Fast direct methods for molecular electrostatics

by

Kenneth L. Ho

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Program in Computational Biology

New York University

May 2012

Prof. Leslie Greengard

Acknowledgements

First and foremost, I would like to thank my adviser, Prof. Leslie Greengard, whose guidance and advice has been indispensable over these past five years. I am especially grateful for the freedom that he has allowed in my research, as well as his unique perspective on computational biology, which will no doubt shape my own philosophy as I move forward.

Profs. Charlie Peskin, Jerry Percus, and Mark Tygert, and Dr. Jeff Bell have likewise played an outsized role, and I am thankful to them, too, for their insight and encouragement. I am also grateful to my committee for their continued interest and support: Profs. Leslie Greengard, Rich Bonneau, Charlie Peskin, Yingkai Zhang, and Dan Tranchina.

Thank you also to the many members of the group for the useful discussions, especially to Profs. Leslie Greengard, Mark Tygert, and June-Yub Lee; and Drs. Zydrunas Gimbutas, Andras Pataki, Mike O'Neil, Andreas Klöckner, and Josef Sifuentes.

I am indebted as well to my fellow students (past and present), with whom I have shared this incredible journey. Special thanks go to Andrew Matteson, Sandra May, Heather Harrington, Segun Jung, Dharshi Devendran, Eduardo Corona, Kela Lushi, and Emily Chan. I would also like to express my deep appreciation to the wonderful staff at Courant: Tamar Arnon, Susan Mrsic, and Brittany Shields.

Lastly, I thank all my friends and family, but most of all my parents, who have supported me without question throughout my time away, first in Pasadena and then in New York—thank you for your kindness, your love, and your patience.

Abstract

Electrostatic interactions are vital for many aspects of biomolecular structure and function, including stability, activity, and specificity. Thus, a central problem in biophysical modeling is the electrostatic analysis of biomolecular systems. Here, we consider the numerical solution of the linearized Poisson-Boltzman equation (LPBE), a widely used model for the electrostatic potential of a large, solvated biomolecule. By combining boundary integral techniques with new multilevel matrix compression algorithms, we develop a fast direct solver for the LPBE that is accurate, robust, and can be more efficient than current methods by several orders of magnitude.

The fast direct solver is general and applies to a wide range of integral operators based on non-oscillatory Green's functions, including those for the Laplace, low-frequency Helmholtz, Stokes, and LPBEs. The core algorithm uses the interpolative decomposition to compress the matrix discretizations of such operators, producing highly efficient representations that facilitate fast inversion. For boundary integral equations in 2D, the solver has complexity $\mathcal{O}(N)$, where N is the number of discretization elements; in 3D, it incurs an $\mathcal{O}(N^{3/2})$ cost for precomputation, followed by $\mathcal{O}(N \log N)$ solves. As is typical of direct methods, each solve can be performed extremely rapidly, though the cost of precomputation can be high. Thus, the solver is particularly suited to problems where the precomputation time can be amortized, e.g., systems with ill-conditioned matrices or involving multiple right-hand sides.

We demonstrate our solver on a number of examples and discuss various useful extensions. Furthermore, we apply our methods to the calculation of protein pK_a

values, which requires the computation of all pairwise titrating site energies. This corresponds to solving the LPBE on the same molecular geometry with many different boundary conditions on the protein surface, each manifesting as a different right-hand side, and hence presents a prime candidate for acceleration using our direct solver. Preliminary results are favorable and show the viability of our techniques for molecular electrostatics.

Such fast direct methods could well have broad impact on many areas of computational science and engineering. We describe further applications in biology, chemistry, and physics, and outline some directions for future work.

Contents

List of Figures

List of Tables

1 Introduction

The study of macromolecular structure and function is central to modern biology and has provided a rich history of mechanistic insights on such fundamental biological components as DNA [121, 199], protein kinases [113, 147], G protein-coupled receptors [186], and the machinery of cell death [68] and degradation [56], among others. Given a structure, its function and mode of action are in principle determined by the corresponding physics, which at the molecular scale are commonly divided into three classes:

1. bonded forces, characterizing the quantum physics of covalent bonds;
2. electrostatic forces, describing the interaction of charged ions; and
3. van der Waals forces, covering local but non-bonded interactions.

Among these, electrostatics is unique for its long range and hence its ability to coordinate large-scale intra- and intermolecular processes [110]. Indeed, it has been found to participate in many aspects of macromolecular structure and dynamics, including conformational remodeling [33, 114], protein-protein interaction [173], protein stability [206], and enzyme specificity [189]. Given its widespread importance, it is clear that accurate electrostatic analysis is essential for a faithful physical description of biomolecular systems, and thus for a quantitative understanding of biological structure and function at the molecular level.

In this dissertation, we develop new mathematical and computational techniques for the electrostatics of large, solvated biomolecules. Our method relies on a continuum model of the solvent and calculates the electrostatic potential by solving the linearized Poisson-Boltzmann equation (LPBE). The LPBE is recast as a second-kind boundary integral equation, which gives it a strong mathematical foundation, and then solved using a new fast algorithm that can be seen as an extension of the fast multipole method (FMM) [86, 93] for Coulomb summation. In contrast to iteratve FMM-based solvers, however, our algorithm directly computes the solution operator for the LPBE and so can be far more efficient when many solves are required for the same molecular geometry, e.g., with many different boundary conditions as induced by different charge configurations. The result is a direct solver for macromolecular electrostatics that is fast, accurate, and robust, with a well-understood mathematical theory and a controlled numerical error.

We begin in the next section with a discussion of the electrostatic potential of a biomolecular system.

1.1 Electrostatics of biomolecular systems

Consider a biomolecule represented explicitly as a collection of atoms, possibly charged, immersed in a salt solution. Through van der Waals interactions, the atoms in the molecule carve out a solvent-excluded region Ω_1, which we define as the molecular volume; its exterior is composed of the solvent, which permeates all of space $\Omega_0 \equiv \mathbb{R}^3 \setminus \Omega_1$ (Figure 1.1). What is the electrostatic potential of this system?

One way of approaching this is to represent the solvent explicitly as well, discretizing Ω_0 as a collection of solvent molecules, each itself composed of charged atoms,

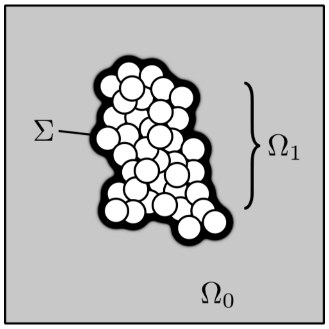

Figure 1.1: A biomolecular system consisting of a solvent-excluded molecular volume Ω_1 with molecular surface Σ, immersed in a salt solution $\Omega_0 = \mathbb{R}^3 \setminus \Omega_1$.

and then to compute the electrostatic potential using Coulomb's law:

$$\varphi(\boldsymbol{r}) = k_e \sum_i \frac{q_i}{|\boldsymbol{r} - \boldsymbol{r}_i|},$$

where k_e is the Coulomb constant, q_i and $\boldsymbol{r}_i$ are the charge and location, respectively, of atom i, and the sum runs over all atoms, both in the molecule and in the solvent. This is generally considered the most realistic treatment since it respects the atomic nature of the solvent, and is the predominant method used in molecular dynamics (MD) simulations [see 127], which are often employed to minimize the energies of biomolecular systems. This realism, however, comes at a significant expense, as a sufficiently large solvent volume must be discretized in order to capture its bulk properties. For protein simulations in water, for example, standard MD protocols call for enough water molecules to buffer the protein by at least 10 Å in each direction. For moderately sized proteins on the order of 100 residues, say, this can add up to 3000 water molecules in total, giving about 10^4 extra atoms, thus vastly increasing the system size.

An alternative approach is to describe the solvent using a continuum theory. Here, the fundamental equation is the Poisson equation

$$-\nabla \cdot (\varepsilon \nabla \varphi) = \rho, \tag{1.1}$$

where $\varepsilon(\boldsymbol{r}) = \varepsilon_i$ for $\boldsymbol{r} \in \Omega_i$ is the dielectric constant, representing some macroscopic average of the microscopic atomic motions, and ρ is the charge. This is a second-order elliptic partial differential equation (PDE) whose study has been central to much of applied mathematics [see, e.g., 211]. In the molecule, we assume that

$$\rho(\boldsymbol{r}) \equiv \sum_i q_i \delta(\boldsymbol{r} - \boldsymbol{r}_i) \quad \text{in } \Omega_1$$

is composed of discrete interior point charges, where δ is the Dirac delta function; thus, the electrostatic potential in the molecule satisfies

$$-\Delta\varphi = \frac{1}{\varepsilon_1} \sum_i q_i \delta(\boldsymbol{r} - \boldsymbol{r}_i) \quad \text{in } \Omega_1. \tag{1.2}$$

The situation in the solvent is slightly more complicated since we must account for the presence of free ions, which can move in response to the electric field and act to counter its effect. For this, using Boltzmann's theory [122], we let

$$\rho(\boldsymbol{r}) \equiv \sum_i q_i c_i(\boldsymbol{r}) = \sum_i q_i c_i^\infty \exp\left[-\frac{q_i \varphi(\boldsymbol{r})}{k_B T}\right] \quad \text{in } \Omega_0,$$

where c_i is the concentration of ion species i, here taken as the mean field distribution due to φ; c_i^∞ is the corresponding bulk concentration; k_B is the Boltzmann constant; and T is the absolute temperature. Substituting this into (1.1), we obtain the Poisson-Boltzmann equation (PBE)

$$-\Delta\varphi = \frac{1}{\varepsilon_0} \sum_i q_i c_i^\infty \exp\left[-\frac{q_i \varphi(\boldsymbol{r})}{k_B T}\right] \quad \text{in } \Omega_0, \tag{1.3}$$

a widely used model that captures the effect of ionic screening.

The PBE is often linearized by assuming that the electrostatic energy is small, i.e., $q_i\varphi(\boldsymbol{r}) \ll k_B T$ for all i. In this case,

$$\exp\left[-\frac{q_i\varphi\left(\boldsymbol{r}\right)}{k_B T}\right] \approx 1 - \frac{q_i\varphi\left(\boldsymbol{r}\right)}{k_B T},$$

so, to first order,

$$-\Delta\varphi = \frac{1}{\varepsilon_0}\sum_i q_i c_i^\infty \left(1 - \frac{q_i\varphi}{k_B T}\right) = -\frac{1}{\varepsilon_0}\sum_i \frac{q_i^2 c_i^\infty}{k_B T}\varphi \quad \text{in } \Omega_0,$$

where the first term vanishes by bulk electroneutrality. This is more commonly written as

$$-\left(\Delta - \kappa^2\right)\varphi = 0 \quad \text{in } \Omega_0, \tag{1.4}$$

where

$$\kappa = \sqrt{\frac{2IF^2}{\varepsilon_0 RT}}$$

is the inverse Debye length, characterizing the distance over which the electric field is screened, for

$$I = \frac{1}{2}\sum_i c_i^\infty z_i^2$$

the ionic strength, where z_i is the charge number of ion species i (so $q_i = z_i e$, where e is the elementary charge); F the Faraday constant; and R the gas constant. Observe that the LPBE (1.4) reduces to the Poisson equation at $\kappa = 0$; thus, (1.4) is often also called the screened Poisson equation.

Note 1.1. The LPBE also appears in the context of nuclear physics as the equation describing the Yukawa interaction potential between elementary particles [210].

In this work, we focus only on the linearized form (1.4). This is due in part to numerical considerations as the linear PDE is more straightforward to solve, especially using FMM-based algorithms. Moreover, the validity of the nonlinear PBE (1.3) is not entirely clear since in the regime of strong interactions, it may be important to include also, e.g., finite size effects and ion correlations, which the PBE neglects [58]. Nonetheless, it should be understood that the methods presented here also have consequences for the nonlinear equation since nonlinear systems are typically solved via iterative linearization. For further details on continuum electrostatic theory and the PBE, we refer the reader to [58, 72, 172]; for an in-depth discussion of the nonlinear PBE, we recommend the excellent treatment in Prof. Michael Holst's thesis [109].

Combining (1.2) and (1.4), the PDE for the electrostatic potential is therefore

$$-\left(\Delta - \kappa^2\right)\varphi = 0 \qquad \text{in } \Omega_0, \tag{1.5a}$$

$$-\Delta\varphi = \frac{1}{\varepsilon_1}\sum_i q_i \delta\left(\boldsymbol{r} - \boldsymbol{r}_i\right) \qquad \text{in } \Omega_1, \tag{1.5b}$$

an interface LPBE system, with the interface conditions

$$[\varphi] = \left[\varepsilon \frac{\partial \varphi}{\partial \nu}\right] = 0 \quad \text{on } \Sigma \tag{1.6}$$

by continuity of the potential and its flux (cf. (1.1)), where $\Sigma \equiv \partial\Omega_1$ is the solvent-excluded molecular surface separating Ω_0 and Ω_1 [see 52], ν is the unit outer surface normal on Σ, and bracket notation denotes the jump across the interface, commonly taken as the exterior value minus the interior value.

Remark 1.2. Several studies have found it useful to represent ions in the solvent explicitly [172, and references therein]. We can incorporate these into the LPBE as fixed counterions by including a source term:

$$-\left(\Delta - \kappa^2\right)\varphi = \sum_i q_i \delta\left(\boldsymbol{r} - \boldsymbol{r}_i\right) \quad \text{in } \Omega_0.$$

The same can be done for the solvent itself, at least for a few layers near the molecule, which has been shown to be important in capturing surface effects [127, 132].

Remark 1.3. The above framework can also accomodate a system of multiple solvated biomolecules by specifying an equation of the form

$$-\Delta\varphi = \frac{1}{\varepsilon_i}\sum_j q_{ij}\delta\left(\boldsymbol{r} - \boldsymbol{r}_{ij}\right) \quad \text{in } \Omega_i$$

for each molecule Ω_i with dielectric ε_i for $i = 1, 2, \dots$, where q_{ij} and $\boldsymbol{r}_{ij}$ are the charge and location, respectively, of atom j in Ω_i. The interface conditions (1.6) must now apply on each molecular surface $\Sigma_i \equiv \partial\Omega_i$.

1.2 Numerical solution of the Poisson-Boltzmann equation

The LPBE (1.5) can be solved in many ways, perhaps the most popular of which is via finite differences. In this approach, the Laplace operator Δ is discretized on a grid using, e.g., a seven-point stencil, and the resulting algebraic system solved using standard techniques [188]; the material parameters ε and κ are typically taken as spatially varying and implicit define the molecular surface Σ. This forms the basis of the so-called finite difference Poisson-Boltzmann methods (FDPB), which count among them the very mature DelPhi program initially developed in Prof. Barry Honig's lab (`http://wiki.c2b2.columbia.edu/honiglab_public/index.php/Software:DelPhi`). For a discussion of the various optimizations in DelPhi, including fast relaxation schemes, electrostatic focusing, and extensions to the nonlinear PBE and multi-dielectric cases, see [81, 148, 162]. Other FDPB schemes include [13, 42, 212].

Another approach is to use finite elements to discretize the solution space [187]. The finite element method has a rich mathematical theory and is routinely used by the engineering community to perform complex structural analyses. One particular advantage of finite elements over finite differences is its support for unstructured meshes, which allows for efficient adaptive refinement [10] of irregular geometries such as molecular surfaces. Progress on finite element-based solvers for the PBE has mainly been driven by the work of Holst and Baker [12, 43, 109, 108], particularly through their software APBS (`http://www.poissonboltzmann.org/apbs`), but results by others have been reported as well [55, 175]

Despite their prevalence, both finite difference and finite element methods have the major drawback that the resulting system matrices are generally ill-conditioned. Specifically, the condition number of the corresponding matrix A typically scales as $\kappa(A) = \mathcal{O}(1/h^2)$, where h is the minimum mesh width, meaning that past a certain threshold, the error actually *increases* with additional refinement due to the amplification of numerical noise. This is in contrast to integral equation methods [97, 184], which can be formulated in a well-conditioned manner, i.e., $\kappa(A) = \mathcal{O}(1)$, and hence enable more robust convergence. This is one of the reasons that we have adopted an integral equation approach. Furthermore, integral equations for problems such as (1.5) often require unknowns only on the domain boundary, leading to a dimensional reduction in the linear system to be solved. In this sense, integral equations reveal the true size of a problem, in contrast to finite differences or finite elements, which must discretize the entire volume, with artificial truncation of infinite domains [24, 145]. Of course, the boundary must now be represented explicitly, which can be a challenging problem in computational geometry, though with the advantage that boundary conditions can be enforced exactly for the discretized system [cf. 42, 212]. Lastly, integral equation methods also support the accurate computation of derivatives by analytic differentiation of the integral kernel (i.e., no numerical differentiation is required), which can be important, e.g., for force calculations.

For a review of recent developments in numerical methods for the PBE, including finite difference, finite volume, finite element, boundary element, and hybrid schemes, see [134].

We now turn to a brief treatment of integral equations, using the Laplace equation as an example. For a more complete account, including potential and Fredholm theory, we refer the interested reader to [91, 97, 184].

Example 1.1. Consider the interior Dirichlet problem for the Laplace equation in a simply connected domain $\Omega \subset \mathbb{R}^3$ with boundary $\partial\Omega$:

$$-\Delta u = 0 \quad \text{in } \Omega, \qquad u = f \quad \text{on } \partial\Omega.$$

Potential theory suggests a solution in the form of a double-layer potential:

$$u(\boldsymbol{r}) \equiv \int_{\partial\Omega} \frac{\partial G}{\partial \nu_{\boldsymbol{s}}}(\boldsymbol{r}, \boldsymbol{s})\, \mu(\boldsymbol{s})\, dS_{\boldsymbol{s}} \quad \text{in } \Omega, \tag{1.7}$$

where

$$G(\boldsymbol{r}, \boldsymbol{s}) = \frac{1}{4\pi |\boldsymbol{r} - \boldsymbol{s}|}$$

is the Green's function (fundamental solution) for the Laplace equation and μ is an unknown surface density. Note that (1.7) satisfies the PDE by construction since

$$-\Delta G(\boldsymbol{r}, \boldsymbol{s}) = \delta(\boldsymbol{r} - \boldsymbol{s}) = 0$$

for $\boldsymbol{r} \in \Omega$ and $\boldsymbol{s} \in \partial\Omega$; the idea then is to use μ to match the boundary data.

Physically, (1.7) represents the electric field due to a surface dipole layer of strength μ. We can therefore think of each local surface patch as an ideal capacitor, which from elementary physics gives rise to a discontinuous field [158]. Hence the double-layer potential should experience a jump in crossing $\partial\Omega$; we formalize this observation with the following theorem.

Theorem 1.1. *Assuming that the quantities involved are all sufficiently smooth, the double-layer potential* (1.7) *satisfies*

$$u\left(\boldsymbol{r}^{\pm}\right) = \pm\frac{1}{2}\mu\left(\boldsymbol{r}\right) + u\left(\boldsymbol{r}\right), \quad \frac{\partial u}{\partial \nu}\left(\boldsymbol{r}^{+}\right) = \frac{\partial u}{\partial \nu}\left(\boldsymbol{r}^{-}\right) \qquad \text{on } \partial\Omega,$$

where $\boldsymbol{r}^{\pm}$ denotes the limit from $\mathbb{R}^3 \setminus \Omega$ and Ω, respectively, and all integrals are taken in the principal value sense.

Returning now to our example, taking the limit of (1.7) as $\boldsymbol{r} \to \partial\Omega$ thus yields

$$-\frac{1}{2}\mu\left(\boldsymbol{r}\right) + \int_{\partial\Omega} \frac{\partial G}{\partial \nu_{\boldsymbol{s}}}\left(\boldsymbol{r}, \boldsymbol{s}\right) \mu\left(\boldsymbol{s}\right) dS_{\boldsymbol{s}} = f\left(\boldsymbol{r}\right) \quad \text{on } \partial\Omega, \tag{1.8}$$

a second-kind integral equation for μ that is provably well-conditioned. Once μ has been obtained, the solution can be evaluated at any point $\boldsymbol{r} \in \Omega$ using (1.7).

Example 1.2. We can proceed similarly for the interior Neumann problem

$$-\Delta u = 0 \quad \text{in } \Omega, \qquad \frac{\partial u}{\partial \nu} = g \quad \text{on } \partial\Omega,$$

provided that the integral of g is zero since otherwise we have an unphysical steady-state heat conduction problem with non-vanishing boundary flow [184]. We use now a single-layer potential

$$u\left(\boldsymbol{r}\right) \equiv \int_{\partial\Omega} G\left(\boldsymbol{r}, \boldsymbol{s}\right) \sigma\left(\boldsymbol{s}\right) dS_{\boldsymbol{s}}, \tag{1.9}$$

which corresponds to the field due to a surface charge layer of strength σ and satisfies the following jump relations:

Theorem 1.2. *Under the same conditions as Theorem 1.1, the single-layer potential* (1.9) *satisfies*

$$u\left(\boldsymbol{r}^{+}\right) = u\left(\boldsymbol{r}^{-}\right), \quad \frac{\partial u}{\partial \nu}\left(\boldsymbol{r}^{\pm}\right) = \mp \frac{1}{2}\sigma\left(\boldsymbol{r}\right) + \frac{\partial u}{\partial \nu}\left(\boldsymbol{r}\right) \qquad \text{on } \partial\Omega.$$

Hence taking the limit as $\boldsymbol{r} \to \partial\Omega$, we have the second-kind integral equation

$$\frac{1}{2}\sigma\left(\boldsymbol{r}\right) + \int_{\partial\Omega} \frac{\partial G}{\partial \nu_{\boldsymbol{r}}}\left(\boldsymbol{r}, \boldsymbol{s}\right)\sigma\left(\boldsymbol{s}\right) dS_{\boldsymbol{s}} \quad \text{on } \partial\Omega. \tag{1.10}$$

Remark 1.4. We use so-called indirect representations such as (1.7) or (1.9) since the direct representation

$$u\left(\boldsymbol{r}\right) \equiv \int_{\partial\Omega} \left[\frac{\partial G}{\partial \nu_{\boldsymbol{s}}}\left(\boldsymbol{r}, \boldsymbol{s}\right) u\left(\boldsymbol{s}\right) - G\left(\boldsymbol{r}, \boldsymbol{s}\right)\frac{\partial u}{\partial \nu}\left(\boldsymbol{s}\right)\right] dS_{\boldsymbol{s}}$$

from Green's identity using the actual boundary data often gives only integral equations of the first kind, e.g.,

$$\int_{\partial\Omega} G\left(\boldsymbol{r}, \boldsymbol{s}\right)\frac{\partial u}{\partial \nu}\left(\boldsymbol{s}\right) dS_{\boldsymbol{s}} = -f\left(\boldsymbol{r}\right) + \int_{\partial\Omega} \frac{\partial G}{\partial \nu_{\boldsymbol{s}}}\left(\boldsymbol{r}, \boldsymbol{s}\right) f\left(\boldsymbol{s}\right) dS_{\boldsymbol{s}} \quad \text{on } \partial\Omega$$

for the Dirichlet problem. Here, the unknown $\partial u/\partial \nu$ appears only under the integral, which is a smoothing operator and hence ill-conditioned to invert. In contrast, second-kind integral operators such as (1.8) and (1.10) are of the form $I + K$, where I is the identity and K is compact, which affords superior solvability properties.

Remark 1.5. The same jump conditions and second-kind integral equations hold for the LPBE (1.4) if we use the Green's function

$$G_{\kappa}\left(\boldsymbol{r}, \boldsymbol{s}\right) \equiv \frac{e^{-\kappa\left|\boldsymbol{r}-\boldsymbol{s}\right|}}{4\pi\left|\boldsymbol{r}-\boldsymbol{s}\right|}.$$

In this notation, the Laplace Green's function is just G_0.

Definition 1.1. Hereafter, we use the shorthand

$$S_k[\sigma](\boldsymbol{r}) \equiv \int_\Gamma G_k(\boldsymbol{r},\boldsymbol{s})\,\sigma(\boldsymbol{s})\,dS_{\boldsymbol{s}}, \quad D_k[\mu](\boldsymbol{r}) \equiv \int_\Gamma \frac{\partial G_k}{\partial \nu_{\boldsymbol{s}}}(\boldsymbol{r},\boldsymbol{s})\,\mu(\boldsymbol{s})\,dS_{\boldsymbol{s}}$$

for the single- and double-layer potentials, respectively, where Γ is to be understood from the context. We will also make use of prime notation to denote normal differentiation, i.e., $S_k' \equiv \partial S_k/\partial\nu$ and $D_k' \equiv \partial D_k/\partial\nu$.

We are now armed with the necessary tools to formulate a well-conditioned integral equation for the molecular electrostatics system (1.5). To this end, we write the solution as a combination of Laplace and Yukawa single- and double-layer potentials on the molecular surface Σ:

$$\varphi \equiv \begin{cases} S_\kappa\sigma + D_\kappa\mu & \text{in } \Omega_0, \\ S_0\sigma + \alpha D_0\mu + \varphi_s & \text{in } \Omega_1, \end{cases} \tag{1.11}$$

where $\alpha \equiv \varepsilon_0/\varepsilon_1$ is the dielectric ratio, and

$$\varphi_s(\boldsymbol{r}) \equiv \frac{1}{\varepsilon_1}\sum_i q_i G_0(\boldsymbol{r},\boldsymbol{r}_i)$$

is the potential due to the sources. By Theorems 1.1 and 1.2,

$$\lim_{\boldsymbol{r}\to\Sigma} \varphi(\boldsymbol{r}) = \begin{cases} \mu/2 + S_\kappa\sigma + D_\kappa\mu & \text{in } \Omega_0, \\ -\alpha\mu/2 + S_0\sigma + D_0\mu + \varphi_s & \text{in } \Omega_1, \end{cases}$$

and

$$\lim_{\boldsymbol{r}\to\Sigma} \frac{\partial\varphi}{\partial\nu}(\boldsymbol{r}) = \begin{cases} -\sigma/2 + S_\kappa'\sigma + D_\kappa'\mu & \text{in } \Omega_0, \\ \sigma/2 + S_0'\sigma + \alpha D_0'\mu + \varphi_s' & \text{in } \Omega_1, \end{cases}$$

so enforcing the interface conditions (1.6) gives

$$\begin{aligned} \frac{1}{2}\left(1+\alpha\right)\mu + \left(S_\kappa - S_0\right)\sigma + \left(D_\kappa - \alpha D_0\right)\mu &= \varphi_s, \\ -\frac{1}{2}\left(1+\alpha\right)\sigma + \left(\alpha S'_\kappa - S'_0\right)\sigma + \alpha\left(D'_\kappa - D'_0\right)\mu &= \frac{\partial \varphi_s}{\partial \nu}, \end{aligned}$$

or, in operator notation, the block system

$$\left(I + \lambda K\right)\begin{bmatrix} \mu \\ \sigma \end{bmatrix} = \lambda \begin{bmatrix} \varphi_s \\ -\partial \varphi_s / \partial \nu \end{bmatrix}, \tag{1.12}$$

where $\lambda = 2/(1+\alpha)$ and

$$K = \begin{bmatrix} D_\kappa - \alpha D_0 & S_\kappa - S_0 \\ -\alpha\left(D'_\kappa - D'_0\right) & -\left(\alpha S'_\kappa - S'_0\right) \end{bmatrix}.$$

Note the difference $D'_\kappa - D'_0$, which mollifies the hypersingular terms D'_κ and D'_0 by singularity subtraction [117]; essentially the same approach was taken by Juffer et al. in [116]. The system (1.12) is a well-conditioned second-kind boundary integral equation for σ and μ.

Remark 1.6. This formulation extends naturally to the case of multiple biomolecules, viz. Remark 1.3, for which the analogue of (1.11) is

$$\varphi \equiv \begin{cases} \sum_i \left(S_\kappa \sigma_i + D_\kappa \mu_i\right) & \text{in } \Omega_0, \\ S_0 \sigma_i + \alpha_i D_0 \mu_i + \varphi_{s,i} & \text{in } \Omega_i, \end{cases}$$

where σ_i and μ_i are layer densities on Σ_i, $\alpha_i \equiv \varepsilon_0 / \varepsilon_i$, and

$$\varphi_{s,i}\left(\boldsymbol{r}\right) \equiv \frac{1}{\varepsilon_i} \sum_j q_{ij} G_0\left(\boldsymbol{r} - \boldsymbol{r}_{ij}\right).$$

The resulting system then becomes

$$(I+K)\begin{bmatrix}\mu_1\\ \sigma_1\\ \mu_2\\ \sigma_2\\ \vdots\end{bmatrix} = \begin{bmatrix}\lambda_1\varphi_{s,1}\\ -\lambda_1\varphi'_{s,1}\\ \lambda_2\varphi_{s,2}\\ -\lambda_2\varphi'_{s,2}\\ \vdots\end{bmatrix},$$

where $\lambda_i = 2/(1+\alpha_i)$ and K is composed of 2×2 blocks of the form

$$K_{ii} = \lambda_i \begin{bmatrix} D_\kappa - \alpha_i D_0 & S_\kappa - S_0 \\ -\alpha_i\left(D'_\kappa - D'_0\right) & -\left(\alpha_i S'_\kappa - S'_0\right)\end{bmatrix}$$

with

$$K_{ij} = \lambda_i \begin{bmatrix} D_\kappa & S_\kappa \\ -\alpha_i D'_\kappa & -\alpha_i S'_\kappa\end{bmatrix} \quad \text{for } i \neq j.$$

Equation (1.12) is an exact statement for the continuous functions σ and μ, and must be discretized to render it amenable to numerical computation. For this, we represent Σ as a collection of flat triangles T_i, on each of which we take σ and μ as constant with values σ_i and μ_i, respectively. This is a first-order discretization; second-order methods based on curved triangles and polynomial densities have recently been developed [4, 17], but this is sufficient for our current purposes. Many programs are available for producing such triangulations, including [53, 156, 169]. We complete the discretization by employing a collocation scheme and enforcing the integral equation at each centroid $\boldsymbol{c}_i$ of T_i, which gives

$$(I + \lambda K) \begin{bmatrix} \mu_1 \\ \sigma_1 \\ \vdots \\ \mu_{N_{\text{tri}}} \\ \sigma_{N_{\text{tri}}} \end{bmatrix} = \lambda \begin{bmatrix} \varphi_s(\boldsymbol{c}_1) \\ -\varphi_s'(\boldsymbol{c}_1) \\ \vdots \\ \varphi_s(\boldsymbol{c}_{N_{\text{tri}}}) \\ -\varphi_s'(\boldsymbol{c}_{N_{\text{tri}}}) \end{bmatrix}, \tag{1.13}$$

where N_{tri} is the total number of triangles and the operators in K are now specified in discrete form as the matrix entries

$$S_k(\boldsymbol{c}_i) = \sum_j \int_{T_j} G_k(\boldsymbol{c}_i, \boldsymbol{s})\, dS_{\boldsymbol{s}}, \quad D_k(\boldsymbol{c}_i) = \sum_j \int_{T_j} \frac{\partial G_k}{\partial \nu_{\boldsymbol{s}}}(\boldsymbol{c}_i, \boldsymbol{s})\, dS_{\boldsymbol{s}},$$

and similarly for their derivatives; the operators in (1.11) for computing φ once the σ_i and μ_i have been recovered are discretized in the same way. For $k = 0$, these integrals are evaluated analytically, while for $k \neq 0$, Gauss-Legendre quadrature is used on the smooth difference, e.g., $S_k - S_0$, and the result added to S_0. The discretized system (1.13) is a matrix equation of the form $Ax = b$, where $A \in \mathbb{R}^{N \times N}$ for $N = 2N_{\text{tri}}$.

Note 1.7. On flat surfaces,

$$\int_{T_i} \frac{\partial G_k}{\partial \nu_{\boldsymbol{c}_i}}(\boldsymbol{c}_i, \boldsymbol{s})\, dS_{\boldsymbol{s}} = \int_{T_i} \frac{\partial G_k}{\partial \nu_{\boldsymbol{s}}}(\boldsymbol{c}_i, \boldsymbol{s})\, dS_{\boldsymbol{s}} = 0 \quad \text{for all } i.$$

In other words, the operators S_k' and D_k in (1.13) vanish on the diagonal.

Note 1.8. Although we consider only collocation here, it is worthwhile to point out alternative discretization schemes such as qualocation [178], which has been shown to be particularly effective for integrated quantities [18, 192], and the Galerkin method [e.g., 28], which is more expensive as it requires double integrals but has the benefit of symmetrizing A.

We now turn to the problem of solving (1.13). Following our discussion above, the system matrix is nonsingular, well-conditioned, and nonsymmetric; it is, however, also dense, in contrast to matrices produced by finite difference or finite element discretizations, which are generally sparse and so enable highly efficient inversion schemes [75, 166]. Thus, sophisticated techniques are required to treat (1.13) in a computationally tractable way, in particular to beat the $\mathcal{O}(N^3)$ cost of direct inversion, which severely limits the size and resolution of any possible numerical simulation. In the following sections, we briefly review the current state of the art for solving dense integral equation systems, starting with those based on iteration.

1.3 Fast iterative solvers

The first fast solvers for integral equations appeared in the late 1980s and were based on two key developments:

1. Krylov subspace methods like GMRES [167] and BiCGSTAB [195], which can solve linear systems on the basis of only matrix-vector products; and
2. fast algorithms to rapidly compute such products [87], e.g., tree codes [9, 20], panel clustering methods [101], and FMMs [36, 48, 86], among others [3, 31, 157].

Since A is well-conditioned, the number of iterations required for convergence is typically quite small and independent of N, i.e., $\mathcal{O}(1)$. Therefore, the cost of performing a solve is simply proportional to that of applying A to a vector, which the methods above have all been very successful at reducing from the generic $\mathcal{O}(N^2)$ to the optimal or near-optimal $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$, respectively.

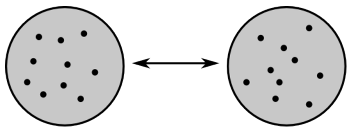

Figure 1.2: Well-separated point clusters with constant interaction rank.

Such fast multiplication schemes were originally designed for the Laplace kernel G_0, e.g., for gravitational or Coulombic potentials, and exploited the simple observation that, to any specified precision $\epsilon > 0$, a separated point cluster can be accurately described by only a constant number of degrees of freedom of the order $\mathcal{O}(\log(1/\epsilon))$, irrespective of the number of points in the cluster or its detailed structure (Figure 1.2). This is supported by physical intuition, wherein a collection of charges, for instance, can be replaced by a single effective charge provided that the observation point is sufficiently far away, i.e., in the so-called far field. In linear algebraic terms, this means that large submatrices of A are low-rank and hence can be approximated very efficiently. Combining this with a hierarchical decomposition of space via, e.g., an octree [168], we can apply this idea in a multiscale manner, resulting in a highly compressed representation of A requiring only $\mathcal{O}(N \log N)$ entries (Figure 1.3). The optimal $\mathcal{O}(N)$ estimate for FMMs requires some additional machinery relating to recursively merging and splitting data-sparse cluster representations; for details, see [86, 93].

The same reasoning gives rise to fast algorithms for other integral kernels based on Green's functions for non-oscillatory elliptic PDEs, including the Yukawa kernel G_k [30, 95, 112, 130]. This has led to the construction of kernel-independent FMMs based only on smooth interpolation [73, 83, 141, 208]. FMMs for oscillatory kernels such as those for the Helmholtz and Maxwell equations have also been developed, but

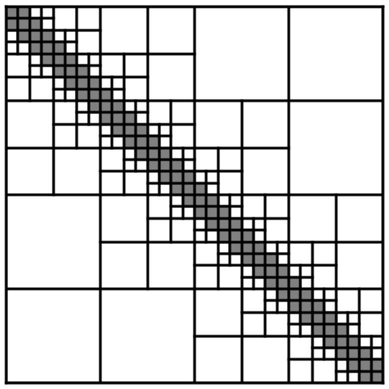

Figure 1.3: Structure of an FMM matrix on the line, where gray blocks denote full-rank near-field interactions, and white blocks denote low-rank far-field interactions. All white blocks have the same rank.

these rely on somewhat different techniques [45, 46, 69, 70, 164, 181]. Altogether, such fast algorithms have had a tremendous impact on computational science and engineering [49, 133, 150, and references therein], and are now used to solve problems that previously would have been infeasible [103, 123, 160, 209].

1.4 Fast direct solvers

Despite their successes, however, the fast solvers above are still iterative by nature, and as such have several significant disadvantages when compared with their direct counterparts:

1. Although second-kind integral equation systems are well-conditioned with respect to N, they can still be ill-conditioned with respect to some problem parameter. Such ill conditioning arises, for example, in the solution of problems near resonance (particularly in the high-frequency regime), in geometries with

"close-to-touching" interactions, and in multi-component physics models with large material contrasts [see 90]. Under these circumstances, the number of iterations required by an iterative solver can be far greater than expected. Direct methods, on the other hand, are robust in the sense that their solution time does not degrade with conditioning. Thus, they are often preferred in production environments, where reliability of the solver and predictability of the solve time are important.

2. When solving a collection of problems governed by a fixed matrix and multiple right-hand sides, as often occurs in the modeling of physical processes in fixed geometries [see, e.g., 15, 50], most iterative methods are unable to effectively exploit the fact that the system matrix is the same and simply treat each right-hand side anew [cf. 39, 71, 196]. In contrast, direct methods are highly efficient in this regard: once the system matrix has been factored, the matrix inverse can be applied to each right-hand side at a much lower cost.

