Front. Math. China
DOI 10.1007/s11464-012-0188-3

RESEARCH ARTICLE

A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains

Adrianna GILLMAN, Patrick M. YOUNG, Per-Gunnar MARTINSSON

Department of Applied Mathematics, University of Colorado at Boulder, Boulder, CO 80309-0526, USA

Abstract An algorithm for the direct inversion of the linear systems arising from Nyström discretization of integral equations on one-dimensional domains is described. The method typically has $O(N)$ complexity when applied to boundary integral equations (BIEs) in the plane with non-oscillatory kernels such as those associated with the Laplace and Stokes' equations. The scaling coefficient suppressed by the "big-O" notation depends logarithmically on the requested accuracy. The method can also be applied to BIEs with oscillatory kernels such as those associated with the Helmholtz and time-harmonic Maxwell equations; it is efficient at long and intermediate wave-lengths, but will eventually become prohibitively slow as the wave-length decreases. To achieve linear complexity, rank deficiencies in the o -diagonal blocks of the coefficient matrix are exploited. The technique is conceptually related to the $\mathscr{H}$- and $\mathscr{H}^2$-matrix arithmetic of Hackbusch and co-workers, and is closely related to previous work on *Hierarchically Semi-Separable* matrices.

Keywords Direct solver, integral equation, fast direct solver, boundary value problem, boundary integral equation, hierarchically semi-separable matrix
MSC 65R20,65F05

1 Introduction

1.1 Problem formulation

This paper describes techniques for numerically solving equations of the type

$$a(t)q(t) + \int_0^T b(t,t^{\times})q(t^{\times})\mathrm{d}t^{\times} = f(t), \quad t \in [0,T], \tag{1.1}$$

Received February 21, 2011; accepted August 28, 2011
Corresponding author: Per-Gunnar MARTINSSON, E-mail: martinss@colorado.edu

where $I = [0, T]$ is an interval on the line, and $a\colon I \to \mathbb{R}$ and $b\colon I \times I \to \mathbb{R}$ are given functions. We observe that a boundary integral equation (BIE) such as

$$a(\boldsymbol{x})q(\boldsymbol{x}) + \int b(\boldsymbol{x}, \boldsymbol{x}^{\backslash})q(\boldsymbol{x}^{\backslash})\mathrm{d}l(\boldsymbol{x}^{\backslash}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Gamma, \tag{1.2}$$

where Γ is a simple curve in the plane, takes the form (1.1) upon parameterization of the curve. The case of a domain Γ that consists of several non-connected simple curves can also be handled.

Upon discretization, equation (1.1) takes the form

$$\mathsf{A}\boldsymbol{q} = \boldsymbol{f}, \tag{1.3}$$

where A is a dense matrix of size, say, $N \times N$. When N is large, standard practice for rapidly solving a system such as (1.3) is to use an iterative solver (such as generalized minimum residual (GMRES), conjugate gradients) in which the matrix-vector multiplications are accelerated via a "fast" method such as the fast multipole method (FMM) [14], panel clustering [16], or Barnes-Hut [2]. When the integral equation (1.1) is a Fredholm equation of the second kind, the iteration typically converges rapidly, and a linear solver of e ectively $O(N)$ complexity results. In contrast, this paper reviews and extends a recently developed *direct* solver that in a single pass computes a data-sparse representation of a matrix S (a "solution operator") that satisfies

$$\mathsf{S} \quad \mathsf{A}^{-1}.$$

Once a representation of S is available, the solution of (1.3) is of course easily constructed:

$$\boldsymbol{q} \quad \mathsf{S}\boldsymbol{f}. \tag{1.4}$$

We will demonstrate that in many important environments (such as, e.g., the BIEs associated with Laplace's equation in the plane), the matrix S can be constructed in $O(N)$ operations.

1.2 Applications

The direct solver presented is applicable to most boundary integral equations associated with the classical boundary value problems of mathematical physics (Laplace, elasticity, Helmholtz, Yukawa, Stokes, etc.) with the two important exceptions that it is not efficient for (1) problems involving highly oscillatory kernels such as Helmholtz equation at short wavelengths, and (2) domain boundaries that tend to "fill space" in the sense illustrated in Fig. 1. We will demonstrate that boundaries with corners can be handled.

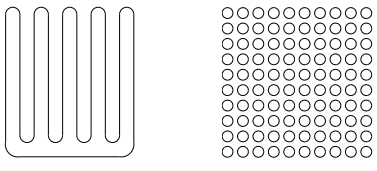


Fig. 1 Contours for which direct solver will *not* achieve $O(N)$ complexity

The direct solver is also applicable to many integral equations of the form (1.1) that arise in the analysis of special functions [27], in evaluating conformal maps [23], and in the analysis of two-point boundary value problems [26].

1.3 Advantages of direct solvers

Direct solvers o er several advantages over iterative ones.

Speed-up by large factors for problems involving multiple right-hand sides. In many situations, an equation such as (1.1) needs to be solved for several di erent data functions f. Iterative techniques can only to a limited extent take advantage of the fact that the operator is the same in each solve. For a direct method, on the other hand, each solve beyond the first simply involves applying a pre-computed inverse to a vector. The time required for applying the (compressed) inverse to a vector is typically much smaller than even the time required for a single application of the original operator using standard techniques.

The ability to solve relatively ill-conditioned problems. Direct solvers allow for the rapid and robust solution of linear systems involving relatively ill-conditioned matrices. In the context of boundary value problems, such ill-conditioning can be caused by physical ill-conditioning as observed, e.g., when solving the equations of elasticity on domains with high aspect ratios, or when solving scattering problems near a resonant frequency. It may also be introduced as a side-e ect of the mathematical formulation, e.g., when a formulation based on a Fredholm equation of the first kind is used, or when a model of a scattering problem introduces so called "spurious" resonances. Direct solvers are capable of computing solutions to such problems with very little loss of accuracy beyond that incurred by the ill-conditioning itself.

Increased reliability. Direct methods are inherently more robust than iterative methods. This point is less important in an academic setting where there is often time to tweak a code until it performs well for a particular problem (for instance, by designing a customized pre-conditioner). However, it has proven difficult to design industrial codes using iterative methods and commercial software developers sometimes tend to shun iterative methods in favor of direct ones, even at significant cost in terms of speed.

1.4 How the direct solver works

Letting ε denote a user specified computational tolerance, the direct solver for (1.1) can be viewed as consisting of four steps.

(i) *Quadrature nodes and quadrature weights for a Nyström discretization are created.* The interval $[0, T]$ is split into panels, and Gaussian nodes are placed on each panel. Customized quadrature weights are constructed using the method of [28] which ensures high accuracy even in the presence of weakly singular kernels (and for BIEs on domains with corners).

(ii) *Construction of the coe cient matrix.* The matrix A in (1.3) is an $N \times N$ matrix that is dense, but whose o -diagonal blocks are to high accuracy

rank-deficient. We exploit this fact, and compute an approximant $\mathsf{A}_{\text{approx}}$ which is stored in the data-sparse format very similar to the *hierarchically semi-separable* (HSS) format of [6,25].

(iii) *Inversion of the coe cient matrix.* The approximant $\mathsf{A}_{\text{approx}}$ of the coefficient matrix is inverted using a variation of the technique of [21,26] to produce the solution operator $\mathsf{S} = \mathsf{A}_{\text{approx}}^{-1}$. The inversion is exact up to round-o errors.

(iv) *Application of the approximate inverse.* The solution operator S is applied to the given data vector $\boldsymbol{f}$ to produce the solution $\boldsymbol{q}$, cf. (1.4).

Each of the four steps typically requires $O(N)$ work when applied to the standard equations of mathematical physics (with the two exceptions mentioned in Section 1.2). The constants of proportionality depend on the specific environment, but step (ii) is often the most expensive. The cost of step (iv) is tiny, meaning that the proposed procedure is particularly e ective in environments where a sequence of equations with the same coefficient matrix need to be solved.

Remark 1.1 The computations in steps (iii) and (iv) are independent of the specific problem being solved, and can be implemented as "black-box" codes.

1.5 Relationship to earlier work

The general idea of exploiting rank-deficiencies in the o -diagonal blocks of the matrix A in (1.3) underlies several "fast" algorithms (e.g., the Fast Multipole Method [14], panel clustering [16], Barnes-Hut [2]) for executing the matrix-vector multiply in an iterative solver. The observation that such rank-deficiencies can be also used in a systematic way to execute matrix inversion, matrix-matrix multiplies, matrix factorizations, etc., was made in the early work on $\mathscr{H}$-matrices by Hackbusch and co-workers [17]. The basic version of these methods have $O(N(\log N)^p)$ complexity for some small integer p. Certain operations were later accelerated to $O(N)$ complexity in the context of $\mathscr{H}^2$-matrices, see [4] and the references therein.

More specifically, the direct solver described in this paper is an evolution of the scheme of [21], which in turn draws on the earlier work [3,22,26]. Since [21] appeared, the algorithms have been significantly improved, primarily in that the compression and inversion steps have been separated. In addition to making the presentation much clearer, this separation leads to several concrete improvement, including:

Improved versatility. Separating the task of compression from the task of inversion makes it much easier to apply the direct solver to new applications. If a BIE with a di erent kernel is to be solved, then a slight modification of the compression step (step (ii) in Section 1.4) is sufficient. It also opens up the possibility of combining the direct solver with generic compression techniques based on randomized sampling, e.g., those described in [20].

Improved quadratures. The version of the algorithm described in this paper is compatible with the quadratures of [5,18] which enable the handling of BIEs

defined on domains with corners, and the quadratures of [28] which simplify the handling of singular kernels.

Improved theoretical analysis. The direct solver is in the present paper expressed transparently as a telescoping matrix factorization. This allows for a simplified error and stability analysis, as illustrated by, e.g., Lemma 3.1 and Corollary 3.2 below.

Improved interoperability with other data-sparse matrix formats. The new version of the algorithm makes it clear that the data-sparse format used to represent both the coefficient matrix and its inverse are essentially identical to the *hierarchically semi-separable* (HSS) format of [6,25]. This opens up the possibility of combining the compression techniques described in this paper with recently developed inversion and factorization algorithms for HSS matrices [8].

Remark 1.2 This paper uses the terms *block separable* (BS) and *hierarchically block separable* (HBS). The format we refer to as HBS is sufficiently similar to the *hierarchically semi-separable* (HSS) format that one should arguably avoid introducing a new term. However, we believe that at least for the purposes of this paper, the term HBS is clarifying since the matrix structure has almost nothing to do with the concept of a "semi-separable" matrix.

1.6 Outline

Section 2 introduces notation and reviews the Nyström discretization method for integral equations. Section 3 describes an accelerated direct solver based on a simplistic tessellation of an $N \times N$ matrix A into $p \times p$ blocks in such a way that all o -diagonal blocks are rank deficient. This method has complexity $O(p^{-2}N^3 + p^3k^3)$, where k is the rank of the o -diagonal blocks. To attain better asymptotic complexity, a more complicated hierarchical tessellation of the matrix must be implemented. This data structure is described in Section 4, and an $O(Nk^2)$ inversion technique is then described in Section 5. Section 6 describes efficient techniques for computing the data-sparse representation in the first place. Section 7 describes some numerical experiments, and Section 8 describes possible extensions of the work.

2 Preliminaries

This section introduces notation, and brie"y reviews some known techniques.

2.1 Notation

We say that a matrix U is *orthonormal* if its columns form an orthonormal set. An orthonormal matrix U preserves geometry in the sense that $|\mathsf{U}\boldsymbol{x}| = |\boldsymbol{x}|$ for every vector $\boldsymbol{x}$. We use the notation of [11] to denote submatrices: if A is an $m \times n$ matrix with entries $\mathsf{A}(i, j)$, and if

$$I = [i_1, i_2, \ldots, i_p], \quad J = [j_1, j_2, \ldots, j_q]$$

are two index vectors, then the associated $p \times q$ submatrix is expressed as

$$\mathsf{A}(I,J) = \begin{bmatrix} a_{i_1,j_1} & \cdots & a_{i_1,j_q} \\ \vdots & & \vdots \\ a_{i_p,j_1} & \cdots & a_{i_p,j_q} \end{bmatrix}.$$

For column- and row-submatrices, we use the standard abbreviations

$$\mathsf{A}(:,J) = \mathsf{A}([1,2,\ldots,m],J), \quad \mathsf{A}(I,:) = \mathsf{A}(I,[1,2,\ldots,n]).$$

2.2 Interpolatory decomposition (ID)

An $m \times n$ matrix B of rank k admits the factorization

$$\mathsf{B} = \mathsf{U}\mathsf{B}(J,:),$$

where $J = [j_1,\ldots,j_k]$ is a vector of integers such that $1 \quad j_i \quad m$, and U is an $m \times k$ matrix that contains the $k \times k$ identity matrix I_k (specifically, $\mathsf{U}(J,:) = \mathsf{I}_k$). Moreover, no entry of U is larger than one. Computing the ID of a matrix is in general combinatorially expensive, but if the restriction on element size of U is relaxed slightly to say that, for instance, each entry of U is bounded by 2, then very efficient schemes are available. See [9,15] for details.

