A Complete $\mathcal{O}(N)$ Elliptic PDE Solver

Alexandru Fikl

December 6, 2017

1 Introduction

In this report we will describe a complete solver for elliptic partial differential equations with linear time complexity. This is achieved by taking advantage of the special structure of elliptic operators and a plethora of advanced algorithms and data structures designed for them. The typical problem we will be looking at is the 2D Laplace equation with Dirichlet boundary conditions:

$$\begin{cases} \nabla^2 u = 0, & \mathbf{x} \in \Omega, \\ u = g, & \mathbf{x} \in \Gamma = \partial\Omega. \end{cases} \tag{1}$$

Historically, this problem has been approached from two different points of view: discretizing the differential equation or discretizing an equivalent integral equation. Discretizing the differential equation itself on the volume Ω can be performed with classic finite element methods and it leads to a linear system that can be solved with iterative methods such as GMRES, Bi-CGSTAB, etc. However, the discrete derivative operators inherit the behavior of their continuous counterparts and the resulting systems are not particularly well-conditioned so, in many cases, this results in $\mathcal{O}(N^2)$ algorithms. Alternatively, we can use an equivalent integral representation using fundamental solutions given by Green's functions. In our case, the 2D Laplace Green function is:

$$G(\mathbf{x}, \mathbf{y}) = \frac{1}{2\pi} \log \|\mathbf{x} - \mathbf{y}\|_2. \tag{2}$$

The solution can then be represented in many different ways. For example, using a *double-layer potential* [3, Chapter 6], we have:

$$u(\mathbf{x}) = \int_\Gamma \mathbf{n}_y \cdot \nabla_{\mathbf{y}} G(\mathbf{x}, \mathbf{y}) \sigma(\mathbf{y}) \, \mathrm{d}y, \tag{3}$$

where $\sigma \in \mathcal{C}(\Gamma)$ is called a *density*. To find the density, we can take the limit $\mathbf{x} \to \Gamma$ to the boundary where the value of u is known and obtain:

$$\frac{1}{2}\sigma(\mathbf{x}) \pm \int_\Gamma \mathbf{n}_y \cdot \nabla_{\mathbf{y}} G(\mathbf{x}, \mathbf{y}) \sigma(\mathbf{y}) \, \mathrm{d}y = g(\mathbf{x}), \tag{4}$$

for all $\mathbf{x} \in \Gamma$. In contrast to the system obtained by directly discretizing the original PDE (1), this formulation has several obvious benefits:

- The dimension was reduced from a problem solved on the volume Ω to one on the surface Γ. This is not possible for all elliptic equations on all domains, but it still gives a very significant improvement when applicable.
- The resulting system is well-defined and well-conditioned for our problem and for a very wide variety of other integral operators [3, Chapter 3]. This implies that even an iterative solver will perform very well for the resulting linear system.
- The formulation allows us to solve an "interior" problem on Ω, but also an "exterior" problem on Ω^c, which is not possible when discretizing the PDE directly.

However, there are still some issues that require careful consideration when attempting a linear time solver:

- The existence and uniqueness of (4). From integral operator theory, we know that even if the Green's function (2) is singular, the resulting integral equation is well-defined and has a unique solution. We will take this as given and note that further information on this subject can be sought in classic monographs such as [3].
- As we have mentioned, the fundamental solution (2) is singular at $\mathbf{x} = \mathbf{y}$. Bypassing theoretical considerations, this introduces many issues when numerically discretizing (4) with classic quadrature rules. In particular, even adaptive high-order Gauss quadrature methods eventually fail to capture the singularity correctly. For this reason, we will investigate a different type of quadrature called *quadrature by expansion* [11] that can give consistent high-order results.
- A major roadblock in achieving linear time consists in actually evaluating the integrals in (3) and (4). If evaluated directly, this would lead to an $\mathcal{O}(N^2)$ algorithm, irrespective of the quadrature algorithm used. The most well-known methods for accelerating this type of particle-particle interactions are the Barnes-Hut method [1] and the Fast Multipole Method [2]. We are only interested in the Fast Multipole Method because it provides $\mathcal{O}(N)$ construction and evaluations of layer potentials like (3). Our use of quadrature by expansion for computing the various integrals also implies some subtle changes in the FMM method, which are detailed in [12].
- As mentioned, the resulting linear system from (4) is well-conditioned and most iterative solvers would achieve $\mathcal{O}(N)$ complexity (assuming the FMM is used to compute the matrix-vector product). However, they will fail for other types of representations, e.g. using a single-layer potential, do not handle multiple right-hand sides efficiently, do not take advantage of the special structure of the system, etc. For this reason, we will attempt to use the direct solver described in [5] for this exact type of problem. The presented solver has been very successful with many extensions [8, 9].

In the next sections we will attempt to further detail each of these algorithms and provide some numerical evidence that they do indeed give the correct result and the expected complexity. However, before moving further, it is important to clarify some of the notation and nomenclature we will make heavy use of:

- Layer potentials are usually written in the form:

$$\mathcal{A}\sigma(\mathbf{x}) = \int_\Omega K(\mathbf{x}, \mathbf{y})\sigma(\mathbf{y})\,\mathrm{d}\mathbf{y},$$

  where K is usually referred to as a *kernel*. For a single layer potential the kernel is simply $G(\mathbf{x}, \mathbf{y})$, for the double layer potential the kernel is given by $\mathbf{n} \cdot \nabla_y G(\mathbf{x}, \mathbf{y})$, etc.

- We refer to $\mathbf{y}$ as the *source* points and $\mathbf{x}$ as the target points. The naming is clear when looking at (3): all the source points $\mathbf{y}$ are part of the "sum" used to compute the solution at the target point $\mathbf{x}$. The two are not necessarily disjoint sets.
- The integral equation (4) is called a *second-order integral equation*. This type of integral equation is well-conditioned even if the double layer potential is a compact operator. A *first-order integral equation* can be obtained by using a single-layer potential, but this is avoided, if possible, because the resulting operator is compact and leads to slow convergence. Further details can be found in [3].

2 Quadrature by Expansion

In this section we will focus exclusively on accurately computing different types of weakly singular kernels required for discretizing (4). We are mainly interested the single-layer and double layer potentials:

$$\begin{aligned}\mathcal{S}\sigma(\mathbf{x}) &= \int_\Gamma G(\mathbf{x},\mathbf{y})\sigma(\mathbf{y})\,\mathrm{d}\mathbf{y},\\ \mathcal{D}\sigma(\mathbf{x}) &= \int_\Gamma \mathbf{n}_y\cdot\nabla_{\mathbf{y}}G(\mathbf{x},\mathbf{y})\sigma(\mathbf{y})\,\mathrm{d}\mathbf{y}.\end{aligned}$$

At this level, the main difference between them is the strength of the singularity, so we will mostly be treating them in an equivalent manner. There are some subtleties when $\mathbf{x}\in\Gamma$, e.g. the value required of the double-layer potential is generally the *principal value* of the integral. The quadrature by expansion method relies on the fact that the kernel $G(\mathbf{x},\mathbf{y})$ is a smooth function when the distance between the two points is sufficiently large. Therefore, at some distance any high-order quadrature rule will give the expected results.

In what follows, we will assume that the boundary Γ is $\mathcal{C}^2$ continuous to ensure compactness of some of the operators and sidestep any problems regarding sharp corners or discontinuous densities. The boundary Γ is assumed to be parametrized by its arclength s and the points on the boundary are given by a mapping $\mathbf{x}(s)=(x_1(s),x_2(s))$. The following values and quantities will be heavily used in what follows:

- M, the number of *panels* in which Γ has been discretized. A panel is a subinterval of Γ, which we will denote by Γ_m. The arclength of a panel is given by:

$$h_m=\int_{\Gamma_m}\mathrm{d}s.$$

- q, the number of Gauss-Legendre quadrature nodes and weights used in each panel. We will denote the kth quadrature node in panel m by $\mathbf{y}_{m,k}$ and the weights by $\omega_{m,k}$.
- $\sigma_{m,k}=\sigma(\mathbf{y}_{m,k})$, the density at the kth quadrature node in panel m. We will often use the alternative one-dimensional indexing $\sigma_j\equiv\sigma_{m,k}$, for $j=m\times q+k$.
- $\mathbf{x}_i$ and $\mathbf{y}_j\equiv\mathbf{y}_{m,k}$, the coordinates of each target and source point, respectively. When solving (4), the two will coincide, but when evaluating (3) the targets can be anywhere in $\mathbb{R}^2$.
- $\mathbf{n}_x$ and $\mathbf{n}_y$, the exterior target and source normal vectors, respectively. Note that target normal vectors only make sense when $\mathbf{x}\in\Gamma$.
- r_j, the QBX expansion radius. It is recommended in [11] to take $r_j\equiv r_{m,k}=h_m/2$, for all $k\in[\![0,q-1]\!]$, i.e. take the same expansion radius for all the points inside a given panel.

- $\mathbf{c}_j = \mathbf{x}_j \pm r_j \mathbf{n}_{x_j}$, the center of the expansion.

2.1 Continuous Theory

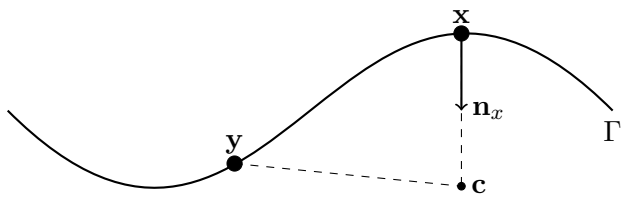


Figure 1: QBX center of expansion $\mathbf{c}$ from a target point $\mathbf{x}$.

To take advantage of the smoothness of G, we will expand it using a local Taylor expansion around an off-surface center $\mathbf{c}$ (see Figure 1). To formalize this expansion, we introduce the function:

$$f(\tau) = \phi(\mathbf{x} \pm r(1-\tau)\mathbf{n}_x),$$

where $\phi = \mathcal{S}\sigma$ or $\phi = \mathcal{D}\sigma$ and the $\pm$ reflects an exterior "+" or interior "−" approximation. Effectively, this new function varies between the expansion center $f(0) = \phi(\mathbf{x} \pm r\mathbf{n}_x) = \phi(\mathbf{c})$ and the on-surface target point $f(1) = \phi(\mathbf{x})$. The one-dimensional Taylor expansion can then be written as:

$$\phi(\mathbf{x}) = f(1) \approx \sum_{k=0}^{p} \frac{1^k}{k!} f^{(k)}(0),$$

where, e.g. for the double-layer potential:

$$f^{(k)}(0) = \frac{\partial^k}{\partial \tau^k} \int_\Gamma \mathbf{n}_y \cdot \nabla_{\mathbf{y}} G(\mathbf{x} \pm r(1-\tau)\mathbf{n}_x, \mathbf{y}) \sigma(\mathbf{y}) \, \mathrm{d}\mathbf{y} \bigg|_{\tau=0}.$$

Therefore, by exchanging the order of summation and integration, we can write:

$$\mathcal{D}^{\pm}\sigma(\mathbf{x}) \approx \int_\Gamma G_p^{\pm}(\mathbf{x}, \mathbf{y}) \sigma(\mathbf{y}) \, \mathrm{d}\mathbf{y}, \tag{5}$$

where:

$$G_p^{\pm}(\mathbf{x}, \mathbf{y}) = \sum_{k=0}^{p} \frac{1}{k!} \frac{\partial^k}{\partial \tau^k} \Big\{ \mathbf{n}_y \cdot \nabla_{\mathbf{y}} G(\mathbf{x} \pm r(1-\tau)\mathbf{n}_x, \mathbf{y}) \Big\} \bigg|_{\tau=0}.$$

By definition, the new kernel $G_p^{\pm}$ is smooth enough that it can be accurately integrated, so we can look at the quadrature by expansion method as a kernel regularization method. This point of view gives a subtle catch when using quadrature by expansion (see [11]): since the scheme is basically a regularization, the spatial discretization h and the order of the Taylor expansion p are linked together. The intuitive way to think about it is that when we increase p, the original singular kernel is better captured, leading to less regularization, so a decrease in h is required to maintain accuracy.