3. In much the same way, when solving problems where the system matrix is altered by low-rank modifications, e.g., in the context of optimization and design, standard iterative methods similarly experience no speedup, whereas direct methods can update the matrix factorization using the Sherman-Morrison-Woodbury formula [102] or use the existing factorization as a preconditioner.

Of course, their strengths notwithstanding, the overriding weakness of all classical direct methods is their $\mathcal{O}(N^3)$ cost. A natural question, therefore, is to ask to what extent direct solvers can be likewise accelerated.

This task was first addressed in 1D by Greengard, Rokhlin, and Starr in [94, 124, 185]. The approach was then analyzed by Hackbusch et al. in a more general context as part of a research program on fast linear algebra for so-called $\mathscr{H}$-matrices [98–100]; for a discussion of current $\mathscr{H}$-matrix methods, see [29]. Since then, a number of fast direct solvers have been developed, most notably within the framework of hierarchically semiseparable matrices [40, 41, 204] and skeletonization procedures [77, 78, 88, 139], among others [35, 155]. All are very closely related and rely on essentially the same FMM-type compression machinery. To date, however, such solvers have largely been shown to be practical only for boundary integral equations in 2D; with few exceptions, effective, multilevel solvers in higher dimensions have been lacking [cf. 200].

In this dissertation, we describe a multilevel fast direct solver for non-oscillatory integral equations that is quite general and formulated without reference to the underlying physical problem dimension. Special attention has been paid to keeping the presentation clear and describing the algorithm in purely linear algebraic terms; this also greatly simplifies its implementation. The solver is based heavily on [88, 139] and uses a recursive skeletonization scheme to produce a data-sparse matrix representation that serves as a platform for fast matrix algebra, including matrix-vector multiplication, matrix factorization, and matrix inverse application. For boundary integral equations in 3D, e.g., the electrostatics system (1.13), our algorithm is capable of $\mathcal{O}(N \log N)$ solves (following precomputation) that are extremely efficient, often beating FMM/GMRES by several orders of magnitude.

1.5 Outline of the dissertation

The remainder of this dissertation is organized as follows.

In Chapter 2, we give a complete description of our fast direct solver, starting with an analysis of the hierarchical low-rank block structure of non-oscillatory integral equation matrices and how to exploit it using the interpolative decomposition, a recently introduced matrix approximation technique. We then give algorithms for multilevel matrix compression, compressed matrix-vector multiplication, and compressed matrix factorization and inverse application via a highly structured sparse embedding. This is followed by complexity and error estimates, which we verify through extensive numerical experiments, including results on low-frequency wave scattering. Much of this chapter has previously appeared as [107].

In Chapter 3, we discuss some extensions of the direct solver, e.g., for block and composite operators, preconditioning, local geometric perturbations, and overdetermined least squares. The first three are rather straightforward applications of the existing technology; the last is slightly more involved and constitutes a semi-direct approach comprising a fast sparse QR factorization followed by rapid iterative corrections. We also describe how the same framework can support an $\mathcal{O}(N)$ compression-based FMM as a higher dimensional generalization of [141], which should prove quite useful for kernels which are difficult to handle analytically.

In Chapter 4, we apply our techniques to protein pK_a calculations, which forms one example in which the LPBE (1.13) must be solved with many different right-hand sides, each corresponding to a different charge configuration. Thus, this presents an enticing opportunity for our direct solver, which can achieve an impressive speedup through its relative efficiency. In this regard, our work here can be considered a heavily accelerated version of [115], which used a similar integral equation approach but was limited by classical numerical methods. Furthermore, our program incorporates various optimizations including the reduced site approximation [22], a novel procedure for sampling tightly coupled charge clusters, and a simple statistical technique for estimating pK_a errors. The validity of the overall approach is confirmed with results for several commonly studied proteins.

Finally, in Chapter 5, we end with some generalizations and concluding remarks, including prescriptions on adapting our algorithm to other biophysical problems such as biomolecular charge optimization, protein structure prediction and design, and protein-protein docking. These are all characterized by immense search spaces and so provide very attractive applications for our direct solver. We also offer some brief comments and observations on future fast direct solvers, including those for the oscillatory case. The dissertation concludes with a summary of our main results and contributions.

2 A fast direct solver for non-oscillatory integral equations

In this chapter, we present a fast direct solver for structured linear systems based on multilevel matrix compression. The relevant matrix structure is characterized by a hierarchical block low-rank property similar to that utilized by fast matrix-vector product techniques like the fast multipole method (FMM). Such matrices commonly arise from the discretization of integral equations, e.g., the molecular electrostatics system (1.13), where the rank structure can be understood in terms of far-field interactions between point clusters, but the algorithm is general and makes no a priori assumptions about rank. Our scheme is a multilevel extension of [88], which itself is based on the fast direct solver for 2D boundary integral equations developed by Martinsson and Rokhlin [139]. We have also constructed what we hope is a useful formalism that simplifies the theoretical analysis and allows for a more straightforward implementation.

The core algorithm in our work computes a compressed matrix representation using the interpolative decomposition [47, 131, 203] via a multilevel procedure that we call recursive skeletonization. Once obtained, the compressed representation enables fast matrix-vector multiplication, matrix factorization, and matrix inverse application. As a fast matrix-vector product scheme, the algorithm may be viewed as a generalized or kernel-independent FMM [73, 83, 141, 208]; we explore this application in §2.8. For matrix inversion, the additional steps involve embedding the compressed representation into an equivalent sparse system much in the style of [40, 155], and then using standard sparse direct solver technology such as UMFPACK [62], SuperLU [64], MUMPS [5], or PARDISO [170]. For maximum flexibility, the solver is divided into two phases: a precomputation phase comprising matrix compression and factorization, followed by a solution phase to apply the matrix inverse. As expected, the solution phase is very inexpensive, often beating the FMM by several orders of magnitude. For boundary integral equations without highly oscillatory kernels, e.g., the Green's functions for the Laplace or low-frequency Helmholtz equations, both phases typically have complexity $\mathcal{O}(N)$ in 2D, where N is the system size; in 3D, these are $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$ for precomputation and solution, respectively.

A particularly interesting aspect of our algorithm is that its specification is purely algebraic, while its performance is due to analysis, namely approximation theory. Thus, it achieves a powerful combination of both efficiency and robustness. A version of this chapter has previously appeared as [107].

2.1 Mathematical preliminaries

Let $A \in \mathbb{C}^{N \times N}$ be a matrix whose index vector $J \equiv (1, 2, \dots, N)$ is grouped into p contiguous blocks of n_i elements each, where $\sum_{i=1}^{p} n_i = N$:

$$J_i = \left(\sum_{j=1}^{i-1} n_j + 1, \sum_{j=1}^{i-1} n_j + 2, \dots, \sum_{j=1}^{i} n_j \right) \quad \text{for } i = 1, \dots, p.$$

Then the linear system $Ax = b$ can be written in the form

$$\sum_{j=1}^{p} A_{ij} x_j = b_i \quad \text{for } i = 1, \dots, p,$$

where $x_i, b_i \in \mathbb{C}^{n_i}$ and $A_{ij} \in \mathbb{C}^{n_i \times n_j}$. Solution of the full system by classical Gaussian elimination is well-known to require $\mathcal{O}(N^3)$ work [84].

Definition 2.1. The matrix A is said to be *block separable* if each off-diagonal submatrix A_{ij} can be decomposed as the product of three low-rank matrices:

$$A_{ij} = L_i S_{ij} R_j \quad \text{for } i \neq j, \tag{2.1}$$

where $L_i \in \mathbb{C}^{n_i \times k_i^{\mathrm{r}}}$, $S_{ij} \in \mathbb{C}^{k_i^{\mathrm{r}} \times k_j^{\mathrm{c}}}$, and $R_j \in \mathbb{C}^{k_j^{\mathrm{c}} \times n_j}$, with $k_i^{\mathrm{r}}, k_i^{\mathrm{c}} \ll n_i$. Note that the left matrix L_i depends only on the index i and the right matrix R_j only on the index j.

We will see how such a factorization arises below. The term "block separable" was introduced in [78] and is closely related to that of semiseparable matrices [40, 41, 204] and $\mathscr{H}$-matrices [98–100]. In [88], the term "structured" was used, but "block separable" is somewhat more informative.

Definition 2.2. The *ith off-diagonal block row* of A is the submatrix

$$\begin{bmatrix} A_{i1} & \cdots & A_{i(i-1)} & A_{i(i+1)} & \cdots & A_{ip} \end{bmatrix}$$

consisting of the ith block row of A with the diagonal block A_{ii} deleted; the *off-diagonal block columns* of A are defined analogously.

Clearly, the block separability condition (2.1) is equivalent to requiring that the off-diagonal block rows and columns have low rank.

When A is block separable, it can be written as

$$A = D + LSR, \tag{2.2}$$

where

$$D = \begin{bmatrix} A_{11} & & \\ & \ddots & \\ & & A_{pp} \end{bmatrix} \in \mathbb{C}^{N \times N}$$

is block diagonal, consisting of the diagonal blocks of A,

$$L = \begin{bmatrix} L_1 & & \\ & \ddots & \\ & & L_p \end{bmatrix} \in \mathbb{C}^{N \times K_\mathrm{r}}, \qquad R = \begin{bmatrix} R_1 & & \\ & \ddots & \\ & & R_p \end{bmatrix} \in \mathbb{C}^{K_\mathrm{c} \times N}$$

are block diagonal, with $K_\mathrm{r} = \sum_{i=1}^{p} k_i^\mathrm{r}$ and $K_\mathrm{c} = \sum_{i=1}^{p} k_i^\mathrm{c}$, and

$$S = \begin{bmatrix} 0 & S_{12} & \cdots & S_{1p} \\ S_{21} & 0 & \cdots & S_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ S_{p1} & S_{p2} & \cdots & 0 \end{bmatrix} \in \mathbb{C}^{K_\mathrm{r} \times K_\mathrm{c}}$$

is dense with zero diagonal blocks. It is convenient to let $z \equiv Rx$ and $y \equiv Sz$. We can then write the original system as

$$\begin{bmatrix} D & L & \\ R & & -I \\ & -I & S \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix}, \tag{2.3}$$

which is highly structured and sparse, and can be factored efficiently using standard sparse matrix techniques. If we assume that each block corresponds to $n_i \equiv N/p$

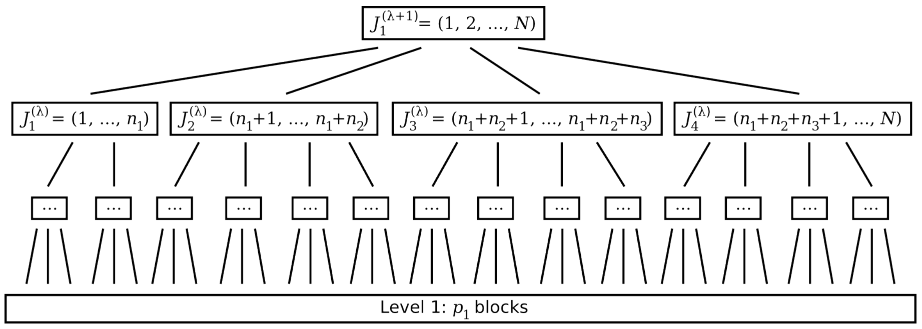
$J_1^{(\lambda+1)} = (1, 2, \ldots, N)$
$J_1^{(\lambda)} = (1, \ldots, n_1)$
$J_2^{(\lambda)} = (n_1+1, \ldots, n_1+n_2)$
$J_3^{(\lambda)} = (n_1+n_2+1, \ldots, n_1+n_2+n_3)$
$J_4^{(\lambda)} = (n_1+n_2+n_3+1, \ldots, N)$
Level 1: p_1 blocks

Figure 2.1: An example of a tree structure imposed on the index vector $(1, 2, \ldots, N)$. At each level of the hierarchy, a contiguous block of indices is divided into a set of children, each of which corresponds to a contiguous subset of those indices.

unknowns and that the ranks $k_i^{\mathrm{r}} = k_i^{\mathrm{c}} \equiv k$ of the off-diagonal blocks are all the same, it is straightforward to see [78, 88] that a scheme based on (2.2) or (2.3) requires an amount of work of the order $\mathcal{O}(p(N/p)^3 + p^3k^3)$.

In many contexts, including integral equations, the notion of block separability is applicable on a hierarchy of subdivisions of the index vector. That is to say, a decomposition of the form (2.2) can be constructed at each level of the hierarchy. When a matrix has this structure, much more powerful solvers can be developed. Hence assume now that a tree τ is imposed on J which is $\lambda + 1$ levels deep. At level l, we assume that there are p_l nodes, with each such node $J_i^{(l)}$ corresponding to a contiguous subsequence of J such that $\left\{J_1^{(l)}, J_2^{(l)}, \ldots, J_{p_l}^{(l)}\right\} = J$. We denote the *finest level* as level 1 and the coarsest level as level $\lambda + 1$, which consists of a single block. Each node $J_i^{(l)}$ at level $l > 1$ has a finite number of children at level $l - 1$ whose concatenation yields the indices in $J_i^{(l)}$ (Figure 2.1).

Following [78], we say that A is *hierarchically block separable* (or hierarchically structured) if it is block separable at each level of the hierarchy defined by τ. In

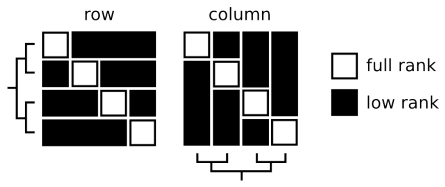

Figure 2.2: Matrix rank structure. At each level of the index tree, the off-diagonal block rows and columns (black) must have low numerical rank; the diagonal blocks (white) can in general be full-rank.

other words, it is structured in the sense of the present chapter if, on each level of τ, the off-diagonal block rows and columns are low-rank (Figure 2.2). Such matrices arise, for example, when discretizing integral equations with non-oscillatory kernels (up to a specified precision).

Example 2.1. Consider the integral operator

$$\phi(\boldsymbol{r}) \equiv \int G(\boldsymbol{r}, \boldsymbol{s})\, \rho(\boldsymbol{s})\, dV_{\boldsymbol{s}} \tag{2.4}$$

where

$$G(\boldsymbol{r}, \boldsymbol{s}) \equiv -\frac{1}{2\pi} \log|\boldsymbol{r} - \boldsymbol{s}| \tag{2.5}$$

is the Green's function for the 2D Laplace equation, and the domain of integration is a square B in the plane. This is a 2D *volume integral operator*. Suppose now that we discretize (2.4) on a $\sqrt{N} \times \sqrt{N}$ grid:

$$\phi(\boldsymbol{r}_i) = \frac{1}{N} \sum_{j \neq i} G(\boldsymbol{r}_i, \boldsymbol{r}_j)\, \rho(\boldsymbol{r}_j)\,. \tag{2.6}$$

(This is not a high-order quadrature but that is really a separate issue.) Let us superimpose on B a quadtree of depth $\lambda + 1$, where B is the root node at level $\lambda + 1$.

Level λ is obtained from level $\lambda+1$ by subdividing the box B into four equal squares and reordering the points $\boldsymbol{r}_i$ so that each child holds a contiguous set of indices. This procedure is carried out until level 1 is reached, reordering the nodes at each level so that the points contained in every node at every level correspond to a contiguous set of indices. It is clear that, with this ordering, the matrix corresponding to (2.6) is hierarchically block separable since the interactions between non-adjacent boxes at each level are low-rank from standard multipole estimates [86, 93]; adjacent boxes are low-rank for a more subtle reason (see §2.5 and Figure 2.7).

Example 2.2. Suppose now that we wish to solve an interior Dirichlet problem for the Laplace equation in a simply connected 3D domain Ω with boundary $\partial\Omega$:

$$\Delta u = 0 \quad \text{in } \Omega, \qquad u = f \quad \text{on } \partial\Omega, \tag{2.7}$$

viz. Example 1.1. Proceeding as before, we write the solution in the form of a double-layer potential

$$u\left(\boldsymbol{r}\right) \equiv \int_{\partial\Omega} \frac{\partial G}{\partial \nu_{\boldsymbol{s}}} \left(\boldsymbol{r}, \boldsymbol{s}\right) \sigma\left(\boldsymbol{s}\right) dS_{\boldsymbol{s}} \quad \text{in } \Omega, \tag{2.8}$$

where

$$G\left(\boldsymbol{r}, \boldsymbol{s}\right) = \frac{1}{4\pi \left|\boldsymbol{r} - \boldsymbol{s}\right|} \tag{2.9}$$

is the Green's function for the 3D Laplace equation, ν is the unit outer surface normal on $\partial\Omega$, and σ is an unknown surface density. Letting $\boldsymbol{r}$ approach the boundary, this gives rise to the second-kind Fredholm equation

$$-\frac{1}{2}\sigma\left(\boldsymbol{r}\right) + \int_{\partial\Omega} \frac{\partial G}{\partial \nu_{\boldsymbol{s}}} \left(\boldsymbol{r}, \boldsymbol{s}\right) \sigma\left(\boldsymbol{s}\right) dS_{\boldsymbol{s}} = f\left(\boldsymbol{r}\right). \tag{2.10}$$

Using a standard collocation scheme over a triangulated surface, we enclose $\partial\Omega$ in a box and bin sort the triangle centroids using an octree where, as in the previous example, we reorder the nodes so that each box in the hierarchy contains contiguous indices. It can be shown that the resulting matrix is also hierarchically block separable (see §2.5 and [88]).

We turn now to a discussion of the interpolative decomposition (ID), the compression algorithm that we will use to compute low-rank approximations of off-diagonal blocks. Many decompositions exist for low-rank matrix approximation, including the singular value decomposition, which is well-known to be optimal [84]. Here, we consider instead the ID [47, 131, 203], which produces a near-optimal representation that is more effective for our purposes as it permits an efficient scheme for multilevel compression when applied in a hierarchical setting. A useful feature of the ID is that it is able to compute the rank of a matrix on the fly, since the exact ranks of the blocks are difficult to ascertain a priori—that is to say, the ID is rank-revealing.

Definition 2.3. Let $A \in \mathbb{C}^{m \times n}$ be a matrix, and $\| \cdot \|$ the matrix 2-norm. A rank-k approximation of A in the form of an *interpolative decomposition (ID)* is a representation $A \approx BP$, where $B \in \mathbb{C}^{m \times k}$, whose columns constitute a subset of the columns of A, and $P \in \mathbb{C}^{k \times n}$, a subset of whose columns makes up the $k \times k$ identity matrix, such that $\|P\|$ is small and $\|A - BP\| \sim \sigma_{k+1}$, where σ_{k+1} is the $(k+1)$st greatest singular value of A. We call B and P the *skeleton* and *projection matrices*, respectively.

Clearly, the ID compresses the column space of A; to compress the row space, simply apply the ID to A^{T}, which produces an analogous representation $A = \tilde{P}\tilde{B}$, where $\tilde{P} \in \mathbb{C}^{m \times k}$ and $\tilde{B} \in \mathbb{C}^{k \times n}$.

Definition 2.4. The indices corrresponding to the retained rows in the ID are called the *row* or *incoming skeletons*. Similarly, the indices corrresponding to the retained columns are called the *column* or *outgoing skeletons*.

Reasonably efficient schemes for constructing an ID exist [47, 131, 203]. By combining such schemes with methods for estimating the approximation error, we can compute an ID to any relative precision $\epsilon > 0$ by adaptively determining the required rank k [131]. This is the sense in which we will use the ID. While previous related work [88, 139] employed the deterministic $\mathcal{O}(kmn)$ algorithm of [47], we take advantage here of the latest compression technology based on random sampling, which typically requires only $\mathcal{O}(mn \log k + k^2 n)$ operations [131, 203].

2.2 Matrix compression

In this section, we describe our fast multilevel matrix compression algorithm, which forms the basis of our fast matrix algebra techniques. Thus, let $A \in \mathbb{C}^{N \times N}$ be a matrix with $p \times p$ blocks, structured in the sense of §2.1, and $\epsilon > 0$ a target relative precision. We first outline a one-level matrix compression scheme:

1. For $i = 1, \ldots, p$, use the ID to compress the row space of the ith off-diagonal block row to precision ϵ. Let L_i denote the corresponding row projection matrix.

2. Similarly, for $j = 1, \ldots, p$, use the ID to compress the column space of the jth off-diagonal block column to precision ϵ. Let R_j denote the corresponding column projection matrix.

3. Approximate the off-diagonal blocks of A by $A_{ij} \approx L_i S_{ij} R_j$ for $i \neq j$, where S_{ij} is the submatrix of A_{ij} defined by the row and column skeletons associated with L_i and R_j, respectively.

This yields precisely the matrix structure discussed in §2.1, following (2.1). The one-level scheme is illustrated graphically in Figure 2.3.

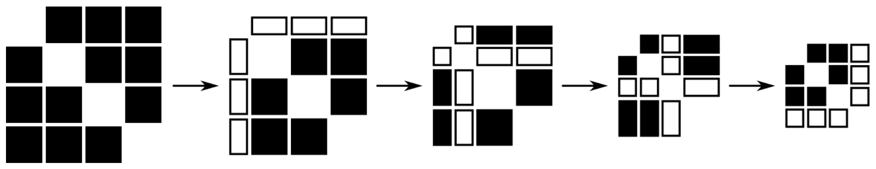

Figure 2.3: One level of matrix compression, obtained by sequentially compressing the off-diagonal block rows and columns. At each step, the matrix blocks whose row or column spaces are being compressed are highlighted in white.

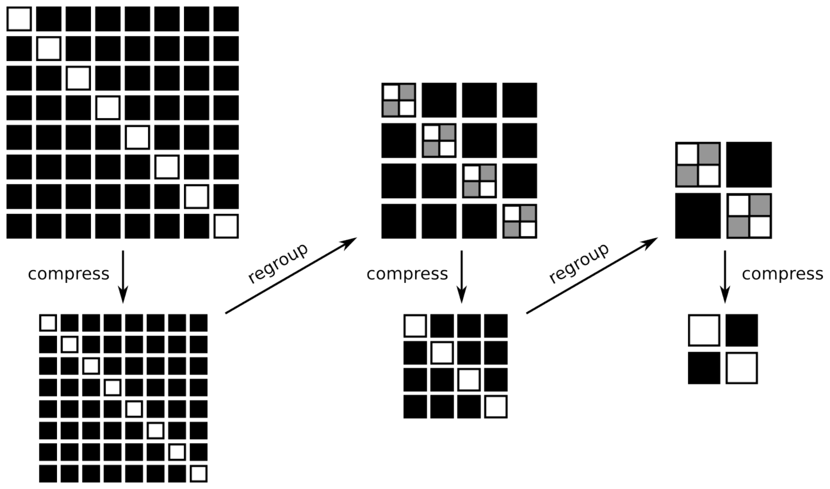

Figure 2.4: Multilevel matrix compression, comprising alternating steps of compression and regrouping via ascension of the index tree. The diagonal blocks (white and gray) are not compressed, but are instead extracted at each level of the tree; they are shown here only to illustrate the regrouping process.

The multilevel algorithm is now just a simple extension based on the observation that by ascending one level in the index tree and regrouping blocks accordingly, we can compress the skeleton matrix S in (2.2) in exactly the same form, leading to a procedure that we naturally call *recursive skeletonization* (Figure 2.4). This is because S is composed of submatrices of A and hence inherits many of its properties. The full algorithm may be specified as follows:

1. Starting at the leaves of the tree, extract the diagonal blocks and perform one level of compression of the off-diagonal blocks.

2. Move up one level in the tree and regroup the matrix blocks according to the tree structure. Terminate if the new level is the root; otherwise, extract the diagonal blocks, recompress the off-diagonal blocks, and repeat.

It is easy to see that any disturbance of the original matrix structure at a given level occurs only on the diagonal, which is immediately extracted at the next level. Thus, the off-diagonal blocks at each level are just submatrices of the original matrix. The result is a telescoping representation

$$A \approx D^{(1)} + L^{(1)} \left[D^{(2)} + L^{(2)} \left(\cdots D^{(\lambda)} + L^{(\lambda)} S R^{(\lambda)} \cdots \right) R^{(2)} \right] R^{(1)}, \tag{2.11}$$

where the superscript indexes the compression level $l = 1, \ldots, \lambda$.

Example 2.3. As a demonstration of the multilevel compression technique, consider the matrix defined by $N = 8192$ points uniformly distributed in the unit square, interacting via the 2D Laplace Green's function (2.5) and sorted according to a quadtree ordering. The sequence of skeletons remaining after each level of compression to $\epsilon = 10^{-3}$ is shown in Figure 2.5, from which we see that compression creates a sparsification of the sources which, in a geometric setting, leaves skeletons along the boundaries of each block.

The computational cost of the algorithm described so far is dominated by the fact that each step is global: that is, compressing the row or column space for each block

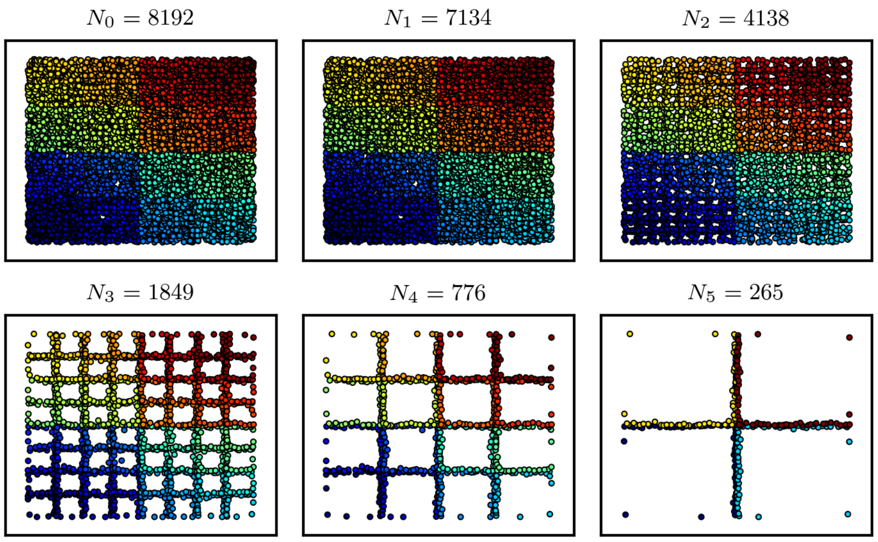

Figure 2.5: Sparsification by recursive skeletonization. Logarithmic interactions between $N = 8192$ points in the unit square are compressed to relative precision $\epsilon = 10^{-3}$ using a five-level quadtree-based scheme. At each level, the surviving skeletons are shown, colored by block index, with the total number of skeletons remaining given by N_l for compression level $l = 0, \ldots, 5$, where $l = 0$ denotes the original uncompressed system.

requires accessing all other blocks in that row or column. If no further knowledge of the matrix is available, this is indeed necessary. However, as noted in [47, 88, 139], this global work can often be replaced by a local one, resulting in considerable savings.

A sufficient condition for this acceleration is that the matrix correspond to evaluating a potential field for which some form of Green's identity holds. It is easiest to present the main ideas in the context of the Laplace equation. For this, consider Figure 2.6, which depicts a set of sources in the plane. We assume that block index i corresponds to the sources in the central square B. The ith off-diagonal block

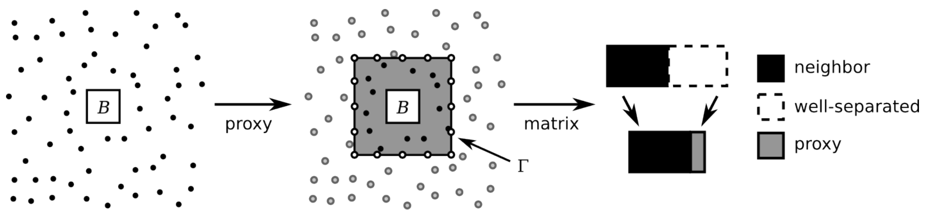

Figure 2.6: Accelerated compression using proxy surfaces. The field within a region B due to a distribution of exterior sources (left) can be decomposed into neighboring and well-separated contributions. By representing the latter via a proxy surface Γ (center), the matrix dimension to compress against for the block corresponding to B (right) can be reduced to the number of neighboring points plus a constant set of points on Γ, regardless of how many points lie beyond Γ.

row then corresponds to the interaction of all points outside B with all points inside B. We can separate this into contributions from the near neighbors of B, which are local, and the distant sources, which lie outside the near-neighbor domain, whose boundary is denoted by Γ. But any field induced by the distant sources induces a harmonic function inside Γ and can therefore be replicated by a charge density on Γ itself. Thus, rather than using the detailed structure of the distant points, the row (incoming) skeletons for B can be extracted by considering just the combination of the near-neighbor sources and an artifical set of charges placed on Γ, which we refer to as a *proxy surface*. Likewise, the column (outgoing) skeletons for B can be determined by considering only the near neighbors and the proxy surface. If the potential field is correct on the proxy surface, it will be correct at all more distant points (again via some variant of Green's identity).

The interaction rank between Γ and B is constant (depending on the desired precision) from standard multipole estimates [86, 93]. In summary, the number of points required to discretize Γ is constant, and the dimension of the matrix to compress against for the block corresponding to B is essentially just the number of points in the physically neighboring blocks.

Similar arguments hold for other kernels of potential theory including the heat, Helmholtz, Yukawa, Stokes, and elasticity kernels, though care must be taken for oscillatory problems which could require a combination of single and double layer potentials to avoid spurious resonances in the representation for the exterior.

2.3 Compressed matrix-vector multiplication

The compressed representation (2.11) admits an obvious fast algorithm for computing the matrix-vector product $y = Ax$: one simply applies the matrices in (2.11) from right to left. Like the FMM, this procedure can be thought of as occurring in two passes:

1. An upward pass, corresponding to the sequential application of the column projection matrices $R^{(l)}$, which hierarchically compress the input data x to the column (outgoing) skeleton subspace.

2. A downward pass, corresponding to the sequential application of the row projection matrices $L^{(l)}$, which hierarchically project onto the row (incoming) skeleton subspace and, from there, back onto the output elements y.

The same framework can be used to apply A^{T} as well, which is not easy to do via standard FMMs.

Remark 2.1. Essentially the same reasoning allows fast matrix-matrix multiplication, provided that the index trees of the two matrices are the same; for non-identical trees, the computation is substantially more involved.

2.4 Compressed matrix inversion

The representation (2.11) also permits a fast algorithm for the direct inversion of nonsingular matrices. The one level scheme was discussed in §2.1; in the multilevel scheme, the system $Sz = y$ in (2.3) is itself expanded in the same form, leading to the recursive embedding

$$\begin{bmatrix} D^{(1)} & L^{(1)} & & & & & \\ R^{(1)} & & -I & & & & \\ & -I & D^{(2)} & L^{(2)} & & & \\ & & R^{(2)} & \ddots & \ddots & & \\ & & & \ddots & D^{(\lambda)} & L^{(\lambda)} & \\ & & & & R^{(\lambda)} & & -I \\ & & & & & -I & S \end{bmatrix} \begin{bmatrix} x \\ y^{(1)} \\ z^{(1)} \\ \vdots \\ \vdots \\ y^{(\lambda)} \\ z^{(\lambda)} \end{bmatrix} = \begin{bmatrix} b \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ 0 \end{bmatrix}. \tag{2.12}$$

To understand the consequences of this sparse representation, it is instructive to study first the one-level embedding (2.3) in the special case that the row and column skeleton dimensions are identical for each block, so that the total skeleton dimension is $K \equiv K_{\mathrm{r}} = K_{\mathrm{c}}$. Then assuming that D is invertible, block elimination of x and y gives

$$(\Lambda + S)\, z = \Lambda R D^{-1} b,$$

where $\Lambda = (RD^{-1}L)^{-1} \in \mathbb{C}^{K \times K}$ is block diagonal. Back substitution then yields

$$x = \left[D^{-1} - D^{-1}L\Lambda RD^{-1} + D^{-1}L\Lambda\left(\Lambda + S\right)^{-1}\Lambda RD^{-1}\right] b.$$

In other words, the matrix inverse is

$$A^{-1} \approx \mathcal{D} + \mathcal{L}\mathcal{S}^{-1}\mathcal{R}, \tag{2.13}$$

where

$$\mathcal{D} = D^{-1} - D^{-1}L\Lambda RD^{-1} \in \mathbb{C}^{N \times N}$$

and

$$\mathcal{L} = D^{-1}L\Lambda \in \mathbb{C}^{N \times K}, \qquad \mathcal{R} = \Lambda RD^{-1} \in \mathbb{C}^{K \times N}$$

are all block diagonal, and

$$\mathcal{S} = \Lambda + S \in \mathbb{C}^{K \times K}$$

is dense, equal to the skeleton matrix S with its diagonal blocks filled in. Therefore, (2.13) is a compressed representation of A^{-1} with minimal fill-in over the original compressed representation (2.2) of A. In the multilevel setting, one carries out the above factorization recursively, since $\mathcal{S}$ can now be inverted in the same manner:

$$A^{-1} \approx \mathcal{D}^{(1)} + \mathcal{L}^{(1)}\left[\mathcal{D}^{(2)} + \mathcal{L}^{(2)}\left(\cdots \mathcal{D}^{(\lambda)} + \mathcal{L}^{(\lambda)}\mathcal{S}^{-1}\mathcal{R}^{(\lambda)} \cdots\right)\mathcal{R}^{(2)}\right]\mathcal{R}^{(1)}. \tag{2.14}$$

In the general case, this procedure will fail if D happens to be singular. Rather than construct our own stabilized block elimination scheme, where some sort of pivoting would be required, we simply use the sparse direct solver software UMFPACK [59, 60, 62, 63], supplying it with the sparse representation (2.12). Numerical results show that the performance is similar to that expected from (2.14).

2.5 Complexity analysis

We now analyze the complexity of the presented algorithm for a typical example: discretization of the integral operator (2.4), where the integral kernel has smoothness properties similar to that of the Green's function for the Laplace equation. We ignore quadrature issues and assume that we are given a matrix A acting on N points distributed randomly in a d-dimensional domain, sorted by an orthtree that uniformly subdivides until all block sizes are $\mathcal{O}(1)$. (In 1D, an orthtree is a binary tree; in 2D, it is a quadtree; and in 3D, it is an octree [see 168].)

For each compression level $l = 1, \ldots, \lambda$, with $l = 1$ being the finest, let p_l be the number of matrix blocks, and n_l and k_l the uncompressed and compressed block dimensions, respectively, assumed equal for all blocks and identical across rows and columns, for simplicity. We first make the following observations:

1. The total matrix dimension is $p_1 n_1 = N$, where $n_1 = \mathcal{O}(1)$, so $p_1 \sim N$.

2. Each subdivision increases the number of blocks by a factor of roughly 2^d, so $p_l \sim p_{l-1}/2^d \sim p_1/2^{d(l-1)}$. In particular, $p_\lambda = \mathcal{O}(1)$, so $\lambda \sim \log_{2^d} N = (1/d) \log N$.

3. The total number of points at level $l > 1$ is equal to the total number of skeletons at level $l - 1$, i.e., $p_l n_l = p_{l-1} k_{l-1}$, so $n_l \sim 2^d k_{l-1}$.

Furthermore, we note that k_l is on the order of the interaction rank between two adjacent blocks at level l, which can be analyzed by recursive subdivision of the source block to expose well-separated structures with respect to the target (Figure 2.7). Assuming only that the interaction between a source subregion separated by its

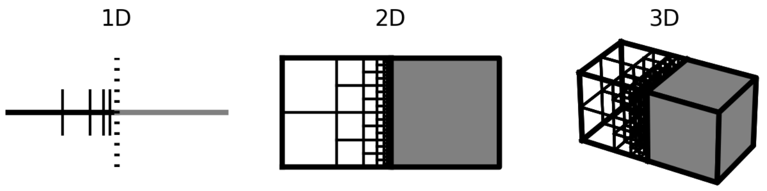

Figure 2.7: The interaction rank between two adjacent blocks can be calculated by recursively subdividing the source block (white) into well-separated subblocks with respect to the target (gray), each of which have constant rank.

own size from a target is of constant rank (to a fixed precision ϵ), we have

$$k_l \sim \sum_{l=1}^{\log_{2^d} n_l} 2^{(d-1)l} \sim \begin{cases} \log n_l & \text{if } d = 1, \\ n_l^{1-1/d} & \text{if } d > 1, \end{cases}$$

where, clearly, $n_l \sim (p_1/p_l) n_1 \sim 2^{d(l-1)} n_1$, so

$$k_l \sim \begin{cases} (l-1)\log 2 + \log n_1 & \text{if } d = 1, \\ 2^{(d-1)(l-1)} n_1^{1-1/d} & \text{if } d > 1. \end{cases}$$

Matrix compression

From §2.1, the cost of computing a rank-k ID of an $m \times n$ matrix is $\mathcal{O}(mn \log k + k^2 n)$. We will only consider the case of proxy compression, for which $m = \mathcal{O}(n_l)$ for a block at level l, so the total cost is

$$T_{\text{cm}} \sim \sum_{l=1}^{\lambda} p_l \left(n_l^2 \log k_l + k_l^2 n_l \right) \sim \begin{cases} N & \text{if } d = 1, \\ N^{3(1-1/d)} & \text{if } d > 1. \end{cases} \tag{2.15}$$

Matrix-vector multiplication

The cost of applying $D^{(l)}$ is $\mathcal{O}(p_l n_l^2)$, while that of applying $L^{(l)}$ or $R^{(l)}$ is $\mathcal{O}(p_l k_l n_l)$. Combined with the $\mathcal{O}((p_\lambda k_\lambda)^2)$ cost of applying S, the total cost is hence

$$T_{\mathrm{mv}} \sim \sum_{l=1}^{\lambda} p_l n_l \left(k_l + n_l\right) + \left(p_\lambda k_\lambda\right)^2 \sim \begin{cases} N & \text{if } d = 1, \\ N \log N & \text{if } d = 2, \\ N^{2(1-1/d)} & \text{if } d > 2. \end{cases} \tag{2.16}$$

Matrix factorization and inverse application

We turn now to the analysis of factorization using (2.14). At each level l, the cost of constructing D^{-1} and Λ is $\mathcal{O}(p_l n_l^3)$, after which forming $\mathcal{D}^{(l)}$, $\mathcal{L}^{(l)}$, and $\mathcal{R}^{(l)}$ all require $\mathcal{O}(p_l n_l^2)$ operations. At the final level, the cost of constructing and inverting $\mathcal{S}$ is $\mathcal{O}((p_\lambda k_\lambda)^3)$. Thus, the total cost is

$$T_{\mathrm{lu}} \sim \sum_{l=1}^{\lambda} p_l n_l^3 + \left(p_\lambda k_\lambda\right)^3,$$

which has complexity (2.15).