2.3 Nyström discretization of integral equations in one dimension

In this subsection, we very brie"y describe some variations of the classical Nyström method for discretizing an integral equation such as (1.1). The material is well known and we refer to [1] for details.

For an integral equation with a smooth kernel $k(t,t^{\times})$, the Nyström method is particularly simple. The starting point is a quadrature rule for the interval $[0,T]$ with nodes $\{t_i\}_{i=1}^N \quad [0,T]$ and weights $\{ \ _i\}_{i=1}^N$ such that

$$\int_0^T b(t_i,t^{\times})q(t^{\times})\mathrm{d}t^{\times} \quad \sum_{j=1}^n b(t_i,t_j)q(t_j) \ _j, \quad i = 1,2,\ldots,N.$$

Then the discretized version of (1.1) is obtained by enforcing that

$$a(t_i)q(t_i) + \sum_{j=1}^n b(t_i,t_j)q(t_j) \ _j = f(t_i), \quad i = 1,2,\ldots,N. \tag{2.1}$$

We write (2.1) compactly as

$$\mathsf{A}\boldsymbol{q} = \boldsymbol{f},$$

where A is the $N \times N$ matrix with entries

$$\mathsf{A}(i,j) = \ _{i,j}a(t_i) + b(t_i,t_j) \ _j, \quad i,j = 1,2,\ldots,N,$$

$\boldsymbol{f}$ is the vector with entries

$$\boldsymbol{f}(i) = f(t_i), \quad i = 1,2,\ldots,N,$$

and $\boldsymbol{q}$ is the approximate solution which satisfies

$$\boldsymbol{q}(i) \quad q(t_i), \quad i = 1, 2, \ldots, N.$$

We have found that using a composite quadrature rule with a 10-point standard Gaussian quadrature on each panel is a versatile and highly accurate choice.

Remark 2.1 (Singular kernels) Some of the numerical examples described in Section 7 involve kernels with logarithmically singular kernels:

$$k(t, t^{\times}) \quad \log|t - t^{\dagger} \quad (t^{\times} \rightarrow t).$$

A standard quadrature rule designed for smooth functions would lose almost all accuracy on the panels where t and $t^{\times}$is close, but this can be remedied by modifying the matrix entries near the diagonal. For instance, when Gaussian quadrature nodes are used, the procedure described in [28] gives very accurate results. Alternatively, the Rokhlin-Kapur [19] procedure starts with a standard trapezoidal rule and modifies the weights near the end points to achieve high order convergence. This is a simpler method than the modified Gaussian rule of [28] but typically also produces lower accuracy.

Remark 2.2 (Contours with corners) Discretizing an integral equation such as (1.2) can be challenging if the contour Γ is not smooth. When $\boldsymbol{x} \in \Gamma$ is a corner point, the function $\boldsymbol{x}^{\times} \rightarrow b(\boldsymbol{x}, \boldsymbol{x}^{\times})$ typically has a singularity at $\boldsymbol{x}$. It has been demonstrated [5,18] that in many cases of practical interest, it is nevertheless possible to use standard quadrature weights designed for smooth functions, as long as the discretization is locally refined near the corner. The drawback is that such refinement can increase the system size in an undesirable way but as [18] demonstrates, the system size can be reduced via a local pre-computation. In this paper, we demonstrate that it is alternatively possible to use general purpose direct solvers to achieve the same e ect.

3 Inversion of block separable matrices

In this section, we define what it means for a matrix to be "block separable" and describe a simple technique for inverting such a matrix.

Let A be an $np \times np$ matrix that is blocked into $p \times p$ blocks, each of size $n \times n$:

$$\mathsf{A} = \begin{bmatrix} \mathsf{D}_1 & \mathsf{A}_{1,2} & \mathsf{A}_{1,3} & \cdots & \mathsf{A}_{1,p} \\ \mathsf{A}_{2,1} & \mathsf{D}_2 & \mathsf{A}_{2,3} & \cdots & \mathsf{A}_{2,p} \\ \vdots & \vdots & \vdots & & \vdots \\ \mathsf{A}_{p,1} & \mathsf{A}_{p,2} & \mathsf{A}_{p,3} & \cdots & \mathsf{D}_p \end{bmatrix}. \tag{3.1}$$

We say that A is "block separable" with "block-rank" k if for $= 1, 2, \ldots, p$, there exist $n \times k$ matrices U_τ and V_τ such that each o -diagonal block $\mathsf{A}_{,\tau}$ of A admits the factorization

$$\underset{n\times n}{\mathsf{A}}{}_{,\tau} = \underset{n\times k}{\mathsf{U}}\ \underset{k\times k}{\widetilde{\mathsf{A}}}{}_{,\tau}\ \underset{k\times n}{\mathsf{V}_\tau^*}, \qquad , \quad \in \{1,2,\ldots,p\}, \quad = \quad . \tag{3.2}$$

Observe that the columns of U must form a basis for the columns of all o -diagonal blocks in row , and analogously, the columns of V_τ must form a basis for the rows in all the o -diagonal blocks in column . When (3.2) holds, the matrix A admits a block factorization

$$\underset{np\times np}{\mathsf{A}} = \underset{np\times kp}{\underline{\mathsf{U}}}\ \underset{kp\times kp}{\widetilde{\mathsf{A}}}\ \underset{kp\times np}{\underline{\mathsf{V}}^*} + \underset{np\times np}{\underline{\mathsf{D}}}, \tag{3.3}$$

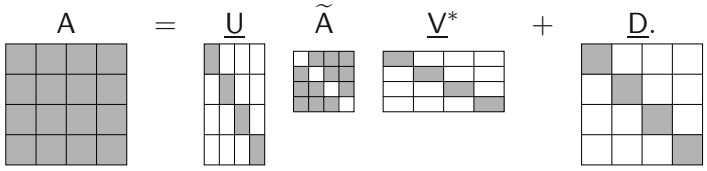
where

$$\underline{\mathsf{U}} = \operatorname{diag}(\mathsf{U}_1, \mathsf{U}_2, \ldots, \mathsf{U}_p),$$
$$\underline{\mathsf{V}} = \operatorname{diag}(\mathsf{V}_1, \mathsf{V}_2, \ldots, \mathsf{V}_p),$$
$$\underline{\mathsf{D}} = \operatorname{diag}(\mathsf{D}_1, \mathsf{D}_2, \ldots, \mathsf{D}_p),$$

and

$$\widetilde{\mathsf{A}} = \begin{bmatrix} 0 & \widetilde{\mathsf{A}}_{12} & \widetilde{\mathsf{A}}_{13} & \cdots \\ \widetilde{\mathsf{A}}_{21} & 0 & \widetilde{\mathsf{A}}_{23} & \cdots \\ \widetilde{\mathsf{A}}_{31} & \widetilde{\mathsf{A}}_{32} & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

The block structure of formula (3.3) for $p = 4$ is illustrated below:



(3.4)

The idea is that by excising the diagonal blocks from A, we obtain a rank-deficient matrix $\mathsf{A} - \underline{\mathsf{D}}$ that can be factored with block diagonal "anking matrices:

$$\mathsf{A} - \underline{\mathsf{D}} = \underline{\mathsf{U}}\widetilde{\mathsf{A}}\underline{\mathsf{V}}^*.$$

The inverse of a block-separable matrix can rapidly be constructed using the following simple variation of the classical Sherman-Morrison-Woodbury formula.

Lemma 3.1 *Suppose that* A *is an* $N \times N$ *invertible matrix. Suppose further that* K *is a positive integer such that* $K < N$*, that* A *admits the decomposition*

$$\underset{N\times N}{\mathsf{A}} = \underset{N\times K}{\mathsf{U}}\ \underset{K\times K}{\widetilde{\mathsf{A}}}\ \underset{K\times N}{\mathsf{V}^*} + \underset{N\times N}{\mathsf{D}}, \tag{3.5}$$

and that the matrices D, $\mathsf{V}^*\mathsf{D}^{-1}\mathsf{U}$, *and* $\widetilde{\mathsf{A}} + (\mathsf{V}^*\mathsf{D}^{-1}\mathsf{U})^{-1}$ *are invertible. Then*

$$\mathsf{A}^{-1} = \mathsf{E}(\widetilde{\mathsf{A}} + \mathsf{D})^{-1}\mathsf{F}^* + \mathsf{G}, \tag{3.6}$$

where

$$\mathsf{D} = (\mathsf{V}^*\mathsf{D}^{-1}\mathsf{U})^{-1}, \tag{3.7}$$

$$\mathsf{E} = \mathsf{D}^{-1}\mathsf{U}\mathsf{D}, \tag{3.8}$$

$$\mathsf{F} = (\mathsf{D}\mathsf{V}^*\mathsf{D}^{-1})^*, \tag{3.9}$$

$$\mathsf{G} = \mathsf{D}^{-1} - \mathsf{D}^{-1}\mathsf{U}\mathsf{D}\mathsf{V}^*\mathsf{D}^{-1}. \tag{3.10}$$

When A is block-separable, (3.5) holds with block diagonal matrices U, V, and D. The matrices D, E, F, and G can then be evaluated rapidly, and Lemma 3.1 can be said to reduce the task of inverting the $np \times np$ matrix A, to the task of inverting the $kp \times kp$ matrix $\widetilde{\mathsf{A}} + \mathsf{D}$.

Proof of Lemma 3.1 Consider the equation

$$(\mathsf{U}\widetilde{\mathsf{A}}\mathsf{V}^* + \mathsf{D})\boldsymbol{q} = \boldsymbol{u}. \tag{3.11}$$

We will prove that (3.6) holds by proving that the solution $\boldsymbol{q}$ of (3.11) is the right-hand side of (3.6) applied to $\boldsymbol{u}$. First, we set

$$\boldsymbol{q} = \mathsf{V}^*\boldsymbol{q}. \tag{3.12}$$

Then (3.11) can be written as

$$\mathsf{U}\widetilde{\mathsf{A}}\boldsymbol{q} + \mathsf{D}\boldsymbol{q} = \boldsymbol{u}. \tag{3.13}$$

Solving (3.13) for $\boldsymbol{q}$ and inserting the result into (3.12), we obtain

$$(\mathsf{I} + \underbrace{\mathsf{V}^*\mathsf{D}^{-1}\mathsf{U}}_{=\mathsf{D}^{-1}}\widetilde{\mathsf{A}})\boldsymbol{q} = \mathsf{V}^*\mathsf{D}^{-1}\boldsymbol{u}. \tag{3.14}$$

Multiplying (3.14) by D, we find that

$$(\mathsf{D} + \widetilde{\mathsf{A}})\boldsymbol{q} = \underbrace{\mathsf{D}\mathsf{V}^*\mathsf{D}^{-1}}_{=\mathsf{F}}\boldsymbol{u}. \tag{3.15}$$

Now, note that from (3.13) it also follows that

$$\boldsymbol{q} = -\mathsf{D}^{-1}\mathsf{U}\widetilde{\mathsf{A}}\boldsymbol{q} \ + \mathsf{D}^{-1}\boldsymbol{u}. \tag{3.16}$$

From (3.15), we know that

$$\widetilde{\mathsf{A}}\boldsymbol{q} = -\mathsf{D}\boldsymbol{q} + \mathsf{F}^*\boldsymbol{u}. \tag{3.17}$$

Inserting (3.17) into (3.16), we obtain

$$\boldsymbol{q} = -\mathsf{D}^{-1}\mathsf{U}(-\mathsf{D}\boldsymbol{q} + \mathsf{F}^*\boldsymbol{u}) + \mathsf{D}^{-1}\boldsymbol{u} = \underbrace{\mathsf{D}^{-1}\mathsf{U}\mathsf{D}}_{=\mathsf{E}}\boldsymbol{q} + \underbrace{(\mathsf{D}^{-1} - \mathsf{D}^{-1}\mathsf{U}\mathsf{F}^*)}_{=\mathsf{G}}\boldsymbol{u}. \tag{3.18}$$

Solving (3.15) for $\boldsymbol{q}$ and inserting the result into (3.18), we obtain the expression (3.6). □

The technique provided by Lemma 3.1 is a useful tool in its own right. It can often reduce the cost of inverting an $N \times N$ matrix from the $O(N^3)$ cost of Gaussian elimination, to $O(N^{1.8})$, see Remark 3.1. Even more significant is that for many matrices, including the ones under consideration in this paper, the process described can be continued recursively which leads to an $O(N)$ inversion scheme. The required hierarchical representations of matrices are described in Section 4, and the $O(N)$ inversion scheme is given in Section 5.