We also know from [3, Chapter 6] that the jump relations for our layer potentials are given by:

$$\begin{aligned} [\![\mathcal{S}\sigma]\!] &= 0, \\ [\![\mathcal{D}\sigma]\!] &= \sigma, \end{aligned}$$

so we expect that an interior / exterior expansion will make no difference for the single-layer potential. However, for the double layer potential we will obtain the respective one-sided limits:

$$\lim_{r\to 0^+} \mathcal{D}\sigma(\mathbf{x} \pm r\mathbf{n}_x) = \mathcal{D}\sigma(\mathbf{x}) \pm \frac{1}{2}\sigma(\mathbf{x}). \tag{6}$$

Therefore, if the on-surface value is required, one can compute the average of the two one-sided limits. This is also recommended when attempting to numerically solve the linear system resulting from a discretization using QBX, because one-sided limits can lead to ill-conditioned systems for certain choices of (h, p).

2.2 Discrete Theory

We have seen how to construct the expansion, so we can now proceed to completely discretize the resulting integral (5). To this end, we divide our boundary Γ into M panels of arclength h_m. On each panel, we define a Gauss-Legendre quadrature with q nodes $\mathbf{y}_{m,k}$ and weights $\omega_{m,k}$. Using the previously defined notation, we can write the discrete layer potential as:

$$\mathcal{D}^{\pm}_{p,q}\sigma(\mathbf{x}) = \sum_{j=0}^{N} \omega_j G^{\pm}_p(\mathbf{x}, \mathbf{y}_j)\sigma_j, \tag{7}$$

where $N = Mq$. Using this discrete scheme, we have the following error bound from [11, Theorem 1]:

Theorem 2.1. *Suppose that Γ is a smooth, bounded curve embedded on $\mathbb{R}^2$ that $\mathcal{B}_r(\mathbf{c})$ is the ball of radius r centered at $\mathbf{c}$, and that $\overline{\mathcal{B}_r(\mathbf{c})} \cap \Gamma = \{\mathbf{x}\}$. Let Γ be divided into M panels, each of length h and let q be a non-negative integer that defines the number of nodes of a Gaussian quadrature. Then:*

$$|\mathcal{S}\sigma(x) - \mathcal{S}_{p,q}\sigma(\mathbf{x})| \leq C_1(p,\Gamma) r^{p+1} \log r^{-1} \|\sigma\|_{\mathcal{C}^p(\Gamma)} + C_2(q,p,\Gamma) \left(\frac{h}{4r}\right)^{2q} \|\sigma\|_{\mathcal{C}^{2q}(\Gamma)},$$

where the first term corresponds to the truncation error from the Taylor expansion and the second term corresponds to the quadrature error.

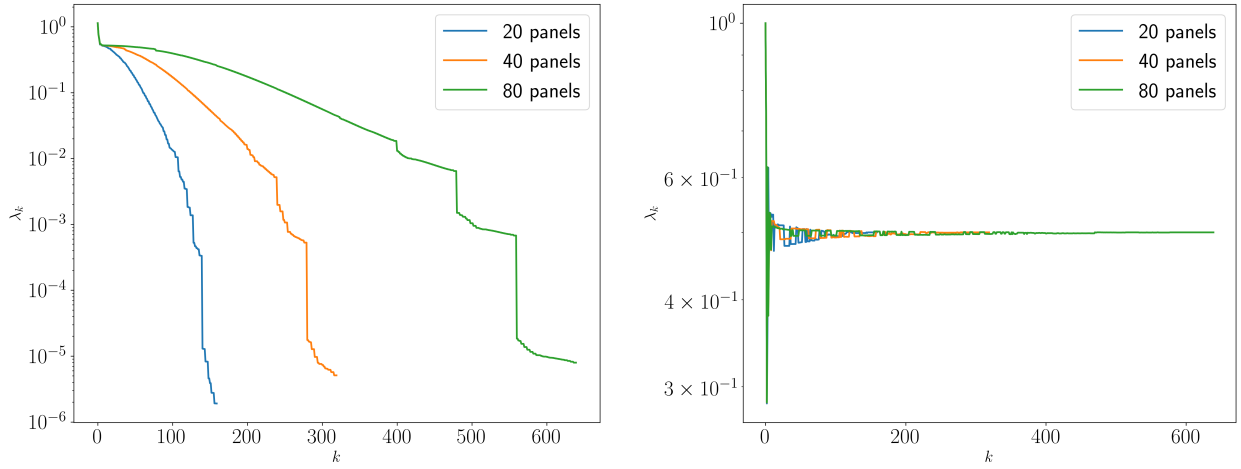
There are several important points in this theorem that need to be taken into account when using quadrature by expansion:

- The radius of expansion must be $r \geq h/4$ if we expect the term corresponding to the quadrature error to decrease. At the same time, if $r = \mathcal{O}(h)$ we get a truncation error of h^{p+1} from the Taylor expansion. Therefore, we require a large value of r to decrease the quadrature error and a small value to decrease the truncation error. As a compromise, [11] recommends a value of $r \approx h/2$.
- The above result is valid for the single-layer potential. For the double-layer potential, we will loose an order in the truncation error.
- The error estimate depends not only on the distance to the target point $\mathbf{x}$ at the current expansion center, but also on the nearby values. Specifically, if the center $\mathbf{c}_m$ is at the desired distance from the panel Γ_m, but $\mathcal{B}_r(\mathbf{c}_m)$ intersects another panel $\Gamma_{m'}$, the approximation will not be accurate.

Having discretized the layer potential, it remains to discretized the full integral equation (4). In our case, we have two choices. First, we can use just the interior expansion to define the full approximation of (4), since we know from the limit (6) that this gives the desired result. However, it has been shown in [11] and can be seen in Figure 2a that this operator is ill-conditioned. Second, we can average the two limits to obtain:

$$\frac{1}{2}\sigma_i + \frac{1}{2}(\mathcal{D}^+_{p,q}\sigma_i + \mathcal{D}^-_{p,q}\sigma_i) = g_i, \tag{8}$$

for $i \in [\![0, N-1]\!]$. We can see in Figure 2b that this is a well-conditioned operator. It is unfortunate that we require to compute both limits to obtain a well-conditioned operator, but the gains are generally more important, especially when using an iterative solver.



(a) (b)

Figure 2: QBX: Discrete operator spectrum for (a) the interior limit (6) and (b) the average (8).

2.3 Adaptivity

We have seen that both the distance of the expansion center from the surface and intersections with other parts of the geometry can cause issues when using quadrature by expansion. Fortunately, both of these issues can be solved by adaptively refining the panels of Γ. Furthermore, refinement is a local operation that requires looking only at the geometry in the neighborhood of a panel m. In [12], the following conditions are defined to ensure good accuracy:

Condition 1 This condition ensures that the ball around each expansion center does not contain any source points. In particular, we can write:

$$\mathrm{d}(\mathbf{c}_{m,k}, \Gamma_{m'}) \geq \frac{h_m}{2},$$

for all nodes k in the panel m and all panels $m' \neq m$.

Condition 2 This condition relates more to the quadrature error and makes sure that two adjacent panels do not interact with each other. We impose that, for each panel m and adjacent panels m':

$$\frac{h_{m'}}{h_m} \in [0.5, 2],$$

which leads to a 2:1 panel sizing in the adaptive mesh. This is a very common condition used in many adaptive mesh algorithms, see e.g. [7].

Condition 3 For each expansion center $\mathbf{c}_{m,k}$, we must ensure that:

$$\mathrm{d}(\mathbf{c}_{m,k}, \Gamma_{m'}) \geq \frac{h_{m'}}{4},$$

for all panels $m' \neq m$. This condition give sufficient resolution at an expansion center from all panels.

From what we can see, Condition 2 can be checked locally in constant time. However, the other two conditions are not local in nature and would require a general "all centers to all panels" computation, which could result in a $\mathcal{O}(N^2)$ algorithm and negate the work from the other fast methods. A fast algorithm for checking and imposing the conditions above is presented in [12, Section 5].

The main remaining issue is related to the complexity of computing the sums in (8). A brute-force algorithm will only achieve $\mathcal{O}(N^2)$ complexity, but the fast multipole method from the next section will reduce this cost to the desired complexity. Alternatively, we will see that a direct solver can achieve the same gains by using a skeletonization method, which is very similar in spirit to the Fast Multipole method.

3 Fast Multipole Method

In this section we will focus on the Fast Multipole Method and the modifications necessary to make it compatible with quadrature by expansion. We will generally follow the outline from [12] and some classic FMM articles such as [2]. The fast multiple method will be used to accelerate the computation of integrals such as (5) and discretizations such as (8). In particular, we are interested in approximations of integrals of the form:

$$\phi(\mathbf{x}) = \int_\Gamma K(\mathbf{x}, \mathbf{y})\sigma(\mathbf{y})\,\mathrm{d}\mathbf{y} \tag{9}$$

and their discretization:

$$\phi(\mathbf{x}_i) = \sum_{j=1}^{N} a_j K(\mathbf{x}_i, \mathbf{y}_j), \tag{10}$$

where the coefficients are defined as $a_j \equiv \sigma(\mathbf{y}_j)\omega_j$, for some given quadrature weights ω_j and quadrature points $\mathbf{y}_i$. We can apply the fast multipole method to this type of problems if the kernel K has certain decay properties, which, in our case, are satisfied by both the single layer and the double layer potential.

The two main mathematical tools we will require are a local expansion and a multipole expansion of K from a set of sources to a set of well-separated targets. Since this requires expansions in multiple dimensions, we first define the notion of a multi-index and some basic operations on it. A

multi-index α is a tuple $(\alpha_1, \cdots, \alpha_n)$ on which we define the following operations relevant to our work:

- The magnitude of a multi-index is defined as:

$$|\alpha| \equiv \alpha_1 + \cdots + \alpha_n.$$

- The factorial of a multi-index is defined element-wise as:

$$\alpha! \equiv \alpha_1! \cdots \alpha_n!.$$

- Raising a vector $\mathbf{x}$ to a multi-index power α is defined as:

$$\mathbf{x}^\alpha \equiv x_1^{\alpha_1} \times \cdots \times x_n^{\alpha_n}.$$

- Differentiating using a multi-index is defined as:

$$\frac{\partial^\alpha f}{\partial \mathbf{x}^\alpha} \equiv \frac{\partial^{|\alpha|} f}{\partial x_1^{\alpha_1} \cdots \partial x_n^{\alpha_n}}.$$

We will define our local and multipole expansions in terms of polynomials, since this is the most common formulation. However, in more general terms any basis functions can be used, e.g. Bessel functions in [12].