Finally, we note that the dimensions of $\mathcal{D}^{(l)}$, $\mathcal{L}^{(l)}$, $\mathcal{R}^{(l)}$, and $\mathcal{S}^{-1}$ are the same as those of $D^{(l)}$, $L^{(l)}$, $R^{(l)}$, and S, respectively. Thus, the total cost of applying the inverse, denoted by T_{sv}, has the same complexity as T_{mv}, namely (2.16).

Storage

An important issue for direct solvers, of course, is that of storage requirements. In the present setting, the relevant matrices are the compressed representations (2.11) and (2.14). Since the costs of storing and applying are the same here, the formulas are identical to those given above, with total complexity (2.16) for both.

2.6 Error analysis

Some simple error estimates can be derived for applying and inverting a compressed matrix. For this, let A be the original matrix and A_ϵ its compressed representation, constructed using the described algorithm such that

$$\frac{\|A - A_\epsilon\|}{\|A\|} \leq \epsilon$$

for some $\epsilon > 0$. Moreover, let x and b be vectors such that $Ax = b$.

Matrix-vector multiplication

Let $b_\epsilon \equiv A_\epsilon x$. Then

$$b - b_\epsilon = (A - A_\epsilon)\, x = (A - A_\epsilon)\, A^{-1} b,$$

so

$$\frac{\|b - b_\epsilon\|}{\|b\|} \leq \epsilon \, \|A\| \left\|A^{-1}\right\| = \epsilon \kappa\, (A)\,,$$

where $\kappa(A)$ is the condition number of A.

Matrix inverse application

Let $x_\epsilon \equiv A_\epsilon^{-1} b$. Then

$$x - x_\epsilon = \left(A^{-1} - A_\epsilon^{-1}\right) b = A_\epsilon^{-1}\, (A_\epsilon - A)\, A^{-1} b = A_\epsilon^{-1}\, (A_\epsilon - A)\, x,$$

so

$$\frac{\|x - x_\epsilon\|}{\|x\|} \leq \epsilon \, \|A\| \left\|A_\epsilon^{-1}\right\|.$$

From the identity

$$A_\epsilon^{-1} = A^{-1}\left[I + \left(A - A_\epsilon\right) A_\epsilon^{-1}\right],$$

we have

$$\left[1 - \epsilon\kappa\left(A\right)\right] \left\|A_\epsilon^{-1}\right\| \leq \left\|A^{-1}\right\|,$$

so if $\kappa(A) < 1/\epsilon$, then

$$\left\|A_\epsilon^{-1}\right\| \leq \frac{\left\|A^{-1}\right\|}{1 - \epsilon\kappa\left(A\right)}. \tag{2.17}$$

Hence,

$$\frac{\left\|x - x_\epsilon\right\|}{\left\|x\right\|} \leq \frac{\epsilon\kappa\left(A\right)}{1 - \epsilon\kappa\left(A\right)}.$$

In particular, if A is well-conditioned, e.g., A is the discretization of a second-kind integral equation, then $\kappa(A) = \mathcal{O}(1)$, so

$$\frac{\left\|x - x_\epsilon\right\|}{\left\|x\right\|} = \mathcal{O}\left(\epsilon\right).$$

2.7 Implementation

We implemented our algorithm in Fortran, using mostly Fortran 77 for compatibility and performance, but also with select Fortran 90 constructs for expressiveness. Our code contains the following primary functionalities:

1. matrix compression;
2. compressed matrix extraction (i.e., “uncompression”);
3. compressed matrix-vector multiplication; and

4. compressed matrix sparse inverse embedding.

The implementation also incorporates various features beyond those mentioned previously, including support for rectangular matrices, zero block sizes, and the capability to apply or factor matrix transposes or adjoints (i.e., conjugate transposes). Two versions of the code were produced, one for arithmetic over $\mathbb{R}$ and another over $\mathbb{C}$. Randomized ID software was generously provided by Prof. Mark Tygert (`http://cims.nyu.edu/~tygert/software.html`). The high-performance linear algebra library LAPACK (`http://www.netlib.org/lapack/`) was exploited whenever possible.

A key feature of our implementation is its generality. This is achieved through code modularity, by letting the user specify the matrix to be compressed along with its index tree. Thus, the same code can handle, for example, problems in both 2D and 3D involving various matrix kernels. It is worth noting therefore that our core algorithm has no knowledge of any geometry: all geometric considerations are synthesized by the user and translated to an equivalent algebraic structure (geometric tree-building codes are provided as auxiliary subroutines).

The code is currently being prepared for release under an open-source license.

2.8 Numerical results

In this section, we investigate the efficiency and flexibility of our algorithm by considering some representative examples. We begin with timing benchmarks for the Laplace and Helmholtz kernels in 2D and 3D, using the algorithm both as an FMM and as a direct solver, followed by applications in molecular electrostatics and multiple scattering.

All matrices were blocked using quadtrees in 2D and octrees in 3D, uniformly subdivided until all block sizes were $\mathcal{O}(1)$, but adaptively truncating empty boxes during the refinement process. We used proxy compression in all cases, with proxy surfaces constructed on the boundary of the supercell enclosing the neighbors of each block. We discretized all proxy surfaces using a constant number of points, independent of the matrix size N: for the Laplace equation, this constant depended only on the compression precision ϵ, while for the Helmholtz equation, it depended also on the wave frequency, chosen to be consistent with the Nyquist-Shannon sampling theorem. Computations were performed over $\mathbb{R}$ instead of $\mathbb{C}$, where possible. The algorithm was compiled using `gfortran` with optimization level `-O3`, and all experiments were performed on a 2.66 GHz processor with 8 GB of RAM in double precision.

In many instances, we compare our results against those obtained using LAPACK/ATLAS [6, 201] and the FMM [46, 48, 86, 163]. All FMM calculations are performed using the open-source FMMLIB package (see `http://www.cims.nyu.edu/cmcl/fmm3dlib/fmm3dlib.html`), which is a fairly efficient implementation but does not include the plane-wave optimizations of [46, 92] or the diagonal translation operators of [163, 164].

Generalized fast multipole method

We first studied the use of recursive skeletonization as a generalized FMM for the rapid computation of matrix-vector products.

The Laplace equation

We considered two point distributions in the plane: points on the unit circle and within the unit square, hereafter referred to as the 2D surface and volume cases, respectively. The surface case is typical of layer-potential evaluation when using boundary integral equations. Such a domain boundary can be described by a single parameter (such as arclength), so it is a 1D domain, hence the expected complexities from §2.5 correspond to $d = 1$, i.e., $\mathcal{O}(N)$ work for both matrix compression and application [cf. 78]. In the volume case, the dimension is $d = 2$, so the expected complexities are $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$ for compression and application, respectively.

For the 3D Laplace kernel (2.9), we considered surface and volume point geometries on the unit sphere and within the unit cube, respectively. The corresponding dimensions are $d = 2$ and $d = 3$; thus, the expected complexities for the 3D surface case are $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$ for compression and application, respectively, while those for the 3D volume case are $\mathcal{O}(N^2)$ and $\mathcal{O}(N^{4/3})$, respectively.

We present timing results for each case and compare with LAPACK/ATLAS and the FMM for a range of N at $\epsilon = 10^{-9}$. Detailed data are provided in Tables 2.1–2.4 and plotted in Figure 2.8. It is evident that our algorithm scales as predicted. Its performance in 2D is particularly strong: not only does our algorithm beat the $\mathcal{O}(N^2)$ uncompressed matrix-vector product for modest N, it is faster even than the $\mathcal{O}(N)$ FMM, at least after compression. In 3D, the same is true over the range of N tested, although the increase in asymptotic complexity would eventually make the scheme less competitive. In all cases studied, the compression time T_{cm} was larger than the time to apply the FMM by one (2D surface) to two (all other cases) orders of magnitude, while the compressed matrix-vector product time T_{mv} was consistently

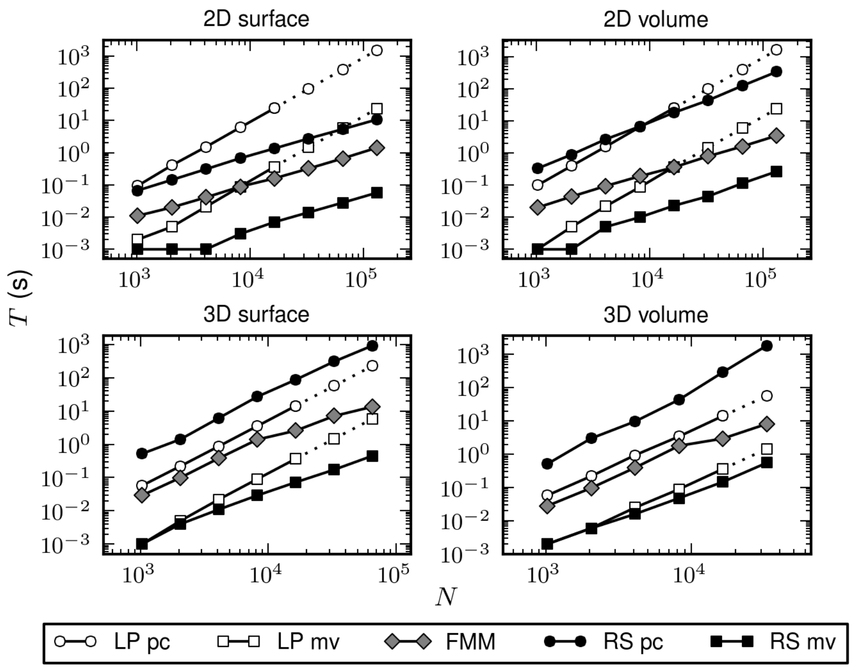

Figure 2.8: CPU times for applying the Laplace kernel in various cases using LAPACK/ATLAS (LP), the FMM, and recursive skeletonization (RS) as a function of the matrix size N. For LP and RS, the computation is split into two parts: precomputation (pc), for LP consisting of matrix formation and for RS of matrix compression, and matrix-vector multiplication (mv). The precision of the FMM and RS was set at $\epsilon = 10^{-9}$. Dotted lines indicate extrapolated data.

Table 2.1: Numerical results for applying the Laplace kernel in the 2D surface case at precision $\epsilon = 10^{-9}$: N, uncompressed matrix dimension; K_{r}, row skeleton dimension; K_{c}, column skeleton dimension; T_{cm}, matrix compression time (s); T_{mv}, matrix-vector multiplication time (s); E, relative error; M, required storage for compressed matrix (MB).

N	K_{r}	K_{c}	T_{cm}	T_{mv}	E	M
1024	94	94	6.7E−2	1.0E−3	3.1E−8	8.5E−1
2048	105	104	1.4E−1	1.0E−3	4.5E−8	1.7E+0
4096	113	114	3.1E−1	1.0E−3	1.1E−7	3.4E+0
8192	123	123	6.7E−1	3.0E−3	4.4E−7	6.4E+0
16384	133	134	1.4E+0	7.0E−3	4.0E−7	1.3E+1
32768	142	142	2.7E+0	1.4E−2	4.7E−7	2.5E+1
65536	150	149	5.4E+0	2.8E−2	9.4E−7	5.0E+1
131072	159	158	1.1E+1	5.7E−2	9.8E−7	1.0E+2

smaller by the same amount. Thus, our algorithm also shows promise as a fast iterative solver for problems requiring more than $\sim$ 10–100 iterations. Furthermore, we note the effectiveness of compression: for $N = 131072$, the storage requirement for the uncompressed matrix is 137 GB, whereas that for the compressed representations are only 100 MB and 1.2 GB in the 2D surface and volume cases, respectively; at a lower precision of $\epsilon = 10^{-3}$, these become just 40 and 180 MB. Finally, to provide some intuition about the behavior of the algorithm as a function of precision, we report the following timings for the 2D volume case with $N = 131072$: for $\epsilon = 10^{-3}$,

Table 2.2: Numerical results for applying the Laplace kernel in the 2D volume case at precision $\epsilon = 10^{-9}$; notation as in Table 2.1.

N	K_r	K_c	T_cm	T_mv	E	M
1024	299	298	3.3E−1	1.0E−3	3.6E−10	2.9E+0
2048	403	405	8.9E−1	1.0E−3	3.7E−10	7.1E+0
4096	570	570	2.7E+0	5.0E−3	1.0E−09	1.8E+1
8192	795	793	6.8E+0	1.0E−2	8.8E−10	4.3E+1
16384	1092	1091	1.8E+1	2.3E−2	7.7E−10	1.0E+2
32768	1506	1505	4.4E+1	4.5E−2	1.0E−09	2.3E+2
65536	2099	2101	1.3E+2	1.1E−1	1.1E−09	5.3E+2
131072	2904	2903	3.4E+2	2.7E−1	1.1E−09	1.2E+3

Table 2.3: Numerical results for applying the Laplace kernel in the 3D surface case at precision $\epsilon = 10^{-9}$; notation as in Table 2.1.

N	K_r	K_c	T_cm	T_mv	E	M
1024	967	967	5.2E−1	1.0E−3	1.0E−11	7.7E+0
2048	1531	1532	1.4E+0	4.0E−3	1.8E−10	2.2E+1
4096	2298	2295	6.1E+0	1.1E−2	1.4E−10	6.2E+1
8192	3438	3426	2.7E+1	2.9E−2	1.2E−10	1.7E+2
16384	4962	4950	8.7E+1	7.2E−2	3.0E−10	4.2E+2
32768	6974	6987	3.1E+2	1.7E−1	4.3E−10	9.9E+2
65536	9899	9925	9.2E+2	4.5E−1	7.7E−10	2.3E+3

Table 2.4: Numerical results for applying the Laplace kernel in the 3D volume case at precision $\epsilon = 10^{-9}$; notation as in Table 2.1.

N	K_r	K_c	T_cm	T_mv	E	M
1024	1024	1024	5.1E−1	2.0E−3	9.3E−16	8.4E+0
2048	1969	1969	3.0E+0	6.0E−3	5.6E−12	3.2E+1
4096	3285	3287	9.7E+0	1.6E−2	6.8E−11	9.8E+1
8192	5360	5362	4.4E+1	4.8E−2	6.3E−11	3.0E+2
16384	8703	8707	2.9E+2	1.5E−1	5.7E−11	9.3E+2
32768	14015	14013	1.9E+3	5.5E−1	7.5E−11	2.9E+3

$T_\mathrm{cm} = 41$ s and $T_\mathrm{mv} = 0.09$ s; for $\epsilon = 10^{-6}$, $T_\mathrm{cm} = 161$ s and $T_\mathrm{mv} = 0.18$ s; and for $\epsilon = 10^{-9}$, $T_\mathrm{cm} = 339$ s and $T_\mathrm{mv} = 0.27$ s.

The Helmholtz equation

We next considered the 2D and 3D Helmholtz kernels

$$G(\boldsymbol{r}, \boldsymbol{s}) = \frac{\imath}{4} H_0^{(1)}(k|\boldsymbol{r} - \boldsymbol{s}|) \tag{2.18}$$

and

$$G(\boldsymbol{r}, \boldsymbol{s}) = \frac{e^{\imath k|\boldsymbol{r} - \boldsymbol{s}|}}{4\pi|\boldsymbol{r} - \boldsymbol{s}|}, \tag{2.19}$$

respectively, where $H_0^{(1)}$ is the zeroth order Hankel function of the first kind and k is the wavenumber. We used the same representative geometries as for the Laplace equation. The size of each domain Ω in wavelengths was given by

$$\omega \equiv \frac{k}{2\pi} \operatorname{diam}(\Omega).$$

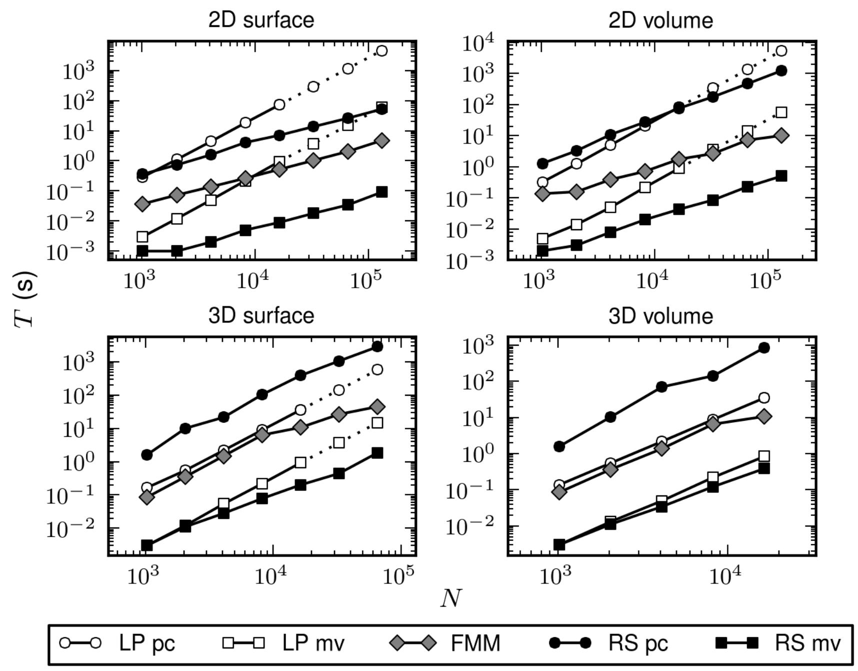

Figure 2.9: CPU times for applying the Helmholtz kernel in various cases at low frequency ($\omega = 10$ in 2D and $\omega = 5$ in 3D) using LAPACK/ATLAS, the FMM, and recursive skeletonization at precision $\epsilon = 10^{-9}$; notation as in Figure 2.8.

Timing results against LAPACK/ATLAS and the FMM at low frequency ($\omega = 10$ in 2D and $\omega = 5$ in 3D) with $\epsilon = 10^{-9}$ are shown in Figure 2.9, with detailed data presented in Tables 2.5–2.8. In this regime, the performance is very similar to that for the Laplace equation, as both kernels are essentially non-oscillatory. However, as discussed in [139], the compression efficiency deteriorates as ω increases, due to the growing ranks of the matrix blocks. In the high-frequency regime, there is no gain in asymptotic efficiency. Still, numerical results suggest that the algorithm remains viable up to $\omega \sim 200$ in 2D and $\omega \sim 10$ in 3D. In all cases, the CPU times and storage requirements are larger than those for the Laplace equation by a factor of about two

Table 2.5: Numerical results for applying the Helmholtz kernel in the 2D surface case with frequency $\omega = 10$ at precision $\epsilon = 10^{-9}$; notation as in Table 2.1.

N	K_{r}	K_{c}	T_{cm}	T_{mv}	E	M
1024	155	154	3.6E−1	1.0E−3	2.9E−9	2.1E+0
2048	166	166	7.3E−1	1.0E−3	2.7E−9	4.1E+0
4096	173	173	1.6E+0	2.0E−3	3.0E−9	7.6E+0
8192	184	182	4.0E+0	5.0E−3	3.5E−9	1.4E+1
16384	192	190	7.0E+0	9.0E−3	5.3E−9	2.7E+1
32768	201	201	1.4E+1	1.8E−2	4.9E−9	5.3E+1
65536	208	208	2.6E+1	3.4E−2	5.4E−9	1.0E+2
131072	219	215	5.2E+1	9.1E−2	8.4E−9	2.0E+2

since all computations are performed over $\mathbb{C}$ instead of $\mathbb{R}$; in 2D, there is also the additional expense of computing $H_0^{(1)}$.

Fast direct solver

We then studied the behavior of our algorithm as a fast direct solver. More specifically, we considered the interior Dirichlet problem for the Laplace and Helmholtz equations in 2D and 3D, recast as a second-kind boundary integral equation using the double-layer representation (2.8). Contour integrals in 2D were discretized using the trapezoidal rule, while surface integrals in 3D were discretized using piecewise constant unknowns on flat triangles (see §1.2). In 2D, the Laplace double-layer kernel

Table 2.6: Numerical results for applying the Helmholtz kernel in the 2D volume case with frequency $\omega = 10$ at precision $\epsilon = 10^{-9}$; notation as in Table 2.1.

N	$K_{\rm r}$	$K_{\rm c}$	$T_{\rm cm}$	$T_{\rm mv}$	E	M
1024	320	321	1.3E+0	2.0E−3	1.6E−9	6.5E+0
2048	426	425	3.3E+0	3.0E−3	3.4E−9	1.6E+1
4096	603	603	1.1E+1	8.0E−3	3.4E−9	4.0E+1
8192	829	833	2.7E+1	2.0E−2	6.5E−9	9.4E+1
16384	1134	1136	7.6E+1	4.3E−2	8.1E−9	2.2E+2
32768	1566	1573	1.8E+2	8.6E−2	1.1E−8	5.0E+2
65536	2200	2197	4.6E+2	2.3E−1	9.3E−9	1.1E+3
131072	3017	3016	1.2E+3	5.0E−1	1.5E−8	2.5E+3

Table 2.7: Numerical results for applying the Helmholtz kernel in the 3D surface case with frequency $\omega = 5$ at precision $\epsilon = 10^{-9}$; notation as in Table 2.1.

N	$K_{\rm r}$	$K_{\rm c}$	$T_{\rm cm}$	$T_{\rm mv}$	E	M
1024	1024	1024	1.6E+0	3.0E−3	1.3E−15	1.7E+1
2048	1958	1958	1.0E+1	1.1E−2	2.8E−10	6.3E+1
4096	2864	2874	2.2E+1	2.8E−2	7.4E−10	1.6E+2
8192	4071	4092	1.0E+2	7.8E−2	4.4E−09	4.3E+2
16384	5658	5660	3.9E+2	2.0E−1	6.9E−09	1.0E+3
32768	7742	7742	1.1E+3	4.5E−1	1.5E−08	2.3E+3
65536	10664	10693	2.8E+3	1.9E+0	3.7E−08	5.0E+3

Table 2.8: Numerical results for applying the Helmholtz kernel in the 3D volume case with frequency $\omega = 5$ at precision $\epsilon = 10^{-9}$; notation as in Table 2.1.

N	K_{r}	K_{c}	T_{cm}	T_{mv}	E	M
1024	1024	1024	1.6E+0	3.0E−3	1.6E−15	1.7E+1
2048	2044	2044	1.0E+1	1.1E−2	4.7E−11	6.7E+1
4096	3603	3603	7.0E+1	3.4E−2	1.8E−10	2.2E+2
8192	5685	5691	1.4E+2	1.2E−1	2.3E−09	6.4E+2
16384	9079	9072	8.5E+2	3.8E−1	9.9E−10	2.0E+3

has a removable singularity:

$$\lim_{\boldsymbol{r} \to \boldsymbol{s}} \frac{\partial G}{\partial \nu_{\boldsymbol{s}}} (\boldsymbol{r}, \boldsymbol{s}) = \frac{1}{2} \kappa (\boldsymbol{s}) \quad \text{on } \partial\Omega,$$

where κ is the signed curvature. Sparse inverses were computed and applied using UMFPACK (`http://www.cise.ufl.edu/research/sparse/umfpack/`). In each case, we took as boundary data the field generated by an exterior point source; the error was assessed by comparing the field evaluated using the numerical solution via (2.8) against the exact field due to that source at an interior location. As a benchmark, we also solved each system directly using LAPACK/ATLAS, as well as iteratively using GMRES with matrix-vector products accelerated by the FMM.

The Laplace equation

For the Laplace equation (2.7), the Green's function G in (2.10) is given by (2.5) in 2D and (2.9) in 3D. As model geometries, we considered an ellipse with aspect ratio $r = 2$ (semi-major and -minor axes $a = 2$ and $b = 1$, respectively) in 2D and the

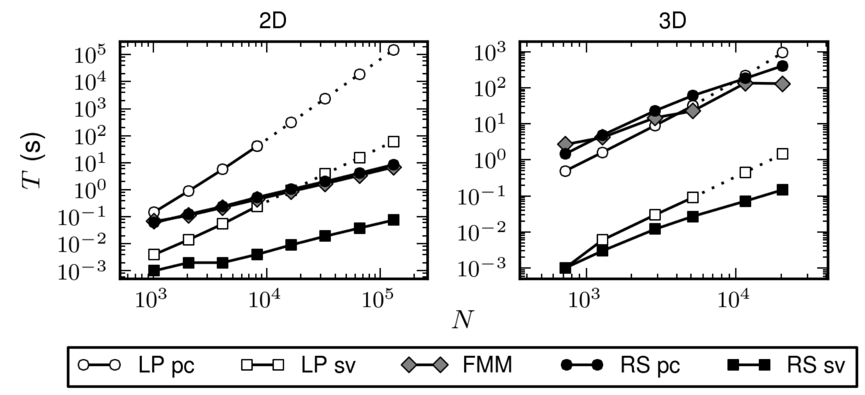

Figure 2.10: CPU times for solving the Laplace equation in various cases using LAPACK/ATLAS (LP), FMM/GMRES (FMM), and recursive skeletonization (RS) as a function of the system size N. For LP and RS, the computation is split into two parts: precomputation (pc), for LP consisting of matrix formation and factorization, and for RS of matrix compression and factorization; and system solution (sv), consisting of matrix inverse application. The precision of the FMM and RS was set at $\epsilon = 10^{-9}$ in 2D and $\epsilon = 10^{-6}$ in 3D. Dotted lines indicate extrapolated data.

unit sphere in 3D; these boundaries have dimensions $d = 1$ and $d = 2$, respectively. Timing results are shown in Figure 2.10, with detailed data given in Tables 2.9 and 2.10; the precision was set to $\epsilon = 10^{-9}$ in 2D and $\epsilon = 10^{-6}$ in 3D.

In 2D, the solver has linear complexity and is exceptionally fast, handily beating the $\mathcal{O}(N^3)$ uncompressed direct solver, but also coming very close to the $\mathcal{O}(N)$ FMM/GMRES iterative solver: at $N = 131072$, for example, the total solution time for the recursive skeletonization algorithm was $T_{\mathrm{RS}} = 8.5$ s, while that for FMM/GMRES was $T_{\mathrm{FMM}} = 6.9$ s over $n_{\mathrm{FMM}} = 7$ iterations. It is worth emphasizing, however, that our solver is direct and possesses obvious advantages over

Table 2.9: Numerical results for solving the Laplace equation in 2D at precision $\epsilon = 10^{-9}$: N, uncompressed matrix dimension; K_{r}, row skeleton dimension; K_{c}, column skeleton dimension; T_{cm}, matrix compression time (s); T_{lu}, sparse matrix factorization time (s); T_{sv}, inverse application time (s); E, relative error; M, required storage for compressed matrix inverse (MB).

N	K_{r}	K_{c}	T_{cm}	T_{lu}	T_{sv}	E	M
1024	30	30	3.4E−2	2.5E−2	1.0E−3	9.0E−11	1.6E+0
2048	29	30	7.0E−2	5.1E−2	2.0E−3	9.0E−12	3.3E+0
4096	30	30	1.4E−1	9.8E−2	2.0E−3	8.3E−11	6.8E+0
8192	30	31	3.0E−1	2.1E−1	4.0E−3	1.6E−10	1.4E+1
16384	31	31	5.5E−1	4.5E−1	9.0E−3	5.5E−10	2.8E+1
32768	30	30	1.1E+0	8.5E−1	1.9E−2	4.9E−12	5.6E+1
65536	30	30	2.3E+0	1.8E+0	3.8E−2	1.1E−11	1.1E+2
131072	29	29	4.6E+0	3.7E+0	7.5E−2	8.5E−11	2.2E+2

FMM/GMRES, as described in §1.4; in particular, the algorithm is relatively insensitive to geometric ill-conditioning. Indeed, the direct solver edged out FMM/GMRES even at modest aspect ratios (for $N = 8192$ at $\epsilon = 10^{-12}$ with $r = 8$: $T_{\mathrm{RS}} = 0.76$ s, $T_{\mathrm{FMM}} = 0.98$ s, $n_{\mathrm{FMM}} = 15$); for larger r, the effect was even more pronounced ($r = 512$: $T_{\mathrm{RS}} = 2.5$ s, $T_{\mathrm{FMM}} = 3.9$ s, $n_{\mathrm{FMM}} = 44$). Furthermore, the compressed inverse representation allows subsequent solves to be performed extremely rapidly; for instance, at $N = 131072$, the solve time was just $T_{\mathrm{sv}} = 0.07$ s, i.e., $T_{\mathrm{FMM}}/T_{\mathrm{sv}} \sim 100$. Thus, our algorithm is especially efficient in regimes where T_{sv} dominates [see, e.g.,

Table 2.10: Numerical results for solving the Laplace equation in 3D at precision $\epsilon = 10^{-6}$; notation as in Table 2.9.

N	K_{r}	K_{c}	T_{cm}	T_{lu}	T_{sv}	E	M
720	628	669	1.3E+0	1.1E−1	1.0E−3	9.8E−5	4.6E+0
1280	890	913	4.5E+0	4.0E−1	3.0E−3	5.5E−5	1.1E+1
2880	1393	1400	2.1E+1	2.0E+0	1.2E−2	2.4E−5	5.5E+1
5120	1886	1850	5.5E+1	5.4E+0	2.7E−2	1.3E−5	1.3E+2
11520	2750	2754	1.6E+2	1.7E+1	7.2E−2	6.2E−6	3.5E+2
20480	3592	3551	3.7E+2	4.1E+1	1.5E−1	3.3E−6	6.9E+2

138]. Finally, we remark that although direct methods are traditionally very memory-intensive, our algorithm appears quite manageable in this regard: at $N = 131072$, the storage required for the compressed inverse was only 106 MB for $\epsilon = 10^{-3}$, 172 MB for $\epsilon = 10^{-6}$, and 222 MB for $\epsilon = 10^{-9}$.

In 3D, our solver has complexity $\mathcal{O}(N^{3/2})$. Hence, asymptotics dictate that it must eventually lose. However, our results demonstrate that even up to $N = 20480$, the solver remains surprisingly competitive. For example, at $N = 20480$, $T_{\mathrm{RS}} = 409$ s, while $T_{\mathrm{FMM}} = 131$ s with $n_{\mathrm{FMM}} = 3$; at $\epsilon = 10^{-9}$, the difference is almost negligible: $T_{\mathrm{RS}} = 850$ s, $T_{\mathrm{FMM}} = 839$ s, $n_{\mathrm{FMM}} = 5$. Thus, our algorithm remains a viable alternative for medium-scale problems. It is important to note that the *solve time* advantage is not lost even for large N, since the cost of each solve is only $\mathcal{O}(N \log N)$. In fact, the advantage is, remarkably, even more striking than in 2D: at $N = 20480$, $T_{\mathrm{FMM}}/T_{\mathrm{sv}} \sim 1000$; for $\epsilon = 10^{-9}$, $T_{\mathrm{FMM}}/T_{\mathrm{sv}} \sim 2500$.

The Helmholtz equation

We then considered the Helmholtz equation

$$\left(\Delta + k^2\right) u = 0 \quad \text{in } \Omega, \qquad u = f \quad \text{on } \partial\Omega,$$

recast as a boundary integral equation (2.10) with Green's function (2.18) in 2D and (2.19) in 3D. This representation does not work for all frequencies, encountering spurious discrete resonances for k beyond a critical value. We ignore that (well-understood) issue here and assume that the integral equation we obtain is invertible, though the method itself does not falter in such cases, as discussed in [139]. We used the same geometries and precisions as for the Laplace equation. In 2D, the double-layer kernel is weakly singular, so we modified the trapezoidal rule with tenth-order endpoint corrections [119]. The frequency was set to $\omega = 10$ in 2D and $\omega = 3.18$ in 3D.

Timing results are shown in Figure 2.11, with details given in Tables 2.11 and 2.12. The data are very similar to that for the Laplace equation, but with the direct solver actually beating FMM/GMRES in 2D. This is because the number of iterations required for FMM/GMRES scales as $n_{\text{FMM}} = \mathcal{O}(\omega)$. Interestingly, even at moderately high frequencies, where we would expect the direct solver to break down as previously discussed, the performance drop is more than compensated for by the increase in the number n_{FMM} of iterations. In short, we find that recursive skeletonization is faster than FMM/GMRES at low to moderate frequencies, provided that the memory requirement is not excessive. The story is much the same in 3D and the compressed solve time is again very fast: at $N = 20480$, $T_{\text{FMM}}/T_{\text{sv}} \sim 2000$.

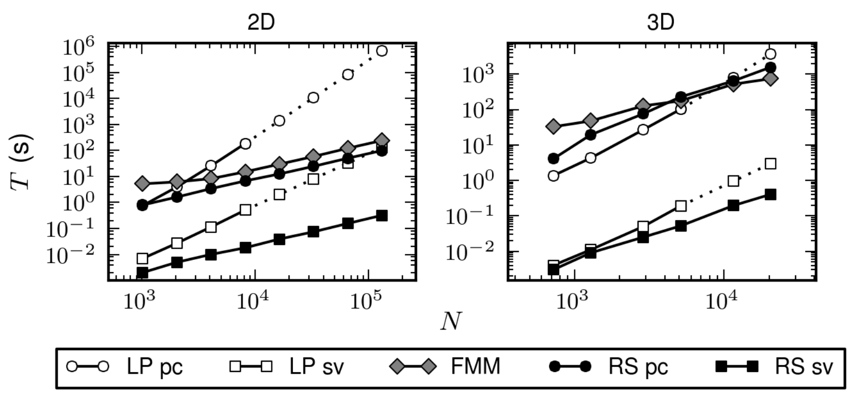

Figure 2.11: CPU times for solving the Helmholtz equation in various cases at low frequency ($\omega = 10$ in 2D and $\omega = 3.18$ in 3D) using LAPACK/ATLAS, FMM/GMRES, and recursive skeletonization; notation as in Figure 2.10. The precision was set to $\epsilon = 10^{-9}$ in 2D and $\epsilon = 10^{-6}$ in 3D.

Table 2.11: Numerical results for solving the Helmholtz equation in 2D with frequency $\omega = 10$ at precision $\epsilon = 10^{-9}$; notation as in Table 2.9.

N	K_{r}	K_{c}	T_{cm}	T_{lu}	T_{sv}	E	M
1024	90	91	6.6E−1	1.4E−1	2.0E−3	6.7E−9	7.6E+0
2048	96	94	1.4E+0	2.3E−1	5.0E−3	5.3E−9	1.4E+1
4096	95	96	2.9E+0	4.5E−1	1.0E−2	9.4E−9	2.9E+1
8192	98	98	5.8E+0	8.6E−1	1.8E−2	9.7E−9	5.5E+1
16384	99	98	1.1E+1	1.8E+0	3.9E−2	4.8E−9	1.1E+2
32768	100	100	2.1E+1	3.5E+0	7.4E−2	6.4E−9	2.2E+2
65536	100	101	4.3E+1	7.2E+0	1.6E−1	1.3E−8	4.3E+2
131072	99	99	8.3E+1	1.5E+1	3.2E−1	4.6E−8	8.5E+2

Table 2.12: Numerical results for solving the Helmholtz equation in 3D with frequency $\omega = 3.18$ at precision $\epsilon = 10^{-6}$; notation as in Table 2.9.

N	K_{r}	K_{c}	T_{cm}	T_{lu}	T_{sv}	E	M
720	720	720	3.7E+0	3.3E−1	3.0E−3	3.0E−3	1.0E+1
1280	1088	1236	1.8E+1	1.2E+0	9.0E−3	2.0E−3	3.2E+1
2880	1653	1786	7.0E+1	5.2E+0	2.5E−2	1.1E−3	1.0E+2
5120	2188	2329	2.2E+2	1.0E+1	5.2E−2	6.7E−4	2.2E+2
11520	3042	3225	5.9E+2	5.2E+1	1.9E−1	3.3E−4	8.5E+2
20480	3867	4034	1.4E+3	1.3E+2	4.0E−1	1.9E−4	1.6E+3

Molecular electrostatics

An important application area for our solver is molecular electrostatics (viz. Chapter 1). A simplified model for this involves consideration of a molecular surface Σ, dividing $\mathbb{R}^3$ into Ω_0 and Ω_1, denoting the solvent and the molecule, respectively. We also suppose that the molecule has interior charges of strengths q_i at locations $x_i \in \Omega_1$ for $i = 1, \ldots, n$. The electrostatic potential φ (ignoring salt effects in the solvent) then satisfies the Poisson equation:

$$-\nabla \cdot (\varepsilon \nabla \varphi) = \sum_{i=1}^{n} q_i \delta \left(\boldsymbol{r} - \boldsymbol{r}_i \right),$$

where $\varepsilon(\boldsymbol{r}) = \varepsilon_i$ in Ω_i is a piecewise constant dielectric (cf. (1.5)).