Remark 3.1 To assess the computational cost of applying Lemma 3.1, suppose that A is a BS matrix whose $p \times p$ blocks of size $n \times n$ satisfy (3.2). Then evaluating the factors D, E, F, and G requires $O(pn^3)$ operations. Evaluating $(\widetilde{\mathsf{A}} + \mathsf{D})^{-1}$ requires $O(p^3k^3)$ operations. The total cost T_{BS} of evaluating the factors in formula (3.6) therefore satisfies

$$T_{\mathrm{BS}} \quad pn^3 + p^3k^3. \tag{3.19}$$

The cost (3.19) should be compared to the $O(p^3n^3)$ cost of evaluating A^{-1} directly. To elucidate the comparison, suppose temporarily that we are given a matrix A of fixed size $N \times N$, and can choose how many blocks p we wish to partition it into. Then $n \quad N/p$, and the total cost satisfies

$$T_{\mathrm{BS}} \quad p^{-2}N^3 + p^3k^3. \tag{3.20}$$

If the rank of interaction k is independent of the block size, we can set $p \quad N^{3/5}$, whence

$$T_{\mathrm{BS}} \quad k^3N^{9/5}. \tag{3.21}$$

In practice, the numerical rank of the o -diagonal blocks typically increases slightly as the block size n is increased, but the increase tends to be moderate. For instance, for the matrices under consideration in this paper, one typically sees

$$k \quad \log n \quad \log \frac{N}{p}.$$

In such an environment, setting

$$p \quad N^{3/5}(\log N)^{-3/5}$$

transforms (3.20) to, at worst,

$$T_{\mathrm{BS}} \quad (\log N)^{6/5}N^{9/5}.$$

The calculations in this remark do not include the cost of actually constructing the factors U, V, and D. For a general matrix, this cost is $O(kN^2)$, but for the matrices under consideration in this paper, techniques with better asymptotic complexity are described in Section 6.

We close by showing that when the matrix A is symmetric positive definite (spd), the assumption in Lemma 3.1 that certain intermediate matrices are invertible can be omitted.

Corollary 3.2 *Let* A *be a symmetric positive de nite* (*spd*) *matrix that admits a factorization*

$$\underset{N\times N}{\mathsf{A}} = \underset{N\times K}{\mathsf{U}}\ \underset{K\times K}{\widetilde{\mathsf{A}}}\ \underset{K\times N}{\mathsf{U}^*} + \underset{N\times N}{\mathsf{D}}, \tag{3.22}$$

where $\ker(\mathsf{U}) = \{\mathbf{0}\}$ *and* D *is a block diagonal submatrix of* A. *Then the matrices* D, $\mathsf{U}^*\mathsf{D}^{-1}\mathsf{U}$, *and* $\widetilde{\mathsf{A}} + (\mathsf{U}^*\mathsf{D}^{-1}\mathsf{U})^{-1}$ *are spd* (*and hence invertible*).

Proof That D is spd follows immediately from the fact that it is a block diagonal submatrix of an spd matrix.

To show that $\mathsf{U}^*\mathsf{D}^{-1}\mathsf{U}$ is spd, we pick any $\boldsymbol{x} = \mathbf{0}$, set $\boldsymbol{y} = \mathsf{D}^{-1}\mathsf{U}\boldsymbol{x}$, observe that $\boldsymbol{y} = 0$ since $\ker(\mathsf{U}) = \{\mathbf{0}\}$, and then we find that

$$\mathsf{U}^*\mathsf{D}^{-1}\mathsf{U}\boldsymbol{x}, \boldsymbol{x} \ = \ \mathsf{D}^{-1}\mathsf{U}, \mathsf{U}\boldsymbol{x} \ = \ \boldsymbol{y}, \mathsf{D}\boldsymbol{y} \ > 0$$

since D is spd.

It remains only to prove that $\widetilde{\mathsf{A}} + (\mathsf{U}^*\mathsf{D}^{-1}\mathsf{U})^{-1}$ is spd. To this end, define D and E via

$$\mathsf{D} = (\mathsf{U}^*\mathsf{D}^{-1}\mathsf{U})^{-1}, \quad \mathsf{E} = \mathsf{D}^{-1}\mathsf{U}\mathsf{D}.$$

Then

$$\begin{aligned}
\widetilde{\mathsf{A}} + \mathsf{D} &= \mathsf{D}(\mathsf{D}^{-1}\widetilde{\mathsf{A}}\mathsf{D}^{-1} + \mathsf{D}^{-1})\mathsf{D} \\
&= \mathsf{D}(\mathsf{U}^*\mathsf{D}^{-1}\mathsf{U}\widetilde{\mathsf{A}}\mathsf{U}^*\mathsf{D}^{-1}\mathsf{U} + \mathsf{D}^{-1})\mathsf{D} \\
&= \mathsf{D}(\mathsf{U}^*\mathsf{D}^{-1}(\mathsf{A} - \mathsf{D})\mathsf{D}^{-1}\mathsf{U} + \mathsf{U}^*\mathsf{D}^{-1}\mathsf{U})\mathsf{D} \\
&= \mathsf{D}\mathsf{U}^*\mathsf{D}^{-1}\mathsf{A}\mathsf{D}^{-1}\mathsf{U}\mathsf{D} \\
&= \mathsf{E}^*\mathsf{A}\mathsf{E}.
\end{aligned}$$

That $\widetilde{\mathsf{A}} + (\mathsf{U}^*\mathsf{D}^{-1}\mathsf{U})^{-1}$ is spd now follows since $\ker(\mathsf{E}) = \{\mathbf{0}\}$ and A is spd. □

Remark 3.2 The proof of Corollary 3.2 demonstrates that the stability of the method can readily be assessed by tracking the conditioning of the matrices E. If these matrices are close to orthogonal (i.e., all their singular values are similar in magnitude), then the compressed matrix $\widetilde{\mathsf{A}} + \mathsf{D}$ has about the same distribution of singular values as A since

$$\widetilde{\mathsf{A}} + \mathsf{D} = \mathsf{E}^*\mathsf{A}\mathsf{E}.$$

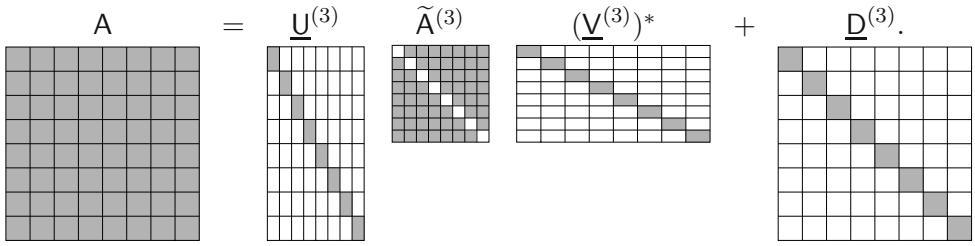
4 Hierarchically block separable matrices

In this section, we define what it means for an $N \times N$ matrix A to be HBS. Section 4.1 informally describes the basic ideas. Section 4.2 describes a simple binary tree of subsets of the index vector $[1, 2, \ldots, N]$. Section 4.3 provides

a formal definition of the HBS property. Section 4.4 describes how an HBS matrix can be expressed a telescoping factorization. Section 4.5 describes an $O(N)$ procedure for applying an HBS matrix to a vector.

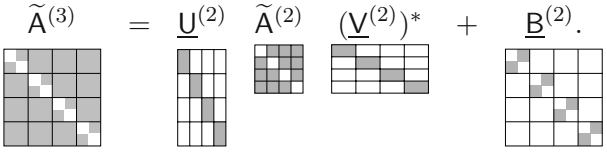
4.1 Heuristic description

The HBS property, first of all, requires A to be BS. Supposing that A consists of 8×8 blocks, this means that A admits a factorization, cf. (3.4),



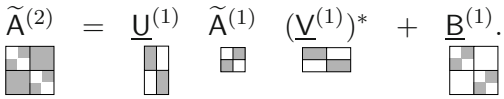
(4.1)

The superscripts on the right-hand side of (4.1) indicate that the factorization is at what we call "level 3" (the motivation for this labeling should become clear once we reach equation (4.4)). We next require the smaller matrix $\widetilde{\mathsf{A}}^{(3)}$ to be BS, and to admit the analogous factorization:



(4.2)

In forming (4.2), we reblocked the matrix $\widetilde{\mathsf{A}}^{(3)}$ by merging blocks in groups of four. The purpose is to "reintroduce" rank deficiencies in the o -diagonal blocks. In the final step in the hierarchy, we require that upon reblocking, $\widetilde{\mathsf{A}}^{(2)}$ is BS and admits a factorization:

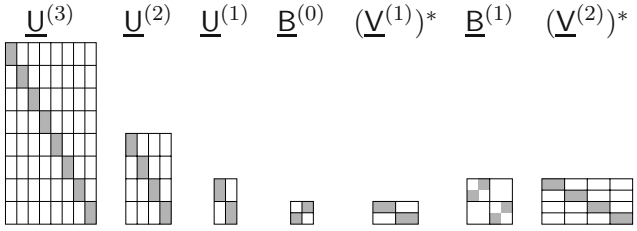


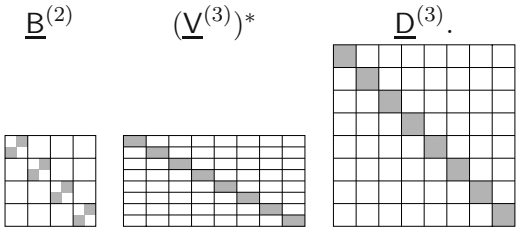
(4.3)

Combining (4.1)–(4.3), we find that A can be expressed as

$$\mathsf{A} = \underline{\mathsf{U}}^{(3)}(\underline{\mathsf{U}}^{(2)}(\underline{\mathsf{U}}^{(1)}\underline{\mathsf{B}}^{(0)}(\underline{\mathsf{V}}^{(1)})^* + \underline{\mathsf{B}}^{(1)})(\underline{\mathsf{V}}^{(2)})^* + \underline{\mathsf{B}}^{(2)})(\underline{\mathsf{V}}^{(3)})^* + \underline{\mathsf{D}}^{(3)}. \tag{4.4}$$

The block structure of the right-hand side of (4.4) is





In other words, the HBS property lets us completely represent a matrix in terms of certain block diagonal factors. The total cost of storing these factors is $O(Nk)$, and the format is an example of a so-called "data-sparse" representation of the matrix.

Remark 4.1 In describing the HBS property, we assumed that all o -diagonal blocks at all levels have the same rank k. We do this for notational clarity only. In practice, the minimal rank tends to vary slightly from block to block, and to moderately increase on the coarser levels. In the numerical examples in Section 7, all codes determine the rank adaptively for each block.

4.2 Tree structure

The HBS representation of an $N \times N$ matrix A is based on a partition of the index vector $I = [1, 2, \ldots, N]$ into a binary tree structure. For simplicity, we limit attention to binary tree structures in which every level is fully populated. We let I form the root of the tree, and give it the index 1, $I_1 = I$. We next split the root into two roughly equi-sized vectors I_2 and I_3 so that $I_1 = I_2 \quad I_3$. The full tree is then formed by continuing to subdivide any interval that holds more than some preset fixed number n of indices. We use the integers $= 0, 1, \ldots, L$ to label the di erent levels, with 0 denoting the coarsest level. A *leaf* is a node corresponding to a vector that never got split. For a non-leaf node , its *children* are the two boxes $_1$ and $_2$ such that $I_\tau = I_{\ _1} \quad I_{\ _2}$, and is then the *parent* of $_1$ and $_2$. Two boxes with the same parent are called *siblings*. These definitions are illustrated in Figure 2.

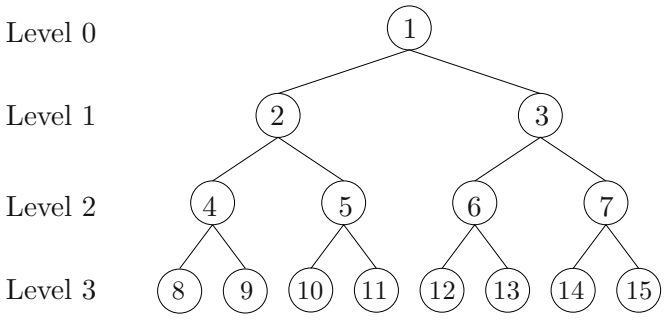


Fig. 2 Numbering of nodes in a fully populated binary tree with $L = 3$ levels. The root is the original index vector $I = I_1 = [1, 2, \ldots, 400]$; and $I_2 = [1, 2, \ldots, 200]$, $I_3 = [201, 202, \ldots, 400]$; $I_4 = [1, 2, \ldots, 100]$, $I_5 = [101, 102, \ldots, 200], \ldots$; $I_8 = [1, 2, \ldots, 50]$, $I_9 = [51, 52, \ldots, 100], \ldots$, $I_{15} = [351, 352, \ldots, 400]$.

