Fast algorithms for PDE and integral equations

18.336J/6.335J Class notes

Laurent Demanet

Draft April 23, 2014

Contents

Chapter 1

Model problems

1.1 The Laplace, Poisson, and Helmholtz equations

1.2 Volume integral equations

1.3 Boundary integral equations

The idea of boundary integral equations is to reduce a PDE in some domain to an integral eqation on the surface of the domain, by means of the free-space Green's function. Let us first consider the Laplace equation in d dimensions ($\mathbf{x} = (x_1, \ldots, x_d)$). Let Ω be a bounded domain in $\mathbb{R}^d$. Four problems are of interest:

- *Interior problems.*
$$\Delta u(\mathbf{x}) = 0, \qquad \mathbf{x} \in \Omega,$$
with either Dirichlet boundary conditions ($u = f$ on $\partial\Omega$) or Neumann boundary conditions ($\partial u/\partial n = g$ on $\partial\Omega$).

- *Exterior problems.*
$$\Delta u(\mathbf{x}) = 0, \qquad \mathbf{x} \in \Omega^c,$$
with either Dirichlet boundary conditions ($u = f$ on $\partial\Omega$) or Neumann boundary conditions ($\partial u/\partial n = g$ on $\partial\Omega$), and a decay conditions at infinity, namely
$$|u(x)| = O(|\mathbf{x}|^{2-d}).$$

The interior Neumann problem is only solvable when g obeys the zero-mean admissibility condition (the topic of an exercise in section 1.4)

$$\int_{\partial\Omega} g(\mathbf{x}) dS_{\mathbf{x}} = 0.$$

Let us focus on the special case of the exterior Dirichlet problem. If Ω is a sphere of radius R in $\mathbb{R}^3$, and $f = 1$, then a multiple of the Green's function provides the solution everywhere outside Ω:

$$u(\mathbf{x}) = q \frac{1}{4\pi|\mathbf{x}|}.$$

Matching with $u(\mathbf{x}) = 1$ when $|\mathbf{x}| = 1$ yields $q = 4\pi R$. The field $u(\mathbf{x})$ solves the Laplace equation by construction, since the Green's function is evaluated away from its singularity (here, the origin.) More generally, we can also build solutions to the exterior Laplace problem as a superposition of charges q_j located at points $\mathbf{y}_j$ inside Ω, as

$$u(\mathbf{x}) = \sum_j G(\mathbf{x}, \mathbf{y}_j) q_j.$$

With enough judiciously chosen points $\mathbf{y}_j$, the boundary condition $u(\mathbf{x}) = g(\mathbf{x})$ on $\partial\Omega$ can often be matched to good accuracy. The fit of the q_j can be done by solving $u(\mathbf{x}_i) = \sum_j G(\mathbf{x}_i, \mathbf{y}_j) q_j$ for an adequate collection of $\mathbf{x}_i$ on $\partial\Omega$. Conversely, if the solution is sought inside Ω, then the charges can be located at $\mathbf{y}_j$ outside Ω.

This type of representation is called a *desingularized* method, because the charges do not lie on the surface $\partial\Omega$. Very little is known about its convergence properties.

A more robust and well-understood way to realize an exterior solution of the Laplace equation is to write it in terms of a monopole density ϕ on the surface $\partial\Omega$:

$$u(\mathbf{x}) = \int G(\mathbf{x}, \mathbf{y}) \phi(\mathbf{y}) dS_{\mathbf{y}}, \qquad x \in \Omega^c, \tag{1.1}$$

called a single-layer potential, and then match the Dirichlet boundary condition by taking the limit of $\mathbf{x}$ approaching $\partial\Omega$. The density $\phi(\mathbf{y})$ can be solved from

$$f(\mathbf{x}) = \int G(\mathbf{x}, \mathbf{y}) \phi(\mathbf{y}) dS_{\mathbf{y}}, \qquad x \in \partial\Omega. \tag{1.2}$$

This latter equation is known as a first-kind Fredholm integral equation. Once ϕ is known, (1.1) can be used to predict the potential everywhere in the exterior of Ω.

Notice that $G(\mathbf{x},\mathbf{y}) = \log(|\mathbf{x}-\mathbf{y}|)$ is integrable regardless of the dimension. The kernel $G(\mathbf{x},\mathbf{y}) = 1/|\mathbf{x}-\mathbf{y}|$ in 3D is also integrable when integrated on a smooth 2D surface passing through the point $\mathbf{y}$, since in local polar coordinates the Jacobian factor r cancels the $1/r$ from the integrand. As long as ϕ itself is bounded, there is no problem in defining the single-layer potential as a Lebesgue integral. The same is true for the high-frequency counterparts of the Poisson kernels. All these kernels belong in the class of weakly singular kernels.

Definition 1. *A kernel $G(\mathbf{x},\mathbf{y})$ is called weakly singular in dimension n (the dimension of the surface over which the integral is taken, not the ambient space) if it can be written in the form*

$$G(\mathbf{x},\mathbf{y}) = A(\mathbf{x},\mathbf{y})|\mathbf{x}-\mathbf{y}|^{-\alpha},$$

for some $0 \le \alpha < n$, and A bounded.