We decompose the solution as $\varphi \equiv \varphi_s + \varphi_p$, where

$$\varphi_s \left(\boldsymbol{r} \right) \equiv \frac{1}{\varepsilon_1} \sum_{i=1}^{n} q_i G \left(\boldsymbol{r}, \boldsymbol{r}_i \right), \tag{2.20}$$

is the potential due to the sources, with G given by (2.9), and φ_p is a piecewise harmonic potential satisfying the jump conditions

$$[\varphi_p] = 0, \qquad \left[\varepsilon \frac{\partial \varphi_p}{\partial \nu}\right] = -\left[\varepsilon \frac{\partial \varphi_s}{\partial \nu}\right] \qquad \text{on } \Sigma.$$

We can write φ_p, called the polarization response, as a single-layer potential

$$\varphi_p(\boldsymbol{r}) \equiv \int_\Sigma G(\boldsymbol{r}, \boldsymbol{s})\, \sigma(\boldsymbol{s})\, dS_{\boldsymbol{s}}, \tag{2.21}$$

which yields the boundary integral equation

$$\frac{1}{2}\sigma(\boldsymbol{r}) + \lambda \int_\Sigma \frac{\partial G}{\partial \nu_{\boldsymbol{r}}}(\boldsymbol{r}, \boldsymbol{s})\, \sigma(\boldsymbol{s})\, dS_{\boldsymbol{s}} = -\lambda \frac{\partial \varphi_s}{\partial \nu}(\boldsymbol{r}),$$

where $\lambda = (\varepsilon_1 - \varepsilon_2)/(\varepsilon_1 + \varepsilon_2)$, in terms of the polarization charge σ.

We generated molecular surfaces for a short segment of DNA [67, PDB ID: 1BNA] using MSMS [169] with a probe radius of 1.4 Å and vertex densities of 1.0 and 3.0 Å^{-2}, resulting in meshes consisting of $N = 7612$ and 19752 triangles, respectively. For each surface, strengths were assigned to each of $n = 486$ heavy atoms using Amber partial charges [37] through PDB2PQR [66], and the resulting system solved with $\varepsilon_0 = 80$ and $\varepsilon_1 = 20$ at precision $\epsilon = 10^{-3}$. The resulting potential φ on Σ for the $N = 19752$ case is shown in Figure 2.12, with numerical data for both cases given in Table 2.13. For both systems, the net solution time was larger than that using FMM/GMRES by a factor of about $\sim$ 10–25. However, the inverse application time was very small: $T_{\text{sv}} = 0.03$ and 0.08 s for $N = 7612$ and 19752, respectively. Thus, when sampling the electrostatic potential for many different charge configurations $\{q_i\}$, as is common in computational chemistry [26], our solver can provide a speedup provided that the number of such configurations is greater than $\sim$ 10–25. We remark that the evaluation of φ at fixed points, e.g., on Σ, via (2.20) and (2.21) can also be accelerated using

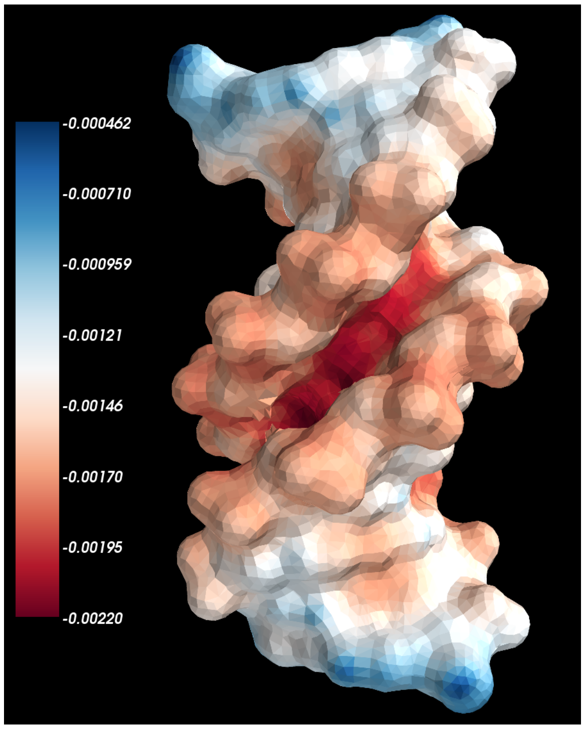

Figure 2.12: Surface potential of DNA in units of the elementary charge, computed using recursive skeletonization to precision $\epsilon = 10^{-3}$. The molecular surface was discretized using $N = 19752$ triangles.

Table 2.13: Numerical results for the molecular electrostatics example at precision $\epsilon = 10^{-3}$: N, number of triangles; T_{FMM}, time for FMM/GMRES solve (s); T_{cm}, matrix compression time (s); T_{lu}, sparse matrix factorization time (s); T_{sv}, inverse application time (s); E, relative error.

N	T_{FMM}	T_{cm}	T_{lu}	T_{sv}	E
7612	1.3E+1	1.5E+2	3.5E+0	2.7E−2	9.3E−2
19752	2.7E+1	5.8E+2	1.3E+1	8.3E−2	8.3E−2

our algorithm in its capacity as a generalized FMM; the computation time for this would be similar to T_{sv}.

Remark 2.2. The Poisson equation is clearly too simplistic a model for this system; following the discussion of §1.1, a more appropriate model is the Poisson-Boltzmann equation, but even this may be not be sufficient due to strong salt effects, which are well-known to be important for DNA stability [171].

Multiple scattering

As a final example, we show how direct solvers can be combined with FMM-based iterative methods to great effect in the context of a multiple scattering problem. For this, let Ω_i, for $i = 1, \ldots, p$, be a collection of acoustic scatterers in 2D with boundaries Σ_i. Then the acoustic pressure field satisfies

$$\left(\Delta + k^2\right) u = 0 \quad \text{in } \mathbb{R}^2 \setminus \bigcup_{i=1}^{p} \Omega_i. \tag{2.22}$$

Assuming that the obstacles are sound-hard, we must compute the exterior solution that satisfies the Neumann boundary condition

$$\frac{\partial u}{\partial \nu} = 0 \quad \text{on } \bigcup_{i=1}^{p} \Sigma_i.$$

If $u \equiv u_i + u_s$, where u_i is an incoming field satisfying (2.22), then the scattered field u_s also satisfies (2.22) with boundary condition

$$\frac{\partial u_s}{\partial \nu} = -\frac{\partial u_i}{\partial \nu} \quad \text{on } \bigcup_{i=1}^{p} \Sigma_i$$

and the Sommerfeld radiation condition [180]

$$\lim_{|\boldsymbol{r}| \to \infty} \sqrt{|\boldsymbol{r}|} \left(\frac{\partial}{\partial |\boldsymbol{r}|} - \imath k \right) u_s\left(\boldsymbol{r}\right) = 0.$$

We write the scattered field as $u_s \equiv \sum_{i=1}^{p} u_{s,i}$, where

$$u_{s,i}\left(\boldsymbol{r}\right) \equiv \int_{\Sigma_i} G\left(\boldsymbol{r},\boldsymbol{s}\right)\sigma_i\left(\boldsymbol{s}\right) dS_{\boldsymbol{s}}$$

with G the single-layer kernel (2.18). Imposing the boundary condition then yields the second-kind integral equation

$$-\frac{1}{2}\sigma_i + \sum_{j=1}^{p} K_{ij}\sigma_j = -\left.\frac{\partial u_i}{\partial \nu}\right|_{\Sigma_i} \quad \text{on } \Sigma_i, \quad \text{for } i = 1,\ldots,p,$$

where

$$K_{ij}\sigma_j\left(\boldsymbol{r}\right) = \int_{\Sigma_j} \frac{\partial G}{\partial \nu_{\boldsymbol{r}}}\left(\boldsymbol{r},\boldsymbol{s}\right)\sigma_j\left(\boldsymbol{s}\right) dS_{\boldsymbol{s}} \quad \text{on } \Sigma_i.$$

In operator notation, the linear system therefore has the form

$$\sum_{i=1}^{p} A_{ij}\sigma_j = -\left.\frac{\partial u_i}{\partial \nu}\right|_{\Sigma_i}, \quad A_{ij} = \begin{cases} -\frac{1}{2}I + K_{ii} & \text{if } i = j, \\ K_{ij} & \text{if } i \neq j. \end{cases}$$

We solve this system using FMM/GMRES with the block diagonal preconditioner

$$P^{-1} \equiv \begin{bmatrix} A_{11}^{-1} & & \\ & \ddots & \\ & & A_{pp}^{-1} \end{bmatrix},$$

where each A_{ii}^{-1} is computed using recursive skeletonization; observe that A_{ii}^{-1} is precisely the solution operator for scatterer Ω_i in isolation. The question is whether this preconditioner will significantly reduce the iteration count required, which is typically quite high for problems of appreciable size. As a test, we embedded two identical scatterers, each described in polar coordinates by the radial function

$$r\left(\theta\right) \equiv \frac{2+\cos\left(3\theta\right)}{6},$$

where θ is the polar angle; each scatterer is smooth, though somewhat complicated, and was taken to be ten wavelengths in size. We assumed an incoming field given by the plane wave $u_i \equiv e^{\imath k r_2}$, where $\boldsymbol{r} \equiv (r_1, r_2)$, and considered the scattering problem at various horizontal separation distances δ between the centers of the scatterers. Each configuration was solved both with and without the preconditioner P^{-1} to precision $\epsilon = 10^{-6}$; each scatterer was discretized using a corrected trapezoidal rule [119] with $N = 1024$ points.

The intensities of the resulting pressure fields are shown in Figure 2.13, with numerical data given in Table 2.14. It is clear that the preconditioner is highly effective: following a precomputation time of 0.76 s to construct P^{-1}, which is amortized over all solves, the number of iterations required was decreased from $n_{\mathrm{FMM}} \sim 700$ to just $n_{\mathrm{RS}} \sim 10$ for each case. As expected, more iterations were necessary for smaller δ, though the difference was not too dramatic. The ratio of the total solution time required for all solves was ~ 60 for the unpreconditioned versus the preconditioned method.

2.9 Summary

We have presented a multilevel matrix compression algorithm and demonstrated its efficiency at accelerating matrix-vector multiplication and matrix inversion in a variety of contexts. The matrix structure required is fairly general and relies only on the assumption that the matrix have low-rank off-diagonal blocks. As a fast direct solver for the boundary integral equations of potential theory, we found our algorithm to be competitive with fast iterative methods based on FMM/GMRES in both 2D and 3D, provided that the integral equation kernel is not too oscillatory, and that the

Table 2.14: Numerical results for the multiple scattering example, consisting of six configurations with various separation distances δ/λ, relative to the wavelength, between the centers of two identical scatterers, solved to precision $\epsilon = 10^{-6}$: T_{FMM}, time for FMM/GMRES solve (s); T_{RS}, time for preconditioned FMM/GMRES solve (s); n_{FMM}, number of iterations required for FMM/GMRES; n_{RS}, number of iterations required for preconditioned FMM/GMRES; E, relative error; T_{cm}, matrix compression time for scatterer (s); T_{lu}, sparse matrix factorization time for scatterer (s).

δ/λ	T_{FMM}	T_{RS}	n_{FMM}	n_{RS}	E
30.0	7.9E+1	8.9E−1	697	8	1.3E−8
20.0	7.7E+1	1.1E+0	694	10	5.8E−9
15.0	8.0E+1	1.2E+0	695	11	6.9E−9
12.5	7.9E+1	1.3E+0	695	12	7.8E−9
11.0	7.9E+1	1.4E+0	704	14	8.7E−9
10.5	8.0E+1	1.5E+0	706	14	1.3E−8
T_{cm}		6.6E−1			
T_{lu}		9.3E−2			
total	4.7E+2	8.1E+0			

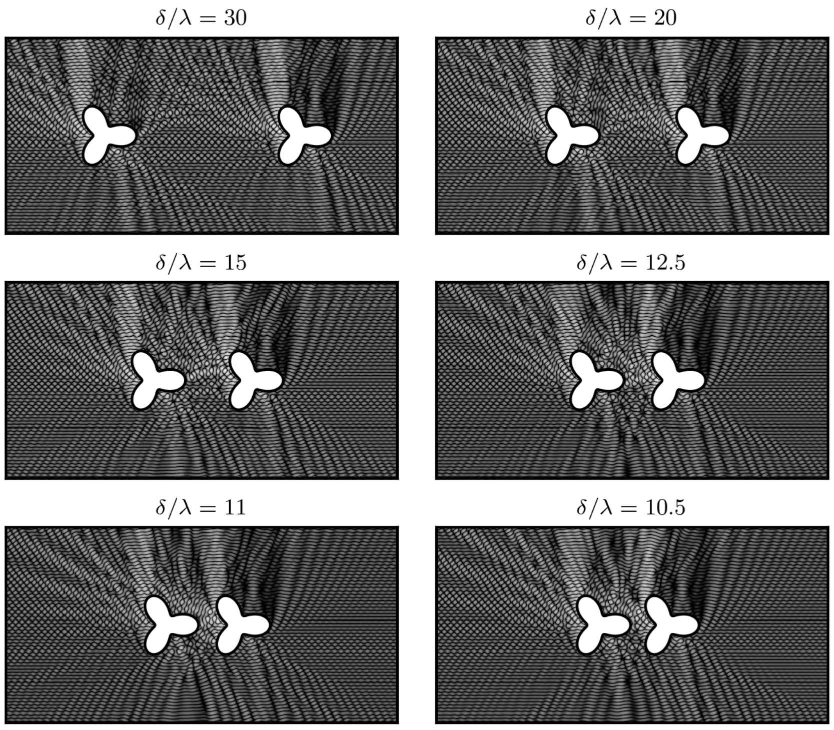

Figure 2.13: Instantaneous intensity $[\Re(u)]^2$ of the pressure field in response to an incoming vertical plane wave for various scattering geometries characterized by the separation distance δ/λ in wavelengths between the centers of two identical scatterers.

system size is not too large in 3D. In such cases, the total solution times for both methods were very comparable. Our solver has clear advantages, however, for problems with ill-conditioned matrices (in which case the number of iterations required by FMM/GMRES can increase dramatically), or those involving multiple right-hand sides (in which case the cost of matrix compression and factorization can be amortized). The latter category includes the use of our solver as a preconditioner for iterative methods, which we expect to be quite promising, particularly for large-scale 3D problems with complex geometries (see §3.2).

A principal limitation of our current approach is the growth in skeleton sizes in 3D or higher, which prohibits the scheme from achieving optimal $\mathcal{O}(N)$ or near-optimal $\mathcal{O}(N \log N)$ complexity. The memory requirement is especially prohibitive. New methods to curtail this growth are an active area of research in several groups. We offer some brief insights in this direction in §5.4.

Although we have presently analyzed our algorithm only for non-oscillatory or low-frequency integral kernels, regardless of whether they represent boundary or volume integral operators, we note that our complexity estimates also apply for high-frequency *volume* integral equations, due to the compression afforded by Green's theorem in moving data from the volume to the boundary. Thus, for instance, the costs of our solver for high-frequency volume wave scattering in 2D are $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$ for precomputation and solution, respectively. For related work, see [44, 202].

Finally, although all numerical results have presently been reported for a single processor, the algorithm is naturally parallelizable: many computations are organized in a block-sweep structure, where each block can be processed independently.

3 Extensions of the direct solver

In the previous chapter, we introduced a fast direct solver for non-oscillatory integral equations and showed its use in a number of important and practical settings. Here, we discuss various extensions to the solver that further expand its range of applicability. These include rather simple observations that enable it to

1. handle integral operators with a more complex structure;
2. effectively precondition problems that continue to be difficult for both direct and iterative schemes, particularly in quasi-2D domains; and
3. efficiently accomodate optimization and design problems characterized by local geometric perturbations.

We also present analysis that allows the solver to be used in unmodified form for overdetermined least squares problems. This is achieved via a semi-direct approach based on sparse QR factorization, and has complexities similar to those reported in §2.5 but slightly worse due to fill-in of the unitary matrix. The complexity is optimal, however, in the special case of partial charge fitting, which can be relevant in computational chemistry, particularly for force field development. Furthermore, we describe the construction of a compression-based fast multipole method (FMM) based on essentially the same tools and algebraic structure developed so far. Early steps in this direction were taken by Martinsson and Rokhlin, who considered the 1D case [141]; we extend this here to higher dimensions and moreover embed the algorithm in a purely linear algebraic framework. As with our direct solver, this scheme is therefore kernel-independent and should prove especially useful for interaction kernels that are difficult to treat analytically.

The flexibility of the direct solver to support such extensions can be attributed to its general algebraic structure: some operations that are unwieldy in the FMM context become quite natural when viewed in terms of matrix compression and fast matrix algebra. This is one advantage of a purely numerical linear algebraic approach. However, it should be noted that this does not come without a price as we must now forfeit certain optimizations based on analysis and geometry, e.g., the diagonal forms of [48, 92].

3.1 Block and composite operators

The fast direct solver of Chapter 2 was designed for second-kind integral operators of the form $L = I + K$, where I is the identity and K is compact, e.g., a layer potential operator. But what if L has a more complex structure? In this section, we consider two extensions: block operators, where each block has the form $I + K$, and composite operators, where L is the composition of multiple operators.

The case of block operators is quite straightforward. For this, let L have blocks L_{ij} for $i, j = 1, \ldots, n$, where each L_{ij} is compressible in the sense of §2.1. Since each L_{ij} can have full rank, L as a whole can contain full-rank off-diagonal blocks and so cannot in general be compressed using our algorithm. However, this can be remedied simply by stacking, e.g., the first row of each block together, then the second row, and so forth, and similarly with the columns. That is, we create a new block structure with blocks $\tilde{L}_{kl}$, where $(\tilde{L}_{kl})_{ij} = (L_{ij})_{kl}$, i.e., the (i, j) element of $\tilde{L}_{kl}$ is the (k, l)

element of L_{ij}. It is easy to see that this interleaved form can have full-rank blocks only on the diagonal and can therefore can be handled by our scheme.

For composite operators, let $L \equiv L_1 \cdots L_n$. Then the system

$$Lx = L_1 \cdots L_n x = b$$

can be rewritten as

$$\begin{bmatrix} & & & L_1 \\ & & L_2 & -I \\ & \iddots & \iddots & \\ L_n & -I & & \end{bmatrix} \begin{bmatrix} x \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} b \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

where $y_{i+1} = L_{n-i} y_i$ with $y_0 \equiv x$. This is precisely of the block form just considered, but with additional sparsity, and so, too, can be treated by our solver.

Remark 3.1. An attractive application of the above methodology is the solution of the exterior Neumann problem for the Helmholtz equation via the solution representation

$$u \equiv (S_k + D_k S_0)\,\sigma,$$

where S_k and D_k are the single- and double-layer potentials, respectively, for the Helmholtz kernel with wavenumber k. This is a modification of the standard single-layer potential representation (cf. Example 1.2) that has the advantage of being free of spurious resonances while leading to a well-conditioned second-kind boundary integral equation [51]:

$$-\frac{1}{2}\sigma + (S'_k + D'_k S_0)\,\sigma = \frac{\partial u}{\partial \nu},$$

which, in operator notation, is just the block system

$$\begin{bmatrix} -\frac{1}{2}I + S'_k & D'_k \\ S_0 & -I \end{bmatrix} \begin{bmatrix} \sigma \\ \mu \end{bmatrix} = \begin{bmatrix} \partial u / \partial \nu \\ 0 \end{bmatrix},$$

where μ is an auxiliary variable.

3.2 Approximate inverse preconditioning

The complexity estimates of §2.5 show that our direct solver has optimal $\mathcal{O}(N)$ cost only on quasi-1D domains, e.g., boundary integral equations in 2D or axisymmetric boundary integral equations in 3D. In theory, this places a severe restriction on the class of problems to which it can be effectively applied. Surprisingly, however, the empirical data suggest that in many situations, the actual cost can be much smaller than predicted, especially if we request only low to moderate precision.

As an example, consider points distributed uniformly on the 2D volume and 3D surface geometries of §2.8, i.e., in the unit square and on the unit sphere, respectively, interacting via the 2D and 3D Laplace Green's functions. These are both quasi-2D domains; therefore, the theoretical cost of compression is $\mathcal{O}(N^{3/2})$ for both, where N is the number of points. However, as the data clearly show (Figure 3.1), the *empirical* cost is only $\mathcal{O}(N^{\alpha})$ with $\alpha = 1.1$–1.3 in 2D and 1.3–1.6 in 3D, depending on the precision ϵ; full scalings are given in Table 3.1. Obviously, these cannot hold as $N \to \infty$, though surprisingly they are retained quite robustly even up to fairly large problem sizes: for instance, in 2D, the $\mathcal{O}(N^{1.1})$ scaling for $\epsilon = 10^{-3}$ holds even up to $N \sim 10^{6}$. Furthermore, while the theoretical complexities do depend logarithmically on ϵ (see §1.3), this appears only as part of the prefactor and so cannot change the ultimate scaling on N. Thus, our observations constitute a rather remarkable discovery, wherein the various constants involved conspire to produce an improved empirical scaling for the algorithm that is stable over a wide range of practical problem sizes.

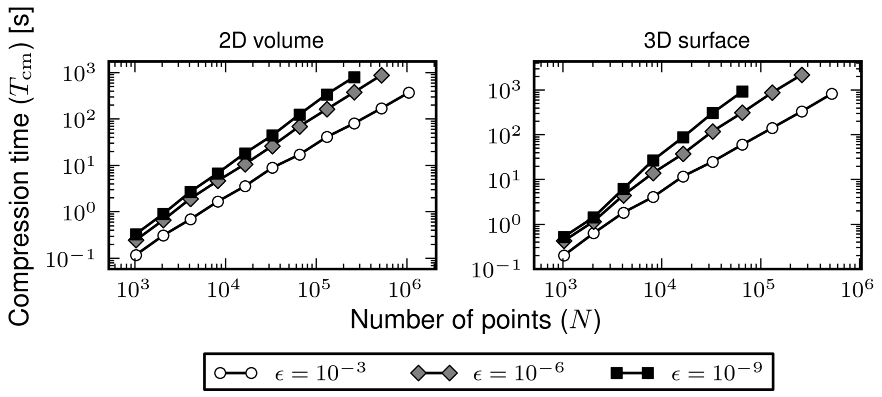

Figure 3.1: Compression time T_{cm} of Laplace interactions in quasi-2D geometries as a function of the matrix size N at various relative precisions ϵ.

Table 3.1: Empirical compression complexities $\mathcal{O}(N^{\alpha})$ of the direct solver in quasi-2D at various relative precisions ϵ, where N is the matrix size.

	2D volume			3D surface		
ϵ	10^{-3}	10^{-6}	10^{-9}	10^{-3}	10^{-6}	10^{-9}
α	1.1	1.2	1.3	1.3	1.4	1.6

Remark 3.2. The same $\mathcal{O}(N^{1.3})$ complexity in 3D was recently observed by Wei, Peng, and Lee in [200], where they used very similar methods to study electromagnetic wave scattering at low precision.

Note 3.3. The $\mathcal{O}(N^{1.6})$ scaling in 3D for $\epsilon = 10^{-9}$ is worse than the predicted $\mathcal{O}(N^{1.5})$ and simply indicates that we are not yet in the asymptotic regime.

This result immediately brings to mind the notion of using the direct solver as a low-precision preconditioner for ill-conditioned systems in quasi-2D, which represent one class of problems for which neither iterative nor direct solvers are currently satisfactory—the former due to the high number of iterations required, and the latter due to their 2D nature. In contrast, preconditioning offers a viable solution by dramatically reducing the number of iterations at a cost of only, e.g., $\mathcal{O}(N^{1.1})$.

To examine the process in more detail, we assume that we use the iterative method GMRES [167], which, recall, can be thought of as a polynomial minimization problem [see, e.g., 193]. Specifically, the residual r_k at the kth iteration satisfies

$$\|r_k\| = \min_{p \in \mathcal{P}_k} \|p(A) r_0\|, \tag{3.1}$$

where A is the system matrix, and $\mathcal{P}_k$ is the space of all polynomials p of degree at most k such that $p(0) = 1$. As a preconditioner, we use the approximate inverse A_ϵ^{-1}, where $A_\epsilon \equiv A + E$ with $\|E\|/\|A\| \leq \epsilon$. Hence, taking $A_\epsilon^{-1} A$ as the system matrix in (3.1), we have

$$\|r_k\| = \min_{p \in \mathcal{P}_k} \left\|p\left(A_\epsilon^{-1} A\right) r_0\right\| \leq \left\|\left(A_\epsilon^{-1} E\right)^k r_0\right\| \leq \left\|A_\epsilon^{-1} E\right\|^k \|r_0\|$$

on letting $p(z) \equiv (1 - z)^k$. But from (2.17), if $\kappa(A) < 1/\epsilon$ then

$$\left\|A_\epsilon^{-1}\right\| \leq \frac{\|A^{-1}\|}{1 - \epsilon\kappa(A)},$$

so

$$\left\|A_\epsilon^{-1} E\right\| \leq \left\|A_\epsilon^{-1}\right\| \|E\| \leq \frac{\epsilon\kappa(A)}{1 - \epsilon\kappa(A)},$$

i.e.,

$$\frac{\|r_k\|}{\|r_0\|} \leq \left[\frac{\epsilon\kappa(A)}{1 - \epsilon\kappa(A)}\right]^k. \tag{3.2}$$

In other words, if $\kappa(A)$ is not too large, then we can expect a factor improvement of roughly $\mathcal{O}(\epsilon)$ on each iteration, and therefore a total iteration count of $\mathcal{O}(\log_{\epsilon_0} \epsilon)$, where ϵ_0 is the overall target precision.

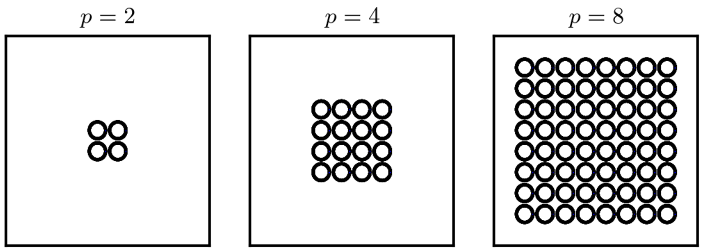

Figure 3.2: A space-filling $p \times p$ grid of disks in 2D. The largest problem considered was $p = 16$, corresponding to $N = 131072$ points.

As a first test, we considered the exterior Neumann problem for the Helmholtz equation about a regular $p \times p$ grid of sound-hard disks in the plane (Figure 3.2). This is a *space-filling* quasi-2D geometry. Each disk has radius 0.4 and is enclosed in a 1×1 cell, which we used to tile the grid. The solution was written as a single-layer potential on a collection of densities defined on the boundary of each disk, following the multiple scattering example of §2.8. We discretized each disk with 512 points; the single-layer integrals were correspondingly discretized using a low-order 'punctured' (i.e., without the singular self-contribution) trapezoidal rule for simplicity. The sampling rate was fixed at about 64 points per wavelength. Larger problems were accessed by increasing the grid dimension p; since the size of each disk is fixed, increasing p results in a larger domain characterized by a higher frequency content ω. Note that this is different from the methodology in §2.8, where we fixed the frequency and let N vary. In each case, accuracy of the solution was assessed by imposing that the exact solution be the field due to point sources in the interior of the disks, against which we compared the computed numerical solution.

Timing results are shown in Figure 3.3 for various grid sizes at compression precision $\epsilon = 10^{-3}$. Detailed data, including for other precisions, are presented in Table

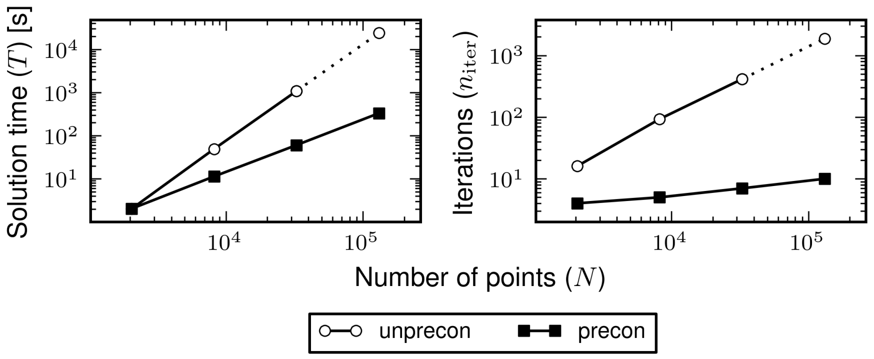

Figure 3.3: CPU times for solving the space-filling 2D example using FMM/GMRES as a function of the system size N, both with (precon) and without (unprecon) approximate inverse preconditioning. The number of iterations required in each case is also shown. The inverse was computed to precision $\epsilon = 10^{-3}$; the overall target precision was $\epsilon_0 = 10^{-12}$. Dotted lines indicate extrapolated data.

3.2. All cases were solved to an overall target precision of $\epsilon_0 = 10^{-12}$. It is immediate that preconditioning is extremely effective, achieving an empirical scaling of just $\mathcal{O}(N^{1.2})$ at $\epsilon = 10^{-3}$ versus $\mathcal{O}(N^{2.2})$ for the unpreconditioned method. For comparison, the expected asymptotic complexity is $\mathcal{O}(N^{3/2} \log N)$ since the number of iterations required is proportional to the size of the domain in wavelengths, i.e., $n_{\mathrm{iter}} = \mathcal{O}(\omega) = \mathcal{O}(N^{1/2})$. Note the weak dependence of n_{iter} on N for the preconditioned solver, which is even more pronounced at higher compression precisions. This leads to very impressive speedups with respect to the unpreconditioned scheme.

For another example of this technique, we next solved the interior Dirichlet problem for the Helmholtz equation on the unit sphere (Figure 3.4), formulated as a second-kind boundary integral equation using a double-layer potential representation and discretized using collocation on flat triangles (as described in §1.2). The sampling

Table 3.2: Numerical results for solving the space-filling 2D example using FMM/GMRES with approximate inverse preconditioning: p, grid dimension; N, system size; ω, grid diameter in wavelengths; ϵ, compression precision; T, total solution time (s); n_{iter}, number of iterations; T_{dir}, direct solver computation time (s); T_{iter}, iterative solver computation time (s); E, relative error.

p	N	ω	ϵ	T	n_{iter}	T_{dir}	T_{iter}	E
2	2048	9.0	10^{-3}	2.0E+0	4	1.5E+0	5.1E−1	7.6E−4
			10^{-6}	2.6E+0	2	2.3E+0	3.2E−1	
			10^{-9}	3.5E+1	1	3.3E+0	2.1E−1	
4	8192	18.0	10^{-3}	1.1E+1	5	8.6E+0	2.7E+0	4.3E−4
			10^{-6}	1.5E+1	2	1.4E+1	1.3E+0	
			10^{-9}	2.1E+1	1	2.0E+1	9.1E−1	
8	32768	36.0	10^{-3}	6.1E+1	7	4.4E+1	1.6E+1	8.8E−5
			10^{-6}	8.0E+1	2	7.4E+1	5.7E+0	
			10^{-9}	1.2E+2	1	1.1E+2	3.9E+0	
16	131072	72.0	10^{-3}	3.3E+2	10	2.3E+2	1.0E+2	7.5E−5
			10^{-6}	4.1E+2	3	3.8E+2	3.2E+1	
			10^{-9}	6.7E+2	1	6.5E+2	1.9E+1	

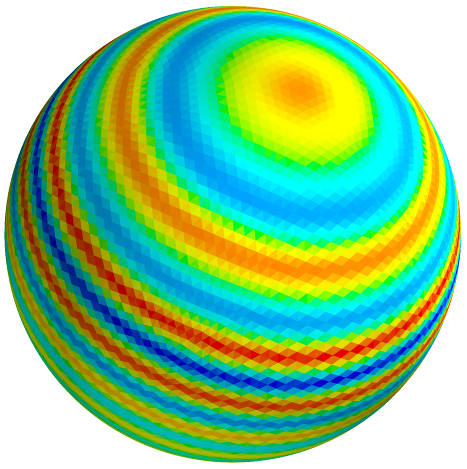

Figure 3.4: Real part of a Helmholtz potential on the unit sphere for a discretization of $N = 20480$ triangles. The sphere is approximately 10 wavelengths in size.

rate was set at 8 triangles per wavelength; the compression and target precisions were $\epsilon = 10^{-3}$ and $\epsilon_0 = 10^{-12}$, respectively. As with the previous case, we assessed the accuracy by comparison against an exact solution.

Timing results are shown in Figure 3.5, from which we again see a very clear advantage for the preconditioned solver. Here, it is evident that the complexities of the two methods are quite similar, but preconditioning gains a substantial constant speedup of about 10 or so throughout. Detailed numerical data are given in Table 3.3.

Remark 3.4. In addition to providing acceleration for certain ill-conditioned problems, the same methods can also be used in general to trade memory for speed, assuming that an appreciable number of iterations are necessary.

Remark 3.5. The present term "approximate inverse preconditioner" is reminiscent of work on sparse preconditioners, e.g., incomplete LU and other factorizations, which

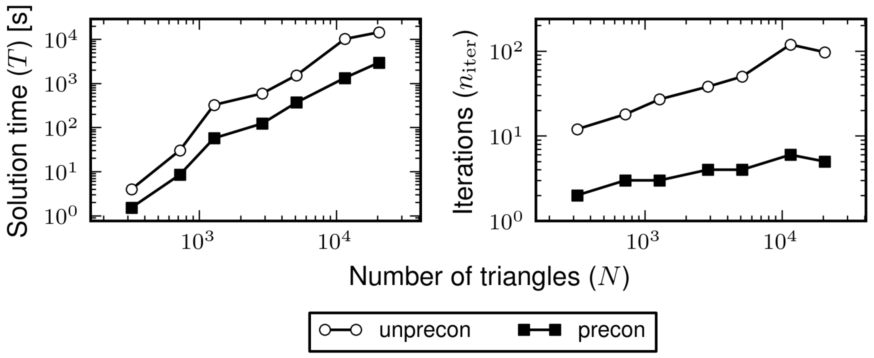

Figure 3.5: CPU times for solving the 3D surface Helmholtz example using FMM/GMRES at compression and target precisions $\epsilon = 10^{-3}$ and $\epsilon_0 = 10^{-12}$, respectively; notation as in Figure 3.3.

Table 3.3: Numerical results for solving the 3D Helmholtz example using FMM/GMRES with approximate inverse preconditioning at compression and target precisions $\epsilon = 10^{-3}$ and $\epsilon_0 = 10^{-12}$, respectively; notation as in Table 3.2.

N	ω	T	n_{iter}	T_{dir}	T_{iter}	E
320	1.3	1.5E+0	2	6.9E−1	8.3E−1	3.8E−3
720	1.9	8.5E+0	3	2.8E+0	5.7E+0	3.7E−3
1280	2.5	5.7E+1	3	1.7E+1	4.0E+1	3.8E−3
2880	3.8	1.2E+2	4	5.9E+1	6.4E+1	5.1E−3
5120	5.0	3.7E+2	4	2.4E+2	1.3E+2	2.3E−3
11520	7.6	1.3E+3	6	7.8E+2	5.5E+2	4.5E−3
20480	10.0	3.0E+3	5	2.2E+3	7.7E+2	4.4E−3

typically impose some sparsity constraint on the inverse approximation [166]. Our approach is quite different: we use fully dense inverses that are only data-sparse and hence cheap to construct and apply. Furthermore, we do not approximate the inverse directly but rather produce the inverse of an approximation of the original matrix. Since our methods heavily exploit a specific matrix structure, we can achieve economical approximations that are generally far more accurate for matrices having that structure [see 96].

Remark 3.6. Direct solvers have also recently been used for preconditioning in domain decomposition settings [104], in which the same observations made here apply.

Other candidates for inverse preconditioning include the 2D volume integral formulations of the Lippmann-Schwinger equation for inhomogeneous wave scattering and the variable-coefficient Poisson equation (1.1) for, e.g., variable-dielectric electrostatics and compressible fluid flow. The iteration count for the former has recently been demonstrated to scale *quadratically* with the frequency [176], i.e., $n_{\mathrm{iter}} = \mathcal{O}(\omega^2)$; in contrast, the condition number is just $\kappa(A) = \mathcal{O}(\omega)$. Therefore, according to our earlier analysis, cf. (3.2), we can take only a relatively low compression precision and still achieve a significant reduction in n_{iter} and hence in the solution time. For the Poisson equation, n_{iter} increases with the magnitude of the coefficient gradient, which determines the extent to which the corresponding integral equation is first-kind (though this may be ameliorated by density scaling [32]). This is especially important in the context of molecular electrostatics, where steep dielectric gradients are common near the molecular boundary. Some additional tools, however, are needed to properly treat volume integral operators [e.g., 82]; numerical results will be reported at a later date.