Remark 4.2 The HBS format works with a broad range of di erent tree structures. It is permissible to split a node into more than two children if desirable, to distribute the points in an index set unevenly among its children, to split only some nodes on a given level, etc. This "exibility is essential when A approximates a non-uniformly discretized integral operator; in this case, the partition tree it constructed based on a geometric subdivision of the domain of integration. The *spatial* geometry of a box then dictates whether and how it is to be split. The algorithms of this paper can easily be modified to accommodate general trees.

4.3 Definition of HBS property

We are now prepared to rigorously define what it means for an $N \times N$ matrix A to be *hierarchically block seperable* with respect to a given binary tree $\mathscr{T}$ that partitions the index vector $J = [1, 2, \ldots, N]$. For simplicity, we suppose that the tree has L fully populated levels, and that for every leaf node , the index vector I_τ holds precisely n points, so that $N = n2^L$. Then A is HBS with block rank k if the following two conditions hold.

(1) *Assumption on ranks of o -diagonal blocks at the nest level.* For any two distinct leaf nodes and $^{\times}$, define the $n \times n$ matrix

$$\mathsf{A}_{\tau,\tau^*} = \mathsf{A}(I_\tau, I_{\tau^*}). \tag{4.5}$$

Then there must exist matrices U_τ, V_{τ^*}, and $\widetilde{\mathsf{A}}_{\tau,\tau^*}$ such that

$$\underset{n\times n}{\mathsf{A}_{\tau,\tau^*}} = \underset{n\times k}{\mathsf{U}_\tau}\ \underset{k\times k}{\widetilde{\mathsf{A}}_{\tau,\tau^*}}\ \underset{k\times n}{\mathsf{V}^*_{\tau^*}}. \tag{4.6}$$

(2) *Assumption on ranks of o -diagonal blocks on level* $= L-1, L-2, \ldots, 1$. The rank assumption at level is defined in terms of the blocks constructed on the next finer level $+ 1$. For any distinct nodes and $^{\times}$on level with children $_1$, $_2$ and $^{\times}_1$, $^{\times}_2$, respectively, define

$$\mathsf{A}_{\tau,\tau^*} = \begin{bmatrix} \widetilde{\mathsf{A}}_{\ 1,\ ^*_1} & \widetilde{\mathsf{A}}_{\ 1,\ ^*_2} \\ \widetilde{\mathsf{A}}_{\ 2,\ ^*_1} & \widetilde{\mathsf{A}}_{\ 2,\ ^*_2} \end{bmatrix}. \tag{4.7}$$

Then there must exist matrices U_τ, V_{τ^*}, and $\widetilde{\mathsf{A}}_{\tau,\tau^*}$ such that

$$\underset{2k\times 2k}{\mathsf{A}_{\tau,\tau^*}} = \underset{2k\times k}{\mathsf{U}_\tau}\ \underset{k\times k}{\widetilde{\mathsf{A}}_{\tau,\tau^*}}\ \underset{k\times 2k}{\mathsf{V}^*_{\tau^*}}. \tag{4.8}$$

The two points above complete the definition. An HBS matrix is now fully described if the basis matrices U_τ and V_τ are provided for each node , and in addition, we are for each leaf given the $n \times n$ matrix

$$\mathsf{D}_\tau = \mathsf{A}(I_\tau, I_\tau), \tag{4.9}$$

and for each parent node τ with children τ_1 and τ_2, we are given the $2k \times 2k$ matrix

$$\mathsf{B}_\tau = \begin{bmatrix} 0 & \widetilde{\mathsf{A}}_{\tau_1,\tau_2} \\ \widetilde{\mathsf{A}}_{\tau_2,\tau_1} & 0 \end{bmatrix}. \tag{4.10}$$

Observe in particular that the matrices $\widetilde{\mathsf{A}}_{\tau_1,\tau_2}$ are only required when $\{\tau_1, \tau_2\}$ forms a sibling pair. Table 1 summarizes the required matrices.

Table 1 An HBS matrix A associated with a tree $\mathscr{T}$ is fully specified if factors listed are provided

	name	size	function
for each leaf node	D_τ	$n \times n$	diagonal block $\mathsf{A}(I_\tau, I_\tau)$
	U_τ	$n \times k$	basis for columns in blocks in row
	V_τ	$n \times k$	basis for rows in blocks in column
for each parent node	B_τ	$2k \times 2k$	interactions between children of
	U_τ	$2k \times k$	basis for columns in (reduced) blocks in row
	V_τ	$2k \times k$	basis for rows in (reduced) blocks in column

Remark 4.3 The definition of the HBS property given in this section is "exible in the sense that we do not enforce any conditions on the factors U_τ, V_τ, and $\widetilde{\mathsf{A}}_{\tau,\tau^*}$ other than that (4.6) and (4.8) must hold. For purposes of numerical stability, further conditions are sometimes imposed. The perhaps strongest such condition is to require the matrices U_τ and V_{τ^*} in (4.6) and (4.8) be orthonormal, see, e.g., [6,25] (one can in this case require that the matrices $\widetilde{\mathsf{A}}_{\tau,\tau^*}$ be diagonal, so that (4.6) and (4.8) become singular value decompositions). If a "general" HBS matrix is given, it can easily be converted to this more restrictive format via, e.g., Algorithm 1. A choice that we have found highly convenient is to require (4.6) and (4.8) to be interpolatory decompositions (see Section 2.2). Then every U_τ and V_{τ^*} contains a $k \times k$ identity matrix (which greatly accelerates computations), and each $\widetilde{\mathsf{A}}_{\tau,\tau^*}$ is a submatrix of the original matrix A.

Remark 4.4 The definition (4.6) transparently describes the functions of the basis matrices U_τ and V_τ whenever τ is a leaf node. The definition (4.8) of basis matrices for non-leaf nodes is perhaps less intuitive. Their meaning can be made clearer by defining what we call "extended" basis matrices $\mathsf{U}_\tau^{\text{extend}}$ and $\mathsf{V}_\tau^{\text{extend}}$. For a leaf node τ, we simply set

$$\mathsf{U}_\tau^{\text{extend}} = \mathsf{U}_\tau, \quad \mathsf{V}_\tau^{\text{extend}} = \mathsf{V}_\tau.$$

For a parent node τ with children τ_1 and τ_2, we set

$$\mathsf{U}_\tau^{\text{extend}} = \begin{bmatrix} \mathsf{U}_{\tau_1}^{\text{extend}} & 0 \\ 0 & \mathsf{U}_{\tau_2}^{\text{extend}} \end{bmatrix} \mathsf{U}_\tau, \quad \mathsf{V}_\tau^{\text{extend}} = \begin{bmatrix} \mathsf{V}_{\tau_1}^{\text{extend}} & 0 \\ 0 & \mathsf{V}_{\tau_2}^{\text{extend}} \end{bmatrix} \mathsf{V}_\tau.$$

Then for any distinct nodes τ and τ^* on level ℓ, we have

$$\underset{n2^{L-\ell} \times n2^{L-\ell}}{\mathsf{A}(I_\tau, I_{\tau^*})} = \underset{n2^{L-\ell} \times k}{\mathsf{U}_\tau^{\text{extend}}} \; \underset{k \times k}{\widetilde{\mathsf{A}}_{\tau,\tau^*}} \; \underset{k \times n2^{L-\ell}}{(\mathsf{V}_\tau^{\text{extend}})^*}.$$

ALGORITHM 1 (reformatting an HBS matrix)

Given the factors U_τ, V_τ, $\widetilde{\mathsf{A}}_{\ell,\ 1,\ 2}$, D_τ *of an HBS matrix in general format, this algorithm computes new factors (that overwrite the old) such that all* U_τ *and* V_τ *are orthonormal, and all* $\widetilde{\mathsf{A}}_{\ell,\ 1,\ 2}$ *are diagonal.*

loop over levels, finer to coarser, $\ell = L-1, L-2, \ldots, 0$,
  loop over all parent boxes $\ell\ 1$ on level ℓ,
    Let $\ell\ 1$ and $\ell\ 2$ denote the children of $\ell\ 1$.

$$[\mathsf{U}_1, \mathsf{R}_1] = \mathtt{qr}(\mathsf{U}_{\ell+1\ 1}). \qquad [\mathsf{U}_2, \mathsf{R}_2] = \mathtt{qr}(\mathsf{U}_{\ell+1\ 2}).$$
$$[\mathsf{V}_1, \mathsf{S}_1] = \mathtt{qr}(\mathsf{V}_{\ell+1\ 1}). \qquad [\mathsf{V}_2, \mathsf{S}_2] = \mathtt{qr}(\mathsf{V}_{\ell+1\ 2}).$$
$$[\mathsf{X}_1, \ \mathsf{T}_{12}, \mathsf{Y}_2] = \mathtt{svd}(\mathsf{R}_1 \widetilde{\mathsf{A}}_{\ell,\ 1,\ 2} \mathsf{S}_2^*). \qquad [\mathsf{X}_2, \ \mathsf{T}_{21}, \mathsf{Y}_1] = \mathtt{svd}(\mathsf{R}_2 \widetilde{\mathsf{A}}_{\ell,\ 2,\ 1} \mathsf{S}_1^*).$$
$$\mathsf{U}_{\ell\ 1} \leftarrow \mathsf{U}_1 \mathsf{X}_1. \qquad \mathsf{U}_{\ell\ 2} \leftarrow \mathsf{U}_2 \mathsf{X}_2.$$
$$\mathsf{V}_{\ell\ 1} \leftarrow \mathsf{V}_1 \mathsf{Y}_1. \qquad \mathsf{V}_{\ell\ 2} \leftarrow \mathsf{V}_2 \mathsf{Y}_2.$$
$$\mathsf{B}_{\ell\ 1\ 2} \leftarrow \mathsf{T}_{12}. \qquad \mathsf{B}_{\ell\ 2\ 1} \leftarrow \mathsf{T}_{21}.$$

    if $\ell > 0$,

$$\mathsf{U}_\tau \leftarrow \begin{bmatrix} \mathsf{X}_1^* \mathsf{R}_1 & 0 \\ 0 & \mathsf{X}_2^* \mathsf{R}_2 \end{bmatrix} \mathsf{U}_\tau. \qquad \mathsf{V}_\tau \leftarrow \begin{bmatrix} \mathsf{Y}_1^* \mathsf{S}_1 & 0 \\ 0 & \mathsf{Y}_2^* \mathsf{S}_2 \end{bmatrix} \mathsf{V}_\tau.$$

    end if
  end loop
end loop

4.4 Telescoping factorizations

In the heuristic description of the HBS property in Section 4.1, we claimed that any HBS matrix can be expressed as a telescoping factorization with block diagonal factors. We will now formalize this claim. Given the matrices defined in Section 4.3 (and summarized in Table 1), we define the following block diagonal factors:

$$\underline{\mathsf{D}}^{(\ell)} = \mathrm{diag}(\mathsf{D}_\tau : \quad \tau \text{ is a box on level } \ell), \qquad \ell = 0, 1, \ldots, L, \tag{4.11}$$

$$\underline{\mathsf{U}}^{(\ell)} = \mathrm{diag}(\mathsf{U}_\tau : \quad \tau \text{ is a box on level } \ell), \qquad \ell = 1, 2, \ldots, L, \tag{4.12}$$

$$\underline{\mathsf{V}}^{(\ell)} = \mathrm{diag}(\mathsf{V}_\tau : \quad \tau \text{ is a box on level } \ell), \qquad \ell = 1, 2, \ldots, L, \tag{4.13}$$

$$\underline{\mathsf{B}}^{(\ell)} = \mathrm{diag}(\mathsf{B}_\tau : \quad \tau \text{ is a box on level } \ell), \qquad \ell = 0, 1, \ldots, L-1. \tag{4.14}$$

Furthermore, we let $\widetilde{\mathsf{A}}^{(\ell)}$ denote the block matrix whose diagonal blocks are zero, and whose o-diagonal blocks are the blocks $\widetilde{\mathsf{A}}_{\tau,\tau^*}$ for all distinct τ, τ^* on level ℓ. With these definitions,

$$\underset{n2^L \times n2^L}{\mathsf{A}} = \underset{n2^L \times k2^L}{\underline{\mathsf{U}}^{(L)}} \ \underset{k2^L \times k2^L}{\widetilde{\mathsf{A}}^{(L)}} \ \underset{k2^L \times n2^L}{(\underline{\mathsf{V}}^{(L)})^*} + \underset{n2^L \times n2^L}{\underline{\mathsf{D}}^{(L)}}; \tag{4.15}$$

for $\ell = L-1, L-2, \ldots, 1$, we have

$$\underset{k2^{\ell+1} \times k2^{\ell+1}}{\widetilde{\mathsf{A}}^{(\ell+1)}} = \underset{k2^{\ell+1} \times k2^\ell}{\underline{\mathsf{U}}^{(\ell)}} \ \underset{k2^\ell \times k2^\ell}{\widetilde{\mathsf{A}}^{(\ell)}} \ \underset{k2^\ell \times k2^{\ell+1}}{(\underline{\mathsf{V}}^{(\ell)})^*} + \underset{k2^{\ell+1} \times k2^{\ell+1}}{\underline{\mathsf{B}}^{(\ell)}}; \tag{4.16}$$

and finally,

$$\widetilde{\mathsf{A}}^{(1)} = \underline{\mathsf{B}}^{(0)}. \tag{4.17}$$

4.5 Matrix-vector multiplication

The telescoping factorizations in Section 4.4 easily translate into a formula for evaluating the matrix-vector product $\boldsymbol{u} = \mathsf{A}\boldsymbol{q}$ once all factors in an HBS representation have been provided. The resulting algorithm has computational complexity $O(Nk)$ (assuming that $n = O(k)$), and is given as Algorithm 2.

