$ be the Cholesky decomposition of the $L^2(\Gamma)$ mass matrix M in a given discrete basis and A the Galerkin approximation in the same basis. Since $\|\phi\|_{L^2(\Gamma)} = \|C\mathbf{x}\|_2$ for a function ϕ living in the discrete subspace of $L^2(\Gamma)$ with given coefficient vector $\mathbf{x}$ it follows that

$$\|A\|_{L^2(\Gamma)} \approx \max_{\mathbf{x} \neq 0} \frac{\|CM^{-1}A\mathbf{x}\|_2}{\|C\mathbf{x}\|_2} = \|C^{-T}AC^{-1}\|_2,$$

which is generally not the same as $\|M^{-1}A\|_2$. So while the strong form correctly represents spectral information it does not recover norm or similar singular value based approximations.

5.2 Calderón preconditioning for acoustic transmission problems

As a final application we consider the Calderón preconditioned formulation of the following acoustic transmission problem.

$$\begin{aligned}
-\Delta u^+ - k^2 u^+ &= 0, &&\text{in } \Omega^+, \\
-\Delta u^- - n^2 k^2 u^- &= 0, &&\text{in } \Omega, \\
\gamma_0^- u^- &= \gamma_0^+ u^+ + \gamma_0^+ u^{\text{inc}}, &&\text{on } \Gamma, \\
\gamma_1^- u^- &= \gamma_1^+ u^+ + \gamma_1^+ u^{\text{inc}}, &&\text{on } \Gamma,
\end{aligned}$$

$$\lim_{|\mathbf{x}| \to \infty} |\mathbf{x}| \left(\frac{\partial}{\partial |\mathbf{x}|} u^+(\mathbf{x}) - iku^+(\mathbf{x}) \right) = 0. \tag{17}$$

Here, $n = c^+/c^-$ is the ratio of the speed of sound c^+ in the surrounding medium to the speed of sound c^- in the interior medium. The incident field is denoted by u^{inc}. The formulation that we present is based on [4]. A generalized framework for scattering through composites is discussed in [3]. We denote by $V^- := \begin{bmatrix} \gamma_0^- u^- & \gamma_1^- u^- \end{bmatrix}^T$, $V^+ := \begin{bmatrix} \gamma_0^+ u^+ & \gamma_1^+ u^+ \end{bmatrix}^T$, and $V^{\text{inc}} := \begin{bmatrix} \gamma_0^+ u^{\text{inc}} & \gamma_1^+ u^{\text{inc}} \end{bmatrix}^T$ the Cauchy data of u^-, u^+ and u^{inc}. Let A^+ be the multitrace operator associated with the wavenumber $k^+ := k$ and A^- the multitrace operator associated with $k^- := nk$ as defined in (5). From the Calderón projector it now follows that

$$\begin{aligned}
\left(\tfrac{1}{2}\mathsf{Id} + \mathsf{A}^- \right) V^- &= V^- \\
\left(\tfrac{1}{2}\mathsf{Id} - \mathsf{A}^+ \right) V^+ &= V^+
\end{aligned} \tag{18}$$

Together with the interface condition $V^- = V^+ + V^{\text{inc}}$ we can derive from these relationships the formulation