3.3 Local geometric perturbations

As we have seen throughout, many integral equations of practical interest possess a geometric interpretation, in the sense that the action of the integral operator describes interactions between physical elements in space. A natural question then to ask is what can be said if we start moving some of those elements around. This is particularly relevant in computational engineering and design, where one often wishes to optimize a geometric structure for some quantity of interest, e.g., binding affinity in drug discovery or flow patterns in microfluidics. This is typically achieved via local geometric perturbations, wherein a small part of the structure is modified at a time, which, in the linear algebraic context, correspond to low-rank matrix updates. As mentioned briefly in the introduction to Chapter 2, such problems can be solved very efficiently with direct methods. The purpose of this section is therefore to outline how this can be done, and mostly follows the presentation of [88] but with additional comments as appropriate for clarity.

To be precise, let Ω be some reference geometry, discretized, for example, as a collection of triangles T_i, and let K be an integral operator acting on Ω, e.g.,

$$K\left(T_i, T_j\right) = -\frac{1}{2}\delta_{ij} + \int_{T_j} \frac{\partial G}{\partial \nu_{\boldsymbol{s}}}\left(\boldsymbol{c}_i, \boldsymbol{s}\right) dS_{\boldsymbol{s}}$$

for the interior Dirichlet problem, where

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j \end{cases}$$

is the Kronecker delta and $\boldsymbol{c}_i$ is the centroid of triangle T_i. Furthermore, let the discretized integral equation be denoted by $Ax = b$, where $A \in \mathbb{C}^{N \times N}$ with $A_{ij} =$

$K(T_i, T_j)$. We assume that A is nonsingular and that we have precomputed A^{-1}, for instance, using our fast direct solver.

Suppose now that we wish to add p triangles $T_{N+1}, \ldots, T_{N+p}$ to Ω, where typically $p \ll N$. The augmented system then takes the form

$$\begin{bmatrix} A & B_+ \\ C_+ & D_+ \end{bmatrix} \begin{bmatrix} x \\ x_+ \end{bmatrix} = \begin{bmatrix} b \\ b_+ \end{bmatrix},$$

where $B_+ \in \mathbb{C}^{N \times p}$, $C_+ \in \mathbb{C}^{p \times N}$, and $D_+ \in \mathbb{C}^{p \times p}$, with

$$(B_+)_{ij} = K\left(T_i, T_{N+j}\right), \quad (C_+)_{ij} = K\left(T_{N+i}, T_j\right), \quad (D_+)_{ij} = K\left(T_{N+i}, T_{N+j}\right).$$

Note that the original system is embedded as part of this block structure. The solution can be obtained by forming the Schur complement system for x_+:

$$\left(I - D_+^{-1} C_+ A^{-1} B_+\right) x_+ = D_+^{-1}\left(b_+ - C_+ A^{-1} b\right),$$

which has dimension only p and hence is inexpensive to solve, after which we can recover x as

$$x = A^{-1}\left(b - B_+ b_+\right).$$

The total cost is hence $\mathcal{O}(C_{\text{sv}} p + N p^2 + p^3)$, where C_{sv} is the cost of applying A^{-1}.

The case of deleting triangles is similar and can be reduced to that of adding 'anti-triangles' by requiring that the added densities exactly cancel out their original counterparts. For this, let $T_{j_1}, \ldots, T_{j_q}$ be the triangles to be removed. We add new triangles $T_{N+1}, \ldots, T_{N+q}$ at the same spatial locations as the T_{j_i}, then form the system

$$\begin{bmatrix} A & B_- \\ C_- & I \end{bmatrix} \begin{bmatrix} x \\ x_- \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix},$$

where $B_- \in \mathbb{C}^{N\times q}$ and $C_- \in \mathbb{C}^{q\times N}$, with

$$(B_-)_{ik} = \begin{cases} 0 & \text{if } i = j_k, \\ A_{ij_k} & \text{if } i \neq j_k, \end{cases} \qquad (C_-)_{ki} = \begin{cases} 1 & \text{if } i = j_k, \\ 0 & \text{if } i \neq j_k. \end{cases}$$

The solution for the remaining densities consists precisely of those components of x not corresponding to the j_i.

Putting these together, we thus have the following system for simultaneously adding and subtracting triangles:

$$\begin{bmatrix} A & B_+ & B_- \\ C_+ & D_+ & D_* \\ C_- & & I \end{bmatrix} \begin{bmatrix} x \\ x_+ \\ x_- \end{bmatrix} = \begin{bmatrix} b \\ b_+ \\ 0 \end{bmatrix}, \tag{3.3}$$

where $D_* \in \mathbb{C}^{p\times q}$ with $(D_*)_{ik} = K(T_{N+i}, T_{j_k})$, and all other block are as defined previously. This can be solved efficiently by forming the Schur complement in the auxiliary variables $x_\pm$, which gives a system of dimension $m \equiv p + q$.

The cost of the above algorithm is essentially that of applying A^{-1} m times. For m small, say, $\lesssim 50$, this is quite feasible and should lead to very rapid solution times. However, for more sizable m on the order of 100 or more, as might be common for large structures with complicated geometries, iterative solvers present an attractive alternative. In this approach, we iterate on (3.3) with, for example, the block diagonal preconditioner

$$P^{-1} \equiv \begin{bmatrix} A^{-1} & & \\ & I & \\ & & I \end{bmatrix}.$$

Each iteration requires one inverse application, so if the total number of iterations is substantially less than m, then this is a superior method. Further acceleration is also possible: since p and q are now modest, it may be profitable to apply the matrices B_+, B_-, and C_+, which are all either ‘tall and skinny’ or ‘short and fat’, using some fast algorithm. Naturally, the FMM comes to mind, but for matrices with such extreme aspect ratios, even the cost of simply building the index tree can be significant. We propose instead to perform only operations that are local with respect to the modified data, i.e., the auxiliary triangles $T_{N+1}, \dots, T_{N+m}$, by compressing their far field just with respect to the smaller matrix dimension using a proxy-accelerated scheme (or the equivalent analytic FMM formulation). Considering, say, $B_+ \in \mathbb{C}^{N \times p}$ to be concrete, we do this by finding the skeletons and interpolation coefficients for the far-field interactions outgoing from the T_{N+i} to their proxy surface, which, recall, has only a constant number of degrees of freedom. Thus, this step is independent of the larger dimension N. The near field cannot be as efficiently compressed, so we simply account for it directly; the triangles $T_1, \dots, T_N$ lying in the neighborhood of the T_{N+i} can be found quickly by traversal of the existing tree on A. This procedure reduces the cost of applying B_+ from $\mathcal{O}(Np)$ to $\mathcal{O}(N + p)$, with a constant that is considerably smaller than that for using the full FMM.

Numerical experiments are now underway. Early estimates suggest that it may not be unreasonable to expect sub-second solves for systems as large as $50,000$ triangles or more; see Chapter 5 for some compelling applications in biology and chemistry.

3.4 Overdetermined least squares

The core procedure in our fast direct solver is a recursive skeletonization scheme for matrix compression. In this regard, the remainder of the solver, i.e., sparse embedding and inversion, may be viewed simply as manipulations of the compressed representation in order to obtain an LU decomposition of the original matrix. Perhaps unsurprisingly, it is also possible (to some extent) to construct other matrix factorizations from the compressed representation. Here, we consider the QR decomposition for the purpose of solving overdetermined least squares problems.

Let $A \in \mathbb{C}^{M \times N}$ be a compressible matrix in the sense of §2.1, but now with $M \geq N$ and $\operatorname{rank}(A) = N$. Then the system $Ax = b$ cannot in general be solved exactly and must instead be considered in the least squares sense: find x such that

$$\|Ax - b\| \leq \|Ay - b\| \quad \text{for all } y \in \mathbb{C}^N.$$

The solution is given by $x = A^+ b$, where

$$A^+ = (A^* A)^{-1} A^*$$

is the Moore-Penrose pseudoinverse of A. This is often solved via the QR decomposition $A = QR$, where Q is unitary and R is upper triangular, which gives

$$A^+ = (R^* R)^{-1} R^* Q^* = R^+ Q^*. \tag{3.4}$$

For further details, see [84].

Note 3.7. The formula (3.4) is only for theoretical convenience; in practice, R^+ is applied by performing back substitution on $R_1 \in \mathbb{C}^{N \times N}$, where $R = \operatorname{col}(R_1, 0)$.

We now assume that A has been compressed, hereafter working only with its

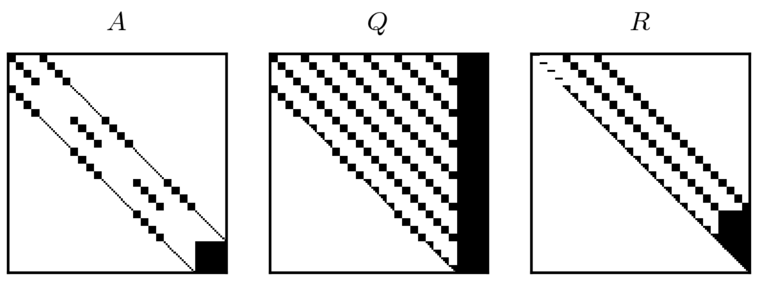

Figure 3.6: Example sparsity pattern of the QR decomposition $\boldsymbol{A} = \boldsymbol{QR}$, where $\boldsymbol{A}$ is the multilevel sparse embedding (2.12). Nonzeros are marked in black.

sparse embedding

$$\boldsymbol{A} \equiv \begin{bmatrix} D^{(1)} & L^{(1)} & & & & & \\ R^{(1)} & & -I & & & & \\ & -I & D^{(2)} & L^{(2)} & & & \\ & & R^{(2)} & \ddots & \ddots & & \\ & & & \ddots & D^{(\lambda)} & L^{(\lambda)} & \\ & & & & R^{(\lambda)} & & -I \\ & & & & & -I & S \end{bmatrix},$$

viz. §2.4. Can we follow an analogous procedure for the corresponding sparse problem $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{b} \equiv \operatorname{col}(b, 0, \dots, 0)$, and then extract the solution from the first block of $\boldsymbol{x}$? (For notational convenience, all quantities relating to the sparse system will be set in bold.) Due to its block tridiagonal structure, the QR decomposition $\boldsymbol{A} = \boldsymbol{QR}$ can be constructed rather efficiently (Figure 3.6). However, the solution is not quite as easy as simply applying $\boldsymbol{A}^{+}$ since the solution of the sparse system will not in general recover that of the original. The reason is that the zero constraints in $\boldsymbol{b}$, which, recall, are meant to enforce exact identities among the variables in $\boldsymbol{x}$ (see

§2.4), will typically be violated (in some least squares sense). Therefore, $\boldsymbol{x} = \boldsymbol{A}^{+}\boldsymbol{b}$ may not correspond to a valid solution with respect to the original variables.

Nevertheless, this can be fixed in a very straightforward manner with iterative refinement. Starting with $\boldsymbol{x}_0 \equiv \boldsymbol{A}^{+}\boldsymbol{b}$, we successively compute

$$\boldsymbol{x}_{k+1} \equiv \boldsymbol{x}_k + \Delta\boldsymbol{x}_k, \quad \Delta\boldsymbol{x}_k \equiv -\boldsymbol{A}^{+}\boldsymbol{A}_I\boldsymbol{x}_k, \tag{3.5}$$

where

$$\boldsymbol{A}_I \equiv \begin{bmatrix} \\ R^{(1)} & & -I & & & & & \\ & -I & D^{(2)} & L^{(2)} & & & & \\ & & R^{(2)} & \ddots & \ddots & & & \\ & & & \ddots & D^{(\lambda)} & L^{(\lambda)} & & \\ & & & & R^{(\lambda)} & & -I \\ & & & & & -I & S \end{bmatrix}$$

is $\boldsymbol{A}$ with the first block row zeroed out, i.e., the part corresponding only to the identities in $\boldsymbol{x}$. Intuitively, (3.5) hence corrects for the violation of those identities. We postpone the required analysis until a later work, noting here only that we have achieved success with the generalization

$$\Delta\boldsymbol{x}_k \equiv -\omega\boldsymbol{A}^{+}\boldsymbol{A}_I\boldsymbol{x}_k \tag{3.6}$$

for an appropriate choice of the *relaxation parameter* ω, so named after a similar parameter in successive over-relaxation [166]. Each step of the iteration can be thought of as projecting out the component of $\boldsymbol{x}_k$ that is incompatible with the identities, i.e., not lying in $\ker(\boldsymbol{A}_I)$; the parameter ω thus controls the magnitude of this projection. We find empirically that choosing $0 < \omega \lesssim 2$ generally works, with sometimes substantial improvements in the iteration count. The significance of this modification can be seen by writing $\boldsymbol{x}_k \equiv \boldsymbol{x}^* + \boldsymbol{e}_k$, where $\boldsymbol{x}^* \in \ker(\boldsymbol{A}_I)$ is the true solution. Then (3.5) and (3.6) give the error iteration

$$\boldsymbol{e}_{k+1} = \left(\boldsymbol{e}_k - \boldsymbol{A}^+ \boldsymbol{A}_I \boldsymbol{e}_k\right) \equiv \boldsymbol{H}\left(\omega\right) \boldsymbol{e}_k,$$

where $\boldsymbol{H}(\omega) = I - \omega \boldsymbol{A}^+ \boldsymbol{A}_I$, so convergence is immediate if $\|\boldsymbol{H}(\omega)\| < 1$, with ω now playing the role of a tuning parameter.

Remark 3.8. Alternative approaches are also possible, such as least squares with weighted constraints: $(\boldsymbol{A} + \lambda \boldsymbol{A}_I)\boldsymbol{x} = \boldsymbol{b}$, i.e.,

$$\boldsymbol{x} = \boldsymbol{A}^+ \left(\boldsymbol{b} - \lambda \boldsymbol{A}_I \boldsymbol{x}\right), \tag{3.7}$$

where λ is a penalty with $|\lambda| \to \infty$. Clearly, this does not affect the sparsity pattern of the system matrix. We have not yet tried this, but it is closely related to our current method, for which

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}^+ \left(\boldsymbol{b} - \omega \boldsymbol{A}_I \sum_{i=1}^{k} \boldsymbol{x}_i\right). \tag{3.8}$$

Therefore, (3.8) acts like (3.7) with graded penalty $\lambda_k \sim k\omega$ as a crude estimate. This also suggests that a larger ω should lead to faster convergence, an observation that we confirm numerically below.

We now provide some complexity estimates for our least squares algorithm. For this, we follow the basic outline of §2.5 and consider $M + N$ points distributed uniformly in a d-dimensional domain, with the row points sampled at a higher density since $M \geq N$, and all $M + N$ points sorted together in the same orthtree. Then continuing the notation of §2.5:

1. The number of levels is $\lambda \sim (1/d) \log(M + N)$.

2. The number of blocks at level l is $p_l \sim p_1/2^{d(l-1)} \sim (M+N)/2^{d(l-1)}$.

3. The number of points in each block at level l is

$$n_l \sim k_l \sim \begin{cases} (l-1)\log 2 + \log n_1 & \text{if } d = 1, \\ 2^{(d-1)(l-1)} n_1^{1-1/d} & \text{if } d > 1. \end{cases}$$

Therefore, using Figure 3.6 as a guide, the cost of computing the QR decomposition $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$ is

$$T_{\text{qr}} \sim \sum_{l=1}^{\lambda} p_l n_l^2 \sum_{l'=1}^{l} n_{l'} + (p_\lambda k_\lambda)^2 \sum_{l=1}^{\lambda} p_l n_l \sim \begin{cases} (M+N)\log^2(M+N) & \text{if } d = 1, \\ (M+N)^{3-2/d} & \text{if } d > 1, \end{cases} \tag{3.9}$$

where the first term accounts for the cost of orthogonalizing all column blocks except the last, which is fully dense and described by the second. Similarly, the cost T_{sv} of each solve is

$$T_{\text{svq}} \sim \sum_{l=1}^{\lambda} p_l n_l \sum_{l'=1}^{l} n_{l'} + p_\lambda k_\lambda \sum_{l=1}^{\lambda} p_l n_l \sim \begin{cases} (M+N)\log(M+N) & \text{if } d = 1, \\ (M+N)^{2-1/d} & \text{if } d > 1 \end{cases} \tag{3.10}$$

for applying $\boldsymbol{Q}^*$, plus

$$T_{\text{svr}} \sim \sum_{l=1}^{\lambda} p_l n_l^2 + (p_\lambda k_\lambda)^2 \sim \begin{cases} M+N & \text{if } d = 1, \\ (M+N)\log(M+N) & \text{if } d = 2, \\ (M+N)^{2(1-1/d)} & \text{if } d > 2 \end{cases}$$

for back substitution with $\boldsymbol{R}$, so T_{sv} has complexity (3.10). These are very similar to those in §2.5, but slightly worse due to the fill-in of $\boldsymbol{Q}$.

As a numerical example, we considered a 2D charge fitting problem mimicking the commonly used restrained electrostatic potential method [23] in computational chemistry. Specifically, we distributed 8192 sources $\boldsymbol{s}_i$ of random strengths q_i in the unit circle (of radius one), and observed their potential field

$$\varphi\left(\boldsymbol{t}_i\right) = \sum_j q_j G\left(\boldsymbol{t}_i, \boldsymbol{s}_j\right),$$

where G is the 2D Laplace Green's function (2.5), at M uniformly spaced target points $\boldsymbol{t}_i$ on the ring of radius $1+\delta$ for $\delta > 0$. We also placed N *regression charges* $\boldsymbol{r}_i$ uniformly spaced on the unit circle, whose strengths we manipulated to try to match φ at the $\boldsymbol{t}_i$. This constitutes a least squares problem with matrix $A \in \mathbb{R}^{M\times N}$, where $A_{ij} = G(\boldsymbol{t}_i, \boldsymbol{r}_j)$.

Note 3.9. This is also similar to the construction of equivalent densities in the kernel-independent FMM [208]. If the $\boldsymbol{r}_i$ are a subset of the $\boldsymbol{s}_i$, then this is like computing an interpolative decomposition (ID), cf. §2.1 and [47, 141]. Indeed, the ID requires a least squares solve [47], so this technology may potentially find use there as well.

Remark 3.10. For such problems where the row and column points are separated (here, by a distance δ), the rank of any matrix block is in fact constant, so putting $n_l \sim k_l = \mathcal{O}(1)$ in (3.9) and (3.10) gives *linear* complexity for both in any dimension.

Remark 3.11. This example can also be adapted for partial charge fitting in solvated biomolecules by combining the present methods with the composite operator formulation of §3.1.

We fixed $\delta = 0.01$ and solved the least squares system for various M and N, making sure each time that the resulting matrix has full column rank [see 74]. This was done using both LAPACK/ATLAS [6, 201] and our compression-based code. In each case, we set a compression precision of $\epsilon = 10^{-9}$ and an iteration precision of $\epsilon_0 = 10^{-6}$; the iteration was terminated when $\|\Delta\boldsymbol{x}_k\|/\|\boldsymbol{x}_k\| < \epsilon_0$. All experiments were performed on a 3.10 GHz processor, where the codes for LAPACK/ATLAS and recursive skeletonization (i.e., the direct portion of our least squares solve) were run in Fortran, and the remaining iteration in MATLAB R2011a (The MathWorks, Inc.: Natick, MA) for its interface to SuiteSparseQR [61]. All unitary matrices $\boldsymbol{Q}$ were stored in Householder form for efficiency. The error was assessed by comparing against the solution returned by LAPACK/ATLAS.

We first studied the behavior of the algorithm with respect to the relaxation parameter ω. We hence fixed $M = 8192$ and $N = 1024$, and performed a parameter sweep over $0 \leq \omega \leq 3$. The results are shown in Figure 3.7, from which we see that a larger ω generally corresponds to a lower iteration count, though some care must be taken since convergence is lost for ω larger than some critical value ω^*; in this example, $\omega^* \approx 2.5$. Clearly, the iteration stalls at $\omega = 0$, but we still appear to get some nontrivial accuracy. Based on these results, we next set $\omega \equiv 2$ and solved at many different combinations of M and N. The solution times are consistent with those predicted and show vastly superior scalings over the classical $\mathcal{O}(MN^2)$ method (Figure 3.8). Interestingly, the iteration count seems to scale as $n_{\text{iter}} = \mathcal{O}((1/N)\log M)$, but remains manageable in most cases. The full data are given in Table 3.4.

Remark 3.12. Clearly, this technique can also be used to solve square systems, in which case no iteration is required as the solution is unique. This might be desirable if the system matrix is particularly ill-conditioned since QR methods tend to be more numerically stable.

Remark 3.13. Our methods should also be compared with modern fast algorithms based on randomization [165], which require only $\mathcal{O}(MN + N^3)$ operations for *general*

Table 3.4: Numerical results for least squares charge fitting at compression and iteration precisions $\epsilon = 10^{-9}$ and $\epsilon_0 = 10^{-6}$, respectively: M, matrix row dimension; N, matrix column dimension; T_{ls}, LAPACK/ATLAS least squares solution time (s); T_{cm}, matrix compression time (s); T_{qr}, sparse QR factorization time (s); T_{iter}, corrective iteration time (s); n_{iter}, number of iterations required; E, relative error. Three sets of data are combined here: M = 1024–16384 with N = 1024; M = 8192 with N = 128–2048; and $M = 1024\Delta$ and $N = 128\Delta$ with Δ = 1–16.

M	N	T_{ls}	T_{cm}	T_{qr}	T_{iter}	n_{iter}	E
1024	128	2.2E−2	1.7E−2	1.5E−2	5.3E−1	74	4.3E−5
1024	1024	4.0E−1	6.7E−2	4.5E−2	3.6E−2	1	3.3E−7
2048	256	1.1E−1	4.4E−2	4.1E−2	2.1E+0	121	3.6E−4
2048	1024	9.2E−1	1.0E−1	6.9E−2	5.9E−1	17	5.8E−5
4096	512	6.1E−1	9.6E−2	6.4E−2	2.6E+0	68	3.2E−4
4096	1024	1.9E+0	1.5E−1	1.1E−1	1.6E+0	31	1.9E−4
8192	128	2.7E−1	8.5E−2	6.2E−2	2.0E+1	439	5.9E−4
8192	256	5.2E−1	1.0E−1	8.1E−2	1.2E+1	234	1.4E−3
8192	512	1.3E+0	1.4E−1	9.4E−2	6.2E+0	102	7.1E−4
8192	1024	4.1E+0	2.1E−1	1.3E−1	3.7E+0	48	5.5E−4
8192	2048	1.4E+1	3.2E−1	2.3E−1	2.3E+0	22	4.5E−4
16384	1024	8.9E+0	3.2E−1	2.1E−1	7.5E+0	60	1.1E−3
16384	2048	3.0E+1	4.1E−1	2.5E−1	5.3E+0	35	1.1E−3

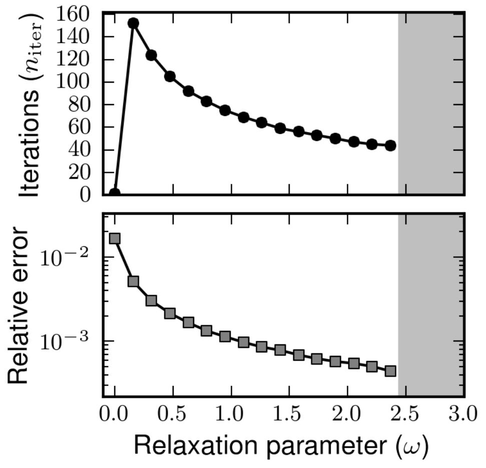

Figure 3.7: Performance of the semi-direct least squares solver as a function of the relaxation parameter ω in terms of the number n_{iter} of iterations required and the resulting error achieved. The shaded region indicates values of ω for which the iteration did not converge (the errors diverged).

matrices.

3.5 A compression-based fast multipole method

Lastly, we outline a compression-based FMM following essentially the approach in [141] but embedded within our matrix framework. The ideas are exactly the same as those from Chapter 2; thus, this section may also be considered a guide on how to modify our direct solver into an FMM. In particular, both algorithms are based on numerical matrix compression, so the resulting FMM is kernel-independent. This makes it especially useful for functions that are difficult to handle analytically, in contrast to traditional FMMs, which require analytic expansions [86, 93]. A good

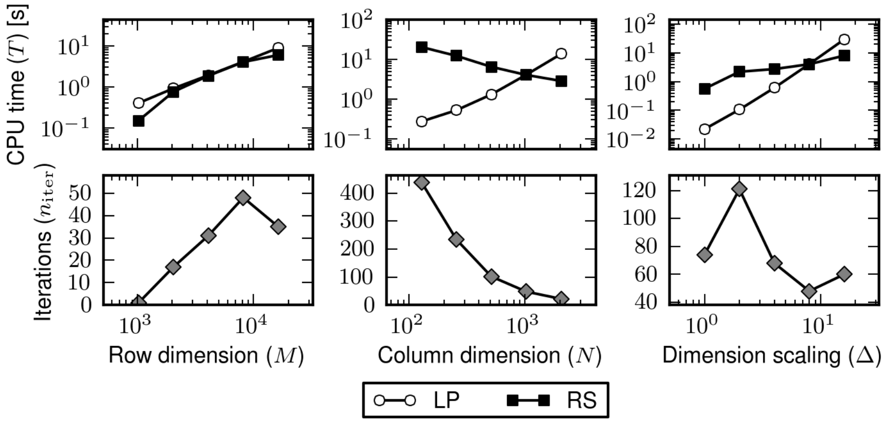

Figure 3.8: CPU times for least squares charge fitting in various cases using LAPACK/ATLAS (LP) and a recursive skeletonization-based semi-direct solver (RS). Three scalings are shown, for M and N the system matrix row and column dimensions, respectively: with varying M and fixed N (left), with fixed M and varying N (center), and with proportionally varying $M, N \propto \Delta$ (right). For each case, the CPU time T required is shown; for RS, the number n_{iter} of iterations needed is also given. The precision of RS was set at $\epsilon = 10^{-9}$ for compression and $\epsilon_0 = 10^{-6}$ for iteration.

example is the solution of the heat equation using potential theory with high-order time integration, which involves exponential integrals [129, 183]. The astute reader may notice that we have already presented such an FMM in Chapter 2. This is indeed true in the sense that both algorithms are capable of fast matrix-vector multiplication, but the current formulation is based on a slight reorganization of the matrix that allows for far greater efficiency, leading to $\mathcal{O}(N)$ complexity in all dimensions. The tradeoff is that inversion is now slower, but this is of no consequence as we are interested only in matrix applications.

We proceed as before and consider a matrix $A \in \mathbb{C}^{N \times N}$ discretizing some integral kernel in a d-dimensional domain with smoothness properties similar to that for the Laplace Green's function. Then A is hierarchically block separable under some appropriate tree ordering, so we can write, on the first level,

$$A = D + LSR$$

following (2.2), where $D \in \mathbb{C}^{N \times N}$ consists of the diagonal blocks of A, $S \in \mathbb{C}^{K_{\mathrm{r}} \times K_{\mathrm{c}}}$ is its skeleton matrix, and L and R are row and column projections, respectively. Since S contains neighboring interactions,

$$K_{\mathrm{r}}, K_{\mathrm{c}} \sim \begin{cases} \log N & \text{if } d = 1, \\ N^{1-1/d} & \text{if } d > 1 \end{cases}$$

by the argument of §2.5. In other words, the skeleton dimension grows with N.

Here, we consider instead the decomposition

$$A = N + LSR,$$

where N characterizes all self- *and* neighboring interactions so that S now accounts only for the far field. Consequently, $K_{\mathrm{r}}, K_{\mathrm{c}} = \mathcal{O}(1)$ to any specified precision. This is the basis for the improved complexity; for further acceleration, we can compress the near field also by writing

$$N = D + UTV,$$

obtained via the ID as well (though other compression schemes can be used, e.g., the singular value decomposition, since the representations need no longer be nested). The multilevel analogue is therefore

$$\begin{aligned} A = D &+ U^{(1)} T^{(1)} V^{(1)} \\ &+ L^{(1)} \left[U^{(2)} T^{(2)} V^{(2)} + L^{(2)} \left(\cdots L^{(\lambda)} U^{(\lambda)} T^{(\lambda)} V^{(\lambda)} R^{(\lambda)} \cdots \right) R^{(2)} \right] R^{(1)}, \quad (3.11) \end{aligned}$$

where D consists of self-interactions at the finest level; $T^{(l)}$ is the skeletonized near field at level l, with row and column projection matrices $U^{(l)}$ and $V^{(l)}$, respectively; and $L^{(l)}$ and $R^{(l)}$ are the far-field row and column projection matrices at level l. Observe that self-interactions appear only once (at the finest level), and furthermore that no far-field skeleton exists, with interactions applied as they emerge in the near field as we move up the tree. Both are consistent with traditional FMM formulations. We can hence think of (3.11) as a sequence of far-field compressions, where at each level the near field portion of the matrix is extracted. As with our direct solver, an algorithm for rapidly computing matrix-vector products is immediate by simply applying the matrices in (3.11) from right to left.

To determine the complexity of the representation (3.11), we adopt the same notation as §2.5, but now with $k_l = \mathcal{O}(1)$ so in fact all $n_l = \mathcal{O}(1)$. Then the cost of compression using proxy acceleration is

$$T_{\mathrm{cm}} \sim \sum_{l=1}^{\lambda} p_l n_l^3 \sim N.$$

Similarly, the cost of matrix-vector multiplication is

$$T_{\mathrm{mv}} \sim \sum_{l=1}^{\lambda} p_l n_l^2 \sim N.$$

Therefore, the algorithm has optimal $\mathcal{O}(N)$ complexity.

The representation (3.11) can also be used for fast matrix inversion by embedding it into a sparse matrix exactly as in (2.12). While the complexities for factorization are the same as those for the direct solver, i.e., (2.15), the matrix factors now fill in so that the solve time is also (2.15). This is because a neighbor grid structure must be inverted at each level, which destroys the sparsity of the operators in the analogue of (2.14). See [155] for a similar FMM-based approach.

As mentioned briefly in the introduction to this chapter, the compression-based FMM, by virtue of its linear algebraic structure, cannot easily accomodate certain important optimizations employed by analytic FMMs, such as diagonal translations [48, 92]. In this case, applying the near-field matrices $T^{(l)}$ costs $\mathcal{O}(p_l k_l^2)$ operations instead of just $\mathcal{O}(p_l k_l)$ in diagonal form. This begins to play a role especially at high precision, where, for example, $k_l \sim 100$ in 3D. However, it should be noted that the ranks k_l emerging from compression are not the same as those from analysis, and, in fact, in many cases are much smaller [see 88]. The reason is that whereas the analytic approach must account for all possible point distributions, compression can specialize only to the particular distribution at hand, thereby producing representations that are tailored to the problem and, in that sense, optimal. Interestingly, this effect seems to be especially pronounced also at high precision, so any tradeoff between the two methods is not immediately clear.

Finally, as with the direct solver, the compressed representation (3.11) can be saved for repeated matrix-vector multiplication, which presents yet another scheme for accelerating the solution of linear systems requiring many iterations. Generally, we can expect the FMM-based solver to prevail in high dimensions due to the current limitations of the direct solver, though whether it will be faster in 2D or 3D remains to be seen. As direct solver technology matures, however, toward optimal or near-optimal complexities (see §5.4), we suspect that it will become dominant for reasons of robustness and adaptability (§1.4).

4 Application to protein pK_a calculations

In this chapter, we return to the linearized Poisson-Boltzmann equation (LPBE) of Chapter 1, formulated as the second-kind boundary integral equation (1.12)

$$(I + \lambda K) \begin{bmatrix} \mu \\ \sigma \end{bmatrix} = \lambda \begin{bmatrix} \varphi_s \\ -\partial \varphi_s / \partial \nu \end{bmatrix}$$

for the densities σ and μ on the molecular surface, where

$$\varphi_s (\boldsymbol{r}) = \frac{1}{\varepsilon_1} \sum_i q_i G_0 (\boldsymbol{r}, \boldsymbol{r}_i)$$

is the electrostatic potential due to charges in the molecule, with compact operator

$$K = \begin{bmatrix} D_\kappa - \alpha D_0 & S_\kappa - S_0 \\ -\alpha (D'_\kappa - D'_0) & -(\alpha S'_\kappa - S'_0) \end{bmatrix},$$

where S_k and D_k are the single- and double-layer potentials, respectively, for the Green's function

$$G_k (\boldsymbol{r}, \boldsymbol{s}) = \frac{e^{-k|\boldsymbol{r}-\boldsymbol{s}|}}{4\pi |\boldsymbol{r} - \boldsymbol{s}|}.$$

Note that the left-hand side depends only on the molecular geometry, and the right-hand side only on the charge configuration. The potential at any point can be expressed in terms of the surface densities as

$$\varphi = \begin{cases} S_\kappa \sigma + D_\kappa \mu & \text{in } \Omega_0, \\ S_0 \sigma + \alpha D_0 \mu + \varphi_s & \text{in } \Omega_1, \end{cases}$$

viz. (1.11), where Ω_0 and Ω_1 denote the solvent and the molecule, respectively; see §1.1 for the full notation. In §1.2, we showed how to discretize this system, while in Chapter 2 we developed a direct numerical algorithm to solve it efficiently. Here, we now apply our techniques to the calculation of protein pK_a values, which provides an important biological setting where fast direct electrostatics can play a significant role.

The pK_a of an acid A is the decimal cologarithm of the equilibrium constant for the ionization reaction $\mathrm{AH} \rightleftharpoons \mathrm{A} + \mathrm{H}$:

$$\mathrm{p}K_\mathrm{a} \equiv -\log_{10} \frac{[\mathrm{A}]\,[\mathrm{H}]}{[\mathrm{AH}]} = \log_{10} \frac{[\mathrm{AH}]}{[\mathrm{A}]} + \mathrm{pH}, \tag{4.1}$$

and is related to the Gibbs free energy change by

$$\mathrm{p}K_\mathrm{a} = \frac{\beta}{\ln 10} \Delta G \left(\mathrm{AH} \to \mathrm{A} + \mathrm{H}\right), \tag{4.2}$$

where $\beta \equiv 1/(RT)$ for R the gas constant and T the absolute temperature. (We use ln for the natural logarithm to maintain consistency with the chemistry literature.) The pK_a hence captures the thermodynamics of acid dissociation and therefore characterizes the quantitative behavior of acid-base reactions. Such protonation or deprotonation of so-called titrating sites can drive changes in binding affinities, enzymatic activities, and structural properties [54, 65, 206]. Consequently, pK_a values are important for a variety of biomolecular processes, and their accurate theoretical prediction is of significant practical interest.

In the next section, we review the theory of protein titration following [21, 194], and show that the main computational bottleneck in pK_a calculations is the solution of the LPBE with multiple right-hand sides. The procedure thus lends itself naturally to direct solvers, which can factor the system matrix once and then reuse it for each solve. In this regard, our work can be considered a heavily accelerated version of that by Juffer et al. [115], who employed a similar boundary integral approach but used only classical $\mathcal{O}(N^3)$ techniques; our compression methods also dramatically reduce the memory footprint, hence allowing far larger problems to be addressed. Furthermore, we incorporate various proven optimizations and introduce two minor but novel contributions:

1. a generalized multi-flip Metropolis criterion for efficient Markov chain Monte Carlo (MCMC) sampling of tightly coupled titrating sites; and
2. a simple statistical procedure to derive error estimates for computed $\mathrm{p}K_\mathrm{a}$ values.