The fact that G is weakly singular in many situations of interest has two important consequences:

- First, when the density ϕ is bounded, it can be shown that $u(\mathbf{x})$ defined by (1.1) is *continuous* for all $\mathbf{x} \in \Omega$. This fact is needed to justify the limit $x \to \partial\Omega$ to obtain the boundary integral equation (1.2).
- Second, the operator T_G mapping ϕ to $f = T_G\phi$ in (1.2) is bounded on all L^p spaces for $1 \le p \le \infty$, and is moreover *compact* (i.e., it is the norm limit of finite rank operators).

Compactness, in particular, imposes a strong condition on the spectrum of T_G. The eigenvalues must cluster at the origin, and nowhere else. This behavior is problematic since it gives rise to ill-conditioning of discretizations of T_G, hence ill-posedness of the problem of solving for ϕ from f. This ill-conditioning is a generic, unfortunate feature of first-kind boundary integral equations.

An interesting alternative is to write the potential in Ω^c from a dipole representation on $\partial\Omega$, as

$$u(\mathbf{x}) = \int \frac{\partial G}{\partial n_{\mathbf{y}}}(\mathbf{x},\mathbf{y})\psi(\mathbf{y})dS_{\mathbf{y}}, \qquad x \in \Omega^c, \tag{1.3}$$

called a double-layer potential. The function $\psi(\mathbf{x})$ can now be thought of as a dipole density, since $\frac{\partial G}{\partial n_{\mathbf{y}}}$ is the potential resulting from a normalized difference of infinitesimally close point charges. Again, we wish to match the Dirichlet data $f(\mathbf{x})$ by letting $\mathbf{x} \to \partial\Omega$ from the exterior, but this time the limit is trickier. Because of the extra $\partial/\partial n_{\mathbf{y}}$ derivative, the kernel $\frac{\partial G}{\partial n_{\mathbf{y}}}(\mathbf{x}, \mathbf{y})$ is singular rather than weakly singular, and the double layer (1.3) is *not continuous* as $\mathbf{x}$ crosses the boundary $\partial\Omega$ (though it is everywhere else.)

For instance, in two and three space dimensions (hence $n = 1, 2$ for line and surface integrals respectively), we compute

$$G(\mathbf{x}, \mathbf{y}) = \log(|\mathbf{x} - \mathbf{y}|) \qquad \Rightarrow \qquad \frac{\partial G}{\partial n_{\mathbf{y}}} = -\frac{(\mathbf{x} - \mathbf{y}) \cdot n_{\mathbf{y}}}{|\mathbf{x} - \mathbf{y}|^2},$$

and

$$G(\mathbf{x}, \mathbf{y}) = \frac{1}{|\mathbf{x} - \mathbf{y}|} \qquad \Rightarrow \qquad \frac{\partial G}{\partial n_{\mathbf{y}}} = -\frac{(\mathbf{x} - \mathbf{y}) \cdot n_{\mathbf{y}}}{|\mathbf{x} - \mathbf{y}|^3}.$$

In order to precisely describe the discontinuity of (1.3) in $\mathbf{x}$, let

$$K(\mathbf{x}, \mathbf{y}) = \frac{\partial G}{\partial n_{\mathbf{y}}}(\mathbf{x}, \mathbf{y}), \qquad \mathbf{x}, \mathbf{y} \in \partial\Omega.$$

Note that we reserve the notation $K(\mathbf{x}, \mathbf{y})$ only for $\mathbf{x} \in \partial\Omega$. Treating the case $\mathbf{x} \in \partial\Omega$ on its own is important, because gains extra regularity in that case. Consider the numerator $(\mathbf{x} - \mathbf{y}) \cdot n_{\mathbf{y}}$ in either expression above: when $\mathbf{x} \in \partial\Omega$, we have orthogonality of the two vectors in the limit $\mathbf{x} \to \mathbf{y}$, yielding

$$|(\mathbf{x} - \mathbf{y}) \cdot n_{\mathbf{y}}| \leq c|\mathbf{x} - \mathbf{y}|^2,$$

where the constant c depends on curvature. This implies that $K(\mathbf{x}, \mathbf{y})$ for $\mathbf{x} \in \partial\Omega$ is less singular by a whole power of $|\mathbf{x}-\mathbf{y}|$ than $\frac{\partial G}{\partial n_{\mathbf{y}}}(\mathbf{x}, \mathbf{y})$ would otherwise be outside of $\partial\Omega$. From the expressions above, we now see that $K(\mathbf{x}, \mathbf{y})$ is bounded in two dimensions, and weakly singular in three dimensions. As a consequence, the operator T_K defined by

$$(T_K\psi)(\mathbf{x}) = \int K(\mathbf{x}, \mathbf{y})\psi(\mathbf{y})dS_{\mathbf{y}}, \qquad x \in \partial\Omega, \tag{1.4}$$

is *compact* like T_G was.

The *jump conditions* for the double-layer potential (1.3) can now be formulated. For $\mathbf{x} \in \partial\Omega$, let $u_-(\mathbf{x})$ be the limit of $u(\mathbf{x})$ for $\mathbf{x}$ approaching $\partial\Omega$

form the inside, and let $u_+(\mathbf{x})$ be the limit of $u(\mathbf{x})$ for $\mathbf{x}$ approaching $\partial\Omega$ form the outside. It is possible to show that

$$u_-(\mathbf{x}) = \frac{1}{2}\psi(\mathbf{x}) + (T_K\psi)(\mathbf{x}),$$

and

$$u_+(\mathbf{x}) = -\frac{1}{2}\psi(\mathbf{x}) + (T_K\psi)(\mathbf{x}).$$

The total jump across the interface is $u_+(\mathbf{x}) - u_-(\mathbf{x}) = \psi(\mathbf{x})$.