$$\left(\mathsf{A}^- + \mathsf{A}^+ \right) V^+ = \left(\tfrac{1}{2}\mathsf{Id} - \mathsf{A}^- \right) V^{\text{inc}}. \tag{19}$$
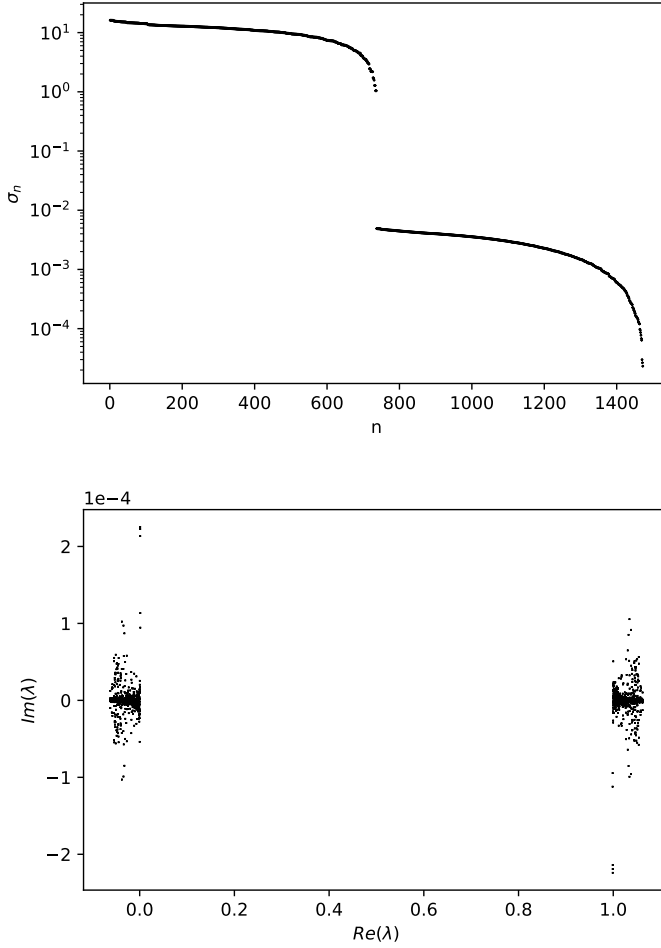
Fig. 3. Top: Singular values of the discrete strong form of the Calderón projector on the unit cube. Bottom: Eigenvalues of the discrete strong form.

This formulation is well defined for all wavenumbers $k > 0$ [4]. Moreover, it admits a simple preconditioning strategy [3] based on properties of the Calderón projector as follows. We note that A^+ is a compact perturbation of A^- [11]. We hence obtain

$$\left(\mathsf{A}^- + \mathsf{A}^+\right)^2 = \left(\mathsf{A}^- + \text{compact}\right)^2 = \tfrac{1}{4}\mathsf{Id} + \text{compact}.$$

We can therefore precondition (19) by squaring the left-hand side to arrive at

$$\left(\mathsf{A}^- + \mathsf{A}^+\right)^2 V^+ = \left(\mathsf{A}^- + \mathsf{A}^+\right)\left(\tfrac{1}{2}\mathsf{Id} - \mathsf{A}^-\right)V^{\text{inc}}. \tag{20}$$

With the block operator algebra in place in Bempp the main code snippet becomes

```
A_minus = multitrace_operator(grid, n * k, spaces="dual")
A_plus = multitrace_operator(grid, k, spaces="dual")
```
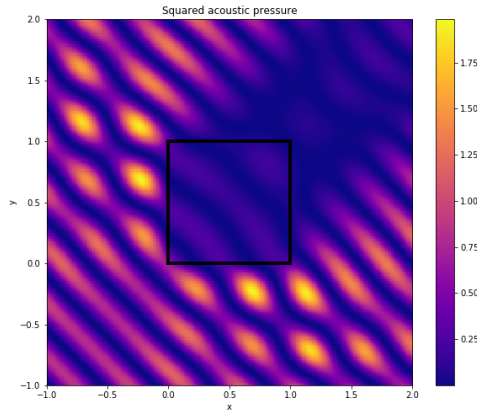
Fig. 4. Squared acoustic pressure distribution of a wave travelling through a piecewise homogeneous medium.

```
ident = multitrace_identity(grid, spaces="dual")
op = A_minus + A_plus
rhs_op = op * (.5 * ident - A_minus)
sol, info = bempp.linalg.gmres(op * op, rhs_op * v_inc,
                               use_strong_form=True)
```

As in the single-operator case we can intuitively write the underlying equations and solve them. All mass matrix transformations are being taken care off automatically. An example is shown in Figure 4. It demonstrates a two-dimensional slice at height 0.5 of a plane wave travelling through the unit cube. In this example $k = 10$ and $n = 0.8$. The system was solved in 7 GMRES iterations to a tolerance of 10^{-5}.