Definition 3.1 (Local Expansion)**.** The local expansion of $K(\mathbf{x}, \mathbf{y})$ in $\mathbf{x}$ around a given center $\mathbf{c}$ is given by:

$$K(\mathbf{x}, \mathbf{y}) = \sum_{|\mathbf{p}|} \frac{(\mathbf{x} - \mathbf{c})^\mathbf{p}}{\mathbf{p}!} \frac{\partial^\mathbf{p}}{\partial \mathbf{x}^\mathbf{p}} K(\mathbf{x}, \mathbf{y}) \bigg|_{\mathbf{x}=\mathbf{c}},$$

where $\mathbf{p}$ is a multi-index. The local expansion of the potential $\phi(\mathbf{x})$ can then be written as:

$$\phi(\mathbf{x}) = \sum_{|\mathbf{p}|} L_\mathbf{p} (\mathbf{x} - \mathbf{c})^\mathbf{p},$$

where the coefficients $L_\mathbf{p}$ are given by:

$$L_\mathbf{p} = \int_\Omega \frac{\sigma(\mathbf{y})}{\mathbf{p}!} \frac{\partial^\mathbf{p}}{\partial \mathbf{x}^\mathbf{p}} K(\mathbf{x}, \mathbf{y}) \bigg|_{\mathbf{x}=\mathbf{c}} \mathrm{d}\mathbf{y}.$$

Definition 3.2 (Multipole Expansion)**.** The multipole expansion of $K(\mathbf{x}, \mathbf{y})$ in $\mathbf{y}$ around a given center $\mathbf{c}$ is given by:

$$K(\mathbf{x}, \mathbf{y}) = \sum_{|\mathbf{p}|} \frac{(\mathbf{c} - \mathbf{y})^\mathbf{p}}{\mathbf{p}!} \frac{\partial^\mathbf{p}}{\partial \mathbf{y}^\mathbf{p}} K(\mathbf{x}, \mathbf{y}) \bigg|_{\mathbf{y}=\mathbf{c}},$$

where $\mathbf{p}$ is a multi-index. The multipole expansion of the potential $\phi(\mathbf{x})$ can then be written as:

$$\phi(\mathbf{x}) = \sum_{|\mathbf{p}|} M_\mathbf{p} \frac{\partial^\mathbf{p}}{\partial \mathbf{y}^\mathbf{p}} K(\mathbf{x}, \mathbf{y}) \bigg|_{\mathbf{y}=\mathbf{c}},$$

where the coefficients $M_\mathbf{p}$ are given by:

$$M_\mathbf{p} = \int_\Omega \frac{(\mathbf{c} - \mathbf{y})^\mathbf{p}}{\mathbf{p}!} \sigma(\mathbf{y}) \, \mathrm{d}\mathbf{y}.$$

The main idea behind the fast multipole method is that we can use a multipole expansion (3.2) to *compress* the information from a cluster of sources $\mathbf{y}$ around a given center $\mathbf{c}$ that are well-separated from the desired target $\mathbf{x}$. Together with a set of translation operators and a hierarchically constructed mesh, this leads to a $\mathcal{O}(N)$ algorithm, where N is the number of desired target points.

Specifically, the target and source domain Ω is partitioned using a quadtree into boxes with an approximately equal number of particles. In this tree, we define the following entities:

- Parent, child, root and leaf nodes retain their standard definitions.
- A *box* b is a node in the tree, which represents a rectangular partition of the domain with m_b particles.
- Two boxes are said to be *k-near neighbors* if they are at the same level and their ℓ_∞ center distance is at most k box widths. In particular, 2-near neighbors are called *colleagues*. Note that a box is always a k-near neighbor of itself, for any k.
- Two boxes are said to be *k-well-separated* if they are at the same level of the tree, but are not k-near neighbors. Specifically, if we denote by $\mathbf{c}$ and $\mathbf{c}'$ the centers of two boxes b and b' on a level l, then they are well-separated if:

  $$|\mathbf{c} - \mathbf{c}'| \geq 4|\Omega|2^{-l},$$

  where $|\Omega|$ is the ℓ_∞ "radius" of the root domain. By this definition, we can see that two boxes can only be non-trivially well-separated if the level $l \geq 2$.

3.1 Translations

Theorem 3.1 (Multipole to Multipole Translation)**.** *Let $\phi(\mathbf{x})$ have a multipole expansion like in* (3.2) *and let $\mathbf{c}'$ be a new center of expansion. We can translate the original multipole expansion at $\mathbf{c}$ to a new multipole expansion at $\mathbf{c}'$, which reads:*

$$\phi(\mathbf{x}) = \sum_{|\mathbf{p}|} \mathcal{M}_{\mathbf{p}} \left. \frac{\partial^{\mathbf{p}}}{\partial \mathbf{y}^{\mathbf{p}}} K(\mathbf{x}, \mathbf{y}) \right|_{\mathbf{y}=\mathbf{c}'},$$

where the translated coefficients are given by:

$$\mathcal{M}_{\mathbf{p}} = \sum_{\mathbf{q} \leq \mathbf{p}} M_{\mathbf{q}} \frac{(\mathbf{c}' - \mathbf{c})^{\mathbf{p}-\mathbf{q}}}{(\mathbf{p} - \mathbf{q})!}.$$

Theorem 3.2 (Multipole to Local Translation)**.** *Let $\phi(\mathbf{x})$ have a multipole expansion like in* (3.2) *and let $\mathbf{c}'$ be a new center of expansion. We can translate the multiple expansion at $\mathbf{c}$ into a local expansion at $\mathbf{c}'$, which reads:*

$$\phi(\mathbf{x}) = \sum_{|\mathbf{p}|} \mathcal{L}_{\mathbf{p}} (\mathbf{x} - \mathbf{c}')^{\mathbf{p}},$$

where the translated coefficients are given by:

$$\mathcal{L}_{\mathbf{p}} = \sum_{\mathbf{q} \leq \mathbf{p}} \frac{M_{\mathbf{q}}}{(\mathbf{p} - \mathbf{q})!} \left. \frac{\partial^{\mathbf{q}} \partial^{\mathbf{p}-\mathbf{q}}}{\partial \mathbf{x}^{\mathbf{q}} \partial \mathbf{y}^{\mathbf{p}-\mathbf{q}}} K(\mathbf{x}, \mathbf{y}) \right|_{(\mathbf{x},\mathbf{y})=(\mathbf{c}',\mathbf{c})}.$$

Theorem 3.3 (Local to Local Translation)**.** *Let $\phi(\mathbf{x})$ have a local expansion like in* (3.1) *and let $\mathbf{c}'$ be a new center of expansion. We can translate the local expansion at $\mathbf{c}$ into a local expansion at $\mathbf{c}'$, which reads:*

$$\phi(\mathbf{x}) = \sum_{|\mathbf{p}|} \mathcal{L}_{\mathbf{p}} (\mathbf{x} - \mathbf{c}')^{\mathbf{p}},$$

where the translated coefficients are given by:

$$\mathcal{L}_{\mathbf{p}} = \sum_{\mathbf{q} \geq \mathbf{p}} \frac{\mathbf{q}!}{\mathbf{p}!(\mathbf{q}-\mathbf{p})!} L_{\mathbf{p}} (\mathbf{c}' - \mathbf{c})^{\mathbf{q}-\mathbf{p}}$$

The proofs for the various translation coefficients and error bounds can be found in many articles, e.g. [2], and the main ideas are:

- To find the multipole to multipole coefficients, perform an additional multipole expansion of the kernel at the new center $\mathbf{c}'$, replace into the original expansion and group by the derivative order.
- To find the multipole to local coefficients, perform an additional local expansion of the kernel around the new center $\mathbf{c}'$.
- Finally, to derive the local to local coefficients, make use of the multi-binomial theorem.

We can now use the translation operators to move information about an expansion between the levels of the the tree. For example, given the multipole expansions at the 4 children c_i of a box b:

$$\phi_{c_i}(\mathbf{x}) = \sum_{|\mathbf{p}|} M_{c_i,\mathbf{p}} \left. \frac{\partial^{\mathbf{p}}}{\partial \mathbf{y}^{\mathbf{p}}} K(\mathbf{x}, \mathbf{y}) \right|_{\mathbf{y}=\mathbf{c}_{c_i}}$$

we can use the translation (3.1) to shift the center of the expansions to the center $\mathbf{c}_b$ of the parent to obtain:

$$\phi_b(\mathbf{x}) = \sum_{|\mathbf{p}|} M_{b,\mathbf{p}} \left. \frac{\partial^{\mathbf{p}}}{\partial \mathbf{y}^{\mathbf{p}}} K(\mathbf{x}, \mathbf{y}) \right|_{\mathbf{y}=\mathbf{c}_b},$$

where the new coefficient is simply the sum of the translated coefficients:

$$M_{b,\mathbf{p}} = \sum_{i=1}^{4} \mathcal{M}_{c_i,\mathbf{p}}.$$

The same approach can be used to translate a local expansion from a parent to its children. We will make heavy use of these translations in the fast multipole algorithm. For a truncated expansion with p terms, the cost of translating expansions from children to parents of vice-versa is $4p^2$.

3.2 Compressing Far-Field Interactions

We will now briefly give a non-recursive version of the computation of the sum (10). This is meant to give an idea of how it is decomposed, how each term is transformed and how the sum is reduced to a fixed number of local interactions for a given target $\mathbf{x}$.

However, because of our use of the QBX method, there are two possible cases when dealing with a given target $\mathbf{x}$:

1. If the target $\mathbf{x}$ is far away from any source point $\mathbf{y}$, we can evaluate the potential (10) using usual methods. Quantifying what far means in this context relies on the expansions done as part of the QBX method. Namely, the target $\mathbf{x}$ must not be in the ball corresponding to any expansion center $\mathbf{c}_i$.
2. If the target $\mathbf{x}$ is close to the a boundary consisting of source points $\mathbf{y}$, we must use the QBX method to accurately compute it. The algorithm for marking a target as close to a boundary Γ is given in [12, Section 6].