The boundary match is now clear for the double-layer potential in the Dirichlet case. The boundary integral is either $f(\mathbf{x}) = \frac{1}{2}\psi(\mathbf{x}) + (T_K\psi)(\mathbf{x})$ in the interior case, and $f(\mathbf{x}) = -\frac{1}{2}\psi(\mathbf{x}) + (T_K\psi)(\mathbf{x})$ in the exterior case. Both equations are called second-kind Fredholm integral equation.

Since T_K is compact, neither $-I/2 + T_K$ nor $I/2 + T_K$ can be (because the identity isn't compact). The respective accumulations points for the eigenvalues are at $-1/2$ and $1/2$. This situation is much more favorable from a numerical point of view: second-kind integrals are in general well conditioned, and Krylov subspace methods need relatively few iterations to converge. The number of eigenvalues close to zero stays small, and the condition number does not grow as the discretization gets refined. For the two Dirichlet problems, the second kind formulation is therefore much preferred.

Both the single-layer potential and the double-layer potential formulations are available for the Neumann problems as well. Matching the normal derivative is trickier. For all $x \notin \partial\Omega$, we have

$$\frac{\partial u}{\partial n_{\mathbf{x}}}(\mathbf{x}) = \int \frac{\partial G}{\partial n_{\mathbf{x}}}(\mathbf{x}, \mathbf{y})\psi(\mathbf{y})dS_{\mathbf{y}}. \tag{1.5}$$

The kernel $\frac{\partial G}{\partial n_{\mathbf{x}}}(\mathbf{x}, \mathbf{y})$ is the adjoint of the double layer potential's $\frac{\partial G}{\partial n_{\mathbf{y}}}(\mathbf{x}, \mathbf{y})$. For $\mathbf{x} \in \partial\Omega$, equation (1.5) becomes the application of the adjoint of T_K, namely

$$(T_K^*\psi)(\mathbf{x}) = \int K(y, x)\psi(\mathbf{y})dS_{\mathbf{y}}, \qquad x \in \partial\Omega.$$

The (1.5) is discontinous as $\mathbf{x}$ crosses $\partial\Omega$, and inherits its jump conditions from those of (1.3). Notice that the signs are reversed:

$$\frac{\partial u}{\partial n_-}(\mathbf{x}) = -\frac{1}{2}\psi(\mathbf{x}) + (T_K^*\psi)(\mathbf{x}),$$

and

$$\frac{\partial u}{\partial n_+}(\mathbf{x}) = \frac{1}{2}\psi(\mathbf{x}) + (T_K^*\psi)(\mathbf{x}).$$

The single-layer potential therefore gives rise to a favorable second-kind integral equation in the Neumann case – it is the method of choice. In the Neumann case the double-layer potential gives rise to an unwieldy hypersingular kernel that we will not deal with in these notes[1].

There exist three prevalent discretization methods for boundary integral equations of the form $u(\mathbf{x}) = \int K(\mathbf{x},\mathbf{y})\phi(\mathbf{y})dS_{\mathbf{y}}$.

1. *Nyström methods.* Direct, though perhaps sophisticated quadrature of the kernel, and pointwise evaluation as

$$u(\mathbf{x}_i) \simeq \sum_{j\neq i} K(\mathbf{x}_i,\mathbf{y}_j)\phi(\mathbf{y}_j)\omega_j,$$

   for some weights ω_j. Then solve for $\phi(\mathbf{y}_j)$.

2. *Collocation methods.* Expand $\phi(\mathbf{y}) = \sum_j \phi_j v_j(\mathbf{y})$ in some basis set $v_j(\mathbf{y})$, and evaluate pointwise to get

$$u(\mathbf{x}_i) \simeq \sum_j \phi_j \int K(\mathbf{x}_i,\mathbf{y})v_j(\mathbf{y})dS_{\mathbf{y}}.$$

   Then solve for ϕ_j.

3. *Galerkin methods.* Pick a basis set $v_j(\mathbf{x})$, expand $\phi(\mathbf{y}) = \sum_j \phi_j v_j(\mathbf{y})$, $u(\mathbf{x}) = \sum_j u_j v_j(\mathbf{x})$, and test against $v_i(\mathbf{x})$ to get

$$\sum_j u_i\langle v_i, v_j\rangle = \sum_j \phi_j\langle v_i, T_K v_j\rangle.$$

   Then solve for ϕ_j.

[1]Hypersingular kernels grow faster than $1/|\mathbf{x}|^n$ in n dimensions, but are well-defined in the sense of distributions thanks to cancellation conditions. They are sometimes handled numerically by the so-called Maue identities.

1.4 Exercises

1. Show that the right-hand side g for the interior Neumann Laplace equation must obey
$$\int_{\partial\Omega} g(\mathbf{x}) dS_{\mathbf{x}} = 0.$$
[Hint: use a Green's identity.]

Chapter 2

Fourier-based methods

2.1 PDE and discrete sine/cosine transforms

(...)