ALGORITHM 2 (HBS matrix-vector multiply)

Given a vector $\boldsymbol{q}$ *and a matrix* A *in HBS format, compute* $\boldsymbol{u} = \mathsf{A}\boldsymbol{q}$.

loop over all leaf boxes ,
  $\boldsymbol{q}_\tau = \mathsf{V}_\tau^* \boldsymbol{q}(I_\tau)$.
end loop

loop over levels, finer to coarser, $= L-1, L-2, \ldots, 1$,
  loop over all parent boxes on level ,
    Let $_1$ and $_2$ denote the children of .
    $$\boldsymbol{q}_\tau = \mathsf{V}_\tau^* \begin{bmatrix} \boldsymbol{q}_{\ 1} \\ \boldsymbol{q}_{\ 2} \end{bmatrix}.$$
  end loop
end loop

$\boldsymbol{u}_1 = 0$
loop over all levels, coarser to finer, $= 1, 2, \ldots, L-1$,
  loop over all parent boxes on level ,
    Let $_1$ and $_2$ denote the children of .
    $$\begin{bmatrix} \boldsymbol{u}_{\ 1} \\ \boldsymbol{u}_{\ 2} \end{bmatrix} = \mathsf{U}_\tau \boldsymbol{u}_\tau + \begin{bmatrix} 0 & \mathsf{B}_{\ 1,\ 2} \\ \mathsf{B}_{\ 2,\ 1} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_{\ 1} \\ \boldsymbol{q}_{\ 2} \end{bmatrix}.$$
  end loop
end loop

loop over all leaf boxes ,
  $\boldsymbol{u}(I_\tau) = \mathsf{U}_\tau \boldsymbol{u}_\tau + \mathsf{D}_\tau \boldsymbol{q}(I_\tau)$.
end loop

5 Inversion of hierarchically block separable matrices

In this section, we describe an algorithm for inverting an HBS matrix A. The algorithm is exact (in the absence of round-o errors) and has asymptotic complexity $O(Nk^2)$. It is summarized as Algorithm 3, and as the description

ALGORITHM 3 (inversion of an HBS matrix)

loop over all levels, finer to coarser, $= L, L-1, \ldots, 1$
  loop over all boxes on level ,
    if is a leaf node
      $\widetilde{\mathsf{D}}_\tau = \mathsf{D}_\tau$
    else
      Let $_1$ and $_2$ denote the children of .
      $\widetilde{\mathsf{D}}_\tau = \begin{bmatrix} \mathsf{D}_{\ 1} & \mathsf{B}_{\ 1,\ 2} \\ \mathsf{B}_{\ 2,\ 1} & \mathsf{D}_{\ 2} \end{bmatrix}$
    end if
    $\mathsf{D}_\tau = \left(\mathsf{V}_\tau^* \widetilde{\mathsf{D}}_\tau^{-1} \mathsf{U}_\tau\right)^{-1}$.
    $\mathsf{E}_\tau = \widetilde{\mathsf{D}}_\tau^{-1} \mathsf{U}_\tau \mathsf{D}_\tau$.
    $\mathsf{F}_\tau^* = \mathsf{D}_\tau \mathsf{V}_\tau^* \widetilde{\mathsf{D}}_\tau^{-1}$.
    $\mathsf{G}_\tau = \mathsf{D}_\tau - \widetilde{\mathsf{D}}_\tau^{-1} \mathsf{U}_\tau \mathsf{D}_\tau \mathsf{V}_\tau^* \widetilde{\mathsf{D}}_\tau^{-1}$.
  end loop
end loop
$\mathsf{G}_1 = \begin{bmatrix} \mathsf{D}_2 & \mathsf{B}_{2,3} \\ \mathsf{B}_{3,2} & \mathsf{D}_3 \end{bmatrix}^{-1}$.

indicates, it is very simple to implement. The output of Algorithm 3 is a set of factors in a data-sparse representation of A^{-1} which can be applied to a given vector via Algorithm 4. Technically, the scheme consists of recursive application of Lemma 3.1, and its derivation is given in the proof of the following theorem.

Theorem 5.1 *Let A be an invertible $N \times N$ HBS matrix with block-rank k. Suppose that at the nest level of the tree structure, there are 2^L leaves that each holds n points (so that $N = n2^L$), and that $n \quad 2k$. Then a data-sparse representation of A^{-1} that is exact up to rounding errors can be computed in $O(Nk^2)$ operations via a process given as Algorithm 3, provided that none of the matrices that need to be inverted is singular. The computed inverse can be applied to a vector in $O(Nk)$ operations via Algorithm 4.*

Proof We can according to equation (4.15) express an HBS matrix as

$$\mathsf{A} = \underline{\mathsf{U}}^{(L)} \widetilde{\mathsf{A}}^{(L)} (\underline{\mathsf{V}}^{(L)})^* + \underline{\mathsf{D}}^{(L)}.$$

Lemma 3.1 immediately applies, and we find that

$$\mathsf{A}^{-1} = \underline{\mathsf{E}}^{(L)} (\widetilde{\mathsf{A}}^{(L)} + \underline{\mathsf{D}}^{(L)})^{-1} (\underline{\mathsf{F}}^{(L)})^* + \underline{\mathsf{G}}^{(L)}, \tag{5.1}$$

where $\underline{\mathsf{E}}^{(L)}$, $\underline{\mathsf{F}}^{(L)}$, $\underline{\mathsf{D}}^{(L)}$, and $\underline{\mathsf{G}}^{(L)}$ are defined via (3.7)–(3.10), respectively.

To move to the next coarser level, we set $= L-1$ in formula (4.16) whence

$$\widetilde{\mathsf{A}}^{(L)} + \underline{\mathsf{D}}^{(L)} = \underline{\mathsf{U}}^{(L-1)} \widetilde{\mathsf{A}}^{(L-1)} (\underline{\mathsf{V}}^{(L-1)})^* + \underline{\mathsf{B}}^{(L-1)} + \underline{\mathsf{D}}^{(L)}. \tag{5.2}$$

ALGORITHM 4 (application of inverse)

Given a vector $\boldsymbol{u}$, *compute* $\boldsymbol{q} = \mathsf{A}^{-1}\boldsymbol{u}$ *using the compressed representation of* A^{-1} *resulting from Algorithm* 3.

loop over all leaf boxes ,
  $\boldsymbol{u}_\tau = \mathsf{F}_\tau^* \boldsymbol{u}(I_\tau)$.
end loop

loop over all levels, finer to coarser, $= L, L-1, \ldots, 1$,
  loop over all parent boxes on level ,
    Let $_1$ and $_2$ denote the children of .
    $\boldsymbol{u}_\tau = \mathsf{F}_\tau^* \begin{bmatrix} \boldsymbol{u}_{\ 1} \\ \boldsymbol{u}_{\ 2} \end{bmatrix}$.
  end loop
end loop

$$\begin{bmatrix} \boldsymbol{q}_2 \\ \boldsymbol{q}_3 \end{bmatrix} = \mathsf{G}_1 \begin{bmatrix} \boldsymbol{u}_2 \\ \boldsymbol{u}_3 \end{bmatrix}.$$

loop over all levels, coarser to finer, $= 1, 2, \ldots, L-1$,
  loop over all parent boxes on level ,
    Let $_1$ and $_2$ denote the children of .
    $\begin{bmatrix} \boldsymbol{q}_{\ 1} \\ \boldsymbol{q}_{\ 2} \end{bmatrix} = \mathsf{E}_\tau \boldsymbol{u}_\tau + \mathsf{G}_\tau \begin{bmatrix} \boldsymbol{u}_{\ 1} \\ \boldsymbol{u}_{\ 2} \end{bmatrix}$.
  end loop
end loop

loop over all leaf boxes ,
  $\boldsymbol{q}(I_\tau) = \mathsf{E}_\tau \boldsymbol{q}_\tau + \mathsf{G}_\tau \boldsymbol{u}(I_\tau)$.
end loop

We define

$$\widetilde{\underline{\mathsf{D}}}^{(L-1)} = \underline{\mathsf{B}}^{(L-1)} + \underline{\mathsf{D}}^{(L)} = \left[\begin{array}{cc|cc|cc|c} \mathsf{D}_{\tau_1} & \mathsf{B}_{\tau_1\tau_2} & 0 & 0 & 0 & 0 & \cdots \\ \mathsf{B}_{\tau_1\tau_2} & \mathsf{D}_{\tau_2} & 0 & 0 & 0 & 0 & \cdots \\ \hline 0 & 0 & \mathsf{D}_{\tau_3} & \mathsf{B}_{\tau_3\tau_4} & 0 & 0 & \cdots \\ 0 & 0 & \mathsf{B}_{\tau_3\tau_4} & \mathsf{D}_{\tau_4} & 0 & 0 & \cdots \\ \hline 0 & 0 & 0 & 0 & \mathsf{D}_{\tau_5} & \mathsf{B}_{\tau_5\tau_6} & \cdots \\ 0 & 0 & 0 & 0 & \mathsf{B}_{\tau_5\tau_6} & \mathsf{D}_{\tau_6} & \cdots \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \end{array}\right],$$

where $\{ \ _1, \ _2, \ldots, \ _{2^L}\}$ is a list of the boxes on level L. Equation (5.2) then takes the form

$$\widetilde{\mathsf{A}}^{(L)} + \underline{\mathsf{D}}^{(L)} = \underline{\mathsf{U}}^{(L-1)} \widetilde{\mathsf{A}}^{(L-1)} (\underline{\mathsf{V}}^{(L-1)})^* + \widetilde{\underline{\mathsf{D}}}^{(L-1)}. \tag{5.3}$$

(We note that in (5.3), the terms on the left-hand side are block matrices with $2^L \times 2^L$ blocks, each of size $k \times k$, whereas the terms on the right-hand side are block matrices with $2^{L-1} \times 2^{L-1}$ blocks, each of size $2k \times 2k$.) Now, Lemma 3.1 applies to (5.3) and we find that

$$(\widetilde{\mathsf{A}}^{(L)} + \underline{\mathsf{D}}^{(L)})^{-1} = \underline{\mathsf{E}}^{(L-1)}(\widetilde{\mathsf{A}}^{(L-1)} + \underline{\mathsf{D}}^{(L-1)})^{-1}(\underline{\mathsf{F}}^{(L-1)})^* + \underline{\mathsf{G}}^{(L-1)},$$

where $\underline{\mathsf{E}}^{(L-1)}$, $\underline{\mathsf{F}}^{(L-1)}$, $\underline{\mathsf{D}}^{(L-1)}$, and $\underline{\mathsf{G}}^{(L-1)}$ are defined via (3.7)–(3.10), respectively.