Standard Interactions

For a conventional target, sufficiently far away from a boundary, the main steps of the algorithm rely on separating the far-field and near-field interactions. As a simple example, we will take a uniform grid and assume that we want to compute the potential at some target point $\mathbf{x}_i$ in a box b of the grid. The full formula would read:

$$\phi(\mathbf{x}_i) = \sum_{j=1}^{N} K(\mathbf{x}_i, \mathbf{y}_j)\omega_j,$$

but we can separate the interactions into a set of source points in the near-field $\mathbf{y}_j \in \mathcal{N}(b)$ and a set of points in the far-field $\mathbf{y}_j \in \mathcal{F}(b)$ of the box b:

$$\begin{aligned}\phi(\mathbf{x}_i) = &\sum_{j\in\mathcal{N}(b)} K(\mathbf{x}_i, \mathbf{y}_j)\omega_j + \sum_{j\in\mathcal{F}(b)} K(\mathbf{x}_i, \mathbf{y}_j)\omega_j \\ = &\sum_{j\in\mathcal{N}(b)} K(\mathbf{x}_i, \mathbf{y}_j)\omega_j + \sum_{b'\in\mathcal{F}(b)}\sum_{j\in b'} K(\mathbf{x}_i, \mathbf{y}_j)\omega_j.\end{aligned}$$

We can now form a pth order multipole expansion in each far-field box b' and replace all the interactions from the particles in that box with the expansion:

$$\phi(\mathbf{x}_i) = \sum_{j\in\mathcal{N}(b)} K(\mathbf{x}_i, \mathbf{y}_j)\omega_j + \sum_{b'\in\mathcal{F}(b)}\sum_{|\mathbf{p}|\le p} M_{\mathbf{p},b'} \left.\frac{\partial^{\mathbf{p}}}{\partial \mathbf{y}^{\mathbf{p}}} K(\mathbf{x}_i, \mathbf{y})\right|_{\mathbf{x}=\mathbf{c}_{b'}}.$$

We can now translate all the multipole expansions to local expansion around the center of the box b in which our target resides. This gives:

$$\begin{aligned}\phi(\mathbf{x}_i) = &\sum_{j\in\mathcal{N}(b)} K(\mathbf{x}_i, \mathbf{y}_j)\omega_j + \sum_{b'\in\mathcal{F}(b)}\sum_{|\mathbf{p}|\le p} M_{\mathbf{p},b'} \left.\frac{\partial^{\mathbf{p}}}{\partial \mathbf{y}^{\mathbf{p}}} K(\mathbf{x}_i, \mathbf{y})\right|_{\mathbf{x}=\mathbf{c}_{b'}} \\ = &\sum_{j\in\mathcal{N}(b)} K(\mathbf{x}_i, \mathbf{y}_j)\omega_j + \sum_{b'\in\mathcal{F}(b)}\sum_{|\mathbf{p}|\le p} \mathcal{L}_{\mathbf{p},b',b}(\mathbf{x}_i - \mathbf{c}_b).\end{aligned}$$

Finally, since $(\mathbf{x}_i - \mathbf{c}_b)$ does not depend on the source box b', we can sum up all the local expansion coefficients to obtain a simple formula:

$$\phi(\mathbf{x}_i) = \sum_{j\in\mathcal{N}(b)} K(\mathbf{x}_i, \mathbf{y}_j)\omega_j + \sum_{|\mathbf{p}|\le p} L_b(\mathbf{x}_i - \mathbf{c}_b),$$

which only depends on the near-field of b and can be evaluated "locally". In the following we will explain in more detail how we can form all the multipole expansions, translate them and sum up all the coefficients in $\mathcal{O}(N)$ on a more complex adaptive grid. The resulting algorithm is what is known as the *fast multipole method*.

QBX Interactions

If a target is on or near the boundary, we want to use accurate methods to compute the desired integral. For this reason, we have introduced the quadrature by expansion method in Section 2. QBX makes use of a local expansion, so we can write it in terms of (3.1):

$$\phi(\mathbf{x}) = \sum_{|\mathbf{p}|} Q_{\mathbf{p}} (\mathbf{x} - \mathbf{c})^{\mathbf{p}}.$$

For targets near the boundary we further distinguish the following two cases:

- The target $\mathbf{x}$ is also a source point and resides on the boundary itself. In this case, we know that the desired expansion center is given by:

$$\mathbf{c}_j = \mathbf{x}_j \pm r_j \mathbf{n}_j.$$

- The target $\mathbf{x}$ is simply near the boundary, close enough to fall into the expansion ball of one of the expansion centers. In this case, the desired expansion center is described by:

$$\mathbf{c} = \min_j \|\mathbf{x} - \mathbf{c}_j\|_\infty,$$

  i.e. the closest expansion center to the desired target.

Even if the expansion center is know, we still need to evaluate $Q_{\mathbf{p}}$, which is given by:

$$Q_{\mathbf{p}} = \frac{1}{\mathbf{p}!} \int_\Gamma \partial_{\mathbf{x}}^{\mathbf{p}} K(\mathbf{x}, \mathbf{y})|_{\mathbf{x}=\mathbf{c}} \, \sigma(\mathbf{y}) \, \mathrm{d}\mathbf{y}, \tag{11}$$

i.e. an integral over the whole domain equivalent to computing our original problem (9). However, the formulation is very similar, so the QBX local coefficients can be naturally computed as part of the fast multipole method. Specifically, after the application of the fast multipole method, we obtain a set of local coefficients L_b in each leaf box. We can use a local to local translation (3.3) to shift the coefficients from the box center $\mathbf{c}_b$ to any expansion center $\mathbf{c}_i$ in the box. It then remains to compute the direct interactions for each $\mathbf{p}$ and we will have all coefficients in (11).

3.3 Algorithm

For each leaf box b, we also define the following interaction lists (see Figure 3):

Near-field The set of all boxes in List 1 and List 3.

Far-field The set of all boxes not in the near-field.

List 1 (U) Contains the set of all colleagues of b. All boxes in this list require a direct evaluation, since they are too close to the target box b.

List 2 (V) Contains the children of the 2-near neighbors of the parent of b that are 2-well-separated from b. This list is required for all boxes b, not just leaf boxes.

List 3 (W) Contains children of colleagues of b that are not adjacent to b themselves. Therefore, the boxes b' in List 3 are always smaller than b and separated by a distance equal to $2|b'|$.

List 4 (X) Contains all leaf boxes b' such that b is in List 3 of b'. Therefore, the boxes in List 4 are always leaf boxes larger than b contained in the near-field of b's parent.

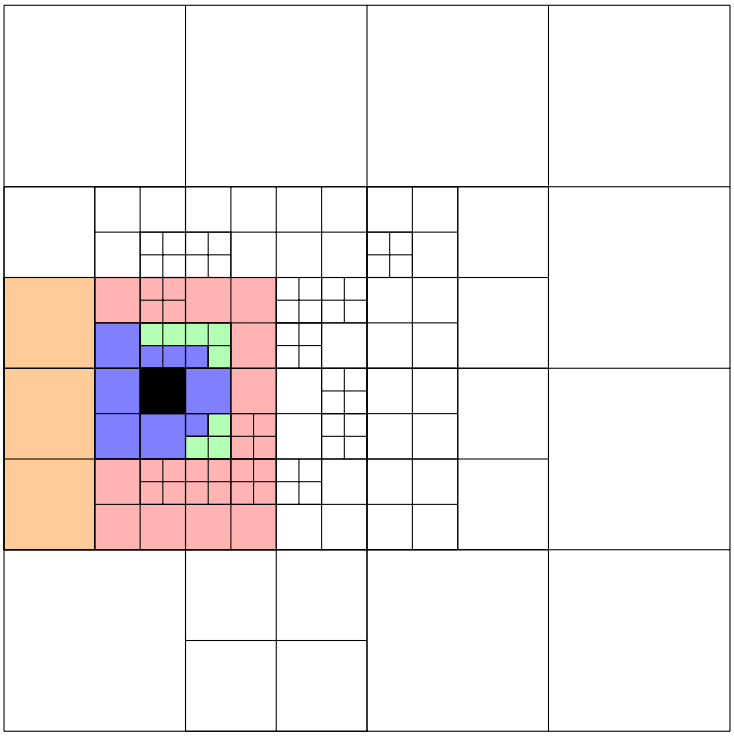


Figure 3: FMM: List 1 (blue), List 2 (red), List 3 (green), List 4 (orange), List 5 (white) and the target box (black).

List 5 Contains all the other boxes.

Furthermore, in each box we define the following quantities:

- $\mathbf{x}_i^{FMM}$ will denote targets far away from the boundary Γ and $\mathbf{x}_i^{QBX}$ will denote targets that require QBX evaluations and are near the boundary. When the superscript is not present, we refer to all targets.
- $\phi_i = \phi(\mathbf{x}_i)$ will be the potential at a target point $\mathbf{x}_i$.
- $\mathbf{M}_b = (M_{1,b}, \ldots, M_{|\mathbf{p}|,b})$ will denote the coefficients of the multipole expansion in the box b.
- $\mathbf{L}_b = (L_{1,b}, \ldots, L_{|\mathbf{p}|,b})$ will denote the coefficients of the local expansion in the box b.
- $\mathbf{Q}_{\mathbf{c}_i} = (Q_{1,\mathbf{c}_i}, \ldots Q_{|\mathbf{p}|,\mathbf{c}_i})$ will denote the local QBX coefficients at the expansion center $\mathbf{c}_i$.

We now have all the necessary mathematical tools and notation to describe the fast multipole algorithm and the required modifications for QBX quadrature. The following broad steps are necessary to evaluate (10):

Initialization Set problem data (see Algorithm 1) and construct a quadtree with all the required interaction lists (see Algorithm 2).

Upward pass Compute multipole expansions in all leaf boxes and recursively translate them to all parents (see Algorithm 3).

Downward pass Translate multipole expansions (3.2) from List 2 to each box b and then recursively translate the resulting local expansions (3.3) to each child of b (see Algorithm 5). Additionally, evaluate all multipole expansions from boxes in List 3 and form additional local expansions from all boxes in List 4 (see Algorithm 4).

QBX Coefficients The QBX coefficients are computed in the same way as the local expansions in each box: we first compute the multipole expansions from List 3 (see Algorithm 4), we then translate the local expansion in each box (see Algorithm 5) and finally compute the local interactions from List 1 (see Algorithm 6).

FMM Targets Compute direct interactions and evaluate local expansions (see Algorithm 7)

QBX Targets Evaluate the local expansion with the QBX coefficients obtained in the previous steps (see Algorithm 7).

Algorithm 1: Stage 0: Initialize parameters.

Data: Accuracy ϵ, QBX order p_{QBX}.
Data: Maximum number of particles per leaf node m.
Compute multipole order p_{FMM} from ϵ;
Create list of target points $\mathbf{x}_i$;
Create list of source points $\mathbf{y}_j$;
Create list of QBX expansion centers $\mathbf{c}_k$;
Mark all targets as $\mathbf{x}_i^{FMM}$ or $\mathbf{x}_i^{QBX}$;
Associate each $\mathbf{x}_i^{QBX}$ to its closest expansion center $\mathbf{c}_k$;

Algorithm 2: Stage 1: Form quadtree.

Data: Points $\mathbf{x}_i, \mathbf{y}_j$ and $\mathbf{c}_k$.
Result: Quadtree where each leaf box has no more than m points.
for $\mathbf{p} \in \mathbf{x}_i \cup \mathbf{y}_j \cup \mathbf{c}_k$ **do**
    Recursively insert $\mathbf{p}$ into tree at box b_l;
    if *box* b_l *has more than* m *points* **then**
        Subdivide b_l into 4 children and redistribute the points;
    end
end
for *each box* b **do**
    Form interactions lists: List 1, List 2, etc;
end

Algorithm 3: Stage 2: Form multipole expansions.