4.1 Theory of protein titration

We begin by analyzing the simple case of a solvated protein with a single titrating site, i.e., a residue that can be either protonated or unprotonated, for which

$$\mathrm{p}K_\mathrm{a} = \frac{\beta}{\ln 10} \Delta G \left(\mathrm{A_p H} \rightarrow \mathrm{A_p} + \mathrm{H} \right),$$

which is just (4.2) but with the subscript p to emphasize that the site exists in the environment of the protein. This free energy change cannot be calculated directly in a straightforward way, so we consider instead the thermodynamic cycle shown in Figure 4.1, which gives

$$\mathrm{p}K_\mathrm{a} = \frac{\beta}{\ln 10} \left[\Delta G \left(\mathrm{A_s H} \rightarrow \mathrm{A_s} + \mathrm{H} \right) + \Delta G \left(\mathrm{A_s} \rightarrow \mathrm{A_p} \right) - \Delta G \left(\mathrm{A_s H} \rightarrow \mathrm{A_p H} \right) \right],$$

where the subscript s refers to the titrating site isolated in the solvent. It is useful to consider a decomposition of this form because the *model pK_a*

$$\mathrm{p}K_\mathrm{a}^0 \equiv \frac{\beta}{\ln 10} \Delta G \left(\mathrm{A_s H} \rightarrow \mathrm{A_s} + \mathrm{H} \right)$$

$$\begin{tikzcd}
\mathrm{A_sH} \arrow[r, "\Delta G(\mathrm{A_sH} \longrightarrow \mathrm{A_s+H})"] \arrow[d, "\Delta G(\mathrm{A_sH} \longrightarrow \mathrm{A_pH})"'] & \mathrm{A_s + H} \arrow[d, "\Delta G(\mathrm{A_s} \longrightarrow \mathrm{A_p})"] \\
\mathrm{A_pH} \arrow[r, "\Delta G(\mathrm{A_pH} \longrightarrow \mathrm{A_p+H})"] & \mathrm{A_p + H}
\end{tikzcd}$$

Figure 4.1: Thermodynamic cycle for protein titration. The free energy change $\Delta G(\mathrm{A_pH} \to \mathrm{A_p} + \mathrm{H})$ for ionization in the protein can be computed from that for the corresponding reaction in the solvent ($\Delta G(\mathrm{A_sH} \to \mathrm{A_s} + \mathrm{H})$), which is generic and determined by experiment, and the transfer energies $\Delta G(\mathrm{A_s} \to \mathrm{A_p})$ and $\Delta G(\mathrm{A_sH} \to \mathrm{A_pH})$ for the unprotonated and protonated forms, respectively, which cancel to within electrostatic contributions.

can be determined experimentally for each residue type using a generic *model compound*, and therefore can be taken as data. Moreover, if we assume that no structural rearrangements occur upon ionization, then all non-polar contributions to the remaining transfer energies cancel, and so

$$\begin{aligned}
\Delta G\left(\mathrm{A_s} \to \mathrm{A_p}\right) - \Delta G\left(\mathrm{A_sH} \to \mathrm{A_pH}\right) &= \Delta G_{\mathrm{ele}}\left(\mathrm{A_s} \to \mathrm{A_p}\right) - \Delta G_{\mathrm{ele}}\left(\mathrm{A_sH} \to \mathrm{A_pH}\right) \\
&= \Delta G_{\mathrm{ele}}\left(\mathrm{A_s} \to \mathrm{A_sH}\right) - \Delta G_{\mathrm{ele}}\left(\mathrm{A_p} \to \mathrm{A_pH}\right),
\end{aligned}$$

i.e.,

$$\mathrm{p}K_{\mathrm{a}} = \mathrm{p}K_{\mathrm{a}}^{0} - \frac{\beta}{\ln 10}\left[\Delta G_{\mathrm{ele}}\left(\mathrm{A_p} \to \mathrm{A_pH}\right) - \Delta G_{\mathrm{ele}}\left(\mathrm{A_s} \to \mathrm{A_sH}\right)\right]. \tag{4.3}$$

The second term is called the pK_{a} shift and characterizes the electrostatic interactions of the titrating site with the protein environment. Observe that the free energy of protonation at a given pH is hence

$$\begin{aligned}\Delta G\left(\mathrm{A_p} \to \mathrm{A_pH}; \mathrm{pH}\right) &= -RT \ln \frac{[\mathrm{AH}]}{[A]} \\ &= -RT \ln 10 \left(\mathrm{p}K_\mathrm{a} - \mathrm{pH}\right) \\ &= -RT \ln 10 \left(\mathrm{p}K_\mathrm{a}^0 - \mathrm{pH}\right) + \Delta G_\mathrm{ele}\left(\mathrm{A_p} \to \mathrm{A_pH}\right) \\ &\qquad - \Delta G_\mathrm{ele}\left(\mathrm{A_s} \to \mathrm{A_sH}\right).\end{aligned}$$

We now take a brief aside to discuss the calculation of electrostatic energies in our integral equation framework. For this, we adopt the premise of Chapter 1 and consider a collection of charges q_i at locations $\boldsymbol{r}_i \in \Omega_1$ for $i = 1, \ldots, N_\mathrm{src}$. Then the electrostatic energy of the system is

$$E = \frac{1}{2} \sum_{i=1} q_i \varphi\left(\boldsymbol{r}_i\right),$$

where φ is the electrostatic potential. In our formulation, $\varphi = \varphi_p + \varphi_s$, where φ_p is the *polarization potential* (or *reaction potential*) due to the solvent, composed of the terms involving the surface densities σ and μ in (1.11), and φ_s is the *direct potential* due to the charges. Following §1.2, these can be written in matrix form as

$$\varphi_p = CA^{-1}Bq, \quad \varphi_s = Dq,$$

where

$$A = I + \lambda \begin{bmatrix} D_\kappa - \alpha D_0 & S_\kappa - S_0 \\ -\alpha\left(D'_\kappa - D'_0\right) & -\left(\alpha S'_\kappa - S'_0\right) \end{bmatrix} \in \mathbb{R}^{2N_\mathrm{tri} \times 2N_\mathrm{tri}}$$

is the system matrix of the discretized integral equation (1.13), for N_tri the number of triangles composing the molecular surface Σ;

$$B = \lambda \begin{bmatrix} \varphi_s \\ -\partial\varphi_s/\partial\nu \end{bmatrix} \in \mathbb{R}^{2N_\mathrm{tri} \times N_\mathrm{src}}, \quad C = \begin{bmatrix} D_0 & \alpha S_0 \end{bmatrix} \in \mathbb{R}^{N_\mathrm{src} \times 2N_\mathrm{tri}}$$

are the matrices generating the right-hand side of (1.13) from the charges and evaluating the polarization potential from the surface densities via (1.11), respectively; and

$$D \in \mathbb{R}^{N_{\mathrm{src}} \times N_{\mathrm{src}}}, \quad D_{ij} = \begin{cases} 0 & \text{if } i = j, \\ (1/\varepsilon_1)\, G_0\left(\boldsymbol{r}_i, \boldsymbol{r}_j\right) & \text{if } i \neq j \end{cases}$$

is the matrix computing the direct potential between the charges. Therefore,

$$\varphi = \left(CA^{-1}B + D\right) q \equiv Wq, \tag{4.4}$$

so

$$E = \frac{1}{2} q^{\mathsf{T}} W q. \tag{4.5}$$

Note that each of A^{-1}, B, C, and D is compressible using the algorithm of Chapter 2—the first in its capacity as a direct solver, and the others as a generalized fast multipole method (FMM)—hence W is compressible as well.

Returning now to (4.3), we consider first the energy difference $\Delta G_{\mathrm{ele}}(\mathrm{A_p} \to \mathrm{A_pH})$ in the protein, which has vector charges b and t corresponding to the background and titrating charges, respectively. Specifically, b gives the charges due to the fixed background and t gives the additional charges introduced by the protonation of the titrating site; in other words, the charge vector in the unprotonated form is $q = b$, whereas that in the protonated form is $q = b + t$. Then by (4.5),

$$\Delta G_{\mathrm{ele}}\left(\mathrm{A_p} \to \mathrm{A_pH}\right) = \frac{1}{2}\left[(b+t)^{\mathsf{T}} W (b+t) - b^{\mathsf{T}} W b\right],$$

where the first term gives the energy of the state $\mathrm{A_pH}$, and the second that of $\mathrm{A_p}$. The same argument adapted to the model compound instead of the full protein gives $\Delta G_{\mathrm{ele}}(\mathrm{A_s} \to \mathrm{A_sH})$.

Remark 4.1. Clearly, our formulation can support the use of a so-called detailed charge model, where protonation can spread charge over a number of different atoms [compare, e.g., 7, 21, 22, 115, 205].

Suppose now that we have N_{titr} titrating sites. For each site i, we compute its *intrinsic pK*$_a$, which we call $\mathrm{p}K_i^0$, according to (4.3), where the protein environment is defined as that corresponding to the fixed background charges only, i.e., with all titrating sites unprotonated. Then to calculate the free energy of an arbitrary protein protonation state, we must first add the energies corresponding to each relevant $\mathrm{p}K_i^0$, and then the energy of interaction between the protonated sites. That is,

$$\Delta G\left(\mathrm{A} \to \mathrm{A}\left(\theta\right);\mathrm{pH}\right) = -RT\ln 10\sum_i \theta_i\left(\mathrm{p}K_i^0 - \mathrm{pH}\right) + \frac{1}{2}\sum_i \theta_i \sum_{j\neq i}\theta_j \Delta G_{ij}, \quad (4.6)$$

where $\theta \in \{0,1\}^{N_{\text{titr}}}$ has entries 0 or 1 indicating whether a site is unprotonated or protonated, respectively, and

$$\Delta G_{ij} = t_i^{\mathsf{T}} W t_j \quad (4.7)$$

is the electrostatic interaction energy between sites i and j, for t_i the titrating charge vector corresponding to the protonation of site i. Note that the background charges do not appear in the formula for the ΔG_{ij}; they are used only to compute the $\mathrm{p}K_i^0$.

Note 4.2. In principle, $\Delta G_{ij} = \Delta G_{ji}$, but we only have approximate equality here since we use an unsymmetric triangle-centroid collocation scheme. (This can be made somewhat clearer by interpreting ΔG_{ij} as the energy associated with the protonation of site i due to the field induced by the protonation of site j.) In what follows, we will try to 'symmetrize' the energies, either by summing over both ΔG_{ij} and ΔG_{ji} as in (4.6) or by considering both in the treatment of energetic cutoffs (see §4.3).

Armed with the free energy (4.6) of an arbitrary protonation state, the next step is to compute the Boltzmann average

$$\langle \theta_i; \mathrm{pH} \rangle \equiv \frac{\sum_\theta \theta_i e^{-\beta \Delta G(\mathrm{A} \longrightarrow \mathrm{A}(\theta); \mathrm{pH})}}{\sum_\theta e^{-\beta \Delta G(\mathrm{A} \longrightarrow \mathrm{A}(\theta); \mathrm{pH})}}, \tag{4.8}$$

over all possible states θ at each pH, and then to take the pK_a of site i as the pH at which $\langle \theta_i; \mathrm{pH} \rangle = 1/2$, following the Henderson-Hasselbalch equation (4.1). The state space, however, is exponentially large in N_titr, so while (4.8) can be computed directly for small proteins, more sophisticated techniques are required in general. In this work, we use MCMC methods to sample from the probability distribution

$$\Pr(\theta; \mathrm{pH}) \propto e^{-\beta \Delta G(\mathrm{A} \longrightarrow \mathrm{A}(\theta); \mathrm{pH})}. \tag{4.9}$$

Before moving to that topic, though, we will find it useful to describe a classical approach based on mean field approximation, which neglects correlations between titrating sites but can provide a useful starting point for our Monte Carlo simulation. Furthermore, as the number of Monte Carlo steps will typically be quite large, it is most computationally efficient to precompute the ΔG_{ij} and then simply to perform table lookups at each step. This can be accomplished by applying W N_titr times, once each to compute the potential $\varphi_j \equiv W t_j$ due to the protonation of site j, from which its interaction energies $\Delta G_{ij} = t_i^\mathsf{T} \varphi_j$ with all sites i can be obtained. This energy precomputation is often the most demanding part of the entire calculation, so any acceleration, for example, using our fast direct solver, is very welcome.

Remark 4.3. A very similar situation is encountered in electrical engineering as *capacitance extraction*, where one wishes to characterize the induced-charge behavior of a collection of electronic devices [see, e.g., 118, 146, 157]. This, too, constitutes a problem requiring multiple electrostatic solves, once for each component involved; indeed, fast direct solvers have recently been applied here as well [38].

4.2 Mean field approximation

Instead of considering each titrating site as either strictly protonated or unprotonated, we now let each site have protonation probability p_i and consider their interaction through this mean field average [22, 190]; this is the same as treating the single-site case with effective background charge $b + \sum_{j \neq i} p_j t_j$ for each site i. Then from (4.6), the protonation energy of site i is

$$\Delta G_i = -RT \ln 10 \left(\mathrm{p}K_i^0 - \mathrm{pH} \right) + \frac{1}{2} \sum_{j \neq i} p_j \left(\Delta G_{ij} + \Delta G_{ji} \right)$$

with closure condition

$$\frac{p_i}{1 - p_i} = e^{-\beta \Delta G_i},$$

which can be solved self-consistently via the iteration

$$\Delta G_i \left(p^k \right) \equiv -RT \ln 10 \left(\mathrm{p}K_i^0 - \mathrm{pH} \right) + \frac{1}{2} \sum_{j \neq i} p_j^k \Delta G_{ij}, \tag{4.10a}$$

$$p_i^{k+1} \equiv \frac{e^{-\beta \Delta G_i \left(p^k \right)}}{1 + e^{-\beta \Delta G_i \left(p^k \right)}} \tag{4.10b}$$

for some initial vector iterate p^0 (we use simply $p^0 = (0, \ldots, 0)$). The probabilistic character of each p_i^k is immediate. Thresholding then gives an effective initial Monte Carlo state:

$$\theta_i \equiv \begin{cases} 0 & \text{if } p_i < 1/2, \\ 1 & \text{if } p_i \geq 1/2. \end{cases} \tag{4.11}$$

Note 4.4. The mean field estimate for the pK_a of site i is just

$$\mathrm{p}K_i \equiv \frac{\beta}{\ln 10} \Delta G_i \left(p \right).$$

For proteins with titrating sites that interact only weakly (at a given pH), the iteration typically converges very rapidly, i.e., within ten iterations or so. Stronger interactions generally require more iterations, and sometimes the iteration does not converge at all. In such cases, we use instead the probabilities p^1 corresponding to the intrinsic energy differences between the $\mathrm{p}K_i^0$ and the pH for each site i without any titrating site interactions. This gives a poorer initial estimate compared to that above, but will only affect the burn-in time for the Markov chain to reach the equilibrium distribution (4.9) by ergodicity.

4.3 Reduced site approximation

Since the interesting protonation behavior of a given titrating site will typically occur near its $\mathrm{p}K_i^0$, it is evident that its state can be fixed as either protonated or unprotonated for many pH values away from $\mathrm{p}K_i^0$, especially near the extremes. This observation was first made by Bashford and Karplus in [22], who formalized it as the *reduced site approximation* and demonstrated its ability to provide exponential reductions in the protonation state space.

The method is very intuitive and is based on calculating the maximum and minimum protonation probabilities for each titrating site. We consider first the minimum protonation, which is clearly achieved when all other titrating sites are protonated as this maximizes the free energy. Thus, for each site i, we compute

$$\Delta G_{\max,i} \equiv -RT \ln 10 \left(\mathrm{p}K_i^0 - \mathrm{pH}\right) + \frac{1}{2} \sum_{j \neq i} \left(\Delta G_{ij} + \Delta G_{ji}\right).$$

Then the minimum protonation probability is

$$p_{\min,i} = \frac{e^{-\beta \Delta G_{\max,i}}}{1 + e^{-\beta \Delta G_{\max,i}}}, \tag{4.12}$$

so if $p_{\min,i} \geq p^*_{\min}$ for some threshold, say, $p^*_{\min} = 0.99$, then we consider site i as completely protonated and remove it from Monte Carlo sampling. Similarly, the minimum free energy is achieved when no other site is protonated, i.e.,

$$\Delta G_{\min,i} \equiv -RT \ln 10 \left(\mathrm{p}K_i^0 - \mathrm{pH} \right)$$

and the maximum protonation probability is

$$p_{\max,i} = \frac{e^{-\beta \Delta G_{\min,i}}}{1 + e^{-\beta \Delta G_{\min,i}}}. \tag{4.13}$$

Hence if $p_{\max,i} \leq p^*_{\max}$ (e.g., $p^*_{\max} = 0.01$), then we consider site i as completely unprotonated. If N_{fix} is the number of sites fixed in this way, then clearly the state space is reduced by a factor of $2^{N_{\mathrm{fix}}}$. Hereafter, for a given pH, we let $N_{\mathrm{free}} \equiv N_{\mathrm{titr}} - N_{\mathrm{fix}}$ be the number of free titrating sites remaining.

Note 4.5. Other approaches of reducing the Monte Carlo workload have also been reported, most notably within the context of hybrid methods employing statistical mechanical treatments within titrating site clusters and mean field approximations between them [79, 206].

4.4 Monte Carlo sampling

Restricting to the N_{free} unfixed sites, we now sample (4.9) over the remaining state space using a standard Metropolis-Hastings MCMC algorithm [26, 142]. To be precise, we start the Markov chain at the initial protein state as determined by thresholding of the mean field protonation probabilities, and accept each transition from the current state θ to a new proposed state θ' with probability

$$\Pr(\theta \to \theta') = \min \left\{ 1, e^{-\beta [\Delta G(\theta') - \Delta G(\theta)]} \right\}, \tag{4.14}$$

where $\Delta G(\theta)$ is shorthand for $\Delta G(\mathrm{A} \to \mathrm{A}\,(\theta); \mathrm{pH})$ as given by (4.6). Although the conventional single-flip proposal function can be used, wherein θ' is derived from θ by flipping the protonation state of a single randomly chosen site, this can be inefficient when strong correlations exist, leading to low acceptance ratios and thus slow distributional convergence. As a remedy, researchers have supplemented the typical formulation with two- [26, 76, 182] and even three-site moves [159], but the choice of this limit is somewhat arbitrary. One of our objectives in this section therefore is to provide a generalized framework that can accomodate extended multi-site moves in a natural manner. Our method consists of two elements:

1. a partition of the free titrating sites into strongly interacting clusters; and

2. a scheme for proposing multi-site moves within clusters.

Thus, multi-site moves are employed only when needed; this is an attractive feature as their unwarranted use generally leads to less efficient sampling.

To determine cluster assignments, we use an energetic threshold based on the pairwise interaction energies ΔG_{ij}; distance considerations can also be used [205, see], but the interaction energy is more informative. Specifically, we consider two sites i and j as strongly interacting if

$$\max\left\{\left|\Delta G_{ij}\right|, \left|\Delta G_{ji}\right|\right\} \geq \left|\Delta G^*\right| \tag{4.15}$$

for some threshold $|\Delta G^*|$. This defines a coupling graph on the titrating sites, whose connected components we define to be the site clusters. (Single sites uncoupled to any other site are considered their own cluster.) The connected components of a graph can be found easily using any standard breath- or depth-first search [191].

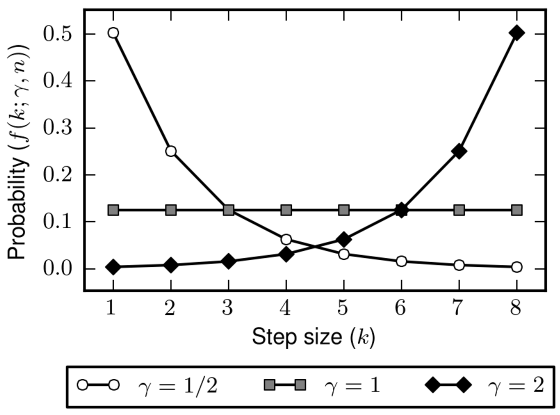

Figure 4.2: Probability density $f(k;\gamma,n)$ of the FGD for $n = 8$ and $\gamma = 1/2$, 1, and 2. For $\gamma = 1$, the FGD is just the uniform distribution.

Within clusters, we propose multi-site moves with a move distance drawn from some appropriate discrete distribution; here, we use a *finite geometric distribution (FGD)*, the natural analogue of the geometric distribution but with finite support. Briefly, for a given cluster with state θ, we consider a proposal density

$$\Pr\left(\theta';\theta\right) \equiv \binom{n}{k}^{-1} f\left(k;\gamma,n\right), \tag{4.16}$$

where n is the dimension of θ (i.e., the number of sites in the cluster), k is the number of sites at which the proposed state θ' differs from θ, and

$$f\left(k;\gamma,n\right) \equiv \left(\frac{1-\gamma}{1-\gamma^n}\right)\gamma^{k-1} \quad \text{for } k = 1,\ldots,n$$

is the probability density of the FGD with decay parameter γ (Figure 4.2). In other words, we choose a proposal distance k from f, then sample the sphere $\{\theta' : |\theta' - \theta| = k\}$ uniformly. This procedure is clearly symmetric, so the Metropolis criterion (4.14) can be applied without modification. The parameter γ is typically taken as $\gamma < 1$ to bias toward local moves, in which case it can be chosen to enforce a desired average move distance by noting that the FGD has mean

$$\mu_{\mathrm{FGD}}(\gamma) = n + \frac{1}{1-\gamma} - \frac{n}{1-\gamma^n},$$

with $\gamma \to 1 - 1/\mu_{\mathrm{FGD}}$ as $n \to \infty$; e.g., the choice $\gamma = 1/2$ corresponds to a mean proposal distance of $\mu_{\mathrm{FGD}} \approx 2$.

At each Monte Carlo step, our full proposal algorithm is then as follows:

1. Select a site cluster to modify at random, weighted by the cluster size.

2. Propose a new state for that cluster via (4.16).

Note 4.6. Clearly, other discrete distributions $f(k)$ on $\{1, \ldots, n\}$ can be used in (4.16). Here, we have chosen the FGD because it is, in some sense, the most natural, especially given that we generally prefer k to be small.

4.5 Estimating the pK_{a}

We have presented a multi-flip MCMC method for sampling the Boltzmann distribution (4.9), seeded by the mean field approximation of §4.2 and accelerated by the reduced site approximation of §4.3. In this section, we assume that this sampling has been performed over a range of pH, leaving only their analysis and the estimation of individual pK_{a} values. We begin by describing how to obtain the distribution of the mean protonation (4.8) for each site i at a given pH, using a scheme similar to that employed by Beroza et al. [26] but based on slightly more sophisticated and robust considerations as outlined by Alan Sokal in [179]. We then show how to estimate each pK_{a} from these quantities, and furthermore how to characterize the distributions of our estimates. This latter contribution appears to be novel and is exceedingly simple, based only on a direct application of the delta method from statistics [152].

Fix the pH and consider only titrating site i for the moment (so that the notation becomes much cleaner), and let $\{\chi_j\}_{j=1}^N$ be a sample of the protonation states θ_i. Then the mean protonation $\langle\theta_i\rangle$ can be estimated by simple averaging as

$$\bar{\chi} \equiv \frac{1}{N}\sum_{j=1}^{N}\chi_j.$$

To estimate the variance of $\langle\theta_i\rangle$, we compute the *integrated autocorrelation time*

$$\tau \equiv \sum_{k=-(N-1)}^{N-1}\rho\left(k\right) = 1 + 2\sum_{k=1}^{N-1}\rho\left(k\right), \tag{4.17}$$

where

$$\rho\left(k\right) \equiv \frac{1}{\sigma_\chi^2}\left(\frac{1}{N-k}\sum_{j=1}^{N-k}\chi_j\chi_{j+k} - \bar{\chi}^2\right) \tag{4.18}$$

is the autocorrelation, for σ_χ^2 the sample variance. The number of independent samples in the data is then approximately N/τ, so we estimate the variance of $\langle\theta_i\rangle$ as

$$\sigma_{\bar{\chi}}^2 \approx \frac{\sigma_\chi^2}{N/\tau}.$$

From (4.18), however, it is easy to see that $\rho(k)$ is increasingly subject to statistical error as k increases due to the diminishing number of samples, so we follow [179] and use instead the windowed analogue

$$\hat{\tau} \equiv 1 + 2\sum_{k=1}^{M}\rho\left(k\right) \tag{4.19}$$

of (4.17), where ideally M is chosen such that $\hat{\tau} \ll M \ll N$. In practice, we compute $\hat{\tau}$ for various values of M, and use the first M such that the consistency criterion $M \geq c\hat{\tau}$ is satisfied, for, e.g., $c = 4$ [see 179].

Once this has been done for each pH, we then estimate the pK_a of site i, denoted pK_i, as the pH at which $\bar{\chi} = 1/2$ by linear interpolation, cf. (4.1). This is generally considered the standard protocol, which we now improve upon by using the distributions of $\langle \theta_i \rangle$ to produce a distribution for pK_i. For this, first recall that our estimate of $\langle \theta_i \rangle \sim \mathcal{N}(\bar{\chi}, \sigma_{\bar{\chi}}^2)$ is normally distributed by the central limit theorem, so let us consider two data points (x_j, Y_j) for $j = 1, 2$, where each $Y_j \sim \mathcal{N}(y_j, \sigma_j^2)$. Using linear interpolation on the means y_j then yields

$$y = \left(\frac{x_2 - x}{x_2 - x_1} \right) y_1 + \left(\frac{x - x_1}{x_2 - x_1} \right) y_2,$$

which can be inverted to give

$$x = \left(\frac{y - y_2}{y_1 - y_2} \right) x_1 + \left(\frac{y_1 - y}{y_1 - y_2} \right) x_2.$$

Applying this to the distribution data, we hence have

$$X = h\left(Y\right) \equiv \left(\frac{y - Y_2}{Y_1 - Y_2} \right) x_1 + \left(\frac{Y_1 - y}{Y_1 - Y_2} \right),$$

where

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix} \right) \equiv \mathcal{N}\left(\mu_Y, \sigma_Y^2\right).$$

Therefore, by the delta method [152], X has the asymptotic distribution

$$X \sim \mathcal{N}\Big(h\left(\mu_Y\right), \nabla h\left(\mu_Y\right)^\mathsf{T} \sigma_Y^2 \nabla h\left(\mu_Y\right) \Big) \equiv \mathcal{N}\left(\mu_X, \sigma_X^2\right),$$

where

$$\mu_X = \left(\frac{y - y_2}{y_1 - y_2} \right) x_1 + \left(\frac{y_1 - y}{y_1 - y_2} \right) x_2, \tag{4.20a}$$

$$\sigma_X^2 = \left[\left(y - y_2\right) \sigma_1^2 + \left(y_1 - y\right) \sigma_2^2 \right] \frac{x_2 - x_1}{\left(y_2 - y_1\right)^2}. \tag{4.20b}$$

It is immediate that this can be used to estimate the distribution of pK_i by identifying x with the pH and y with $\langle \theta_i \rangle$. Repeating this for all sites completes the computation.

4.6 Algorithm

We now have all the ingredients necessary to describe the full pK_a calculation algorithm, which, for simplicity, is divided into four phases:

1. preprocessing, including protein preparation, titrating site identification, charge assignment, and molecular surface triangulation;
2. energy precomputation, comprising the compression of the electrostatic potential matrix W, the computation of the site interaction energies ΔG_{ij}, and the calculation of the intrinsic pK_a values for each site;
3. Monte Carlo sampling, to draw protein protonation states from the Boltzmann distribution (4.9) at each pH; and
4. postprocessing, to derive from the Monte Carlo data the pK_i.

Preprocessing

We use protein structures from the Protein Data Bank (PDB) [25]. These are typically not directly suitable for pK_a calculations as they contain only the coordinates of heavy atoms; moreover, they can contain waters or inorganic ions. Thus, we first prepare them by stripping all non-standard residues, and then adding and optimizing the locations of all hydrogens using PDB2PQR [66]. If a protein has multiple conformations, only the primary one (A-form) is considered. For each atom, we assign a partial charge and an atomic radius using PARSE parameters [177]. We consider only the residues Arg, Asp, Cys, Glu, His, Lys, and Tyr as titrable; we hence ignore the titration of the C- and N-termini. Only non-bridged Cys are titrated, and we assume that the unprotonated form of His has a hydrogen on the ϵ-nitrogen (i.e.,

Table 4.1: Model pK_a values for each titratable residue at temperature $T = 25$ °C.

residue	pK_a^0
Arg	12.0
Asp	4.0
Cys	9.5
Glu	4.4
His	6.3
Lys	10.4
Tyr	9.6

the HIE form). Model pK_a values at $T = 25$ °C are taken from [151] (Table 4.1), with model compound structures chosen as the extractions of the relevant residues from the protein. Molecular surfaces are triangulated using MSMS [169] with a probe radius of 1.4 Å and vertex densities of either 1.0 or 0.5 Å^{-2} for smaller or larger proteins, respectively (see Table 4.3). In total, $N_\mathrm{titr} + 1$ surfaces are generated: one for the protein as a whole, and one for the model compound of each titrating site.

Energy precomputation

For each molecular geometry, we compress the electrostatic potential matrix W by compression of its constituent matrices A^{-1}, B, C, and D, cf. (4.4) and §1.2; the compression precision is set at $\epsilon = 10^{-3}$. Unless otherwise specified, we take $\varepsilon_0 = 80$ and $\varepsilon_1 = 20$ for the dielectric constants of the solvent and the protein, respectively. This choice of ε_1 is somewhat higher than the commonly accepted value of $\varepsilon_1 = 4$

[80] and is used to model the effects of minor pH-dependent conformational changes [see, e.g., 8, 115]. The default ionic strength is 0.1 M, corresponding to a Debye screening length of $\kappa^{-1} = 10$ Å. For each titrating site, we calculate the protonation energies $\Delta G_{\mathrm{ele}}(\mathrm{A_p} \to \mathrm{A_pH})$ and $\Delta G_{\mathrm{ele}}(\mathrm{A_s} \to \mathrm{A_sH})$ in the protein and in the model compound, respectively, which give the pK_i^0 via (4.3) (using $T = 25$ °C). Furthermore, we precompute the interaction energies (4.7) in the protein, in anticipation of their extensive use in each Monte Carlo run.

Monte Carlo sampling

At each pH, we perform the following operations:

1. Use the reduced site approximation to compute $p_{\mathrm{min},i}$ and $p_{\mathrm{max},i}$ for each site i via (4.12) and (4.13), respectively. Fix its protonation state if possible using $p^*_{\mathrm{min}} = 0.99$ and $p^*_{\mathrm{max}} = 0.01$.

2. Find titrating site clusters among the remaining N_{free} sites using an energy threshold of one pK_{a} unit, i.e., $|\Delta G^*| = 1.37$ kcal/mol, in (4.15).

3. Run the mean field iteration (4.10) to determine an appropriate initial MCMC state by (4.11).

We then perform the actual Monte Carlo simulation, where at each step a new state is proposed following §4.4 and accepted according to (4.14). All state transitions are processed at $T = 25$ °C. The free energy is computed via (4.6) using the precomputed ΔG_{ij}. The decay parameter in the FGD is set at $\gamma = 1/2$. We take 1000 Monte Carlo passes for each sample, i.e., $1000 N_{\mathrm{free}}$ steps. This is repeated for each pH over the range $-6 \leq \mathrm{pH} \leq 20$ in increments of 0.5 pH units.

Postprocessing

Finally, the Monte Carlo data are postprocessed using the techniques of §4.5. Specifically, the distribution of $\langle\theta_i\rangle$ at each pH is estimated using the consistency parameter $c = 4$ in (4.19), and then the distribution of each pK_i is estimated via (4.20).

4.7 Numerical results

We implemented the above algorithm using a mix of Fortran and Python, relying on the former for the heavy number crunching (energy precomputation and Monte Carlo sampling) and the latter to drive the overall calculation. Following [128], we applied our methods to five well-studied proteins: bovine pancreatic trypsin inhibitor (BPTI, PDB ID: 4PTI [136]), turkey ovomucoid third domain (OMTKY3, PDB ID: 2OVO [27]), hen egg white lysozyme (HEWL, PDB ID: 2LZT [161]), RNase H (PDB ID: 3NR3 [111]), and RNase A (PDB ID: 2RN2 [120]). These are summarized briefly in Table 4.2. For the larger proteins ($N_{\text{res}} \gtrsim 100$ residues), MSMS sometimes returned triangles with zero area; these were removed from the mesh before proceeding further.

Algorithmic acceleration

Numerical data for each protein titration with respect to the acceleration provided by the algorithm are shown in Table 4.3. It is evident that the fast direct solver was very successful at reducing the energy precomputation time; the estimated speedup over classical methods is ~ 200 (after matrix compression and factorization). However, the cost of matrix compression remained high and was, in fact, several orders greater than that of calculating the energies in all cases. This suggests a fundamental imbalance

Table 4.2: Summary statistics for titrated proteins: N_{res}, number of residues; N_{titr}, number of titrating sites (according to Table 4.1); N_{src}, number of atoms.

name	PDB ID	N_{res}	N_{titr}	N_{src}
BPTI	4PTI	58	18	891
OMTKY3	2OVO	56	15	813
HEWL	2LZT	129	30	1965
RNase A	3RN3	124	34	1865
RNase H	2RN2	155	53	2474

Table 4.3: Numerical data for protein titration: density, triangulation vertex density (Å^{-2}); N_{tri}, number of triangles in protein surface triangulation; T_{cm}, matrix compression time (A) in protein (s); T_{sv}, inverse application time (A^{-1}) in protein (s); T_{nrg}, total energy calculation time after matrix precomputation (s); M, required storage for compressed matrix (A) in protein (MB); r_{free}, average fraction of free sites to titrating sites over all pH sampled.

name	density	N_{tri}	T_{cm}	T_{sv}	T_{nrg}	M	r_{free}
BPTI	1.0	7402	2.5E+3	9.9E−2	1.7E+0	1.5E+2	0.20
OMYTK3	1.0	7278	2.5E+3	1.2E−1	1.5E+0	1.6E+2	0.21
HEWL	0.5	9652	3.3E+3	1.3E−1	4.3E+0	2.1E+2	0.21
RNase A	0.5	9426	3.4E+3	1.4E−1	4.8E+0	2.1E+2	0.25
RNase H	0.5	13014	5.7E+3	2.4E−1	1.3E+1	3.4E+2	0.28

that may be better addressed by other algorithmic tools (see §4.8). Nonetheless, the compression afforded by our current scheme dramatically cuts the memory requirement and therefore allows much larger problems to be pursued, directly translating to a more faithful representation of the molecular geometry and hence to a more accurate result. For RNase H, for instance, the amount of memory required to simply store the matrix A in the protein is about 5.5 GB without compression, but only 340 MB with it. Furthermore, we find the reduced site approximation to be extremely powerful, leading to a uniform four- to five-fold acceleration in the Monte Carlo phase essentially without sacrificing any accuracy (especially since the interpolation of the final pK_a values are local). More detailed data for each protein are given in Figures 4.3–4.7, from which we see that the approximation efficiently pruned out rare titration events near the pH extremes. Also shown are the protein titration curves, which reflect the standard sigmoidal shape.

We moreover emphasize the acceptance ratios for MCMC transitions, which were generally very satisfactory. Only a few stagnation points were observed, and these at pH values for which most, if not all, sites were close to being fixed, i.e., the difficulties of transition were *physical*. Such cases typically occurred in the $5 < \mathrm{pH} < 8$ range, which is exactly intermediate to the two pK_a^0 clusters with Asp and Glu on the low end, and Arg, Cys, Lys, and Tyr on the high end (Table 4.1); His falls exactly within this range, but they were generally rare and so did not have much effect. Our Monte Carlo sampling procedure therefore appears quite efficient, though there were not enough multisite clusters to test the FGD-based scheme extensively.

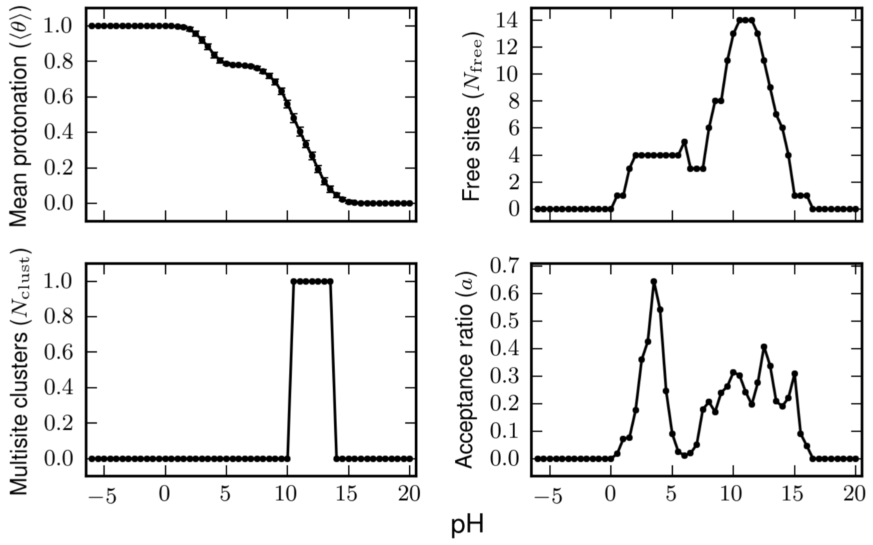

Figure 4.3: Numerical results for titrating BPTI, showing the mean protein protonation $\langle\theta\rangle$ (with standard error), the number N_{free} of free sites, the number N_{clust} of multisite clusters, and the Monte Carlo acceptance ratio a as a function of pH.

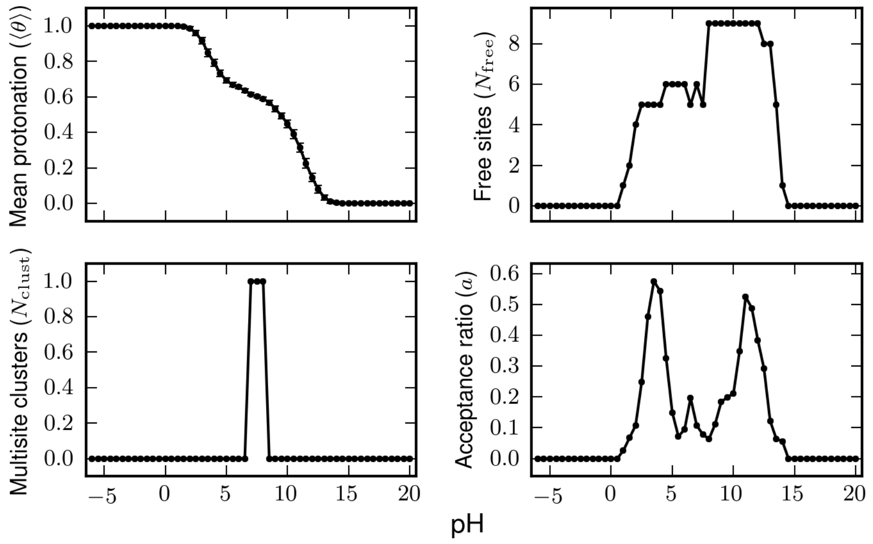

Figure 4.4: Numerical results for titrating OMTKY3; notation as in Figure 4.3.