The process by which we went from step L to step $L-1$ is then repeated to move up to coarser and coarser levels. With each step, the size of the matrix to be inverted is cut in half. Once we get to the top level, we are left with the task of inverting the matrix

$$\widetilde{\mathsf{A}}^{(1)} + \underline{\mathsf{D}}^{(1)} = \begin{bmatrix} \mathsf{D}_2 & \mathsf{B}_{2,3} \\ \mathsf{B}_{3,2} & \mathsf{D}_3 \end{bmatrix}. \tag{5.4}$$

The matrix in (5.4) is of size $2k \times 2k$, and we use brute force to evaluate

$$\mathsf{G}^{(0)} = \mathsf{G}_1 = \begin{bmatrix} \mathsf{D}_2 & \mathsf{B}_{2,3} \\ \mathsf{B}_{3,2} & \mathsf{D}_3 \end{bmatrix}^{-1}.$$

To calculate the cost of the inversion scheme described, we note that in order to compute the matrices $\underline{\mathsf{E}}^{(\ )}$, $\underline{\mathsf{F}}^{(\ )}$, $\underline{\mathsf{G}}^{(\ )}$, and $\underline{\mathsf{D}}^{(\ )}$ on level , we need to perform dense matrix operations on 2 blocks, each of size at most $2k \times 2k$. Since the leaf boxes each hold at most $2k$ points, so that $2^{L+1}k \quad N$, the total cost is

$$\text{COST} \quad \sum_{=1}^{L} 2 \ 8k^3 \quad 2^{L+4}k^3 \quad Nk^2.$$

This completes the proof. □

Remark 5.1 Algorithm 3 produces a representation of A^{-1} that is not exactly in HBS form since the matrices G_τ do not have zero diagonal blocks like the matrices B_τ, cf. (4.10). However, a simple technique given as Algorithm 5 converts the factorization provided by Algorithm 3 into a standard HBS factorization. If a factorization in which the expansion matrices are all orthonormal is sought, then further post-processing via Algorithm 1 will do the job.

Remark 5.2 Algorithm 3 provides four formulas for the matrices $\{\mathsf{E}_\tau, \mathsf{F}_\tau, \mathsf{G}_\tau, \mathsf{D}_\tau\}_\tau$. The task of actually computing the matrices can be accelerated in two ways: (1) Several of the matrix-matrix products in the formulas are recurring, and the computation should be organized so that each matrix-matrix product is evaluated only once. (2) When interpolatory decompositions are used, multiplications involving the matrices U_τ and V_τ can be accelerated by exploiting that they each contain a $k \times k$ identity matrix.

ALGORITHM 5 (reformatting inverse)

Postprocessing the terms computed Algorithm 3 *to obtain an inverse in the standard HBS format.*

loop over all levels, coarser to finer, $= 0, 1, 2, \ldots, L-1$,
  loop over all parent boxes on level ,
    Let $_1$ and $_2$ denote the children of .
    Define the matrices $\mathsf{H}_{1,1}, \mathsf{B}_{1,2}, \mathsf{B}_{2,1}, \mathsf{H}_{2,2}$ so that
    $\mathsf{G}_\tau = \begin{bmatrix} \mathsf{H}_{1,1} & \mathsf{B}_{1,2} \\ \mathsf{B}_{2,1} & \mathsf{H}_{2,2} \end{bmatrix}$.
    $\mathsf{G}_1 \quad \mathsf{G}_1 + \mathsf{E}_1 \mathsf{H}_{1,1} \mathsf{F}^*_1$.
    $\mathsf{G}_2 \quad \mathsf{G}_2 + \mathsf{E}_2 \mathsf{H}_{2,2} \mathsf{F}^*_2$.
  end loop
end loop

loop over all leaf boxes ,
  $\mathsf{D}_\tau = \mathsf{G}_\tau$.
end loop

Remark 5.3 The assumption in Theorem 5.1 that none of the matrices to be inverted is singular is undesirable. When A is spd, this assumption can be done away with (cf. Corollary 3.2), and it can further be proved that the inversion process is numerically stable. When A is non-symmetric, the intermediate matrices often become ill-conditioned. We have *empirically* observed that if we enforce that $\mathsf{U}_\tau = \mathsf{V}_\tau$ for every node (the procedure for doing this for non-symmetric matrices is described in Remark 6.1), then the method is remarkably stable, but we do not have any supporting theory.

Remark 5.4 The assumption in Theorem 5.1 that the block-rank k remains constant across all levels is slightly unnatural. In applications to integral equations on 1D domain, one often finds that the rank of interaction depends logarithmically on the number of points in a block. It is shown in [21] that inclusion of such logarithmic growth of the interaction ranks does not change the $O(N)$ total complexity.

6 Computing HBS representation of a boundary integral operator

Section 4 describes a particular way of representing a class of "compressible" matrices in a hierarchical structure of block-diagonal matrices. For any matrix whose HBS rank is k, these factors can via straight-forward means be computed in $O(N^2 k)$ operations. In this section, we describe an $O(Nk^2)$ technique for computing an HBS representation of the matrix resulting upon Nyström discretization of a BIE.

6.1 A basic compression scheme for leaf nodes

In this section, we describe how to construct for every leaf node , interpolatory matrices U_τ and V_τ of rank k, and index vectors $\widetilde{I}_\tau^{(\mathrm{row})}$ and $\widetilde{I}_\tau^{(\mathrm{col})}$ such that, cf. (4.6),

$$\mathsf{A}(I_\tau, I_{\tau^*}) = \mathsf{U}_\tau \mathsf{A}(\widetilde{I}_\tau^{(\mathrm{row})}, \widetilde{I}_{\tau^*}^{(\mathrm{col})}) \mathsf{V}_{\tau^*}^*, \qquad = \quad \overset{\times}{.} \tag{6.1}$$

The first step in the process is to construct for every leaf a row of blocks R_τ and a column of blocks C_τ via

$$\mathsf{R}_\tau = \mathsf{A}(I_\tau, L_\tau), \quad \mathsf{C}_\tau = \mathsf{A}(L_\tau, I_\tau),$$

where L_τ is the complement of the index vector I_τ within the full index set

$$L_\tau = \{1, 2, 3, \ldots, N\} \backslash I_\tau.$$

The condition (4.6) implies that R_τ and C_τ each has rank at most k. We can therefore construct interpolatory decompositions

$$\mathsf{R}_\tau = \mathsf{U}_\tau \mathsf{R}(J_\tau^{(\mathrm{row})}, :), \tag{6.2}$$

$$\mathsf{C}_\tau = \mathsf{C}(: , J_\tau^{(\mathrm{col})}) \mathsf{V}_\tau^*. \tag{6.3}$$

Now, (6.1) holds if we set

$$\widetilde{I}_\tau^{(\mathrm{row})} = I_\tau(J_\tau^{(\mathrm{row})}), \quad \widetilde{I}_\tau^{(\mathrm{col})} = I_\tau(J_\tau^{(\mathrm{col})}).$$

Remark 6.1 It is often useful to use the same basis matrices to span the ranges of both R_τ and C_τ^*. In particular, this can make a substantial di erence in the stability of the inversion scheme described in Section 4. To accomplish this, form the matrix $\mathsf{X}_\tau = \begin{bmatrix} \mathsf{R}_\tau \mid \mathsf{C}_\tau^* \end{bmatrix}$ and then compute an interpolatory factorization

$$\mathsf{X}_\tau = \mathsf{U}_\tau \mathsf{X}_\tau(J_\tau, :). \tag{6.4}$$

Then, clearly,

$$\mathsf{R}_\tau = \mathsf{U}_\tau \mathsf{R}(J_\tau, :), \quad \mathsf{C}_\tau = \mathsf{C}(: , J_\tau) \mathsf{U}_\tau^*.$$

Enforcing symmetry can slightly increase the HSS-ranks, but typically in a very modest way.

Remark 6.2 In applications, the matrices R_τ and C_τ are typically only *approximately* of rank k and the factorizations (6.2) and (6.3) are then required to hold only to within some preset tolerance ε. Techniques for computing rank-revealing partial factorizations of this type are described in detail in [9,15].

6.2 An accelerated compression scheme for leaf nodes

We will in this section demonstrate how to rapidly construct matrices U_τ, V_τ and index vectors $J_\tau^{(\mathrm{row})}$, $J_\tau^{(\mathrm{col})}$ such that (6.2) and (6.3) hold, respectively. We focus on the construction of U_τ and $J_\tau^{(\mathrm{row})}$, since the construction of V_τ and

$J_\tau^{(\mathrm{col})}$ is analogous. While U_τ and $J_\tau^{(\mathrm{row})}$ can in principle be constructed by directly computing an interpolatory decomposition of R_τ, this is in practice prohibitively expensive since R_τ is very large. The way to get around this is to exploit analytic properties of the kernel to construct a much smaller matrix $\mathsf{R}_\tau^{(\mathrm{small})}$ whose columns span the column space of R_τ. Then we can cheaply form U_τ and $J_\tau^{(\mathrm{row})}$ by factoring this smaller matrix. The process typically overestimates the rank slightly (since the columns of $\mathsf{R}_\tau^{(\mathrm{small})}$ will span a slightly larger space than the columns of R_τ), but this is more than compensated by the dramatic acceleration of the compression step.

To formalize matters, our goal is to construct a small matrix $\mathsf{R}_\tau^{(\mathrm{small})}$ such that

$$\mathrm{Ran}(\mathsf{R}_\tau) \quad \mathrm{Ran}\big(\mathsf{R}_\tau^{(\mathrm{small})}\big). \tag{6.5}$$

Then all we would need to do to compress is to construct the interpolatory decomposition

$$\mathsf{R}_\tau^{(\mathrm{small})} = \mathsf{U}_\tau \mathsf{R}_\tau^{(\mathrm{small})}(J_\tau^{(\mathrm{row})}, :), \tag{6.6}$$

since (6.5) and (6.6) together imply (6.2).

When constructing the matrix $\mathsf{R}_\tau^{(\mathrm{small})}$, we distinguish between near field interactions and far field interactions. The near field interactions cannot readily be compressed, but this is not a problem since they contribute a very small part of R_τ. To define what we mean by "near" and "far", we need to introduce some notation. Let Γ_τ denote the segment of Γ associated with the node , see Fig. 3. We enclose Γ_τ in a circle and then let $\Gamma_\tau^{(\mathrm{proxy})}$ denote a circle with the same center but with a 50% larger radius. We now define $\Gamma_\tau^{(\mathrm{far})}$ as the part of Γ outside of $\Gamma_\tau^{(\mathrm{proxy})}$ and define $\Gamma_\tau^{(\mathrm{near})}$ as the part of Γ inside $\Gamma_\tau^{(\mathrm{proxy})}$ but disjoint from Γ_τ. In other words,

$$\Gamma = \Gamma_\tau \quad \Gamma_\tau^{(\mathrm{near})} \quad \Gamma_\tau^{(\mathrm{far})}$$

forms a disjoint partitioning of Γ. We define $L_\tau^{(\mathrm{near})}$ and $L_\tau^{(\mathrm{far})}$ so that

$$\{1, 2, 3, \ldots, N\} = I_\tau \quad L_\tau^{(\mathrm{near})} \quad L_\tau^{(\mathrm{far})}$$

forms an analogous disjoint partitioning of the index vector $\{1, 2, \ldots, N\}$. We now find that

$$\mathsf{R}_\tau = [\mathsf{A}(I_\tau, L_\tau^{(\mathrm{near})}) \mid \mathsf{A}(I_\tau, L_\tau^{(\mathrm{far})})] \ , \tag{6.7}$$

where is a permutation matrix. We will construct a matrix $\mathsf{R}_\tau^{(\mathrm{proxy})}$ such that

$$\mathrm{Ran}(\mathsf{A}(I_\tau, L_\tau^{(\mathrm{far})})) \quad \mathrm{Ran}(\mathsf{R}_\tau^{(\mathrm{proxy})}), \tag{6.8}$$

and then we set

$$\mathsf{R}_\tau^{(\mathrm{small})} = [\mathsf{A}(I_\tau, L_\tau^{(\mathrm{near})}) \mid \mathsf{R}_\tau^{(\mathrm{proxy})}]. \tag{6.9}$$

That (6.5) holds is now a consequence of (6.7)–(6.9).

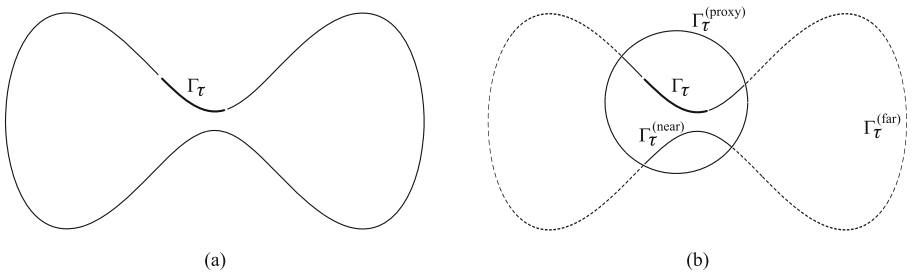


Fig. 3 A contour Γ. (a) Γ_τ is drawn with a bold line. (b) The contour $\Gamma_\tau^{(\text{near})}$ is drawn with a thin solid line and $\Gamma_\tau^{(\text{far})}$ with a dashed line.