Data: Box b (starting from root box).
if b *is a leaf box* **then**
    Compute multipole coefficients $\mathbf{M}_b$ (3.2).
else
    Recurse into 4 children;
    for *Each child* b_i *of* b **do**
        Translate coefficients $\mathbf{M}_{b_i}$ using (3.1) and add to $\mathbf{M}_b$;
    end
end

Algorithm 4: Stage 3: Handle sources from interaction lists.

for *each box* b **do**
  for *Each box* b' *in List 2 of* b **do**
    Translate multipole $\mathbf{M}_{b'}$ to local using (3.2) and add to $\mathbf{L}_b$;
  end
  for *Each box* b' *in List 4 of* b **do**
    Form local expansion from b' and add to $\mathbf{L}_b$;
  end
  for *Each box* b' *in List 3 of* b **do**
    Directly evaluate multipole expansion at each target $\mathbf{x}_i \in b$ and add to ϕ_i;
    Directly evaluate multipole expansion at each expansion center $\mathbf{c}_i \in b$ and add to $\mathbf{Q}_{\mathbf{c}_i}$;
  end
end

Algorithm 5: Stage 4: Form local expansions from parent.

Data: Box b (start with root box).
for *each child* b' *of* b **do**
  Translate local $\mathbf{L}_b$ using (3.3) and add to $\mathbf{L}_{b'}$;
end
if b *is not a leaf* **then**
  Recurse into each child of b;
else
  for *each expansion center* $\mathbf{c}_i$ *in* b **do**
    Translate local $\mathbf{L}_b$ using (3.3) and add to $\mathbf{Q}_{\mathbf{c}_i}$;
  end
end

Algorithm 6: Stage 5: Compute direct interactions in List 1.

for *each leaf box* b **do**
  for *each source* $\mathbf{y}_j$ *in List 1 of* b **do**
    Compute direct interaction with all conventional targets $\mathbf{x}_i$ in b and add to ϕ_i;
    Compute direct interaction with all expansion centers $\mathbf{c}_i$ in b and add to $\mathbf{Q}_{\mathbf{c}_i}$;
  end
end

Algorithm 7: Stage 6: Finalize potential computation.

for *each leaf box* b **do**
  for *each* $\mathbf{x}_i^{FMM}$ *in* b **do**
    Evaluate local expansion with coefficients $\mathbf{L}_b$ and add to potential ϕ_i;
  end
  for *Each* $\mathbf{x}_i^{QBX}$ *in* b **do**
    Evaluate local expansion with coefficients $\mathbf{Q}_{\mathbf{c}_i}$ and add to potential ϕ_i;
  end
end

4 Direct Solver

In this section we will present an efficient way to solve the system of equations resulting from (8). The system matrix will be denoted by $\mathbf{A} \in \mathbb{R}^{n \times n}$ with the individual components given by:

$$a_{ij} = \frac{1}{2}\left[\delta_{ij} + (G_p^+(\mathbf{x}_i, \mathbf{y}_j) + G_p^-(\mathbf{x}_i, \mathbf{y}_j))\omega_j\right].$$

This is a dense system since the kernel $G(\mathbf{x}, \mathbf{y})$ does not have compact support. For the purposes of our algorithm, we will break the operator into blocks using standard notation:

$$\mathbf{A} = \begin{bmatrix} A_{11} & \cdots & A_{r1} \\ \vdots & \ddots & \vdots \\ A_{1q} & \cdots & A_{rq}, \end{bmatrix} \tag{12}$$

where each $A_{ij} \in \mathbb{R}^{r_i \times q_j}$, with $r_i = q_i$. In most of what follows, we will assume that $r_i = q_i = r$, i.e. all the blocks are square. We will also make use of the notation $A_{I,*}$ and $A_{*,J}$ to denote a set of rows and set of columns, respectively, corresponding to the index sets I and J. The two index sets are not necessarily contiguous.

As we have seen, our linear system is dense, but we can take advantage of the underlying problem properties to gain more insight into its exact structure. The most important point is that the off-diagonal elements correspond strictly to the discretization of the chosen layer potential. In our representation, all the layer potentials are compact operators, which in finite dimensions become low-rank matrices. Therefore, looking only at the off-diagonal blocks in (12), we expect they are prime candidates to make use of classic algorithms for low-rank matrix decompositions [4, 6]. Matrices of this type are called *block-separable* [9]:

Definition 4.1 (Block Separability)**.** A matrix $\mathbf{A}$ is called block separable if each off-diagonal block A_{ij} can be decomposed as the product of three low-rank matrices:

$$A_{ij} = L_i S_{ij} R_j,$$

where $L_i \in \mathbb{R}^{r_i \times k_i^r}, S_{ij} \in \mathbb{R}^{k_i^r \times k_j^c}$ and $R \in \mathbb{R}^{k_j^c \times q_j}$. The row rank of A_{ij} is given by $k_i^r \ll r_i$ and the column rank is given by $k_j^c \ll q_j$.

Furthermore, when $\mathbf{A} \in \mathbb{R}^{n \times n}$ is block separable, it can be written as:

$$\mathbf{A} = \mathbf{D} + \mathbf{LSR},$$

where $\mathbf{D} \in \mathbb{R}^{n \times n}, \mathbf{L} \in \mathbb{R}^{n \times k^r}$ and $\mathbf{R} \in \mathbb{R}^{k^c \times n}$ are a block diagonal matrices:

$$\mathbf{D} = \begin{bmatrix} A_{11} & & \\ & \ddots & \\ & & A_{pp} \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} L_1 & & \\ & \ddots & \\ & & L_q \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} R_1 & & \\ & \ddots & \\ & & R_q \end{bmatrix}$$

and $\mathbf{S} \in \mathbb{R}^{k^r \times k^c}$ is dense matrix with a zero diagonal:

$$\mathbf{S} = \begin{bmatrix} 0 & S_{12} & \cdots & S_{1p} \\ S_{21} & 0 & \cdots & S_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ S_{p1} & S_{p2} & \cdots 0 & \end{bmatrix},$$

with $k^r = \sum k_i^r$ and $k^c = \sum k_j^c$. We will see that this decomposition will lead to an efficient algorithm for computing the inverse, as detailed in some form in [5, 9, 8] and others.

4.1 Interpolative Decomposition

(a) (b)

Figure 4: ID: Block decomposition for each (a) row and (b) column where the white diagonal blocks are full rank and the block off-diagonal blocks are low rank.

We have seen that, at least intuitively, we expect the matrix representing a discrete integral equation to be block separable. In this section we will present an efficient way of computing such a low-rank representation using the so-called *interpolative decomposition* [4]. We will first look at a generic decomposition problem and apply the ID to each block row and block column (see Figure 4) and then present a more efficient method that relies on specific interaction defined by our Laplace kernel $G(\mathbf{x}, \mathbf{y})$.

We refer to [9, Definition 2.3] for a generic definition of the interpolative decomposition. For our purposes, we only need to know that the ID gives us the following decomposition for any low-rank matrix $\mathbf{A}$:

$$\mathbf{A} = \mathbf{BP},$$

where $\mathbf{B} \in \mathbb{R}^{n \times k}$ is known as the *skeleton* matrix and $\mathbf{P} \in \mathbb{R}^{k \times n}$ is known as the *projection* matrix. Both of the resulting matrices have a specific structure that gives rise to a very intuitive understanding of the decomposition itself. Namely:

- The matrix $\mathbf{B}$ simply contains a subset of the columns of $\mathbf{A}$, i.e. $\mathbf{B} = A_{*,J}$, for a given J. The subset is very important because it was selected in such a way that it can reconstruct the rest of the columns of $\mathbf{A}$ to a given tolerance ϵ or a given rank k.
- The matrix $\mathbf{P}$ provides a way to interpolate the columns of $\mathbf{B}$ into the full matrix $\mathbf{A}$, thus its name. By construction $\mathbf{P}$ will contain a $k \times k$ identity, usually permuted, that will keep the column subset J intact in $\mathbf{A}$.

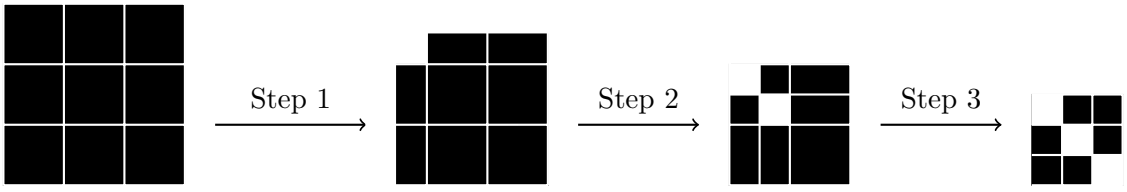


Figure 5: ID: Step-by-step decomposition of row and column blocks. The compressed blocks correspond to the block of the matrix S with a zero diagonal.

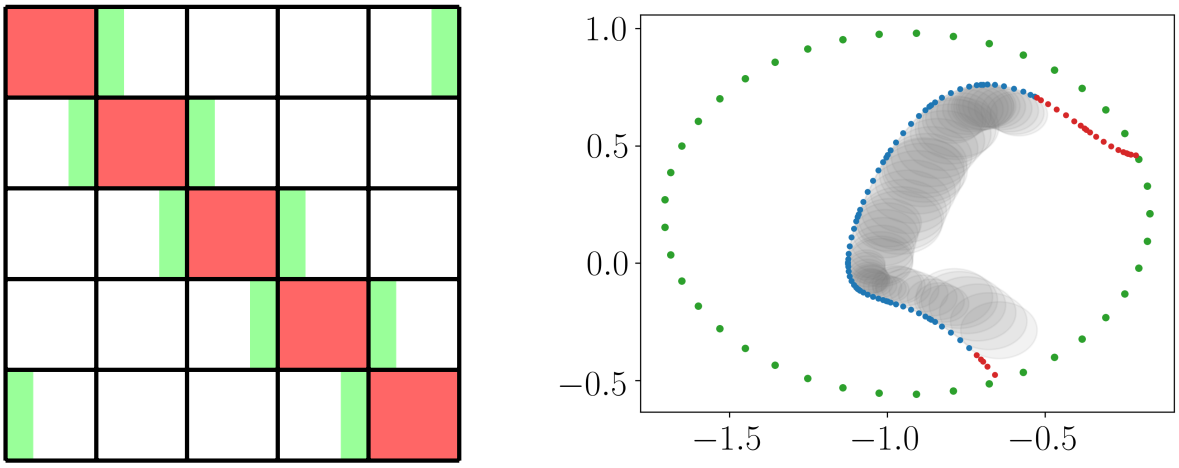
The main way in which we will use the ID is by imposing a desired tolerance ϵ. In this sense, the decomposition is also *rank-revealing*, since it will give us a numerical rank k^c for the column space of $\mathbf{A}$. Alternatively, if the rank is already know or imposed in some way, one can directly construct the decomposition to that specification. From [6], we known that a such a decomposition can be

computed in $\mathcal{O}(mn \log k)$, for a matrix of size $m \times n$ of rank k. We will see that this is sufficient to give an $\mathcal{O}(n)$ algorithm for our problem.