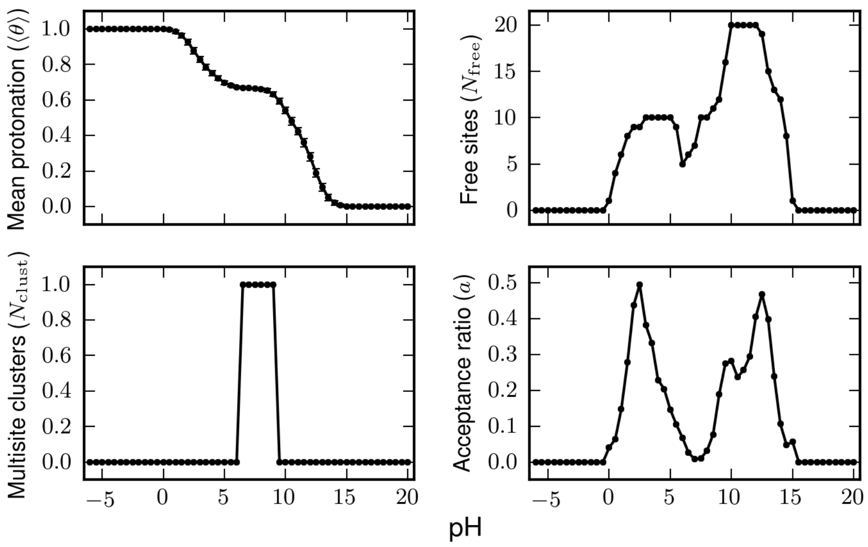

Figure 4.5: Numerical results for titrating HEWL; notation as in Figure 4.3.

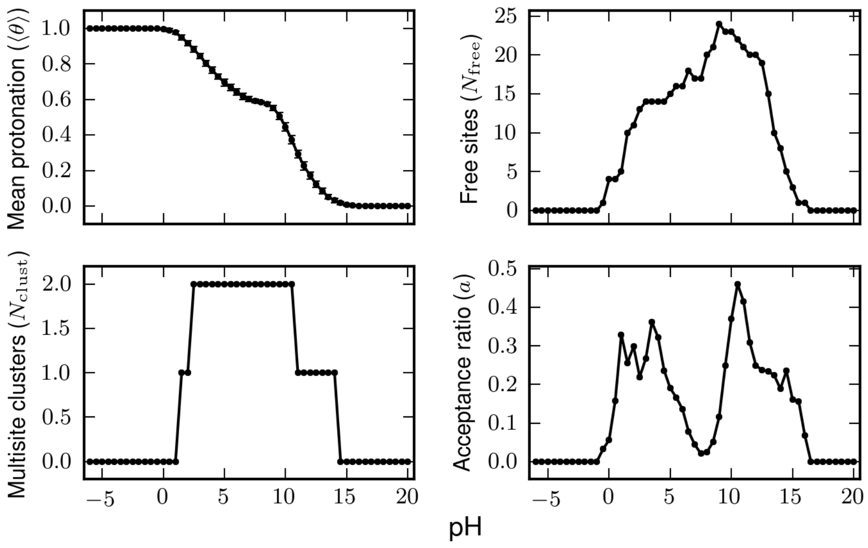

Figure 4.6: Numerical results for titrating RNase A; notation as in Figure 4.3.

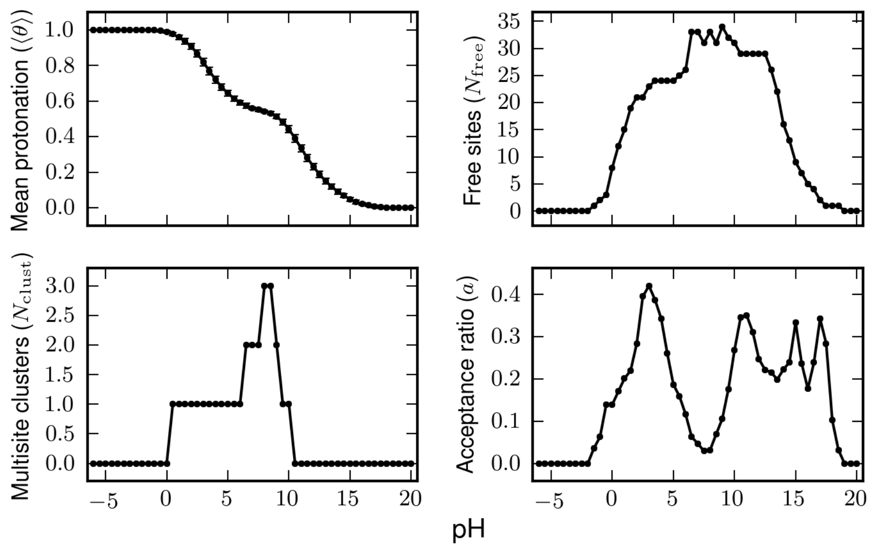

Figure 4.7: Numerical results for titrating RNase H; notation as in Figure 4.3.

Prediction accuracy

We also briefly studied the quality of our p$K_{\rm a}$ calculations by comparing against experimental data as reproduced in [128]. For this, we ran our code using three different protein dielectrics: $\varepsilon_1 = 4$, 8, and 20, corresponding to increasing implicit conformational flexibility. The calculated p$K_{\rm a}$ values for each protein are presented in Tables 4.4–4.8, along with the root mean square deviation (RMSD) for each ε_1. It is immediate that our sampling error is very low, in many cases less than 0.1 protons, as determined by the method of §4.5. Thus, we can be confident that our estimates have converged, though, of course, they may be biased due to the physical or computational model. For all proteins but one (OMTKY3), the RMSD is smallest for $\varepsilon_1 = 20$, sometimes by large margins (HEWL, RNase A, RNase H); for OMTKY3, the smallest RMSD is achieved at $\varepsilon_1 = 8$, but the difference with that for $\varepsilon_1 = 20$ is

Table 4.4: Calculated pK_a values for BPTI at various protein dielectrics $\varepsilon_1 = 4$, 8, and 20, compared against experiment (expt). The standard error for each calculated value is given in parentheses; the best matching pK_a prediction for each site is marked in bold. The RMSD is also shown for each ε_1, with the number of sites in each average given in parentheses.

residue	expt	$\varepsilon_1 = 4$	$\varepsilon_1 = 8$	$\varepsilon_1 = 20$
Arg 1		17.09 (0.07)	15.47 (0.07)	14.21 (0.05)
Asp 3	3.6	3.19 (0.05)	3.45 (0.05)	**3.51** (0.06)
Glu55	3.9	5.14 (0.05)	4.25 (0.05)	**3.65** (0.06)
Tyr10	9.4	9.64 (0.06)	**9.39** (0.05)	9.13 (0.05)
Lys15	10.4	10.73 (0.05)	10.72 (0.05)	**10.59** (0.05)
Arg17		12.30 (0.04)	12.25 (0.05)	12.11 (0.04)
Arg20		11.38 (0.07)	12.33 (0.05)	12.80 (0.06)
Tyr21	10.0	10.60 (0.05)	**10.02** (0.05)	9.67 (0.05)
Tyr23	11.0	13.32 (0.09)	**11.30** (0.08)	10.28 (0.08)
Lys26	10.1	**10.38** (0.04)	**10.38** (0.05)	10.41 (0.05)
Tyr35	10.6	6.39 (0.05)	7.45 (0.05)	**8.08** (0.05)
Arg39		12.10 (0.05)	12.21 (0.05)	12.23 (0.05)
Lys41	10.6	**10.66** (0.07)	10.94 (0.05)	11.02 (0.05)
Arg42		12.96 (0.05)	12.77 (0.05)	12.52 (0.05)
Lys46	9.9	**10.11** (0.05)	10.13 (0.06)	10.24 (0.05)
Glu49	4.0	3.80 (0.05)	**3.92** (0.05)	3.90 (0.05)
Asp50	3.2	**2.61** (0.05)	2.54 (0.05)	2.44 (0.05)

Table 4.4: Calculated pK_a values for BPTI (continued).

residue	expt	$\varepsilon_1 = 4$	$\varepsilon_1 = 8$	$\varepsilon_1 = 20$
Arg53		11.44 (0.05)	12.12 (0.05)	12.64 (0.04)
RMSD		1.47 (12)	0.96 (12)	**0.82** (12)

negligible (1.07 v. 1.09). This is consistent with previous findings advocating a higher protein dielectric [8, 115].

Our predictions are highly accurate for BPTI, HEWL, and RNase A (RMSD ≤ 1), but suffer somewhat for OMTKY3 and RNase H. Of these, RNase H proved particularly difficult (RMSD = 1.36), with the algorithm performing especially poorly for Asp10, Asp70, and Glu129. The latter two can evidently be attributed to strong structural relaxations beyond that captured by our artificially high choice of ε_1. Indeed, much more favorable results were reported by Georgescu, Alexov, and Gunner [76], who explicitly modeled sidechain flexibility, but significantly not by other calculations using only rigid structures [149], which gave very similar values to ours. For all four proteins considered, our overall results are generally comparable with those of previous methods based on Poisson-Boltzmann electrostatics [7, 8, 76, 115, 149, 182, 205, 206], and also those based on other techniques [128, 153, and references therein]. The combined data over all proteins are summarized in Figure 4.8 and Table 4.9.

Table 4.5: Calculated pK_a values for OMTKY3 at various protein dielectrics; notation as in Table 4.4.

residue	expt	$\varepsilon_1 = 4$	$\varepsilon_1 = 8$	$\varepsilon_1 = 20$
Asp 7	2.4	1.67 (0.06)	**2.48** (0.05)	2.89 (0.05)
Glu10	4.1	4.22 (0.05)	**4.18** (0.05)	**4.18** (0.05)
Tyr11	10.2	14.13 (0.07)	11.23 (0.14)	**9.40** (0.05)
Lys13	9.9	**10.86** (0.09)	11.26 (0.10)	11.71 (0.05)
Glu19	3.2	4.93 (0.05)	3.88 (0.05)	**3.28** (0.05)
Tyr20	11.1	8.59 (0.06)	**8.86** (0.07)	**8.86** (0.06)
Arg21		11.44 (0.04)	11.60 (0.04)	12.45 (0.05)
Asp21	2.2	5.07 (0.05)	**2.78** (0.05)	3.64 (0.05)
Lys29	11.1	**11.12** (0.04)	11.06 (0.04)	11.15 (0.06)
Tyr31	$>$ 12.5	16.13 (0.07)	13.43 (0.05)	11.57 (0.07)
Lys34	10.1	11.71 (0.05)	11.43 (0.06)	**11.42** (0.06)
Glu43	4.8	**4.42** (0.05)	3.56 (0.05)	4.29 (0.05)
His52	7.5	**6.89** (0.05)	6.23 (0.04)	6.53 (0.05)
Lys55	11.1	10.82 (0.07)	10.78 (0.07)	**10.91** (0.06)
Cys56		17.33 (0.06)	14.01 (0.05)	10.93 (0.06)
RMSD		1.77 (12)	**1.07** (12)	1.09 (12)

Table 4.6: Calculated pK_a values for HEWL at various protein dielectrics; notation as in Table 4.4.

residue	expt	$\varepsilon_1 = 4$	$\varepsilon_1 = 8$	$\varepsilon_1 = 20$
Lys 1	10.6	9.04 (0.05)	9.75 (0.04)	**10.16** (0.04)
Arg 5		11.42 (0.05)	12.06 (0.05)	12.30 (0.05)
Glu 7	2.9	−1.44 (0.05)	0.72 (0.05)	**2.11** (0.04)
Lys 13	10.3	9.68 (0.04)	9.83 (0.04)	**10.04** (0.04)
Arg 14		10.77 (0.05)	11.67 (0.05)	12.30 (0.05)
His 15	5.6	1.72 (0.05)	3.93 (0.05)	**5.19** (0.05)
Asp 18	2.7	−0.24 (0.06)	1.28 (0.05)	**2.27** (0.05)
Tyr 20	10.3	14.78 (0.04)	12.19 (0.06)	**9.90** (0.05)
Arg 21		11.03 (0.06)	12.41 (0.06)	12.84 (0.06)
Tyr 23	9.8	11.09 (0.05)	**10.07** (0.05)	9.48 (0.06)
Lys 33	10.4	7.61 (0.04)	8.77 (0.04)	**9.53** (0.04)
Glu 35	6.2	**4.93** (0.05)	4.86 (0.05)	4.49 (0.05)
Arg 45		10.76 (0.05)	11.54 (0.06)	12.45 (0.06)
Asp 48	< 2.5	−2.20 (0.05)	0.21 (0.05)	1.74 (0.05)
Asp 52	3.7	−0.30 (0.05)	1.42 (0.05)	**2.35** (0.06)
Tyr 53	12.1	$>$ 20.00 ( )	15.30 (0.06)	**11.12** (0.07)
Arg 61		10.07 (0.05)	11.80 (0.05)	12.85 (0.06)
Asp 66	< 2.0	6.00 (0.07)	4.64 (0.04)	**3.59** (0.05)
Arg 68		11.95 (0.10)	12.33 (0.15)	13.27 (0.05)
Arg 73		10.19 (0.05)	11.53 (0.05)	12.29 (0.05)

Table 4.6: Calculated pK_a values for HEWL (continued).

residue	expt	$\varepsilon_1 = 4$	$\varepsilon_1 = 8$	$\varepsilon_1 = 20$
Asp 87	2.1	**2.28** (0.06)	2.52 (0.05)	2.60 (0.05)
Lys 96	10.7	9.23 (0.06)	9.94 (0.05)	**10.86** (0.05)
Lys 97	10.1	12.06 (0.04)	11.68 (0.05)	**11.53** (0.05)
Asp101	4.1	5.42 (0.05)	**4.31** (0.05)	3.58 (0.05)
Arg112		8.52 (0.06)	10.87 (0.05)	12.07 (0.05)
Arg114		12.45 (0.05)	12.50 (0.04)	12.61 (0.04)
Lys116	10.2	9.25 (0.05)	9.60 (0.04)	**9.99** (0.05)
Asp119	3.2	2.85 (0.05)	2.86 (0.04)	**2.88** (0.05)
Arg125		10.78 (0.04)	11.79 (0.05)	12.38 (0.05)
Arg128		12.02 (0.04)	12.04 (0.05)	11.93 (0.05)
RMSD		2.52 (16)	1.49 (17)	**0.79** (17)

4.8 Summary

In this chapter, we have described a procedure for the theoretical calculation of protein pK_a values and demonstrated its acceleration using the fast direct solver of Chapter 2. We have also provided various enhancements to the conventional algorithm by implementing generalized multi-site transitions and propagating sampling errors to our predictions; the former was shown to be effective over the few cases tested, while the latter allowed us to assess the convergence of our Monte Carlo sampling. Overall, our direct solver was very efficient at computing the interaction energies (4.7), though the cost of the requisite matrix precomputations became somewhat prohibitive for

Table 4.7: Calculated pK_a values for RNase A at various protein dielectrics; notation as in Table 4.4.

residue	expt	$\varepsilon_1 = 4$	$\varepsilon_1 = 8$	$\varepsilon_1 = 20$
Lys 1		10.12 (0.04)	10.30 (0.04)	10.51 (0.05)
Glu 2	2.8	−4.50 (0.04)	−0.95 (0.04)	**1.26** (0.05)
Lys 7		10.15 (0.06)	10.46 (0.05)	10.50 (0.06)
Glu 9	4.0	2.87 (0.04)	3.56 (0.04)	**3.94** (0.05)
Arg 10		14.21 (0.04)	14.26 (0.04)	14.35 (0.05)
His 12	6.2	9.84 (0.21)	6.91 (0.09)	**5.99** (0.06)
Asp 14	< 2.0	4.29 (0.11)	3.07 (0.22)	**1.64** (0.08)
Tyr 25		> 20.00 ( )	15.87 (0.05)	12.04 (0.05)
Lys 31		9.31 (0.04)	9.73 (0.05)	10.20 (0.05)
Arg 33		9.69 (0.15)	11.71 (0.05)	13.21 (0.05)
Lys 37		11.58 (0.04)	11.16 (0.05)	10.83 (0.05)
Asp 38	3.5	1.26 (0.04)	2.31 (0.04)	**2.86** (0.04)
Arg 39		11.38 (0.04)	12.14 (0.05)	12.86 (0.05)
Lys 41		4.37 (0.13)	8.03 (0.08)	9.74 (0.06)
His 48	6.0	< -6.00 ( )	−0.20 (0.10)	**6.01** (0.10)
Glu 49	4.7	5.63 (0.04)	**5.08** (0.06)	4.30 (0.05)
Asp 53	3.9	3.18 (0.04)	3.39 (0.05)	**3.53** (0.05)
Lys 61		10.41 (0.06)	10.76 (0.06)	11.10 (0.04)
Lys 66		10.32 (0.05)	10.60 (0.05)	10.71 (0.05)
Tyr 73		13.47 (0.06)	12.52 (0.10)	11.37 (0.08)

Table 4.7: Calculated pK_a values for RNase A (continued).

residue	expt	$\varepsilon_1 = 4$	$\varepsilon_1 = 8$	$\varepsilon_1 = 20$
Tyr 76		9.63 (0.06)	9.67 (0.05)	9.73 (0.06)
Asp 83	3.5	**2.71** (0.05)	2.19 (0.05)	1.92 (0.04)
Arg 85		9.46 (0.05)	11.42 (0.05)	12.54 (0.05)
Glu 86	4.1	1.84 (0.04)	2.97 (0.04)	**3.47** (0.04)
Lys 91		11.48 (0.05)	11.11 (0.05)	10.88 (0.05)
Tyr 92		9.93 (0.04)	10.07 (0.06)	10.04 (0.06)
Tyr 97		< -6.00 ( )	15.18 (0.05)	11.44 (0.07)
Lys 98		10.20 (0.04)	10.18 (0.04)	10.22 (0.05)
Lys104		8.14 (0.04)	9.22 (0.05)	9.84 (0.05)
His105	6.7	2.27 (0.04)	4.18 (0.04)	**5.26** (0.05)
Glu111	3.5	1.62 (0.05)	2.98 (0.04)	**3.66** (0.05)
Tyr115		16.60 (0.07)	13.44 (0.11)	11.67 (0.08)
His119	6.1	2.42 (0.10)	6.01 (0.07)	**6.07** (0.10)
Asp121	3.1	5.92 (0.11)	**2.08** (0.07)	2.02 (0.05)
RMSD		3.22 (12)	2.25 (13)	**0.85** (13)

larger surface meshes. This is especially relevant since preliminary investigations with molecular representations hinted that our accuracy improves appreciably with surface refinement. Since the total number of potential solves is relatively small (only N_titr as opposed to the N_tri governing all matrix calculations), it may therefore be more profitable to use instead an iterative scheme driven by a compression-based FMM

Table 4.8: Calculated pK_a values for RNase H at various protein dielectrics; notation as in Table 4.4.

residue	expt	$\varepsilon_1 = 4$	$\varepsilon_1 = 8$	$\varepsilon_1 = 20$
Lys 3		10.45 (0.04)	10.69 (0.05)	10.81 (0.06)
Glu 6	4.5	-1.77 (0.05)	0.98 (0.06)	**2.72** (0.06)
Asp 10	6.1	-2.04 (0.04)	0.39 (0.08)	**2.44** (0.11)
Cys 13		18.68 (0.04)	14.30 (0.05)	11.13 (0.05)
Tyr 22		15.90 (0.11)	12.87 (0.11)	11.15 (0.10)
Arg 27		14.01 (0.04)	14.53 (0.04)	14.97 (0.05)
Tyr 28		16.91 (0.07)	13.67 (0.07)	10.80 (0.07)
Arg 29		12.36 (0.04)	12.56 (0.05)	12.91 (0.06)
Arg 31		11.96 (0.04)	11.93 (0.05)	12.14 (0.05)
Glu 32	3.6	-0.61 (0.04)	1.43 (0.05)	**2.56** (0.06)
Lys 33		6.56 (0.04)	8.17 (0.04)	9.36 (0.05)
Tyr 39		11.95 (0.14)	11.34 (0.11)	10.80 (0.09)
Arg 41		9.39 (0.11)	11.69 (0.06)	12.90 (0.05)
Arg 46		> 20.00 ( )	19.36 (0.05)	16.68 (0.05)
Glu 48	4.4	0.69 (0.04)	2.08 (0.05)	**2.67** (0.07)
Glu 57	3.2	-1.89 (0.05)	0.77 (0.05)	**2.36** (0.05)
Lys 60		10.62 (0.04)	10.68 (0.05)	10.99 (0.05)
Glu 61	3.9	**3.85** (0.04)	3.75 (0.05)	3.26 (0.05)
His 62	7.0	7.89 (0.06)	7.25 (0.06)	**6.91** (0.05)
Cys 63		> 20.00 ( )	18.50 (0.05)	13.65 (0.05)

Table 4.8: Calculated pK_a values for RNase H (continued).

residue	expt	$\varepsilon_1 = 4$	$\varepsilon_1 = 8$	$\varepsilon_1 = 20$
Glu 64	4.4	1.27 (0.04)	2.85 (0.05)	**3.62** (0.05)
Asp 70	2.6	5.46 (0.08)	5.62 (0.08)	**4.43** (0.11)
Tyr 73		13.11 (0.07)	10.61 (0.10)	8.84 (0.06)
Arg 75		9.64 (0.04)	11.28 (0.04)	12.26 (0.05)
His 83	5.5	4.17 (0.04)	5.00 (0.04)	**5.42** (0.04)
Lys 86		11.13 (0.04)	11.40 (0.05)	11.40 (0.05)
Lys 87		10.85 (0.04)	11.78 (0.05)	12.37 (0.05)
Arg 88		10.85 (0.04)	11.78 (0.05)	12.37 (0.05)
Lys 91		10.55 (0.05)	10.61 (0.06)	10.49 (0.06)
Asp 94	3.2	**2.98** (0.04)	2.88 (0.04)	2.96 (0.05)
Lys 95		8.62 (0.04)	9.46 (0.04)	10.02 (0.05)
Lys 96		9.96 (0.04)	10.50 (0.04)	10.69 (0.04)
Lys 99		7.69 (0.05)	9.14 (0.09)	10.90 (0.07)
Asp102	< 2.0	-2.20 (0.09)	0.19 (0.05)	1.58 (0.05)
Arg106		14.15 (0.04)	14.62 (0.05)	14.72 (0.05)
Asp108	3.2	2.51 (0.04)	**2.79** (0.05)	2.78 (0.04)
His114	5.0	-4.61 (0.05)	0.75 (0.05)	**4.61** (0.04)
Lys117		11.09 (0.04)	11.31 (0.04)	11.33 (0.05)
Glu119	4.1	2.33 (0.14)	2.72 (0.07)	**3.04** (0.06)
Lys122		9.86 (0.04)	10.09 (0.05)	10.30 (0.05)
His124	7.1	1.92 (0.04)	4.30 (0.05)	**5.51** (0.05)

Table 4.8: Calculated pK_a values for RNase H (continued).

residue	expt	$\varepsilon_1 = 4$	$\varepsilon_1 = 8$	$\varepsilon_1 = 20$
His127	7.9	-0.81 (0.10)	3.83 (0.07)	**6.08** (0.05)
Glu129	3.6	-3.92 (0.07)	-0.81 (0.05)	**1.09** (0.06)
Glu131	4.3	1.14 (0.04)	2.76 (0.06)	**3.75** (0.06)
Arg132		10.71 (0.04)	12.24 (0.04)	13.39 (0.06)
Cys133		> 20.00 ( )	18.89 (0.05)	13.53 (0.06)
Asp134	4.1	4.82 (0.09)	**4.32** (0.10)	3.31 (0.07)
Glu135	4.3	3.66 (0.04)	**4.16** (0.04)	4.13 (0.05)
Arg138		14.76 (0.04)	14.97 (0.04)	14.89 (0.05)
Glu147	4.2	4.35 (0.04)	4.22 (0.05)	**4.19** (0.04)
Asp148	< 2.0	< -6.00 ( )	-2.78 (0.06)	0.28 (0.05)
Tyr151		12.39 (0.14)	10.40 (0.05)	9.75 (0.06)
Glu154	4.4	2.78 (0.04)	3.35 (0.04)	**3.74** (0.05)
RMSD		4.53 (22)	2.53 (22)	**1.36** (22)

(§3.5), for which the compression time (and memory requirement) should decrease substantially. Work in this direction is now forthcoming.

Although the accuracies that we have presently reported are respectable (provided that $\varepsilon_1 = 20$), they are still quite far from the limits imposed by the experimental data. As noted briefly above, an attractive remedy is to include conformational flexibility [2, 76, 159, 182, 198]. This can be done in a number of ways, but the approach of Gunner et al. [2, 76, 182] is particularly suited to our methods, as they

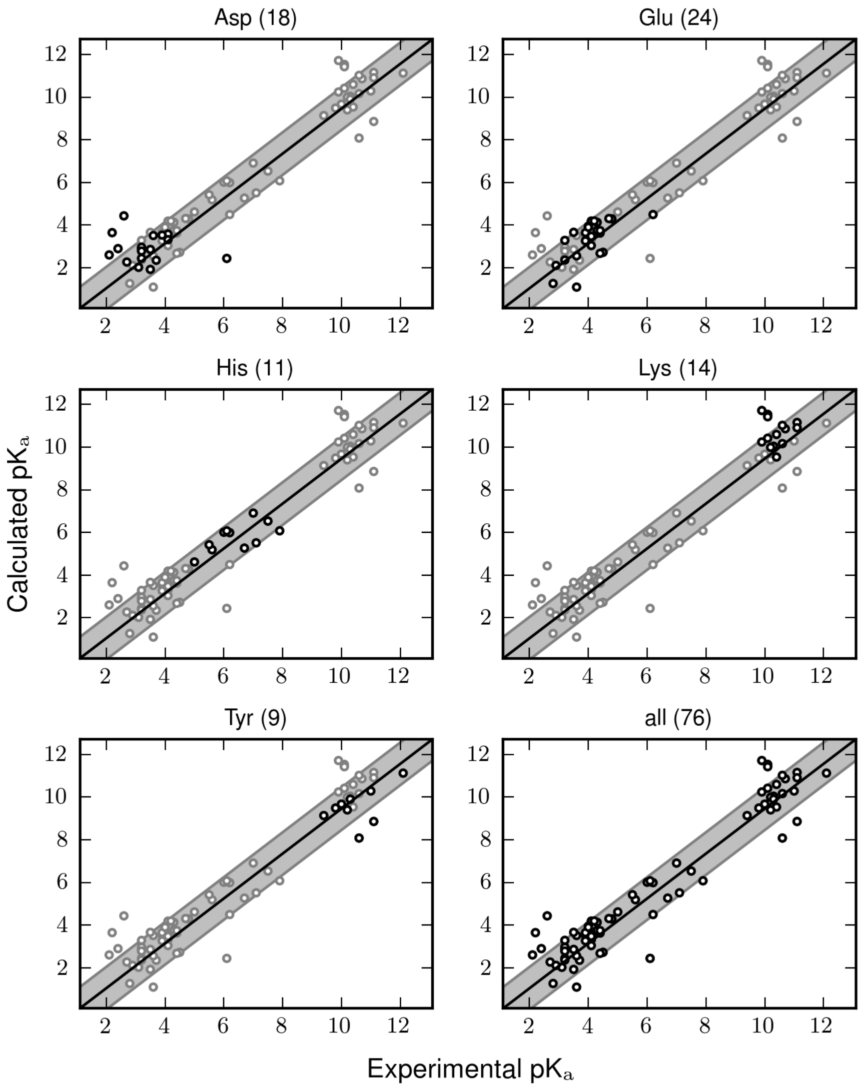

Figure 4.8: Accuracy of pK_a calculations, grouped by residue type. Each plot shows the calculated pK_a values as a function of their experimental values for the group of interest (dark outline), along with the identity transformation (black) and a ± 1 pK_a unit region (gray). The number of experimental values for each group is given in parentheses.

Table 4.9: Summary statistics for calculated pK_a values, grouped by residue type: n, number of experimental values; n_close, number of 'close' predictions within 1 pK_a unit.

type	n	n_close	RMSD
Asp	18	12	1.23
Glu	24	17	1.00
His	11	8	0.92
Lys	14	11	0.79
Tyr	9	7	1.24
all	76	55	1.05

rely on local resampling about an otherwise rigid structure. This hence can be treated using precisely the perturbative techniques of §3.3. The energy state space is now also significantly larger (the number of solves is equal to the total number of conformers across both titratable and non-titratable residues [see 182]), which gives a further advantage to the direct solver as the precomputation time can be more efficiently amortized. For a review of current progress in protein pK_a predictions, see [1].

Finally, we mention some very recent work on nonlocal electrostatic models, which have been shown to better reproduce experimental solvation free energies while partially reconciling the inconsistency of the protein dielectric [14, 105, 106]. Particular formulations of such models can be accomodated using very similar integral equation methods that can be treated numerically with our techniques as well [see 19]. Their application to pK_a calculations are much anticipated.

5 Generalizations and concluding remarks

In this dissertation, we have:

1. presented a well-conditioned second-kind boundary integral formulation of the linearized Poisson-Boltzmann equation (LPBE) for continuum molecular electrostatics (Chapter 1);

2. developed a fast direct solver based on multilevel matrix compression for its efficient numerical solution (Chapter 2);

3. outlined several practical and important extensions of the direct solver methodology (Chapter 3); and

4. applied our techniques to the calculation of protein pK_a values (Chapter 4).

Our main contribution is the fast direct solver, which is general and can treat a wide variety of non-oscillatory integral operators from mathematical physics. Thus, it should prove a useful tool in many areas of computational science and engineering. Its principal feature is its extremely fast solve times following precomputation, often beating classical techniques or even accelerated schemes like the fast multipole method by several orders of magnitude. Hence, our algorithm is particularly efficient for problems involving multiple right-hand sides; such is the case with the calculation of protein pK_a values, and our results reflect its efficiency in this regard. Still, our scheme can become unwieldy for large problems due to the non-optimal complexity of the precomputation phase in higher dimensions; in the protein electrostatics setting, this is $\mathcal{O}(N^{3/2})$, where N is the system size. Thus, an outstanding issue is the development of fast direct solvers with optimal or near-optimal complexities. At the same time, this precomputation cost may be forgiven if the number of solves required is exceedingly large (say, ~ 1000); therefore, it is also of interest to identify important scientific problems for which this is true.

In this concluding chapter, we offer some insights toward both concerns. We begin by surveying other biophysical problems to which the direct solver may be effectively applied, including biomolecular charge optimization, protein structure prediction and design, and protein-protein docking. These all have, in some sense, a much large search space than the pK_a application of Chapter 4, and so can more efficiently amortize the precomputation cost. We then give some brief comments on how a near-optimal fast direct solver may be achieved, as well as how similar ideas might be used to construct direct solvers for *oscillatory* integral equations, for which much remains to be discovered. Such solvers for oscillatory kernels are expected to have tremendous impact, especially in biological imaging. We end with some final remarks and perspectives for the future.

5.1 Biomolecular charge optimization

The first application that we consider is biomolecular charge optimization, for which the canonical problem is the minimization of the binding energy of a solvated protein-protein complex over all admissible charge configurations. This has implications for understanding protein charge complementarity as well as in furthering molecular design [125, 126], and constitutes an optimization problem that is constrained by a partial differential equation (PDE)—in this case, the LPBE. Bardhan et al. [15, 16] has shown that such problems can be treated efficiently using a block co-optimization approach, where an LPBE solve is required at each Newton-Raphson iteration as part of an expanded "reverse-Schur" computation. This can therefore be accelerated by using our direct solver as a block preconditioner as was done for the multiple scattering example in §2.8. Our methods are especially attractive for this application since the search space is discovered on the fly and hence cannot be bounded a priori. Similar remarks hold for other PDE-constrained optimization problems, provided that the integral kernels are compatible with our techniques.

5.2 Protein structure prediction and design

We next consider the task of protein structure prediction and design, which encompasses some of the foremost problems in computational structural biology [11, 57]. We focus only on the electrostatics, and moreover assume a setting in which we have a fixed backbone structure (e.g., from homology modeling), about which we perform sidechain rotamer optimization. Then it turns out that the electrostatic energy of a given rotamer configuration can be well-approximated by only one- and two-body terms (that is, energies depending only on one or two rotamer states) [137, 197]; the required calculation thus presents as a collection of local geometric perturbations, to which we can directly apply the methods of §3.3. As with the multi-conformation approach to pK_a calculations discussed briefly in §4.8, the search space is now far larger, giving the direct solver an extended comparative advantage. Such techniques are expected to greatly accelerate current protein analysis workflows and should find many applications in biotechnology.

5.3 Protein-protein docking

For a final example, we turn to protein-protein docking, which is fundamental to the study of protein-protein interactions, with particular relevance to cell signaling and modern drug discovery. As with the above, we consider only the electrostatic contribution to the binding free energy [see 135]. Then the protocol for rigid-body docking is fairly straightforward and is essentially equivalent to that of the multiple scattering example from §2.8. Specifically, the electrostatic operators for both the ligand and receptor are precomputed individually and then combined to rapidly solve each candidate configuration by using the precomputed matrix factors as a block preconditioner for some outer iterative scheme. This can be seen as a refinement of the electrostatic "steering" Brownian dynamics simulations of McCammon et al. [see, e.g., 189], which allow the ligand to move only in the field of the receptor without accounting for their cross-interactions. The underlying search space here is *continuous*, so the number of required solves in this setting can in general be quite large.

Recently, it has become clear that conformational flexibility is critical to docking processes [85, 174], and this, too, can be handled by our approach. The precomputations are now performed with respect to the backbone structures of each protein, and the different rotamer configurations treated using perturbative techniques as described above.

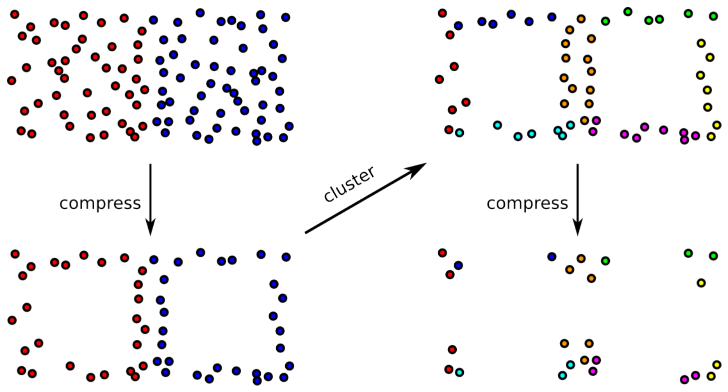

Figure 5.1: Schematic for skeleton recompression in 2D, where skeletons (colored by block index) are reclustered after each round of compression to exploit further geometric structure.

5.4 Towards a near-optimal fast direct solver

Recall from §2.5 that our current direct solver has optimal $\mathcal{O}(N)$ complexity only on 1D domains, with general $\mathcal{O}(N^{3(1-1/d)})$ cost in $\mathbb{R}^d$. The reason for this is most easily seen from Figure 2.5, which shows that skeletons tend to line up along cell boundaries (as a consequence of Green's theorem) and so are effectively objects in $\mathbb{R}^{d-1}$. To temper their growing sizes in high dimensions, it is clear that the skeletons must be *recompressed*, and Figure 2.5 offers some tantalizing clues on how this might be achieved. In particular, we observe that the skeleton distribution is highly structured and hence can be further exploited. This requires a reordering of the skeletons such that all clusters are separated, and leads to a natural scheme for recompression as outlined in Figure 5.1. A higher dimensional analogue is straightforward and corresponds essentially to a recursion down on the effective skeleton dimension. The consequences of such recompression are not yet entirely clear, but it is expected to significantly reduce, at least, the practical cost of the algorithm.

5.5 Towards a fast direct solver for oscillatory kernels

Although we have thus far been concerned only with the solution of non-oscillatory integral equations, similar ideas may prove useful for the oscillatory case, e.g., the Helmholtz equation, for which some progress has been made for volume integral equations [44] but not for boundary integral equations in the general case [cf. 140, 144]. Fast algorithms for apply oscillatory integral operators based on the so-called butterfly algorithm [34, 143, 154, 207] are available, and some preliminary results suggest that the rank structure utilized there may also facilitate rapid inversion.

Briefly, in 1D, a one-level butterfly algorithm computes a matrix decomposition of the form

$$A \equiv \begin{bmatrix} L_1^{(1)} S_{11} R_1^{(1)} & L_1^{(2)} S_{12} R_2^{(1)} \\ L_2^{(1)} S_{21} R_1^{(2)} & L_2^{(2)} S_{22} R_2^{(2)} \end{bmatrix},$$

where the $L_j^{(i)}$ and $R_j^{(i)}$ are interpolation matrices, with the S_{ij} the associated skeletons; $L^{(1)}$ and $L^{(2)}$ can be thought of as the row projection matrices for the left and right halves, respectively, of A, and similarly with $R^{(1)}$ and $R^{(2)}$ for the top and bottom, where

$$L^{(i)} \equiv \begin{bmatrix} L_1^{(i)} & \\ & L_2^{(i)} \end{bmatrix}, \quad R^{(i)} \equiv \begin{bmatrix} R_1^{(i)} & \\ & R_2^{(i)} \end{bmatrix}.$$

Then A admits the representation

$$A = LSR,$$

where

$$L = \begin{bmatrix} L_1^{(1)} & & L_1^{(2)} & \\ & L_2^{(1)} & & L_2^{(2)} \end{bmatrix}, \quad S = \begin{bmatrix} S_{11} & & & \\ & & S_{21} & \\ & S_{12} & & \\ & & & S_{22} \end{bmatrix}, \quad R = \begin{bmatrix} R_1^{(1)} & \\ & R_2^{(1)} \\ R_1^{(2)} & \\ & R_2^{(2)} \end{bmatrix}.$$

This has the same basic structure as that for the present direct solver (but with $D = 0$, cf. (2.2)) and so is amenable to essentially the same methods for fast multiplication and inversion. The multilevel and multi-dimensional generalizations are not difficult but are somewhat cumbersome; their description and analysis will be the subject of a later publication.