All that remains is to construct a small matrix $\mathsf{R}_\tau^{(\text{proxy})}$ such that (6.8) holds. We describe the process for the single layer potential associated with Laplace's equation (for generalizations, see Remarks 6.4–6.7). Since we use Nyström discretization, the matrix $\mathsf{A}(I_\tau, L_\tau^{(\text{far})})$ in this case represents evaluation on Γ_τ of the harmonic potential generated by a set of point charges on $\Gamma_\tau^{(\text{far})}$. We know from potential theory that any harmonic field generated by charges *outside* $\Gamma_\tau^{(\text{proxy})}$ can be generated by placing an equivalent charge distribution *on* $\Gamma_\tau^{(\text{proxy})}$. Since we only need to capture the field to within some preset precision ε, it is sufficient to place charges on a finite collection $\{\boldsymbol{z}_j\}_{j=1}^J$ of points on $\Gamma_\tau^{(\text{proxy})}$. In other words, we set

$$\mathsf{R}_\tau^{(\text{proxy})}(i,j) = \log|\boldsymbol{x}_i - \boldsymbol{z}_j|, \quad i \in I_\tau,\ j \in \{1,2,3,\ldots,J\}.$$

The number of charges J that are needed on the external circle depends on the precision ε required. In fact $J = O(\log(1/\varepsilon))$ as $\varepsilon \to 0$. We have found that using $J = 50$ points is sufficient to attain ten digits of accuracy or better.

The construction of a small matrix $\mathsf{C}_\tau^{(\text{small})}$ that can be used to construct V_τ and $J_\tau^{(\text{col})}$ such that (6.3) holds is entirely analogous since the matrix C_τ^* is also a map of a charge distribution on $\Gamma_\tau^{(\text{far})}$ to a potential field on Γ_τ. The only caveat is that the rows of C_τ^* must be scaled by the quadrature weights used in the Nyström method.

Remark 6.3 As an alternative to placing charges on the exterior circle, one could represent the harmonic field generated on Γ_τ by an expansion in the cylindrical harmonic functions $\{r^j\cos(j\theta), r^j\sin(j\theta)\}_{j=0}^J$, where (r,θ) are the polar coordinates of a point in the circle enclosing Γ_τ. The number of functions needed are about the same, but we found it easier to correctly weigh the two terms $\mathsf{A}(I_\tau, L_\tau^{(\text{far})})$ and $\mathsf{R}_\tau^{(\text{proxy})}$ when using proxy charges on the outer circle.

Remark 6.4 (Extension to the double layer kernel) The procedure described directly generalizes to the double layer kernel associated with Laplace's equation. The compression of R_τ is exactly the same. The compression of C_τ^* is very similar, but now the target field is the normal derivative of the set of harmonic potentials that can be generated by sources outside $\Gamma_\tau^{(\text{proxy})}$.

Remark 6.5 (Extension to Laplace's equation in $\mathbb{R}^3$) The scheme described generalizes immediately to BIEs defined on surfaces in $\mathbb{R}^3$ if we replace the circles must by spheres. In this environment, the asymptotic complexity is typically $O(N^{1.5})$ for the compression and inversion steps, while application of either the original operator or the computed inverse remains $O(N)$ [10]. The scaling constants in the asymptotic estimates appear to be very small at low and intermediate accuracies [10,13].

Remark 6.6 (Extension to Helmholtz and other equations) The scheme described has been generalized to the single and double layer potentials associated with Helmholtz equation, see [21]. The only complication happens when the frequency is close to a resonant frequency of the proxy circle. This potential problem is avoided by placing both monopoles and dipoles on the proxy circle.

Remark 6.7 (Extension to integral equations on the line) The acceleration gets particularly simple for integral equations on a line with smooth non-oscillatory kernels. In this case, the range of $\mathsf{A}(I_\tau, L_\tau^{(\text{far})})$ can typically be represented by a short expansion in a generic set of basis functions such as, e.g., Legendre polynomials.

6.3 Compression of higher levels

The method described in Section 6.2 rapidly constructs the matrices U_τ, V_τ and the index vector $J_\tau^{(\text{row})}$, $J_\tau^{(\text{col})}$ for any leaf node . Using the notation introduced in Section 4.4, it computes the matrices $\underline{\mathsf{U}}^{(L)}$, $\underline{\mathsf{V}}^{(L)}$, and $\widetilde{\mathsf{A}}^{(L)}$. It is important to note that when the interpolatory decomposition is used, $\widetilde{\mathsf{A}}^{(L)}$ is in fact a submatrix of A, and is represented *implicitly* by specifying the relevant index vectors. To be precise, if and $^{\times}$are two nodes on level $L-1$, with children $_1$, $_2$ and $^{\times}_1$, $^{\times}_2$, respectively, then

$$\mathsf{A}_{\tau,\tau^*} = \begin{bmatrix} \mathsf{A}(\widetilde{I}^{(\text{row})}_{\ 1}, \widetilde{I}^{(\text{col})}_{\ \ 1^*}) & \mathsf{A}(\widetilde{I}^{(\text{row})}_{\ 1}, \widetilde{I}^{(\text{col})}_{\ \ 2^*}) \\ \mathsf{A}(\widetilde{I}^{(\text{row})}_{\ 2}, \widetilde{I}^{(\text{col})}_{\ \ 1^*}) & \mathsf{A}(\widetilde{I}^{(\text{row})}_{\ 2}, \widetilde{I}^{(\text{col})}_{\ \ 2^*}) \end{bmatrix}.$$

The observation that $\widetilde{\mathsf{A}}^{(L)}$ is a submatrix of A is critical. It implies that the matrices $\underline{\mathsf{U}}^{(L-1)}$, $\underline{\mathsf{V}}^{(L-1)}$, and $\widetilde{\mathsf{A}}^{(L-1)}$ can be computed using the strategy of Section 6.2 *without any modi cations.*

6.4 Approximation errors

As mentioned in Remark 6.2, factorizations such as (6.2), (6.3), (6.4), and (6.6) are in practice only required to hold to within some preset tolerance. In a single-level scheme, it is straightforward to choose a local tolerance in such a way that $\|\mathsf{A} - \mathsf{A}_{\text{approx}}\|$ ε holds to within some given global tolerance ε. In a multi-level scheme, it is rather difficult to predict how errors aggregate across levels, in particular when the basis matrices U_τ and V_τ are not orthonormal. As an empirical observation, we have found that such error propagation is typically

mild and that the error $\|\mathsf{A} - \mathsf{A}_{\text{approx}}\|$ is well predicted by the local tolerance in the interpolatory decomposition.

While it can be hard to *predict* the norm of the approximation error $\mathsf{E} = \mathsf{A} - \mathsf{A}_{\text{approx}}$ (where $\mathsf{A}_{\text{approx}}$ is the computed HBS approximation to A), it is often a simple matter to accurately estimate the error *à posteriori* via a power iteration on $\mathsf{E}\mathsf{E}^*$. In executing the power iteration, $\mathsf{A}_{\text{approx}}$ and $\mathsf{A}^*_{\text{approx}}$ can be applied very rapidly since the HBS representations of these matrices are available, and A and A^* can be applied rapidly via, e.g., the fast multipole method [14] (executed at a tolerance finer than that sought).

Once $\|\mathsf{A} - \mathsf{A}_{\text{approx}}\|$ has been estimated, the question of controlling the error in the full direct solver presents itself. This error is caused in part by the fact that $\mathsf{A}^{-1}_{\text{approx}}$ need not necessarily approximate A^{-1} well even when $\mathsf{A} - \mathsf{A}_{\text{approx}}$ is small (recall, e.g., that $\mathsf{A}^{-1} - \mathsf{A}^{-1}_{\text{approx}} = \mathsf{A}^{-1}(\mathsf{A}_{\text{approx}} - \mathsf{A})\mathsf{A}^{-1}_{\text{approx}}$), and in part by the fact that even though Algorithm 3 is in principle exact, rounding errors may be in" ated substantially whenever any of the intermediate matrices is ill-conditioned. Since estimating the combined e ects of these errors is likely to result in overly pessimistic bounds, we again recommend the use of à posteriori error estimates. To illustrate how this can be done, we let G denote the output of applying Algorithm 3 to the data-sparse approximation $\mathsf{A}_{\text{approx}}$. Then

$$\frac{\|\mathsf{A}^{-1} - \mathsf{G}\|}{\|\mathsf{A}^{-1}\|} = \frac{\|(\mathsf{I} - \mathsf{G}\mathsf{A})\mathsf{A}^{-1}\|}{\|\mathsf{A}^{-1}\|} \quad \|\mathsf{I} - \mathsf{G}\mathsf{A}\|,$$

and the quantity $\|\mathsf{I} - \mathsf{G}\mathsf{A}\|$ can again be estimated via a power iteration on $(\mathsf{I} - \mathsf{G}\mathsf{A})(\mathsf{I} - \mathsf{A}^*\mathsf{G}^*)$. The quantity $\|\mathsf{I} - \mathsf{G}\mathsf{A}\|$ also provides a direct bound on the error in any computed solution. To demonstrate this, let $\boldsymbol{q}$ denote the exact solution of (1.3), and let $\boldsymbol{q}_{\text{approx}} = \mathsf{G}\boldsymbol{f}$ denote the computed approximate solution. Then

$$\|\boldsymbol{q} - \boldsymbol{q}_{\text{approx}}\| = \|\boldsymbol{q} - \mathsf{G}\boldsymbol{f}\| = \|\boldsymbol{q} - \mathsf{G}\mathsf{A}\boldsymbol{q}\| \quad \|\mathsf{I} - \mathsf{G}\mathsf{A}\|\,\|\boldsymbol{q}\|.$$

7 Numerical examples

In this section, we illustrate the performance of the direct solver when applied to several di erent boundary integral equations (BIEs). For each example, a compressed representation of the coefficient matrix was computed via Matlab implementations of the compression scheme described in Section 6. Then Fortran 77 implementations of Algorithms 3 (inversion of an HBS matrix), 5 (conversion to standard HBS format), and 2 (matrix-vector multiply) were used to directly solve the respective linear systems. All codes were executed on a desktop computer with 2.6 GHz Intel i5 processor and 8 GB of RAM. The speed of the compression step can be improved significantly by moving to a Fortran implementation, but since this step is somewhat idiosyncratic to each specific problem, we believe that it is representative to report the times of an unoptimized Matlab code.

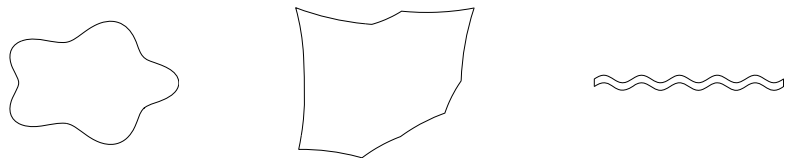
The linear systems considered were obtained by Nyström discretization of the BIEs

$$\frac{1}{2}\,q(\boldsymbol{x}) + \int \frac{\boldsymbol{n}(\boldsymbol{x}') \cdot (\boldsymbol{x} - \boldsymbol{x}')}{2\ \ |\boldsymbol{x} - \boldsymbol{x}'|^2}\, q(\boldsymbol{x}')\mathrm{d}l(\boldsymbol{x}') = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Gamma, \tag{7.1}$$

and

$$-2\mathrm{i}q(\boldsymbol{x}) + \int \Big(\frac{}{\boldsymbol{n}(\boldsymbol{x}')}\, H_0(k|\boldsymbol{x} - \boldsymbol{x}'|) + \mathrm{i}kH_0(k|\boldsymbol{x} - \boldsymbol{x}'|) \Big) q(\boldsymbol{x}')\mathrm{d}l(\boldsymbol{x}') = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Gamma, \tag{7.2}$$

where $\boldsymbol{n}(\boldsymbol{x}')$ is a unit normal to the contour Γ. In (7.2), H_0 is the Hankel function of the first kind of order zero, k is the wave number, and $\mathrm{i} = \overline{-1}$. Equation (7.1) is the standard BIE formulation of a Laplace boundary value problem on the domain external to Γ with Dirichlet boundary data. Equation (7.2) is a variation of the "combined field equation" for an analogous Helmholtz boundary value problem. The geometry Γ was one of the three geometries shown in Fig. 4. Specifically, four di erent environments were considered.



(a) (b) (c)

Fig. 4 Contours Γ used in numerical experiments for BIE (7.1).
(a) Smooth star; (b) star with corners; (c) snake.

Smooth Laplace problem. The BIE (7.1) was considered for the contour Γ shown in Fig. 4 (a). The BIE is discretized via Gaussian quadrature as described in Section 2.3. In the experiment, we fix the size of the computational domain and increase the number of discretization points. This problem is artificial in the sense that the largest experiments use *far* more quadrature points than what is required for any reasonable level of accuracy. It was included simply to demonstrate the asymptotic scaling of the method.