To obtain the decomposition in (4.1) we apply the interpolative decomposition to each block row $A^T_{i,*}$ to obtain $L_i B_i$ and each block column $A_{*,j}$ to obtain $B_j R_j$. Finally, S_{ij} is defined by the row and column skeleton matrices obtain in the previous decompositions. A step by step depiction of this process can be seen in Figure 5.

Skeletonization

Note that performing an interpolative decomposition in the manner presented above will inevitably lead to an $\mathcal{O}(kN^2)$ algorithm. In our particular case, we can take advantage once more of the special structure of the problem and use a *skeletonization* method to dramatically improve the performance of the compression step, as in [9]. Skeletonization relies on the fact that the solution to the Laplace equation is harmonic and satisfies certain Green identities. In particular, this allows us to simulate the interactions from far away sources or targets with a set of proxy points.



(a) (b)

Figure 6: (a) Diagonal (red) and near (green) columns required in the skeletonization.
(b) Skeletonization with proxy points (green), current block points (blue), nearby points (red) and interior expansion centers (gray) for a given block.

The proposed algorithm proceeds as follows: instead of computing the ID of an entire block row $A_{I,*}$, we can construct a proxy matrix:

$$\tilde{A}_I = [A_{I,J_{near}} \; A_{proxy}],$$

where $A_{I,J_{near}}$ corresponds to nearby points and A_{proxy} corresponds to the interactions with a set of proxy points. We can see the required components of A in Figure 6a. In Figure 6b we have a graphical rendering of all the sets of points of interest in this problem.

The most obvious issue with the skeletonization methods is where to place the skeleton points so that they give the greatest amount of information about the far-field interactions. For simplicity, we will place the skeleton points in a circle around the center of mass of the current block $\mathbf{m}$. Any other basic shape can also be used with similar results. Therefore, we have reduced the problem to finding a suitable radius r for the circle. The immediate choice is taking:

$$r_1 = \max_{i \in I} \|\mathbf{m} - \mathbf{y}_i\|_2,$$

i.e. the maximum distance from the center of mass to all the source points in the current block, as represented by the index set I. However, if we are using the QBX method, some of the proxy points may intersect the balls around the expansion centers in the current block. In that case, those intersecting proxy points would need to be associated with an expansion center and computed using QBX. However, we can sidestep this issue by choosing a slightly larger proxy radius. For example, we can take:

$$r_2 = \max_{i \in I} \|\mathbf{m} - \mathbf{c}_i\|_2 + r_i,$$

i.e. the maximum distance to the surface of all the expansion disks. This choice will set all proxy points outside any of the expansion disks and allow us to accurately compute the interactions without the need for QBX. Therefore, we set our final radius to be:

$$r = (1 + \delta_{proxy}) r_2,$$

where δ_{proxy} allows for some control over the exact distance. Another important feature of the skeletonization method is that we do not require to actually construct the entire matrix to solve the system. We can proceed as follows:

- Determine the set of points J_{near} and compute just $A_{I,J_{near}}$ for each index set I.
- Create a set of target proxy points, which will be used to compress the block row I of the matrix. The proxy matrix in this case is given by:

  $$A_{proxy,ij} = K(\mathbf{x}_i, \mathbf{y}_j)\omega_j,$$

  for all sources in the current block $\mathbf{y}_j$ and the set of proxy points $\mathbf{x}_i$.
- Create a set of source proxy points, which will be used to compress the block column I of the matrix. This case is slightly more complicated than the row compression because the source points are tied to the boundary Γ in all our previous definitions. Furthermore, if we are using the double layer potential as our representation, the normal vectors are also only defined for $\mathbf{y}_j \in \Gamma$. As an alternative representation in this case, we take the proxy matrix to be:

  $$A_{proxy,ij} = G(\mathbf{x}_i, \mathbf{y}_j),$$

  for all target points $\mathbf{x}_i$ and the set of proxy source points $\mathbf{y}_j$. The hope is that the monopole representation will give us a similar set of columns in the ID, a hope which we will test numerically in the following sections. If we use the single layer representation, this choice is natural and we do not expect any issues.

Using this method we can dramatically reduce the cost of the method by not forming the entire matrix, reducing the cost of the ID to construct L_i and R_j, etc. In the case of the single layer potential, we also know that the system matrix is symmetric, so only one of L_i or R_i needs to be constructed since $L_i \approx R_i^T$ is a good approximation.

4.2 Single Level Compression

We have seen in the previous section how to perform a decomposition and bring our matrix to the form:

$$\mathbf{A} = \mathbf{LSR} + \mathbf{D},$$

where $\mathbf{L}, \mathbf{R}, \mathbf{D}$ are block diagonal and $\mathbf{S}$ is a dense matrix, as illustrated in Figure 7. Once decomposed in this manner, the inverse can be easily computed as [5]:

$$\mathbf{A}^{-1} = \hat{\mathbf{L}}\hat{\mathbf{S}}^{-1}\hat{\mathbf{R}} + \hat{\mathbf{D}}, \tag{13}$$

where:

$$\hat{\mathbf{D}} = \mathbf{D}^{-1} - \mathbf{D}^{-1}\mathbf{L}\Lambda\mathbf{R}\mathbf{D}^{-1}$$

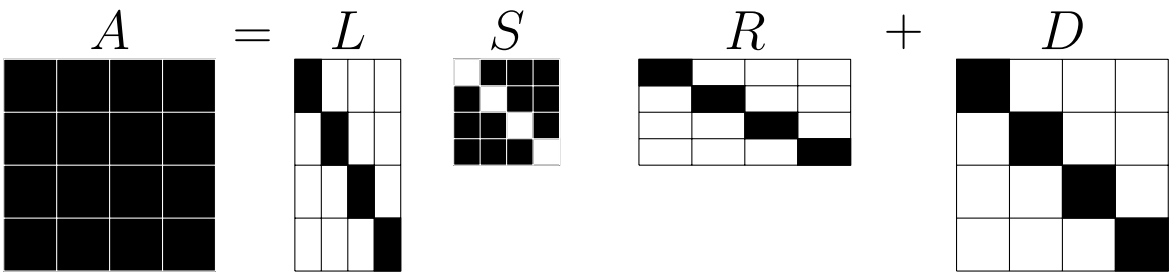


Figure 7: Single level matrix compression.

and

$$\hat{\mathbf{L}} = \mathbf{D}^{-1}\mathbf{L}\Lambda, \quad \hat{\mathbf{R}} = \Lambda\mathbf{R}\mathbf{D}^{-1}$$

are block diagonal matrices, as their counterparts, and:

$$\hat{\mathbf{S}} = \Lambda + \mathbf{S},$$

is a dense matrix with non-zero diagonal because of $\Lambda = (\mathbf{R}\mathbf{D}^{-1}\mathbf{L})^{-1}$, which is an approximation of the compressed diagonal block. If the inverse is not required, but we only desire applying the inverse to a given vector, we can proceed by rewriting our system as:

$$\begin{bmatrix} \mathbf{D} & \mathbf{L}\mathbf{S} \\ -\mathbf{R} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}.$$

We can now proceed by a simple Shur complement-type method. We first multiply the first row by $\mathbf{R}\mathbf{D}^{-1}$ and add it to the second row to obtain:

$$\begin{bmatrix} \mathbf{R} & \mathbf{R}\mathbf{D}^{-1}\mathbf{L}\mathbf{S} \\ \mathbf{0} & \mathbf{I} + \mathbf{R}\mathbf{D}^{-1}\mathbf{L}\mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{R}\mathbf{D}^{-1}\mathbf{b} \\ \mathbf{R}\mathbf{D}^{-1}\mathbf{b} \end{bmatrix}.$$

We can now focus solely on the second row of the equation and multiply it by Λ, as defined above, to obtain:

$$(\Lambda + \mathbf{S})\mathbf{y} = \Lambda\mathbf{R}\mathbf{D}^{-1}\mathbf{b}. \tag{14}$$

This a system of size $\mathbb{R}^{k^r \times k^c}$ and is generally a significantly smaller than the initial system, so it can be solved directly by any available method such as an LU decomposition and Gauss elimination. Once $\mathbf{y}$ has been found, we can use back substitution to obtain $\mathbf{x}$:

$$\mathbf{x} = \mathbf{D}^{-1}\mathbf{b} - \mathbf{D}^{-1}\mathbf{L}\mathbf{S}\mathbf{y}.$$

The main pain points of this algorithm are clearly:

- Obtaining the original decomposition $\mathbf{A} = \mathbf{L}\mathbf{S}\mathbf{R} + \mathbf{D}$. However, once the decomposition has been acquired and the individual blocks of the inverse have been computed , it is very efficient to apply the result to multiple right-hand sides $\mathbf{b}$.
- Obtaining the inverse of $\mathbf{D}$. $\mathbf{D}$ is a block diagonal matrix, so computing its inverse is not very expensive. This applies equally to obtaining Λ, which is even cheaper because the matrix is smaller.

- Solving the system (14). The system is significantly smaller than the original system, so its resolution also incurs a small cost.

As proved in [10], this single level compression algorithm leads to a complexity of $\mathcal{O}(rn^3 + r^3k^3)$, where k is the average rank of each block row or column, r is the number of blocks and n is the size of each block. This approximation applies to our concrete case when solving (8), for a more general case see [5]. For a careful choice of $r = N^{3/5}$, the complexity is given by:

$$\mathcal{O}(k^3N^{9/5}).$$

We will see in the next section how this can be further reduced by making use of recursion. However, it is important to note that the rank k can vary with the choice of n. This problem will persist even in the multilevel case, especially at coarser levels.

4.3 Multilevel Compression

We have all the necessary ingredients to transition to a multilevel scheme at this point:

- The interpolative decomposition, used to compress the matrix between a level l and $l+1$.
- The single level method that can solve level $l+1$ cheaply and use back substitution to obtain the solution at the level l.

The resulting multilevel scheme is shown graphically in Figure 8. The multilevel inverse is given by the telescoping sum:

$$\mathbf{A}^{-1} = \hat{\mathbf{L}}^{(1)} \left[\hat{\mathbf{L}}^{(2)} \left(\cdots \hat{\mathbf{L}}^{(\ell)} \mathbf{S}^{-1} \hat{\mathbf{R}}^{(\ell)} + \hat{\mathbf{D}}^{(\ell)} \cdots \right) \hat{\mathbf{R}}^{(2)} + \hat{\mathbf{D}}^{(2)} \right] \hat{\mathbf{R}}^{(1)} + \hat{\mathbf{D}}^{(1)}, \tag{15}$$

where the various matrices have the same definition as the ones in the single level algorithm (13). Looking at the problem from the point of view of simply applying the inverse to a given vector, we can use the same procedure as presented in the single level case. In particular, at level 2, we can decompose the resulting matrix $\Lambda^{(1)} + \mathbf{S}^{(1)}$ (14) as follows:

- The matrices $\mathbf{L}^{(2)}$ and $\mathbf{R}^{(2)}$ are obtained by performing a second interpolative decomposition on the row and column subsets obtained on level 1. This can be done directly on the original matrix $\mathbf{A}$ by appropriate slicing, so the matrix $\mathbf{S}$ only needs to be constructed on leaf nodes.
- The diagonal matrix $\mathbf{D}^{(2)}$ is given by:

$$\mathbf{D}_{ii}^{(2)} = \begin{bmatrix} \Lambda_{ii}^{(1)} & \mathbf{S}_{i,i+1}^{(1)} \\ \mathbf{S}_{i+1,i}^{(1)} & \Lambda_{i+1,i+1}^{(1)} \end{bmatrix}.$$

  Graphically, this diagonal is exactly the diagonal of the block matrix obtained after clustering in Figure 8.