Throughout these notes we use the convention

$$F_{jk} = e^{-2\pi ijk/N}, \qquad (F^{-1})_{jk} = \frac{1}{N} e^{2\pi ijk/N},$$

with $0 \le j, k \le N-1$. Notice that $F^{-1} = \frac{1}{N} F^*$.

2.2 Integral equations, convolutions

We consider the problem of computing potentials from charges. After discretization, we have

$$u_j = \sum G_{jk} q_k,$$

where G is a Green's function such as $\frac{1}{2\pi} \log(|x-y|)$ in 2D, or $\frac{1}{4\pi|x-y|}$ in 3D, or even a Helmholtz Green's function. The numerical values G_{jk} either follow from a Galerkin formulation (finite elements), or a quadrature of the integral (Nyström). The simplest Galerkin discretization of G consists in clustering charges inside panels.

Direct summation takes $O(N^2)$ operations. In this section we present $O(N \log N)$ FFT-based algorithms for the summation $u = Gq$.

- *Periodic ring of charges.* By translation-invariance, periodicity, and symmetry, we get

$$\begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} g_0 & g_1 & \cdots & g_{N-1} \\ g_1 & g_0 & \cdots & g_{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ g_{N-1} & g_{N-2} & \cdots & g_0 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_{N-1} \end{pmatrix} \tag{2.1}$$

  with $g_j = g_{N-j}$. In other words, $G_{jk} = g_{j-k}$. The resulting matrix is both Toeplitz (constant along diagonals) and circulant (generated by cyclic shifts of the first column). It is the circulation property which enables a fast Fourier-based algorithm. The expression of u can also be written as a cyclic convolution:

$$u_j = (g \star q)_j = \sum_{k=0}^{N-1} g_{(j-k) \bmod N}\, q_k,$$

  where mod is the remainder of the integer division by N.

  The discrete Fourier transform diagonalizes cyclic convolutions: it is a good exercise to show that

$$(F(g \star q))_k = (Fg)_k (Fq)_k,$$

  or, in matrix form,

$$g \star q = F^{-1} \Lambda F q, \qquad \Lambda = \operatorname{diag}(F(g)).$$

  The fast algorithm is clear: 1) Fourier-transform q, 2) multiply componentwise by the Fourier transform of g, and 3) inverse Fourier-transform the result. Notice that we also get fast inversion, since $q = F^{-1} \Lambda^{-1} F u$.

- *Straight rod of charges.* The formulation is the same as previously, namely (2.1) or $G_{jk} = g_{j-k}$, but without the circulant property $g_{N-j} = g_j$. We are now in presence of a Toeplitz matrix that realizes an "ordinary" convolution, without the mod operation:

$$u_j = \sum_{k=0}^{N-1} g_{j-k} q_k,$$

with $0 \le j \le N-1$, hence $-(N-1) \le j-k \le N-1$, and the convention that $g_{-j} = g_j$. The key to a fast algorithm is to view the result of this convolution as a piece of an almost-twice-bigger cyclic convolution via

$$\begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \\ \hline \times \\ \vdots \\ \times \end{pmatrix} = \left(\begin{array}{cccc|cccc} g_0 & g_1 & \cdots & g_{N-1} & g_{N-2} & g_{N-3} & \cdots & g_1 \\ g_1 & g_0 & \cdots & g_{N-2} & g_{N-1} & g_{N-2} & \cdots & g_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{N-1} & g_{N-2} & \cdots & g_0 & g_1 & g_2 & \cdots & g_{N-2} \\ \hline g_{N-2} & g_{N-1} & \cdots & g_1 & g_0 & g_1 & \cdots & g_{N-3} \\ g_{N-3} & g_{N-2} & \cdots & g_2 & g_1 & g_0 & \cdots & g_{N-4} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & \cdots & g_{N-2} & g_{N-3} & g_{N-4} & \cdots & g_0 \end{array}\right) \begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_{N-1} \\ \hline 0 \\ \vdots \\ 0 \end{pmatrix} \tag{2.2}$$

Note that q is zeropadded, while the G matrix is grown from a size N by N to a size $2N-2$ by $2N-2$. We can understand the extended matrix as generated by cyclic shifts of its first column $\tilde{g}$, which is in turn obtained by mirror extension of the first column g of the original G matrix. Note that the components N through $2N-3$ of the output are unneeded.

The fast algorithm is: 1) zeropad q with $N-2$ zeros to get $\tilde{q}$ of length $2N-2$; 2) mirror-extend g (without repetition of the endpoints) to get $\tilde{g}$ of length $2N-2$; 3) perform the cyclic convolution of $\tilde{q}$ with $\tilde{g}$, namely,

$$u_j = \sum_{k=0}^{2N-3} \tilde{g}_{(j-k) \bmod (2N-2)}\, \tilde{q}_k,$$

via FFT as we saw before; and 4) restrict the result to the first N components $0 \le j \le N-1$. This strategy only allow to multiply fast with G, not to invert it (because we do not have access to the unknown components labeled $\times$ above.)