6 CONCLUSIONS

In this paper we have demonstrated how a Galerkin based product algebra can be defined and implemented. The underlying idea is very simple. Instead of an operator being defined just trough a domain and a test space we define it by a triplet of a domain space, range space, and dual to range (test) space. This is more natural in terms of the underlying mathematical description and allows the software implementation of an automatic Galerkin operator product algebra.

We have demonstrated the power of this algebra using three examples, the efficient evaluation of hypersingular boundary operators by single-layer operators, the computation of the singular values and eigenvalues of Calderón projectors, and the Calderón preconditioned solution of an acoustic transmission problem.

As long as an efficient LU decomposition of the involved mass matrices is possible the product algebra can be implemented with little overhead. Multiple LU decompositions of the same mass matrix can be easily avoided through caching.

In this paper we focused on Galerkin discretizations of boundary integral equations. Naturally, operator algebras are equally applicable to Galerkin discretizations of partial

differential equations. The main difference here is that for large-scale three dimensional problems an efficient LU decomposition of mass matrices may not always be possible.

Finally, we would like to stress that the underlying principle of this paper and its implementation in Bempp is to allow the user of software libraries to work as closely to the mathematical formulation as possible. Ideally, a user treats operators as continuous objects and lets the software do the rest while the library ensures mathematical correctness. The framework proposed in this paper and implemented in Bempp provides a step towards this goal.

While in this paper we have focused on acoustic problems the extension to Maxwell problems is straight forward and has been used in [12].

REFERENCES

[1] *SciPy.* www.scipy.org.
[2] A. Buffa and S. H. Christiansen, *A dual finite element complex on the barycentric refinement*, Mathematics of Computation, 76 (2007), pp. 1743–1769.
[3] X. Claeys and R. Hiptmair, *Multi-trace boundary integral formulation for acoustic scattering by composite structures*, Communications on Pure and Applied Mathematics, 66 (2013), pp. 1163–1201.
[4] M. Costabel and E. Stephan, *A direct boundary integral equation method for transmission problems*, Journal of Mathematical Analysis and Applications, 106 (1985), pp. 367 – 413.
[5] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, Journal of Computational Physics, 73 (1987), pp. 325 – 348.
[6] W. Hackbusch, *Hierarchical matrices: algorithms and analysis*, vol. 49 of Springer Series in Computational Mathematics, Springer, Heidelberg, 2015.
[7] M. A. Hamdi, *Une formulation variationnelle par équations intégrales pour la résolution de léquation de helmholtz avec des conditions aux limites mixtes*, CR Acad. Sci. Paris, série II, 292 (1981), pp. 17–20.
[8] R. Hiptmair, *Operator Preconditioning*, Computers & Mathematics with Applications, 52 (2006), pp. 699–706.
[9] R. C. Kirby, *From Functional Analysis to Iterative Methods*, SIAM Review, 52 (2010), pp. 269–293.
[10] J. C. Nedelec, *Integral equations with non integrable kernels*, Integral Equations and Operator Theory, 5 (1982), pp. 562–572.
[11] S. A. Sauter and C. Schwab, *Boundary element methods*, vol. 39 of Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 2011. Translated and expanded from the 2004 German original.
[12] M. W. Scroggs, T. Betcke, E. Burman, W. Śmigaj, and E. van t Wout, *Software frameworks for integral equations in electromagnetic scattering based on Calderón identities*, Computers & Mathematics with Applications, (2017).
[13] W. Śmigaj, T. Betcke, S. Arridge, J. Phillips, and M. Schweiger, *Solving Boundary Integral Problems with BEM++*, ACM Transactions on Mathematical Software, 41 (2015), pp. 1–40.
[14] O. Steinbach, *Numerical approximation methods for elliptic boundary value problems*, Springer, New York, 2008. Finite and boundary elements, Translated from the 2003 German original.
[15] O. Steinbach and W. L. Wendland, *The construction of some efficient preconditioners in the boundary element method*, Advances in Computational Mathematics, 9 (1998), pp. 191–216.