As suggested in [5], there are several important implementation and efficiency considerations regarding the multilevel algorithm:

- The compression matrices $\mathbf{L}^{(l)}$ and $\mathbf{R}^{(l)}$ need to be stored at each level.
- The matrices $\mathbf{S}^{(l)}$ should not be stored. Instead, a simple set of indices $(I^{(l)}, J^{(l)})$ should be kept for each off-diagonal block, as obtained from the compression step.

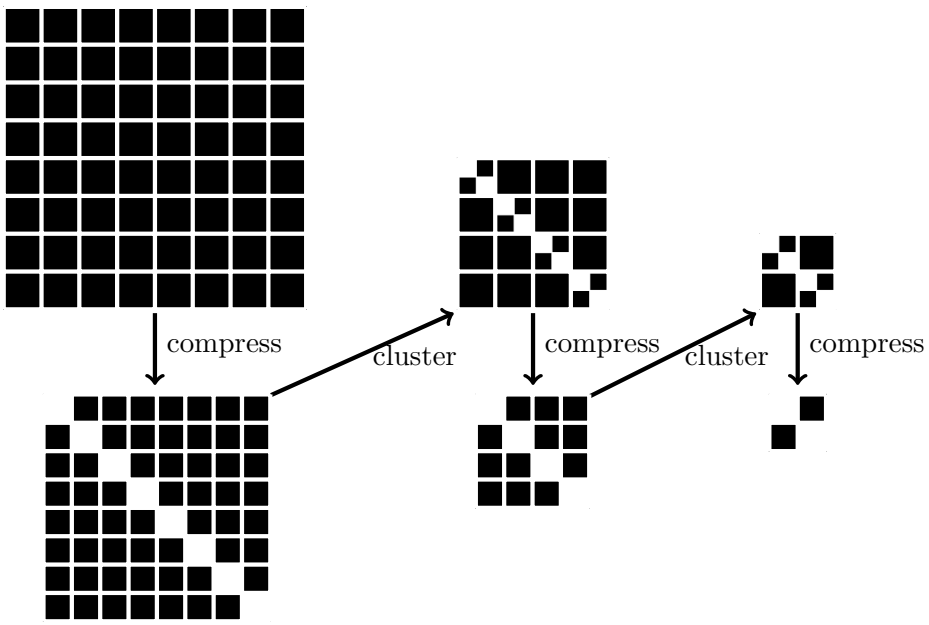


Figure 8: Multilevel compression and clustering. Black blocks are unchanged and low-rank, while white blocks are full rank and correspond to Λ in (13).

- The matrix blocks should be clustered in some manner when moving from one level to the next.The clustering in Figure 8 is done using a binary tree, which is why each 2×2 block interactions are clustered into their parent.
- Using the ID with a tolerance ϵ will invariably give different ranks for the various compressed rows and columns. This means that the off-diagonal blocks will be rectangular after the first compression step. To fix this issue, we impose the tolerance ϵ on either the row or column decomposition and then use the resulting rank in the decomposition of the other one. This will lead to matrices $L_i, R_i^T \in \mathbb{R}^{n \times k_i}$, which implies that $\mathbf{S}$ will be square as well.

The complexity of the multilevel scheme is discussed in [5, Section 5.2], for the problem at hand. Notably, we have seen that the per-level complexity is given by $\mathcal{O}(r^{(l)}(n^{(l)})^3) \approx \mathcal{O}(N^{9/5})$. If we take $n^{(l)} \leq 2k^{(l)}$, where $k^{(l)}$ is the average rank at level l and a clustering method that gives:

$$r^{(l)} = \frac{r^1}{2^{l-1}},$$

we can find that the total cost is:

$$\sum_{l=1}^{l_{max}} r^{(l)}(n^{(l)})^3 \approx \sum_{l=1}^{l_max} \frac{r^{(1)}}{2^l}(l + \log n^{(1)})^3 \approx r^{(1)}(\log n^{(1)})^3.$$

Since at level 1, the number of points in each panel is fixed irrespective of N and $r^{(1)} \in \mathcal{O}(N)$, we get the expected complexity bound.

We now have two different fast methods to solve the original discrete integral equation:

- A fast matrix-vector product powered by the fast multipole method. With a well-conditioned operator, this leads to a $\mathcal{O}(N)$ algorithm.
- A direct solver for the complete discrete system. This method is also $\mathcal{O}(N)$, but will generally be slower because of the rather expensive compression step. However, since the compression

only needs to be done once, the method is still viable if we need to solve for multiple right-hand side vectors.

In fact, the two methods are closer in spirit that immediately obvious. We can interpret the multilevel compression and inverse (15) in the following way [9]:

- First, we perform an upward pass by applying each $\hat{\mathbf{R}}^{(l)}$ to compress $\mathbf{b}$ to the coarsest skeleton subset and solve the compressed system.
- Next, we perform a downward pass by applying each $\hat{\mathbf{L}}^{(l)}$ to decompress the coarse solution vector back to the full solution.

5 Numerical Results

In this section we will be focusing on a testing the accuracy and scaling of our direct solver. Lastly, we will solve the boundary value problem (1) we initially set out to solve.

The geometry we will use for most of our tests is given by:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{3}{4}\cos\left(t - \frac{\pi}{4}\right)\left(1 + \frac{1}{2}\sin 2t\right) \\ \sin\left(t - \frac{\pi}{4}\right)\left(1 + \frac{1}{2}\sin 4t\right) \end{pmatrix}, \tag{16}$$

parametrized by $t \in [0, 2\pi]$. We can see the result in Figure 9, for a set of color-coded panels that have been adapted to satisfy the conditions stated in Section 2.

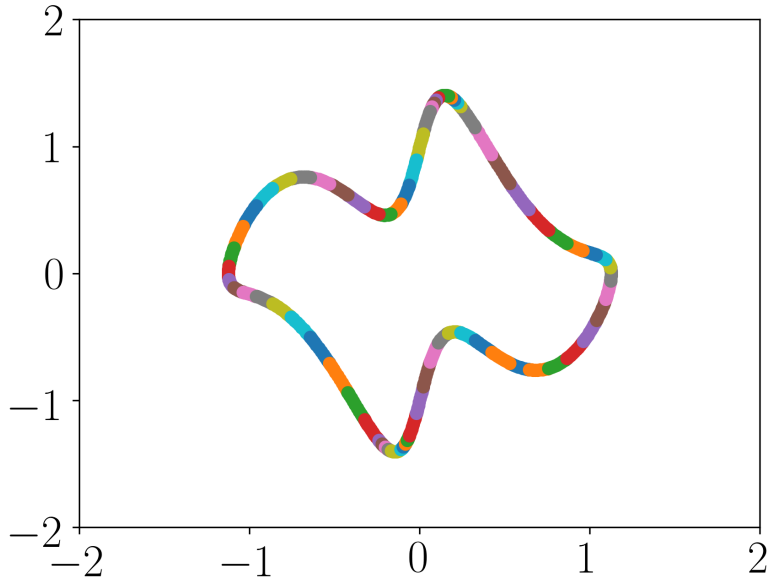


Figure 9: Star geometry given by (16).

The set of parameters we will be using is given as follows:

- $p_{qbx} = 5$, the order of the QBX expansion.
- $q_{qbx} = 16$, the number of quadrature points and weights in each panel.

- $M_{qbx} = 128$, the number of quadrature panels. For our geometry, this is a large enough number of panels that no additional refinement was required.
- $\ell_{ds} = 4$, the number of recursion levels in the direct solver.
- $p_{ds} = 16$, the number of blocks at the finest level of recursion.
- $\epsilon_{ds} = 10^{-10}$, the tolerance for the interpolative decomposition.
- $N_{proxy} = 50$, the number of proxy points in the skeletonization.
- $\delta_{proxy} = 0.25$, the coefficient controlling the distance of the proxy points.

When there is a change for a given test, it will be highlighted. Furthermore, we will use the average representation of the double layer potential as given by (7) and form the usual system of the form:

$$A\mathbf{x} = \mathbf{b}.$$

We will investigate the performance of the two types of direct solvers we have defined based on the type of decomposition they perform:

- First, a direct solver that uses the whole off-diagonal block in the interpolative decomposition (see Figure 4 and the related discussion). We will refer to this method as DS_{block}.
- Second, a direct solver that uses skeletonization to perform the compression (see Figure 6 and the related discussion). We will refer to this method as DS_{skel}.

5.1 Quadrature by Expansion: Condition Number

A first simple test we will perform is to look at the condition number of the resulting matrix operator for the discretized double layer potential with respect to the expansion order p_{qbx} and quadrature order q_{qbx}. We can see the results in Table 1.

Table 1: Condition number of the discrete integral operator (8).

p_{qbx}	$q_{qbx} = 4$	$q_{qbx} = 8$	$q_{qbx} = 16$
3	8.06155	8.39672	8.54373
5	7.92243	8.39019	8.54077
7	7.84448	8.37186	8.54133

As expected, all the resulting matrices have small condition numbers. However, it is interesting to note that the condition number seems to consistently decrease, ever so slightly, with an increase in the QBX expansion order. We expect this is because a higher-order expansion will capture the well-behaving continuous operator better.

5.2 Direct Solver: Accuracy

Next, we will look at the accuracy of our direct solver. Since there are quite a number of parameters in play, we need to ascertain what their effect is and how they interact with each other. We will use a random exact solution and solve the system with the parameters described above.

Test 1 This test will be done for varying expansion orders p_{qbx} and quadrature orders q_{qbx} and a single level of recursion. The results can be seen in Table 2 and Table 3. Several observation are in order here. First, we can see that the errors for the full off-diagonal block ID are smaller than the errors for the skeletonized version of the direct solver. This is to be expected, since a certain amount of information is lost when using a fixed number of proxy points. Better results can be obtained by using more proxy points, at the cost of efficiency. Second, we can see that the relative errors decrease when increasing the QBX expansion order, but there is no significant change with relation to the quadrature order. This is a very interesting phenomenon, which we expect is related to how well the singularity on the boundary is captured. A similar effect can be seen in Table 1 with relation to the condition number of the resulting approximation of the integral equation operator.