- *Square plate of charges.* We are still in presence of translation-invariant interactions in two spatial dimensions, namely

$$G_{j_1,j_2,k_1,k_2} = g_{j_1-k_1,j_2-k_2},$$

which makes G into a "block Toeplitz matrix with Toeplitz blocks" when ordering its elements in lexicographical (comic book) order. To handle the negative indices, we still impose symmetry with $g_{-j_1,j_2} =$

$g_{j_1,-j_2} = g_{-j_1,-j_2} = g_{j_1,j_2}$. We now have a convolution of arrays rather than a convolution of vectors:

$$u_{j_1,j_2} = \sum_{k_1,k_2=0}^{N-1} g_{j_1-k_1,j_2-k_2} q_{k_1,k_2},$$

with $0 \le j_1, j_2 \le N-1$, hence $-(N-1) \le j_1 - k_1, j_2 - k_2 \le N-1$. It can be turned into a cyclic convolution with a similar trick as before: 1) zeropad q into a $2N-2$ by $2N-2$ array $\tilde{q}$, 2) mirror-extend g into a $2N-2$ by $2N-2$ array $\tilde{g}$ via

$$\tilde{g} = \left(\begin{array}{cccc|cccc} g_{0,0} & g_{0,1} & \cdots & g_{0,N-1} & g_{0,N-2} & g_{0,N-3} & \cdots & g_{0,1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,N-1} & g_{1,N-2} & g_{1,N-3} & \cdots & g_{1,1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{N-1,0} & g_{N-1,1} & \cdots & g_{N-1,N-1} & g_{N-1,N-2} & g_{N-1,N-3} & \cdots & g_{N-1,1} \\ \hline g_{N-2,0} & g_{N-2,1} & \cdots & g_{N-2,N-1} & g_{N-2,N-2} & g_{N-2,N-3} & \cdots & g_{N-2,1} \\ g_{N-3,0} & g_{N-3,1} & \cdots & g_{N-3,N-1} & g_{N-3,N-2} & g_{N-3,N-3} & \cdots & g_{N-3,1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{1,0} & g_{1,1} & \cdots & g_{1,N-1} & g_{1,N-2} & g_{1,N-3} & \cdots & g_{1,1} \end{array}\right)$$

then 3) do the cyclic convolution of the extended arrays, namely

$$u_{j_1,j_2} = \sum_{k_1,k_2=0}^{2N-3} \tilde{g}_{(j_1-k_1) \bmod (2N-2),(j_2-k_2) \bmod (2N-2)}\, \tilde{q}_{k_1,k_2},$$

via 2D FFT, and finally 4) restrict the indices of the result as $0 \le j_1, j_2 \le N-1$. Note that this operation of mirror extension of the array g into $\tilde{g}$ has *nothing to do* with the operation of creating the circulant matrix from the Toeplitz matrix that we saw in the straight rod case – it is not the same operation at all.

Forming the large matrix G is never needed – only g enters the computation. The resulting fast matrix-vector multiplication has complexity $O(N^2 \log N)$.

2.3 Krylov subspace methods

2.4 Ewald summation

2.5 Nonuniform Fourier methods

2.6 Exercises

1. Show the discrete convolution theorem, namely $(F(g \star q))_k = (Fg)_k (Fq)_k$, where F is the discrete Fourier transform and $\star$ is cyclic convolution.

Chapter 3

Fast multipole methods

In this chapter, we consider the problem of computing integrals of the form

$$u(x) = \int G(x, y)q(y)\, dy,$$

or its discrete counterpart, the N-body interaction problem

$$u_i = \sum_{j=1}^{N} G(x_i, y_j)q_j, \qquad i = 1, \ldots, N. \tag{3.1}$$

3.1 Projection and interpolation

Consider a source box B, with N_B charge q_j at points $y_j \in B$. Consider an evaluation box A, disjoint and well-separated from B, with N_A evaluation points $x_i \in A$. We first address the simplification of (3.1) where sources are restricted to B, and evaluation points restricted to A:

$$u_i = \sum_{y_j \in B} G(x_i, y_j)q_j, \qquad x_i \in A.$$

Direct sum evaluation costs $N_A N_B$ operations. In this section we explain how to lower this count to $O(N_A+N_B)$ operations with projection or interpolation, or both. In the next section, we lift the requirement of separation of the two boxes and return to the full problem (3.1).

Given a source box B and an evaluation box A, a *projection rule* is a way to replace the N_B charges q_j at $y_j \in B$ by a smaller number of equivalent, or

canonical charges q_m^B at points y_m^B, such that the potential is reproduced to good accuracy in A:

$$\sum_{y_j \in B} G(x, y_j) q_j \simeq \sum_m G(x, y_m^B) q_m^B, \qquad x \in A. \tag{3.2}$$

We also require that the dependence from q_j to q_m^B is linear, hence we write

$$q_m^B = \sum_{y_j \in B} Q_m^B(y_j) q_j. \tag{3.3}$$

The simplest projection rule consists in placing the total charge $q^B = \sum_j q_j$ at the center y^B of the box B. Upon evaluation at the N_A points x_i, the resulting approximation $G(x_i, y^B)q^B$ of the potential is obtained by the two-step algorithm: 1) sum the charges, in N_B operations, and 2) evaluate $G(x_i, y^B)q^B$ for each x_i, in $O(N^A)$ operations, for a total of $O(N_A + N_B)$ operations. More generally if there are r canonical charges, then the complexity becomes $O(r(N_A + N_B))$.