Such techniques cannot yet treat the Helmholtz kernel but can be used, e.g., for Fourier integral operators [34, 154, 207], a prime application of which is magnetic resonance image reconstruction [89, and references therein].

5.6 Conclusion

In this work, we have presented a fast direct solver for non-oscillatory integral equations and have demonstrated its use in the context of molecular electrostatics. Its primary novelty is the extremely low cost required to apply the solution operator, once the initial compressed factorization has been obtained. Thus, it is particularly suited to problems involving multiple right-hand sides, for which we have shown one example in the form of protein pK_a calculations and have outlined several more. We believe that our techniques will enable new large-scale biophysical simulations, especially in areas concerning optimization and design. Furthermore, because of the generality of the linear algebraic approach, our methods can also be applied to many other problems in computational science and engineering. A number of algorithmic issues remain open, notably the efficient extension of the fast direct solver to problems in higher dimensions or in the highly oscillatory regime. We hope that the basic framework established here will be useful in those contexts as well.

Bibliography

[1] Alexov E, Mehler EL, Baker N, Baptista AM, Huang Y, Milletti F, Nielsen JE, Farrell D, Carstensen T, Olsson MHM, Shen JK, Warwicker J, Williams S, Word JM (2011) Progress in the prediction of pK_a values in proteins. *Proteins* 79: 3260–3275.

[2] Alexov EG, Gunner MR (1997) Incorporating protein conformational flexibility into the calculation of pH-dependent protein properties. *Biophys J* 74: 2075–2093.

[3] Altman MD, Bardhan JP, Tidor B, White JK (2006) FFTSVD: A fast multi-scale boundary-element method solver suitable for bio-MEMS and biomolecule simulation. *IEEE Trans Computer Aid Design* 25: 274–284.

[4] Altman MD, Bardhan JP, White JK, Tidor B (2009) Accurate solution of multi-region continuum biomolecule electrostatic problems using the linearized Poisson-Boltzmann equation with curved boundary elements. *J Comput Chem* 30: 132–153.

[5] Amestoy PR, Duff IS, L'Excellent JY, Koster J (2001) A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J Matrix Anal Appl* 23: 15–41.

[6] Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Croz

JD, Greenbaum A, Hammarling S, McKenney A, Sorensen D (1999) *LAPACK Users' Guide*, 3rd ed. SIAM: Philadelphia, PA.

[7] Antosiewicz J, Briggs JM, Elcock AH, Gilson MK, McCammon JA (1996) Computing ionization states of proteins with a detailed charge model. *J Comput Chem* 17: 1633–1644.

[8] Antosiewicz J, McCammon JA, Gilson MK (1994) Prediction of pH-dependent properties of proteins. *J Mol Biol* 238: 415–436.

[9] Appel AW (1985) An efficient program for many-body simulation. *SIAM J Sci Stat Comput* 6: 85–103.

[10] Babuška I, Rheinboldt WC (1978) Error estimates for adaptive finite element computations. *SIAM J Numer Anal* 15: 736–754.

[11] Baker D, Sali A (2001) Protein structure prediction and structural genomics. *Science* 294: 93–96.

[12] Baker N, Holst M, Wang F (2000) Adaptive multilevel finite element solution of the Poisson-Boltzmann equation II. Refinement at solvent-accessible surfaces in biomolecular systems. *J Comput Chem* 21: 1343–1352.

[13] Baker NA, Sept D, Joseph S, Holst MJ, McCammon JA (2001) Electrostatics of nanosystems: application to microtubules and the ribosome. *Proc Natl Acad Sci USA* 98: 10037–10041.

[14] Bardhan JP (2011) Nonlocal continuum electrostatic theory predicts surprisingly small energetic penalties for charge burial in proteins. *J Chem Phys* 135: 104113.

[15] Bardhan JP, Altman MD, Tidor B, White JK (2009) "Reverse-Schur" approach to optimization with linear PDE constraints: application to biomolecule analysis and design. *J Chem Theory Comput* 5: 3260–3278.

[16] Bardhan JP, Altman MD, White JK, Tidor B (2007) Efficient optimization of electrostatic interactions between biomolecules. In *Proceedings of the 46th IEEE Conference on Decision and Control*, pp. 4563–4569.

[17] Bardhan JP, Altman MD, Willis DJ, Lippow SM, Tidor B, White JK (2007) Numerical integration techniques for curved-element discretizations of molecule-solvent interfaces. *J Chem Phys* 127: 014701.

[18] Bardhan JP, Eisenberg ES, Gillespie D (2009) Discretization of the induced-charge boundary integral equation. *Phys Rev E* 80: 011906.

[19] Bardhan JP, Hildebrandt A (2011) A fast solver for nonlocal electrostatic theory in biomolecular science and engineering. In *Proceedings of the 48th ACM/EDAC/IEEE Design Automation Conference*, pp. 801–805.

[20] Barnes J, Hut P (1986) A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature* 324: 446–449.

[21] Bashford D, Karplus M (1990) pK_{a}'s of ionizable groups in proteins: atomic detail from a continuum electrostatic model. *Biochemistry* 29: 10219–10225.

[22] Bashford D, Karplus M (1991) Multiple-site titration curves of proteins: an analysis of exact and approximate methods for their calculation. *J Phys Chem* 95: 9556–9561.

[23] Bayly CI, Cieplak P, Cornell WD, Kollman PA (1993) A well-behaved electrostatic potential based method using charge restraints for deriving atomic charges: the RESP model. *J Phys Chem* 97: 10269–10280.

[24] Berenger JP (1994) A perfectly matched layer for the absorption of electromagnetic waves. *J Comput Phys* 114: 185–200.

[25] Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, Shindyalov IN, Bourne PE (2000) The Protein Data Bank. *Nucleic Acids Res* 28: 235–242.

[26] Beroza P, Fredkin DR, Okamura MY, Feher G (1991) Protonation of interacting residues in a protein by a Monte Carlo method: application to lysozyme and the photosynthetic reaction center of *Rhodobacter sphaeroides*. *Proc Natl Acad Sci USA* 88: 5804–5808.

[27] Bode W, Epp O, Huber R, Laskowski M Jr, Ardelt W (1985) The crystal and molecular structure of the third domain of silver pheasant ovomucoid (OMSVP3). *Eur J Biochem* 147: 387–395.

[28] Bonnet M, Maier G, Polizzotto C (1998) Symmetric Galerkin boundary element method. *Appl Mech Rev* 51: 669–704.

[29] Börm S (2010) *Efficient numerical methods for non-local operators. $\mathcal{H}^2$-matrix compression, algorithms and analysis*. Eur Math Soc: Zürich.

[30] Boschitsch AH, Fenley MO, Zhou HX (2002) Fast boundary element method for the linear Poisson-Boltzmann equation. *J Phys Chem B* 106: 2741–2754.

[31] Brandt A, Lubrecht AA (1990) Multilevel matrix multiplication and fast solution of integral equations. *J Comput Phys* 90: 348–370.

[32] Bremer J, Rokhlin V, Sammis I (2010) Universal quadratures for boundary integral equations on two-dimensional domains with corners. *J Comput Phys* 229: 8259–8280.

[33] Brown LS, Kamikubo H, Zimányi L, Kataoka M, Tokunaga F, Verdegem P, Lugtenburg J, Lanyi JK (1997) A local electrostatic change is the cause of the large-scale protein conformation shift in bacteriorhodopsin. *Proc Natl Acad Sci USA* 94: 5040–5044.

[34] Candès E, Demanet L, Ying L (2009) A fast butterfly algorithm for the computation of Fourier integral operators. *Multiscale Model Simul* 7: 1727–1750.

[35] Canning FX, Rogovin K (1998) Fast direct solution of standard moment-method matrices. *IEEE Antenn Propag Mag* 40: 15–26.

[36] Carrier J, Greengard L, Rokhlin V (1988) A fast adaptive multipole algorithm for particle simulations. *SIAM J Sci Stat Comput* 9: 669–686.

[37] Case DA, Cheatham TE III, Darden T, Gohlke H, Luo R, Merz KM, Onufriev A, Simmerling C, Wang B, Woods RJ (2005) The Amber biomolecular simulation programs. *J Comput Chem* 26: 1668–1688.

[38] Chai W, Jiao D, Koh CK (2009) A direct integral-equation solver of linear complexity for large-scale 3D capacitance and impedance extraction. In *Proceedings of the 46th ACM/IEEE Design Automation Conference*, pp. 752–757.

[39] Chan TF, Wan LW (1997) Analysis of projection methods for solving linear systems with multiple right-hand sides. *SIAM J Sci Comput* 18: 1698–1721.

[40] Chandrasekaran S, DeWilde P, Gu M, Lyons W, Pals T (2006) A fast solver for HSS representations via sparse matrices. *SIAM J Matrix Anal Appl* 29: 67–81.

[41] Chandrasekaran S, Gu M, Pals T (2006) A fast *ULV* decomposition solver for hierarchically semiseparable representations. *SIAM J Matrix Anal Appl* 28: 603–622.

[42] Chen D, Chen Z, Chen C, Geng W, Wei GW (2011) MIBPB: a software package for electrostatic analysis. *J Comput Chem* 32: 756–770.

[43] Chen L, Holst MJ, Xu J (2007) The finite element approximation of the nonlinear Poisson-Boltzmann equation. *SIAM J Numer Anal* 45: 2298–2320.

[44] Chen Y (2002) A fast, direct algorithm for the Lippmann-Schwinger integral equation in two dimensions. *Adv Comput Math* 16: 175–190.

[45] Cheng H, Crutchfield W, Gimbutas Z, Greengard L, Huang J, Rokhlin V, Yarvin N, Zhao J (2006) Remarks on the implementation of the wideband FMM for the Helmholtz equation in two dimensions. *Contemp Math* 408: 99–110.

[46] Cheng H, Crutchfield WY, Gimbutas Z, Greengard LF, Ethridge JF, Huang J, Rokhlin V, Yarvin N, Zhao J (2006) A wideband fast multipole method for the Helmholtz equation in three dimensions. *J Comput Phys* 216: 300–325.

[47] Cheng H, Gimbutas Z, Martinsson PG, Rokhlin V (2005) On the compression of low rank matrices. *SIAM J Sci Comput* 26: 1389–1404.

[48] Cheng H, Greengard L, Rokhlin V (1999) A fast adaptive multipole algorithm in three dimensions. *J Comput Phys* 155: 468–498.

[49] Chew WC, Jin JM, Michielssen E, Song J (2001) *Fast and efficient algorithms in computational electromagnetics*. Artech House: Boston, MA.

[50] Chorin AJ (1968) Numerical solution of the Navier-Stokes equations. *Math Comput* 22: 745–762.

[51] Colton D, Kress R (1998) *Inverse acoustic and electromagnetic scattering theory*, 2nd ed. Springer-Verlag: New York, NY.

[52] Connolly ML (1983) Solvent-accessible surfaces of proteins and nucleic acids. *Science* 221: 709–713.

[53] Connolly ML (1993) The molecular surface package. *J Mol Graph* 11: 139–141.

[54] Cornish-Bowden A (1995) *Fundamentals of enzyme kinetics*, rev ed. Portland Press: London.

[55] Cortis CM, Friesner RA (1997) Numerical solution of the Poisson-Boltzmann equation using tetrahedral finite-element meshes. *J Comput Chem* 18: 1591–1608.

[56] Coux O, Tanaka K, Goldberg AL (1996) Structure and functions of the 20S and 26S proteasomes. *Annu Rev Biochem* 65: 801–847.

[57] Dahiyat BI, Mayo SL (1997) De novo protein design: fully automated sequence selection. *Science* 278: 82–87.

[58] Davis ME, McCammon JA (1990) Electrostatics in biomolecular structure and dynamics. *Chem Rev* 90: 509–521.

[59] Davis TA (2004) A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans Math Softw* 30: 165–195.

[60] Davis TA (2004) Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans Math Softw* 30: 196–199.

[61] Davis TA (2011) Algorithm 915, SuiteSparseQR: multifrontal multithreaded rank-revealing sparse QR factorization. *ACM Trans Math Softw* 38: 8.

[62] Davis TA, Duff IS (1997) An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J Matrix Anal Appl* 18: 140–158.

[63] Davis TA, Duff IS (1999) A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans Math Softw* 25: 1–20.

[64] Demmel JW, Eisenstat SC, Gilbert JR, Li XS, Liu JWH (1999) A supernodal approach to sparse partial pivoting. *SIAM J Matrix Anal Appl* 20: 720–755.

[65] Dill KA (1990) Dominant forces in protein folding. *Biochemistry* 29: 7133–7155.

[66] Dolinsky TJ, Nielsen JE, McCammon JA, Baker NA (2004) PDB2PQR: an automated pipeline for the setup of Poisson-Boltzmann electrostatic calculations. *Nucl Acids Res* 32: W665–W667.

[67] Drew HR, Wing RM, Takano T, Broka C, Tanaka S, Itakura K, Dickerson RE (1981) Structure of a B-DNA dodecamer: conformation and dynamics. *Proc Natl Acad Sci USA* 78: 2179–2183.

[68] Earnshaw WC, Martins LM, Kaufmann SH (1999) Mammalian caspases: structure, activation, substrates, and functions during apoptosis. *Annu Rev Biochem* 68: 383–424.

[69] Engquist B, Ying L (2007) Fast directional multilevel algorithms for oscillatory kernels. *SIAM J Sci Comput* 29: 1710–1737.

[70] Engquist B, Ying L (2009) A fast directional algorithm for high frequency acoustic scattering in two dimensions. *Commun Math Sci* 7: 327–345.

[71] Fischer PF (1998) Projection techniques for iterative solution of $Ax = b$ with successive right-hand sides. *Comput Method Appl Mech Eng* 163: 193–204.

[72] Fogolari F, Brigo A, Molinari H (2002) The Poisson-Boltzmann equation for biomolecular electrostatics: a tool for structural biology. *J Mol Recognit* 15: 377–392.

[73] Fong W, Darve E (2009) The black-box fast multipole method. *J Comput Phys* 228: 8712–8725.

[74] Francl MM, Carey C, Chirlian LE, Gange DM (1996) Charges fit to electrostatic potentials II. Can atomic charges be unambiguously fit to electrostatic potentials? *J Comput Chem* 17: 367–383.

[75] George A (1973) Nested dissection of a regular finite element mesh. *SIAM J Numer Anal* 10: 345–363.

[76] Georgescu RE, Alexov EG, Gunner MR (2002) Combining conformational flexibility and continuum electrostatics for calculating pK_as in proteins. *Biophys J* 83: 1731–1748.

[77] Gillman A (2011) *Fast direct solvers for elliptic partial differential equations.* PhD thesis, University of Colorado Boulder.

[78] Gillman A, Young PM, Martinsson PG (2012) A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains. To appear, *Front Math China*. arXiv:1105.5372.

[79] Gilson MK (1993) Multiple-site titration and molecular modeling: two rapid methods for computing energies and forces for ionizable groups in proteins. *Proteins* 15: 266–282.

[80] Gilson MK, Honig BH (1986) The dielectric of a folded protein. *Biopolymers* 25: 2097–2119.

[81] Gilson MK, Sharp KA, Honig BH (1987) Calculating the electrostatic potential of molecules in solution: method and error assessment. *J Comput Chem* 9: 327–335.

[82] Gimbutas Z, Greengard L, Minion M (2001) Coulomb interactions on planar structures: inverting the square root of the Laplacian. *SIAM J Sci Comput* 22: 2093–2108.

[83] Gimbutas Z, Rokhlin V (2003) A generalized fast multipole method for nonoscillatory kernels. *SIAM J Sci Comput* 24: 796–817.

[84] Golub GH, van Loan CF (1996) *Matrix computations*, 3rd ed. The Johns Hopkins University Press: Baltimore, MD.

[85] Gray JJ, Moughon S, Wang C, Schueler-Furman O, Kuhlman B, Rohl CA, Baker D (2003) Protein-protein dockingwith simultaneous optimization of rigid-body displacement and side-chain conformations. *J Mol Biol* 331: 281–299.

[86] Greengard L, Rokhlin V (1987) A fast algorithm for particle simulations. *J Comput Phys* 73: 325–348.

[87] Greengard L (1994) Fast algorithms for classical physics. *Science* 265: 909–914.

[88] Greengard L, Gueyffier D, Martinsson PG, Rokhlin V (2009) Fast direct solvers for integral equations in complex three-dimensional domains. *Acta Numer* 18: 243–275.

[89] Greengard L, Lee JY (2004) Accelerating the nonuniform fast Fourier transform. *SIAM Rev* 46: 443–454.

[90] Greengard L, Lee JY (2006) Electrostatics and heat conduction in high contrast composite materials. *J Comput Phys* 211: 64–76.

[91] Greengard L, Moura M (1994) On the numerical evaluation of electrostatic fields in composite materials. *Acta Numer* 3: 379–410.

[92] Greengard L, Rokhlin V (1997) A new version of the Fast Multipole Method for the Laplace equation in three dimensions. *Acta Numer* 6: 229–269.

[93] Greengard LF (1988) *The rapid evaluation of potential fields in particle systems.* The MIT Press: Cambridge, MA.

[94] Greengard L, Rokhlin V (1991) On the numerical solution of two-point boundary value problems. *Commun Pure Appl Math* 44: 419–452.

[95] Greengard LF, Huang J (2002) A new version of the fast multipole method for screened Coulomb interactions in three dimensions. *J Comput Phys* 180: 642–658.

[96] Grote MJ, Huckle T (1997) Parallel preconditioning with sparse approximate inverses. *SIAM J Sci Comput* 18: 838–853.

[97] Guenther RB, Lee JW (1988) *Partial differential equations of mathematical physics and integral equations.* Prentice-Hall: Englewood Cliffs, NJ.

[98] Hackbusch W (1999) A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: introduction to $\mathcal{H}$-matrices. *Computing* 62: 89–108.

[99] Hackbusch W, Börm S (2002) Data-sparse approximation by adaptive $\mathscr{H}^2$-matrices. *Computing* 69: 1–35.

[100] Hackbusch W, Khoromskij BN (2000) A sparse $\mathscr{H}$-matrix arithmetic. Part II: application to multi-dimensional problems. *Computing* 64: 21–47.

[101] Hackbusch W, Nowak ZP (1989) On the fast matrix multiplication in the boundary element method by panel clustering. *Numer Math* 54: 463–491.

[102] Hager WW (1989) Updating the inverse of a matrix. *SIAM Rev* 31: 221–239.

[103] Hamada T, Yokota R, Nitadori K, Narumi T, Yasuoka K, Taiji M (2009) 42 TFlops hierarchical N-body simulations on GPUs with applications in both astrophysics and turbulence. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage, and Analysis.*

[104] Helsing J, Ojala R (2008) Corner singularities for elliptic problems: integral equations, graded meshes, quadrature, and compressed inverse preconditioning. *J Comput Phys* 227: 8820–8840.

[105] Hildebrandt A, Blossey R, Rjasanow S, Kohlbacher O, Lenhof HP (2004) Novel formulation of nonlocal electrostatics. *Phys Rev Lett* 93: 108104.

[106] Hildebrandt A, Blossey R, Rjasanow S, Kohlbacher O, Lenhof HP (2007) Electrostatic potentials of proteins in water: a structured continuum approach. *Bioinformatics* 23: e99–e103.

[107] Ho KL, Greengard L (2012) A fast direct solver for structured linear systems by recursive skeletonization. In review. arXiv:1110.3105.

[108] Holst M, Baker N, Wang F (2000) Adaptive multilevel finite element solution of the Poisson-Boltzmann equation I. Algorithms and examples. *J Comput Chem* 21: 1319–1342.

[109] Holst MJ (1993) *Multilevel methods for the Poisson-Boltzmann equation.* PhD thesis, University of Illinois at Urbana-Champaign.

[110] Honig B, Nicholls A (1995) Classical electrostatics in biology and chemistry. *Science* 268: 1144–1149.

[111] Howlin B, Moss DS, Harris GW (1989) Segmented anisotropic refinement of bovine ribonuclease A by the application of the rigid-body TLS model. *Acta Crystallogr A* 45: 851–861.

[112] Huang J, Jia J, Zhang B (2009) FMM-Yukawa: an adaptive fast multipole method for screened Coulomb interactions. *Comput Phys Commun* 180: 2331–2338.

[113] Hubbard SR, Till JH (2000) Protein tyrosine kinase structure and function. *Annu Rev Biochem* 69: 373–398.

[114] Ito Y, Ochiai Y, Park YS, Imanishi Y (1997) pH-sensitive gating by conformational change of a polypeptide brush grafted onto a porous polymer membrane. *J Am Chem Soc* 119: 1619–1623.

[115] Juffer AH, Argos P, Vogel HJ (1997) Calculating acid-dissociation constants of proteins using the boundary element method. *J Phys Chem B* 101: 7664–7673.

[116] Juffer AH, Botta EFF, van Keulen BAM, van der Ploeg A, Berendsen HJC (1991) The electric potential of a macromolecule in a solvent: a fundamental approach. *J Comput Phys* 97: 144–171.

[117] Kantorovich LV, Krylov VI (1958) *Approximate methods of higher analysis.* Interscience: New York, NY.

[118] Kapur S, Long DE (1997) IES^3: a fast integral equation solver for efficient 3-dimensional extraction. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design.*

[119] Kapur S, Rokhlin V (1997) High-order corrected trapezoidal quadrature rules for singular functions. *SIAM J Numer Anal* 34: 1331–1356.

[120] Katayanagi K, Miyagawa M, Matsushima M, Ishikawa M, Kanaya S, Nakamura H, Ikehara M, Matsuzaki T, Morikawa K (1992) Structural details of ribonuclease H from *Escherichia coli* as refined to an atomic resolution. *J Mol Biol* 223: 1029–1052.

[121] Kornberg A, Baker TA (1992) *DNA replication*, 2nd ed. WH Freeman & Co.: New York, NY.

[122] Landau LD, Lifshitz EM (1980) *Statistical physics*, 3rd ed. Pergamon Press: Oxford.

[123] Lashuk I, Chandramowlishwaran A, Langston H, Nguyen TA, Sampath R, Shringarpure A, Vuduc R, Ying L, Zorin D, Biros G (2009) A massively parallel adaptive fast multipole method on heterogeneous architectures. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage, and Analysis*.

[124] Lee JY, Greengard L (1997) A fast adaptive numerical method for stiff two-point boundary value problems. *SIAM J Sci Comput* 18: 403–429.

[125] Lee LP, Tidor B (1997) Optimization of electrostatic binding free energy. *J Chem Phys* 106: 8681–8690.

[126] Lee LP, Tidor B (2001) Optimization of binding electrostatics: charge complementary in the barnase-barstar protein complex. *Protein Sci* 10: 362–337.

[127] Levy RM, Gallicchio E (1998) Computer simulations with explicit solvent: recent progress in the thermodynamic decomposition of free energies and in modeling electrostatic effects. *Annu Rev Phys Chem* 49: 531–567.

[128] Li H, Robertson AD, Jensen JH (2005) Very fast empirical prediction and rationalization of protein pK_a values. *Proteins* 61: 704–721.

[129] Li JR, Greengard L (2009) High order accurate methods for the evaluation of layer heat potentials. *SIAM J Sci Comput* 31: 3847–3860.

[130] Li P, Johnston H, Krasny R (2009) A Cartesian treecode for screened Coulomb interactions. *J Comput Phys* 228: 3858–3868.

[131] Liberty E, Woolfe F, Martinsson PG, Rokhlin V, Tygert M (2007) Randomized algorithms for the low-rank approximation of matrices. *Proc Natl Acad Sci USA* 104: 20167–20172.

[132] Lin JH, Baker NA, McCammon JA (2002) Bridging implicit and explicit solvent approaches for membrane electrostatics. *Biophys J* 83: 1374–1379.

[133] Liu Y (2009) *Fast multipole boundary element method: theory and applications in engineering*. Cambridge University Press: New York, NY.

[134] Lu BZ, Zhou YC, Holst MJ, McCammon JA (2008) Recent progress in numerical methods for the Poisson-Boltzmann equation in biophysical applications. *Commun Comput Phys* 3: 973–1009.

[135] Mandell JG, Roberts VA, Pique ME, Kotlovyi V, Mitchell JC, Nelson E, Tsigelny I, Eyck LFT (2001) Protein docking using continuum electrostatics and geometric fit. *Protein Eng* 14: 105–113.

[136] Marquart M, Walter J, Deisenhofer J, Bode W, Huber R (1983) The geometry of the reactive site and of the peptide groups in trypsin, trypsinogen, and its complexes with inhibitors. *Acta Crystallogr B* 39: 480–490.

[137] Marshall SA, Vizcarra CL, Mayo SL (2005) One- and two-body decomposable Poisson-Boltzmann methods for protein design calculations. *Protein Sci* 14: 1293–1304.

[138] Martinsson PG (2006) Fast evaluation of electro-static interactions in multiphase dielectric media. *J Comput Phys* 211: 289–299.

[139] Martinsson PG, Rokhlin V (2005) A fast direct solver for boundary integral equations in two dimensions. *J Comput Phys* 205: 1–23.

[140] Martinsson PG, Rokhlin V (2007) A fast direct solver for scattering problems involving elongated structures. *J Comput Phys* 221: 288–302.

[141] Martinsson PG, Rokhlin V (2007) An accelerated kernel-independent fast multipole method in one dimension. *SIAM J Sci Comput* 29: 1160–1178.

[142] Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953) Equation of state calculations by fast computing machines. *J Chem Phys* 21: 1087–1092.

[143] Michielssen E, Boag A (1996) A multilevel matrix decomposition algorithm for analyzing scattering from large structures. *IEEE Trans Antenn Propag* 44: 1086–1093.

[144] Michielssen E, Boag A, Chew WC (1996) Scattering from elongated objects: direct solution in $O(N \log^2 N)$ operations. *IEE Proc Microw Antenn Propag* 143: 277–283.

[145] Mur G (1981) Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic-field equations. *IEEE Trans Electromag Compat* 23: 377–382.

[146] Nabors K, White J (1991) FastCap: a multipole accelerated 3-D capacitance extraction program. *IEEE Trans Computer Aid Design* 10: 1447–1459.

[147] Newton AC (1995) Protein kinase C: structure, function, and regulation. *J Biol Chem* 270: 28495–28498.

[148] Nicholls A, Honig B (1991) A rapid finite difference algorithm, utilizing successive over-relaxation to solve the Poisson-Boltzmann equation. *J Comput Chem* 12: 435–445.

[149] Nielsen JE, Vriend G (2001) Optimizing the hydrogen-bond network in Poisson-Boltzmann equation-based pK_a calculations. *Proteins* 43: 403–412.

[150] Nishimura N (2002) Fast multipole accelerated boundary integral equation methods. *Appl Mech Rev* 55: 299–324.

[151] Nozaki Y, Tanford C (1967) Examination of titration behavior. *Methods Enzymol* 11: 715–734.

[152] Oehlert GW (1992) A note on the delta method. *Amer Stat* 46: 27–29.

[153] Olsson MHM, Søndergaard CR, Rostkowski M, Jensen JH (2011) PROPKA3: consistent treatment of internal and surface residues in empirical pK_a predictions. *J Chem Theory Comput* 7: 525–537.

[154] O'Neil M, Woolfe F, Rokhlin V (2010) An algorithm for the rapid evaluation of special function transforms. *Appl Comput Harmon Anal* 28: 203–226.

[155] Pals TP (2004) *Multipole for scattering computations: spectral discretization, stabilization, fast solvers.* PhD thesis, University of California, Santa Barbara.

[156] Perrot G, Cheng B, Gibson KD, Vila J, Palmer KA, Nayeem A, Maigret B, Scheraga HA (1992) MSEED: a program for the rapid analytical determination of accessible surface areas and their derivatives. *J Comput Chem* 13: 1–11.

[157] Phillips JR, White JK (1997) A precorrected-FFT method for electrostatic analysis of complicated 3-D structures. *IEEE Trans Computer Aid Design* 16: 1059–1072.

[158] Purcell EM (1985) *Electricity and magnetism*, vol 2, 2nd ed. McGraw Hill: Boston, MA.

[159] Rabenstein B, Ullman GM, Knapp EW (1998) Calculation of protonation patterns in proteins with structural relaxation and molecular ensembles. *Eur Biophys J* 27: 626–637.

[160] Rahimian A, Lashuk I, Veerapaneni SK, Chandramowlishwaran A, Malhotra D, Moon L, Sampath R, Shringarpure A, Vetter J, Vuduc R, Zorin D, Biros G (2010) Petascale direct numerical simulation of blood flow on 200K cores and heterogeneous architectures. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage, and Analysis.*

[161] Ramanadham M, Sieker LC, Jensen LH (1990) Refinement of triclinic lysozyme: II. The method of stereochemically restrained least squares. *Acta Crystallogr B* 46: 63–69.

[162] Rocchia W, Alexov E, Honig B (2001) Extending the applicability of the nonlinear Poisson-Boltzmann equation: multiple dielectric constants and multivalent ions. *J Phys Chem B* 105: 6507–6514.

[163] Rokhlin V (1990) Rapid solution of integral equations of scattering theory in two dimensions. *J Comput Phys* 86: 414–439.

[164] Rokhlin V (1993) Diagonal forms of translation operators for the Helmholtz equation in three dimensions. *Appl Comput Harmon Anal* 1: 82–93.

[165] Rokhlin V, Tygert M (2008) A fast randomized algorithm for overdetermined linear least squares regression. *Proc Natl Acad Sci USA* 105: 13212–13217.

[166] Saad Y (2003) *Iterative methods for sparse linear systems*, 2nd ed. SIAM: Philadelphia, PA.

[167] Saad Y, Schultz MH (1986) GMRES: a generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM J Sci Stat Comput* 7: 856–869.

[168] Samet H (1984) The quadtree and related hierarchical data structures. *ACM Comput Surv* 16: 187–260.

[169] Sanner MF, Olson AJ, Spehner JC (1996) Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers* 38: 305–320.

[170] Schenk O, Gärtner K (2004) Solving unsymmetric sparse systems of linear equations with PARDISO. *Future Gener Comput Syst* 20: 475–487.

[171] Schildkraut C, Lifson S (1965) Dependence of the melting temperature of DNA on salt concentration. *Biopolymers* 3: 195–208.

[172] Sharp KA, Honig B (1990) Electrostatic interactions in macromolecules: theory and applications. *Annu Rev Biophys Biophys Chem* 19: 301–332.

[173] Sheinerman FB, Norel R, Honig B (2000) Electrostatic aspects of protein-protein interactions. *Curr Opin Struct Biol* 10: 153–159.

[174] Sherman W, Day T, Jacobson MP, Friesner RA, Farid R (2006) Novel procedure for modeling ligand/receptor induced fit effects. *J Med Chem* 49: 534–553.

[175] Shestakov AI, Milovich JL, Noy A (2002) Solution of the nonlinear Poisson-Boltzmann equation using pseudo-transient continuation and the finite element method. *J Colloid Interface Sci* 247: 62–79.

[176] Sifuentes J (2010) *Preconditioned iterative methods for inhomogeneous acoustic scattering applications*. PhD thesis, Rice University.

[177] Sitkoff D, Sharp KA, Honig B (1994) Accurate calculation of hydration free energies using macroscopic solvent models. *J Phys Chem* 98: 1978–1988.

[178] Sloan IH, Wendland WL (1998) Qualocation methods for elliptic boundray integral equations. *Numer Math* 79: 451–483.

[179] Sokal AD (1989) Monte Carlo methods in statistical mechanics: foundations and new algorithms. In *Cours de Troisième Cycle de la Physique en Suisse Romande*, Lausanne.

[180] Sommerfeld A (1949) *Partial differential equations in physics*. Academic Press: New York, NY.

[181] Song J, Lu CC, Chew WC (1997) Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects. *IEEE Trans Antenn Propag* 45: 1488–1493.

[182] Song Y, Mao JJ, Gunner MR (2009) MCCE2: improving protein pK_a calculations with extensive side chain rotamer sampling. *J Comput Chem* 30: 2231–2247.

[183] Spivak M, Veerapaneni SK, Greengard L (2010) The fast generalized Gauss transform. *SIAM J Sci Comput* 32: 3092–3107.

[184] Stakgold I (2000) *Boundary value problems of mathematical physics*. SIAM: Philadelphia, PA.

[185] Starr P, Rokhlin V (1994) On the numerical solution of two-point boundary value problems II. *Commun Pure Appl Math* 47: 1117–1159.

[186] Strader CD, Fong TM, Tota MR, Underwood D, Dixon RAF (1994) Structure and function of G protein-coupled receptors. *Annu Rev Biochem* 63: 101–132.

[187] Strang G, Fix GJ (2008) *An analysis of the finite element method*, 2nd ed. Wellesley-Cambridge Press: Wellesley, MA.

[188] Strikwerda JC (2004) *Finite difference schemes and partial differential equations*. SIAM: Philadelphia, PA.

[189] Tan RC, Truong TN, McCammon JA, Sussman JL (1993) Acetylcholinesterase: electrostatic steering increases the rate of ligand binding. *Biochemistry* 32: 401–403.

[190] Tanford C, Roxby R (1972) Interpretation of protein titration curves. Application to lysozyme. *Biochemistry* 11: 2192–2198.

[191] Tarjan R (1972) Depth-first search and linear graph algorithms. *SIAM J Comput* 1: 146–160.

[192] Tausch J, Wang J, White J (2001) Improved integral formulations for fast 3-D method-of-moments solvers. *IEEE Trans Computer Aid Design* 20: 1398–1405.

[193] Trefethen LN, Bau David III (1997) *Numerical linear algebra*. SIAM: Philadelphia, PA.

[194] Ullman GM, Knapp EW (1999) Electrostatic models for computing protonation and redox equilibria in proteins. *Eur Biophys J* 28: 533–551.

[195] Van der Vorst HA (1992) Bi-CGSTAB: a fast and smooth converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J Sci Stat Comput* 13: 631–644.

[196] Van der Vorst HA, Vuik C (1994) GMRESR: a family of nested GMRES methods. *Numer Linear Alg Appl* 1: 369–386.

[197] Vizcarra CL, Zhang N, Marshall SA, Wingreen NS, Zeng, C, Mayo SL (2008) An improved pairwise decomposable finite-difference Poisson-Boltzmann method for computational protein design. *J Comput Chem* 29: 1153–1162.

[198] Vlijmen HWT, Schaefer M, Karplus M (1998) Improving the accuracy of protein pK_a calculations: conformational averaging versus the average structure. *Proteins* 33: 145–158.

[199] Watson JD, Crick FHC (1953) Molecular structure of nucleic acids. *Nature* 171: 737–738.

[200] Wei JG, Peng Z, Lee JF (2011) A fast direct matrix solver for surface integral equation methods for electromagnetic wave problems in $\mathcal{R}^3$. In *27th Annual Review of Progress in Applied Computational Electromagnetics*, pp. 121–126.

[201] Whaley RC, Petitet A, Dongarra JJ (2001) Automated empirical optimizations of software and the ALTAS project. *Parallel Comput* 27: 3–35.

[202] Winebrand E, Boag A (2009) A multilevel fast direct solver for EM scattering from quasi-planar objects. In *Proceedings of the International Conference on Electromagnetics in Advanced Applications*, pp. 640–643.

[203] Woolfe F, Liberty E, Rokhlin V, Tygert M (2008) A fast randomized algorithm for the approximation of matrices. *Appl Comput Harmon Anal* 25: 335–366.

[204] Xia J, Chandrasekaran S, Gu M, Li XS (2009) Superfast multifrontal method for large structured linear systems of equations. *SIAM J Matrix Anal Appl* 31: 1382–1411.

[205] Yang AS, Gunner MR, Sampogna R, Sharp K, Honig B (1993) On the calculation of pK_as in proteins. *Proteins* 15: 252–265.

[206] Yang AS, Honig B (1993) On the pH dependence of protein stability. *J Mol Biol* 231: 459–474.

[207] Ying L (2009) Sparse Fourier transform via butterfly algorithm. *SIAM J Sci Comput* 31: 1678–1694.

[208] Ying L, Biros G, Zorin D (2004) A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *J Comput Phys* 196: 591–626.

[209] Yokota R, Bardhan JP, Knepley MG, Barba LA, Hamada T (2011) Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUs and a billion unknowns. *Comput Phys Commun* 182: 1272–1283.

[210] Yukawa H (1935) On the interaction of elementary particles. I. *Proc Phys-Math Soc Jpn* 17: 48–57.

[211] Zauderer E (1989) *Partial differential equations of applied mathematics*. John Wiley & Sons: New York, NY.

[212] Zhou YC, Feig M, Wei GW (2007) Highly accurate biomolecular electrostatics in continuum dielectric environments. *J Comput Chem* 29: 87–97.