Table 2: Accuracy: DS_{block} errors with respect to expansion orders.

p_{qbx}	$q_{qbx} = 4$		$q_{qbx} = 8$		$q_{qbx} = 16$	
	ϵ_{error}	$\epsilon_{residual}$	ϵ_{error}	$\epsilon_{residual}$	ϵ_{error}	$\epsilon_{residual}$
3	9.12872e-09	4.86248e-09	2.76578e-08	1.51805e-08	5.01436e-09	2.76596e-09
5	1.39169e-09	7.67312e-10	4.96305e-09	2.78140e-09	1.90317e-09	1.01418e-09
7	2.39623e-10	1.29254e-10	5.33001e-10	2.98953e-10	3.38780e-10	1.86535e-10

Table 3: Accuracy: DS_{skel} errors with respect to expansion orders.

p_{qbx}	$q_{qbx} = 4$		$q_{qbx} = 8$		$q_{qbx} = 16$	
	ϵ_{error}	$\epsilon_{residual}$	ϵ_{error}	$\epsilon_{residual}$	ϵ_{error}	$\epsilon_{residual}$
3	2.54206e-08	1.40420e-08	6.51848e-08	3.49296e-08	3.66725e-08	2.02563e-08
5	4.11533e-09	2.27906e-09	8.23051e-09	4.32480e-09	4.38584e-09	2.37350e-09
7	7.07775e-10	3.74813e-10	5.75930e-10	3.14966e-10	9.00992e-10	5.01606e-10

Test 2 For our next test, we will look at the influence of recursion and the number of blocks in the decomposition of our operator. The results can be seen in Table 4 and Table 5. The main takeaway from these tests is that the relative errors do not increase as a result of adding more levels of recursion. Furthermore, the number of blocks in the decomposition also has a negligible effect on the errors. Both of these facts are very important because the desired $\mathcal{O}(N)$ complexity is only achieved when we take $\ell_{ds} = \log_2 N$ levels and $p_{ds} = N^{3/5}$ blocks. Therefore, we can conclude that there is a very small trade-off between accuracy and efficiency for both versions of our direct solver.

Table 4: Accuracy: DS_{block} errors with respect to recursion levels.

p_{ds}	$\ell_{ds} = 2$		$\ell_{ds} = 3$		$\ell_{ds} = 4$	
	ϵ_{error}	$\epsilon_{residual}$	ϵ_{error}	$\epsilon_{residual}$	ϵ_{error}	$\epsilon_{residual}$
16	2.31175e-09	1.22432e-09	2.26341e-09	1.19034e-09	1.80893e-09	9.59989e-10
32	7.19358e-09	3.97550e-09	6.56725e-09	3.65275e-09	6.68821e-09	3.66717e-09
64	7.96960e-09	4.42266e-09	8.86560e-09	4.93349e-09	8.83224e-09	4.92415e-09

Test 3 This test only relates to the skeletonized direct solver. We will investigate the effect of the number of proxy points and proxy point distance on the accuracy of the resulting solver. The

Table 5: Accuracy: DS_{skel} errors with respect to recursion levels.

p_{ds}	$\ell_{ds} = 2$		$\ell_{ds} = 3$		$\ell_{ds} = 4$	
	ϵ_{error}	$\epsilon_{residual}$	ϵ_{error}	$\epsilon_{residual}$	ϵ_{error}	$\epsilon_{residual}$
16	5.18507e-09	2.83233e-09	8.49552e-09	4.67426e-09	4.44692e-09	2.45182e-09
32	7.67471e-09	4.18553e-09	1.01932e-08	5.61474e-09	1.27750e-08	6.98161e-09
64	6.31291e-09	3.43362e-09	8.10852e-09	4.47831e-09	1.03811e-08	5.76539e-09

results can be seen in Table 6. As expected, if the proxy points intersect the current block or the expansion loci in the current block, we cannot expect to retrieve the desired accuracy. However, for a large enough proxy distance, we retrieve the desired accuracy which seems to decrease with an increase in number of proxy points. It is important to note that a larger distance and a larger number of proxy points will eventually lead to worse performance that approaches the simpler DS_{block} method.

Table 6: Accuracy: DS_{skel} errors with respect to proxy points.

N_{proxy}	$\delta_{proxy} = -0.2$		$\delta_{proxy} = 0.5$		$\delta_{proxy} = 1.2$	
	ϵ_{error}	$\epsilon_{residual}$	ϵ_{error}	$\epsilon_{residual}$	ϵ_{error}	$\epsilon_{residual}$
30	2.17671e-04	1.18686e-04	5.64100e-08	2.96544e-08	1.07313e-08	5.85785e-09
50	1.33253e-04	7.37563e-05	2.63332e-08	1.45011e-08	1.36580e-08	7.39889e-09
70	6.94320e-05	3.86785e-05	1.98028e-08	1.06562e-08	1.04501e-08	5.78254e-09

5.3 Direct Solver: Scaling

A very important step in our numerical tests is to investigate the actual scaling of the different solvers. We will use the same parameters as before. To test the scaling, we will use the walltime as a proxy for the actual cost of the method. There are more rigorous methods to prove scaling, but this should give us a good indication of whether we can hope to reach the theoretical results or not. To determine the scaling we have use the Theil-Sen estimator to linearly fit the curve (M_{qbx}, t). This estimator was chosen because it is meant to reduce the effect of outliers in the dataset.

Test 1 For a first test, we will look at the scaling with respect to the number of panels and the number of recursion levels of the decomposition step of the direct solver. The results can be seen in Table 7. This is the step that is expected to be more costly, so it is important that it follows the desired complexity trends. We can see that this is indeed the case for both the DS_{block} and the DS_{skel} methods. In fact the skeletonization-based method is close to achieving the optimal $\mathcal{O}(N)$ scaling as we increase the number of recursion levels. In either case, both methods are significantly faster than an equivalent LU decomposition, which scales as $\mathcal{O}(N^3)$.

Test 2 Next, we will look at the solving and back substitution step of the direct solver. The results can be seen in Table 8. As expected, this step is significantly faster than the decomposition step. In the limit of increasing number of levels ℓ_{ds}, we again obtain the optimal scaling $\mathcal{O}(N)$ for both variants of the direct solver. One interesting result is that the skeletonization-based solver is slower than its counterpart. We suspect this is because, for the same accuracy, the skeletonization leads to a slightly larger rank for off-diagonal blocks and, thus, to a slower solve step.

Table 7: Scaling: Time (in seconds) of the decomposition step.

M_{qbx}		$\ell_{ds} = 1$		$\ell_{ds} = 2$		$\ell_{ds} = 4$	
	LU	DS_{block}	DS_{skel}	DS_{block}	DS_{skel}	DS_{block}	DS_{skel}
128	0.223459	0.087522	0.054367	0.125639	0.062884	0.149225	0.091416
256	1.743877	0.216969	0.067689	0.257622	0.098431	0.284817	0.135409
512	15.761304	0.868573	0.194028	0.931527	0.238213	0.985756	0.322106
1024	134.603205	4.315787	0.805866	4.293598	0.874878	4.349963	0.957627
Scaling	**3.08**	**1.93**	**1.40**	**1.77**	**1.27**	**1.70**	**1.18**

Table 8: Scaling: Time (in seconds) of the solve step.

M_{qbx}		$\ell_{ds} = 1$		$\ell_{ds} = 2$		$\ell_{ds} = 4$	
	LU	DS_{block}	DS_{skel}	DS_{block}	DS_{skel}	DS_{block}	DS_{skel}
128	0.001124	0.093797	0.118155	0.031990	0.039818	0.008003	0.012252
256	0.003797	0.131069	0.195297	0.041528	0.058118	0.012495	0.020289
512	0.015309	0.188231	0.310070	0.061076	0.088699	0.025766	0.040509
1024	0.056273	0.262167	0.450505	0.102924	0.148544	0.064412	0.093247
Scaling	**1.88**	**0.49**	**0.65**	**0.55**	**0.62**	**1.00**	**0.98**

6 Conclusions

We have presented here several highly efficient methods to solve a boundary value problem and its equivalent integral equation. We have seen that, in the special case where our problem can be reduced to an integral equation on the boundary of the domain, we can solve the resulting system in linear time. These results have been shown both theoretically and numerically in various tests.

We expect the methods and results extend naturally to other boundary value problems, such as the Helmholtz equation. However, as noted in [5] and others, the Helmholtz equation comes with a new set of subtleties related to the wave number and the direct solver is not always expected to perform optimally.

We have also shown that the quadrature by expansion method can be successfully coupled with both the Fast Multipole Method, as described in [12], and with a skeletonization-based direct solver. At least for the well-conditioned integral equation resulting from a double layer potential, we have shown that expected accuracy and scaling results can be retrieved numerically for these methods. Further tests could include applications to the more demanding single layer potential, investigation of the optimal number of recursion levels, how a high-order expansion $p_{qbx} > 10$ impacts efficiency, etc.

References

[1] J. Barnes, P. Hut, *A Hierarchical $\mathcal{O}(n \log n)$ Force Calculation Algorithm*, Nature, Vol. 324, pp. 446–449, 1986.

[2] L. Greengard, V. Rokhlin, *A Fast Algorithm for Particle Simulations*, Journal of Computational Physics, Vol. 135, pp. 280–292, 1997.

[3] R. Kress, *Linear Integral Equations*, Springer, Second Edition, 1999.

[4] H. Cheng, Z. Gimbutas, P. G. Martinsson, V. Rokhlin, *On the Compression of Low Rank Matrices*, SIAM Journal on Scientific Computing, Vol. 26, pp. 1389–1404, 2005.

[5] P. G. Martinsson, V. Rokhlin, *A Fast Direct Solver for Boundary Integral Equations in Two Dimensions*, Journal of Computational Physics, Vol. 205, pp. 1-23, 2005.

[6] E. Liberty, F. Woolfe, P. G. Martinsson, V. Rokhlin, M. Tygert, *Randomized Algorithms for the Low Rank Approximation of Matrices*, Proceedings of the National Academy of Sciences, Vol. 154, pp. 20167–20172, 2007.

[7] H. Sundar, R. S. Sampath, G. Biros, *Bottom-up Construction and 2:1 Balance Refinement of Linear Octrees in Parallel*, SIAM Journal on Scientific Computing, Vol. 30, pp. 2675–2708, 2008.

[8] L. Greengard, D. Gueyffier, P. G. Martinsson, V. Rokhlin, *Fast Direct Solvers for Integral Equations in Complex Three-dimensional Domains*, Acta Numerica, Vol. 18, pp. 243–275, 2009.

[9] K. L. Ho, L. Greengard, *A Fast Direct Solver for Structured Linear Systems by Recursive Skeletonization*, SIAM Journal on Scientific Computing, Vol. 34, pp. 2507–2532, 2012.

[10] A. Gillman, P. M. Young, P. G. Martinsson, *A Direct Solver with $\mathcal{O}(n)$ Complexity for Integral Equations on One-dimensional Domains*, Frontiers of Mathematics in China, Vol. 7, pp. 217–247, 2012.

[11] A. Klöckner, A. Barnett, L. Greengard, M. O'Neil, *Quadrature by Expansion: A New Method for the Evaluation of Layer Potentials*, Journal of Computational Physics, Vol. 252, pp. 332–349, 2013.

[12] M. Rachh, A. Klöckner, M. O'Neil, *Fast Algorithms for Quadrature by Expansion I: Globally Valid Expansions*, Journal of Computational Physics, Vol. 345, pp. 706–731.