Given a source box B and an evaluation box A, an *interpolation rule* is a way to replace the N_A potentials $u(x_i)$ at $x_i \in A$, generated by sources in B, by a smaller number of equivalent, or canonical potentials $u_n^A = u(x_n^A)$ at points x_n^A, such that the potential is reproduced to good accuracy in A. We also require that the map from u_n^A to $u(x_i)$ is linear, hence we write

$$u(x_i) \simeq \sum_n P_n^A(x_i) u_n^A. \tag{3.4}$$

For a single source at $y \in B$, this becomes

$$G(x_i, y) \simeq \sum_n P_n^A(x_i) G(x_n^A, y), \qquad y \in B. \tag{3.5}$$

The simplest interpolation rule consists in assigning the same potential $G(x^A, y)$, where x^A is the center of box A, to every point $x \in A$. Upon summation over N_B charges q_j, the resulting approximation $\sum_j G(x^A, y_j)q_j$ of the potential is obtained by the two-step algorithm: 1) evaluate $G(x^A, y_j)q_j$ for each $y_j \in B$, then sum over j, in $O(N_B)$ operations, 2) assign the result to every $x_i \in A$, in N_A operations, for a total of $O(N_A + N_B)$ operations. More generally if there are r canonical potentials, then the complexity becomes $O(r(N_A + N_B))$.

Projection and interpolation are in a sense dual of one another. Any projection rule Q can serve as an interpolation rule P with inherited accuracy properties, and vice-versa, as we now argue.

- *From interpolation to projection.* Start from $\sum_{y_j \in B} G(x, y_j) q_j$, and interpolate G in the y variable rather than the x variable – which is fine since G is symmetric[1]. We get

$$\begin{aligned} \sum_{y_j \in B} G(x, y_j) q_j &\simeq \sum_{y_j \in B} \sum_m G(x, y_m^B) P_m^B(y_j) q_j \\ &= \sum_m G(x, y_m^B) \left(\sum_{y_j \in B} \sum_m P_m^B(y_j) q_j \right). \end{aligned}$$

  In view of (3.2), we recognize equivalent charges

$$q_m^B = \sum_{y_j \in B} \sum_m P_m^B(y_j) q_j,$$

  where P plays the role of Q.

- *From projection to interpolation.* Start from $G(x_i, y)$, and understand this expression as the potential at y generated by a unit point charge at x_i. Perform a projection in the x variable – which is fine since G is symmetric. We get

$$\begin{aligned} G(x_i, y) &\simeq \sum_n G(x_n^A, y) q_n^A, \qquad \text{with } q_n^A = Q_n^A(x_i). \\ &= \sum_n Q_n^A(x_i) G(x_n^A, y). \end{aligned}$$

  In view of (3.5), we recognize an interpolation scheme with Q in place of P.

An inspection of equations (3.3) and (3.4) reveals that *interpolation is the transpose of projection.* To see this, pick $P = Q$, $m = n$, and $i = j$. Then equation (3.3) can be seen as a matrix-vector product involving the short-and-wide matrix $P_n^A(x_i)$ with row index n and column index i. On the other hand,

[1] An exercise at the end of the chapter concerns the proof of the symmetry of G when it arises as a Green's function for a symmetric elliptic problem.

equation (3.4) can be seen as a matrix-vector product involving the tall-and-thin matrix $P_n^A(x_i)$ with row index i and column index n. Hence the matrices that take part in (3.3) vs. (3.4) are effectively transposes of one another. In this context, Brandt once referred to projection as "anterpolation", though the name doesn't seem to have caught on [**?**].

It is important to point out that the matrices P and Q are constrained by the properties of interpolation and projection. Because $u_n^A = u(x_n^A)$ by definition of interpolation, we can place an evaluation point x_i at each $x_{n'}^A$ and obtain $u(x_{n'}^A) = \sum_n P_n(x_{n'}^A) u(x_n^A)$. When enough degrees of freedom specify the potential[2], this relation is only possible when

$$P_n(x_{n'}^A) = \delta_{n,n'}.$$

Projection matrices should obey an analogous relationship. It makes sense to require a single charge q at $y_{m'}^B$ to be projected onto a collection of canonical charges q_m^B at y_m^B which are all zero except when $m = m'$, where the canonical charge $q_m^B = q$. In view of (3.3), this is only possible when

$$Q_m^B(y_{m'}^B) = \delta_{m,m'}.$$

As a corollary, if Q maps charges at y_j to canonical charges at y_m^B, then performing the same projection a second time no longer affects the charge distribution: that's the " $Q^2 = Q$ " criterion according to which a linear operator is called a projection in linear algebra.

Here are the most important examples of projection / interpolation.

3.1.1 Polynomial interpolation

A familiar example is to let $\sum_n P_n^A(x) f(x_n^A)$ be a multivariable polynomial interpolant of $f(x)$ through the points $(x_n^A, f(x_n^A))$. We call $P_n^A(x)$ the elementary Lagrange polynomials. In the sequel we illustrate constructions in 2D, though the extension to 3D is straightforward.

The simplest example was mentioned earlier: order-zero interpolation with the constant equal to the function evaluation at the center of the box, $x = x^A$. In that case $P_n^A(x)$ ranges over a single index n and takes on the value 1 regardless of x. The next simplest example is bilinear interpolation from the knowledge of the function at the four corners of the box A, say

[2]It is enough that $G(x_n^A, y_j)$, as a matrix with indices n and j, is full row rank.