Smooth Helmholtz problem. The BIE (7.2) was considered for the contour Γ shown in Fig. 4 (a). The BIE is discretized with the 10th order accurate end-point corrected trapezoidal rule [19]. The wave number k is chosen such that the length of the domain increases from 11 wavelengths to 1490 wavelengths long. Then the number of discretization points N is increased to maintain 50 points per wavelength.

Star with corners. The BIE (7.1) was considered for the contour Γ shown in Fig. 4 (b) which consists of ten segments of circles with di erent radii. We

started with a discretization with 6 panels per segment and 17 Gaussian quadrature nodes per panel. Then grid refinement as described in [5,18] (with a so-called "simply graded mesh") was used to increase the number of discretization points, cf. Remark 2.2.

Snake. The BIE (7.1) was considered for the contour Γ shown in Fig. 4 (c) which consists of two sine waves with amplitude 1 that are vertically separated by a distance of 0.2. At the end points, two vertical straight lines connect the waves. Composite Gaussian quadrature was used with 25 nodes per panel, four panels on each vertical straight line (refined at the corners to achieve 10 digits of accuracy), and then we used 10 panels per wavelength. The width of the contour was increased from 2 full periods of the wave to 200, and the number of discretization points N was increased accordingly.

In the experiments, the local tolerance in the compression step was set to $\varepsilon = 10^{-10}$. (In other words, the interpolatory factorization in (6.6) was required to produce a residual in the Frobenius norm no larger than ε.)

The times corresponding to each step in the direct solver for the BIEs (7.1) and (7.2) are reported in Fig. 5. Notice that for the Helmholtz problem, there is only slight deterioration in the performance as the wave number moves into the high frequency regime.

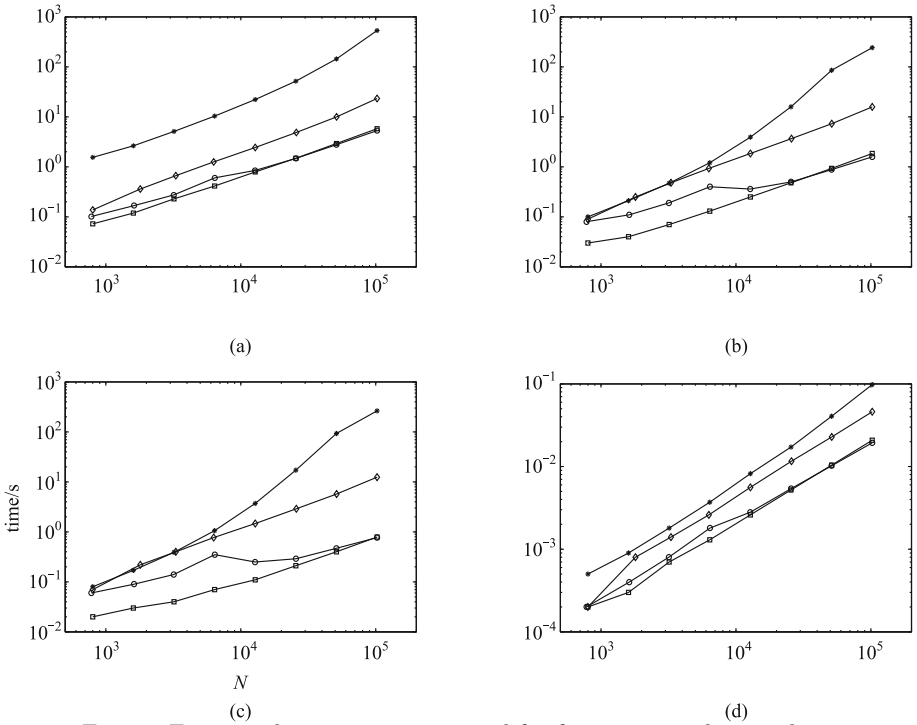


Fig. 5 Four graphs give times required for four steps in direct solver. (a) Compression; (b) inversion; (c) transform inverse; (d) matrix vector multiply. Within each graph, different lines correspond to four environments described in Section 7 as follows: □—smooth Laplace problem, ∗—smooth Helmholtz problem, ○—star with corners, ◇—snake.

The output of the direct solver is a compressed representation of a matrix G that approximates A^{-1}. To assess the accuracy of the approximation, the errors $\|\mathsf{A} - \mathsf{A}_{\text{approx}}\|$ and $\|\mathsf{I} - \mathsf{GA}\|$ (see Section 6.4) were estimated via a power iteration after each compression (the matrix A and its transpose were applied via the classical FMM [14] run at very high accuracy). The quantity $\|\mathsf{G}\|$ was also estimated. The results are reported in Fig. 6. We note that all errors increase with problem size. This error drift can be combatted by enforcing a smaller local tolerance at the finer levels.

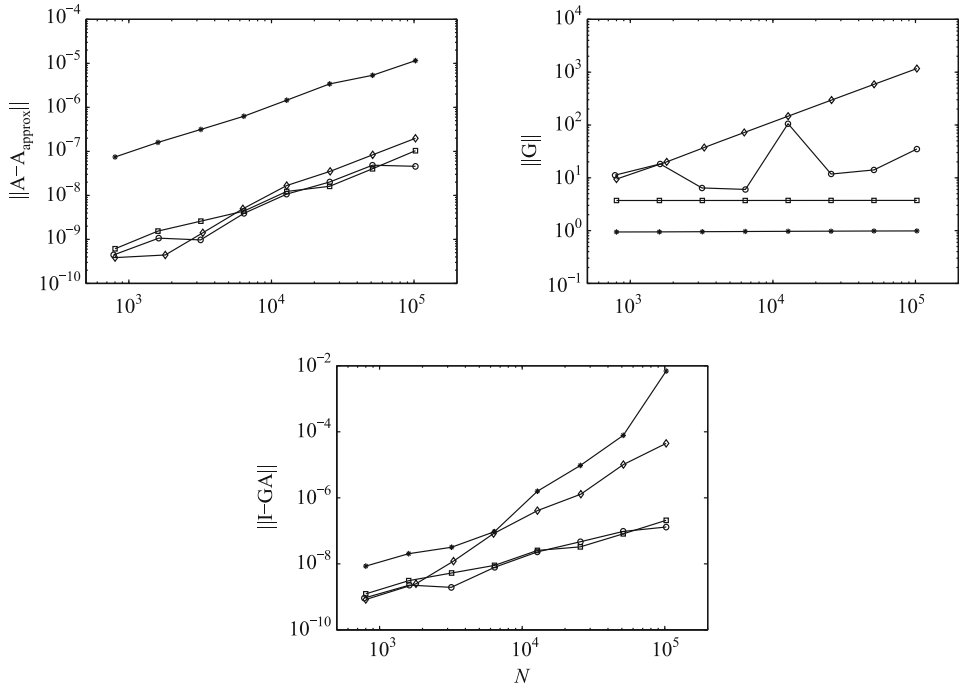


Fig. 6 Error information for each of domains: □—smooth Laplace problem, ∗—smooth Helmholtz problem, ○—star with corners, ◇—snake. Matrix G is computed approximation to A^{-1}.

8 Extensions and future work

The direct solver described can be applied to BIEs defined on surfaces in $\mathbb{R}^3$ without almost any modifications. The complexity then grows from $O(N)$ to $O(N^{1.5})$ for the inversion step, while $O(N)$ complexity is retained for applying the computed inverse. A rudimentary implementation was reported in [13], and more recent experiments [10] demonstrate that the scaling constants suppressed by the "big-O" notation are very modest.

While the simple scheme reported in this paper has complexity $O(N^{1.5})$ when applied to 3D problems, we believe that $O(N)$ complexity can be achieved by executing a "recursion on dimension" in a manner similar to the recently

described $O(N)$ nested dissection schemes reported in [7,12,24]. Work along these lines is currently in progress.

The error analysis in the present paper is very rudimentary. Numerical experiments indicate that the method is stable and accurate in many environments of interest, but this has not yet been demonstrated in any case other than that of symmetric positive definite matrices.

Acknowledgements This work was supported by the NSF grants DMS0748488 and DMS0941476.

References

1. Atkinson K E. The Numerical Solution of Integral Equations of the Second Kind. Cambridge: Cambridge University Press, 1997
2. Barnes J, Hut P. A hierarchical $O(n \log n)$ force-calculation algorithm. Nature, 1986, 324(4): 446 449
3. Beylkin G, Coifman R, Rokhlin V. Wavelets in numerical analysis. In: Wavelets and Their Applications. Boston: Jones and Bartlett, 1992, 181 210
4. Börm S. E cient Numerical Methods for Non-local Operators: $\mathscr{H}^2$-matrix Compression, Algorithms and Analysis. European Mathematics Society, 2010
5. Bremer J, Rokhlin V. E cient discretization of Laplace boundary integral equations on polygonal domains. J Comput Phys, 2010, 229: 2507 2525
6. Chandrasekaran S, Gu M. A divide-and-conquer algorithm for the eigendecomposition of symmetric block-diagonal plus semiseparable matrices. Numer Math, 2004, 96(4): 723 731
7. Chandrasekaran S, Gu M, Li X S, Xia J. Superfast multifrontal method for large structured linear systems of equations. SIAM J Matrix Anal Appl, 2009, 31: 1382 1411
8. Chandrasekaran S, Gu M, Li X S, Xia J. Fast algorithms for hierarchically semiseparable matrices. Numer Linear Algebra Appl, 2010, 17: 953 976
9. Cheng H, Gimbutas Z, Martinsson P G, Rokhlin V. On the compression of low rank matrices. SIAM J Sci Comput, 2005, 26(4): 1389 1404
10. Gillman A. Fast direct solvers for elliptic partial differential equations. Ph D Thesis. Boulder: University of Colorado at Boulder, 2011
11. Golub G H, Van Loan C F. Matrix Computations. 3rd ed. Johns Hopkins Studies in the Mathematical Sciences. Baltimore: Johns Hopkins University Press, 1996
12. Grasedyck L, Kriemann R, Le Borne S. Domain decomposition based H-LU preconditioning. Numer Math, 2009, 112: 565 600
13. Greengard L, Guey er D, Martinsson P G, Rokhlin V. Fast direct solvers for integral equations in complex three-dimensional domains. Acta Numer, 2009, 18: 243 275
14. Greengard L, Rokhlin V. A fast algorithm for particle simulations. J Comput Phys, 1987, 73(2): 325 348
15. Gu M, Eisenstat S C. E cient algorithms for computing a strong rank-revealing QR factorization. SIAM J Sci Comput, 1996, 17(4): 848 869
16. Hackbusch W. The panel clustering technique for the boundary element method (invited contribution). In: Boundary Elements IX, Vol 1 (Stuttgart, 1987), Comput Mech Southampton. 1987, 463 474
17. Hackbusch W. A sparse matrix arithmetic based on H-matrices; Part I: Introduction to H-matrices. Computing, 1999, 62: 89 108
18. Helsing J, Ojala R. Corner singularities for elliptic problems: Integral equations, graded meshes, quadrature, and compressed inverse preconditioning. J Comput Phys, 2008, 227: 8820 8840

19. Kapur S, Rokhlin V. High-order corrected trapezoidal quadrature rules for singular functions. SIAM J Numer Anal, 1997, 34: 1331 1356
20. Martinsson P G. A fast randomized algorithm for computing a hierarchically semi-separable representation of a matrix. SIAM J Matrix Anal Appl, 2011, 32(4): 1251 1274
21. Martinsson P G, Rokhlin V. A fast direct solver for boundary integral equations in two dimensions. J Comp Phys, 2005, 205(1): 1 23
22. Michielssen E, Boag A, Chew W C. Scattering from elongated objects: direct solution in $O(N \log^2 N)$ operations. IEE Proc Microw Antennas Propag, 1996, 143(4): 277 283
23. O Donnell S T, Rokhlin V. A fast algorithm for the numerical evaluation of conformal mappings. SIAM J Sci Stat Comput, 1989, 10: 475 487
24. Schmitz P, Ying L. A fast direct solver for elliptic problems on general meshes in 2d. 2010
25. Sheng Z, Dewilde P, Chandrasekaran S. Algorithms to solve hierarchically semi-separable systems. In: System Theory, the Schur Algorithm and Multidimensional Analysis. Oper Theory Adv Appl, Vol 176. Basel: Birkhäuser, 2007, 255 294
26. Starr P, Rokhlin V. On the numerical solution of two-point boundary value problems. II. Comm Pure Appl Math, 1994, 47(8): 1117 1159
27. Xiao H, Rokhlin V, Yarvin N. Prolate spheroidal wavefunctions, quadrature and interpolation. Inverse Problems, 2001, 17(4): 805 838
28. Young P, Hao S, Martinsson P G. A high-order Nyström discretization scheme for boundary integral equations de ned on rotationally symmetric surfaces. J Comput Phys (to appear)