$(0,0), (0,1), (1,0), (1,1)$. The idea is to interpolate linearly in the first dimension, then again in the other dimension. The result is not linear but quadratic in $x = (x_1, x_2)$, and can be written as

$$\begin{aligned} f(x) \simeq & f(0,0)\,(1-x_2)(1-x_1) + f(0,1)\,(1-x_2)x_1 \\ & + f(1,0)\,x_2(1-x_1) + f(1,1)\,x_2 x_1. \end{aligned}$$

Higher-order polynomial interpolation is advantageously done on a tensor Chebyshev grid. (Polynomial interpolation on Cartesian grids suffers from the Runge phenomenon.)

The transpose of polynomial interpolation gives simple, explicit projection rules of the form $q_n^A = \sum_j P_n^A(x_j) q_j$. Order-zero projection was already mentioned earlier, and it is now clear that it is simply the rule that arises by taking the same $P_n^A(x) = 1$ as order-zero interpolation, hence results in summing the charges. The bilinear projection rule that corresponds to bilinear interpolation is also clear: there are four canonical charges placed at the four corners of the square and given by

$$\begin{aligned} q_{(0,0)} &= \sum_j (1-x_{2,j})(1-x_{1,j}) q_j, & q_{(0,1)} &= \sum_j (1-x_{2,j}) x_{1,j} q_j, \\ q_{(1,0)} &= \sum_j x_{2,j}(1-x_{1,j}) q_j, & q_{(1,1)} &= \sum_j x_{2,j} x_{1,j} q_j. \end{aligned}$$

3.1.2 Collocation

Collocation is the projection rule where the canonical charges are determined by matching, at locations $\check{x}_i$ called check points, the potential generated by arbitrary charges q_j at y_j. These points $\check{x}_i$ are chosen so that

$$\sum_m G(\check{x}_i, y_m^B) q_m^B \simeq \sum_{y_j \in B} G(\check{x}_i, y_j) q_j$$

is not only a well-posed system for q_m^B, but also results in an approximation $\sum_m G(x, y_m^B) q_m^B$ accurate for x in a large region. A good choice is to pick $\check{x}_i$ on a "check curve" (or check surface) enclosing B, in an equispaced or near-equispaced manner, and with a number of $\check{x}_i$ greater than the number of y_m^B. The projection matrix is then given by

$$Q_m^B(y_j) = \sum_i [\, G(\check{x}_i, y_m^B) \,]_{m,i}^{+}\, G(\check{x}_i, y_j). \tag{3.6}$$

This strategy provably gives a good approximation of the potential in the exterior of the check surface, because of the following two facts: 1) the potential is smooth, hence well interpolated everywhere on the check surface from the $\check{x}_i$; and 2) because G typically obeys an elliptic equation with decay conditions at infinity, the error on the potential obeys a maximum principle, hence must decay in the exterior region.

Conversely, we may define interpolation via collocation as follows. The interpolation function $P_n^A(x_i)$ is determined by requiring that the potential generated by unit charges at check points $\check{y}_j$ is reproduced by interpolation from samples at x_n^A. This leads to

$$G(x_i, \check{y}_j) = \sum_n P_n^A(x_i) G(x_n^A, \check{y}_j),$$

For each x_i the above should be a well-posed system for $P_n^A(x_i)$ (indexed by n,) and the resulting interpolation rule $\sum_n P_n^A(x_i) G(x_n^A, y)$ should be accurate for y in a large region. A good choice is to pick $\check{y}_j$ on a check curve (or check surface) enclosing A, in an equispaced or near-equispaced manner, and with a number of $\check{y}_j$ greater than the number of x_n^A. The interpolation matrix is then given by

$$P_n^A(x_i) = \sum_j [\, G(x_n^A, \check{y}_j) \,]^+_{j,n} G(x_i, \check{y}_j). \tag{3.7}$$

The accuracy is inherited from that of the corresponding projection rule (3.6), as we have seen, hence will be good as soon as y is located outside the check surface. By reciprocity of G, we see that (3.7) is the same as (3.6) under the relabeling $B \to A$, $y_j \to x_i$, $\check{y}_j \to \check{x}_i$, and $y_m^B \to x_n^A$.

3.2 Multipole expansions

Linear algebra invites us to think of $G(x, y)$ as a (possibly continuous) matrix with row index x and column index y. Recall that $x \in A$ and $y \in B$, so we are in effect considering a block of G. Expressions like (3.5), or its counterpart in the y variable, are ways of separating this block of G into low-rank factors. Multipole expansions offer an alternative construction of these factors, though not immediately in projection/interpolation form. We return to the topic of low-rank expansions in section 4.1.

In this section, we consider $x, y \in \mathbb{R}^2$, and

$$G(x, y) = \log(|x - y|).$$

As previsouly, consider $x \in A$ and $y \in B$, two well-separated boxes. Let r be the radius of (the circumscribed circle of) B, and $d > 2r$ be the distance from the center of B to the box A. For convenience, but without loss of generality, the box B is centered at the origin.

Theorem 1. *Consider A and B as described above. For all $p > 0$, there exist functions $f_k(x)$, $g_k(y)$ for $1 \leq k \leq 2p+1$, and a constant $C_p > 0$, such that*

$$\max_{(x,y)\in A\times B} |\log(|x - y|) - \sum_{k=1}^{2p+1} f_k(x)g_k(y)| \leq C_p \left(\frac{r}{d}\right)^p.$$

The construction of f_k and g_k is made explicit as part of the proof – which the reader will need to consult to follow the rest of this section.

Proof. Pass to complex variables $z_x = x_1 + ix_2 = \rho_x e^{i\theta_x}$ and $z_y = y_1 + iy_2 = \rho_y e^{i\theta_y}$. The reader can check that

$$\log(|x - y|) = \operatorname{Re} \log(z_x - z_y),$$

so the kernel is harmonic in z_x, as well as in z_y, as long as $z_x \neq z_y$. Perform a complex-variable Taylor expansion in z_y, centered at 0, and valid for $|z_x| > |z_y|$:

$$\log(z_x - z_y) = \log(z_x) - \sum_{k=1}^{\infty} \frac{1}{k} \left(\frac{z_y}{z_x}\right)^k \tag{3.8}$$

Upon taking the real part, a short calculation shows that

$$\operatorname{Re} \left(\frac{z_y}{z_x}\right)^k = \frac{\cos(k\theta_x)}{|x|^k} \cos(k\theta_y)|y|^k + \frac{\sin(k\theta_x)}{|x|^k} \sin(k\theta_y)|y|^k, \tag{3.9}$$

hence each term in the sum has rank 2. Truncate the series at $k = p$, and use $|z_y| = |y| \leq r$, $|z_x| = |x| \geq d$ to conclude

$$|\log(z_x - z_y) - \log(z_x) + \sum_{k=1}^{p} \frac{1}{k} \left(\frac{z_y}{z_x}\right)^k| \leq |\sum_{k=p+1}^{\infty} \frac{1}{k} \left(\frac{z_y}{z_x}\right)^k| \leq \frac{1}{p+1} \frac{(r/d)^p}{1 - r/d}.$$

The same bound holds for the real part of log. Since $d > 2r$, the denominator of the right-hand side is absorbed in the constant. The resulting approximation has $2p + 1$ separated terms. □

Equation (3.8) can either be seen as a Taylor-harmonic expansion in the $z_y = y_1 + iy_2$ variable (because the resulting polynomial is harmonic, as a function of the two variables y_1 and y_2), or a so-called multipole expansion in the $z_x = x_1 + ix_2$ variable. Indeed, we see from equation (3.9) that the term involving $1/z_x$ decomposes into a sum of two functions of $x = (x_1, x_2)$:

$$\frac{\cos(\theta_x)}{|x|} = \frac{x_1}{x_1^2 + x_2^2} = \frac{\partial}{\partial x_1} \log |x|,$$

and

$$\frac{\sin(\theta_x)}{|x|} = \frac{x_2}{x_1^2 + x_2^2} = \frac{\partial}{\partial x_2} \log |x|.$$

These are the expressions of two dipoles, oriented respectively horizontally and vertically. All the other orientations are obtained from their linear combinations. At higher order, there are again two multipoles $\frac{\cos(k\theta_x)}{|x|^k}$ and $\frac{\sin(k\theta_x)}{|x|^k}$, corresponding to the non-mixed higher partials of log.

The multipole expansion can be used to compute interactions from charges in box B to potentials in box A in an efficient way. Use (3.8) to write the potential $u(x_i) = \sum_j \log(|x_i - y_j|)q_j$ as

$$u(x_i) = \operatorname{Re} \log(z_{x_i}) \sum_j q_j - \operatorname{Re} \sum_k \frac{1}{z_{x_i}^k} \left(\sum_j \frac{1}{k} z_{y_j}^k q_j \right). \tag{3.10}$$

We call $\mu_0 = \sum_j q_j$ and $\mu_k = \sum_j \frac{1}{k} z_{y_j}^k q_j$ the *moments* associated with the charges q_j at y_j. The fast algorithm for computing $u(x_i)$ is as follows: 1) compute r moments μ_k in complexity $O(rN_B)$, and 2) assign the moments to multipole components $\log(z_{x_i})$ and $1/z_{x_i}^k$ evaluated at the x_i, in complexity $O(rN_A)$. Finish by taking a real part. (The corresponding expressions that involve only real multipoles instead of their convenient complex counterpart can be obtained by using (3.9).)

Multipole expansions not only provide an explicit construction of a low-rank decomposition on well-separated boxes, but they themselves fit in the previous section's framework of projection and interpolation. In order to justify this, continue with the complex formalism and let $G(x, y) = \log(z_x - z_y)$.

- Let us first argue why equation (3.8), seen as a Taylor-harmonic series in z_y, yields a projection rule. The multipole $1/z_x^k$ is interpreted, like

in the proof of theorem 1, as a partial derivative of the kernel in z_y, via

$$\frac{1}{z_x^k} = -\frac{1}{(k-1)!}\left(\frac{\partial}{\partial z_y}\right)^k \log(z_x - z_y)|_{z_y=0}, \qquad k \geq 1.$$

We can approximate the partial derivative, to arbitrary accuracy, by a finite difference formula at points y_m^B close to the origin, and obtain

$$\left(\frac{\partial}{\partial z_y}\right)^k \log(z_x - z_y)|_{z_y=0} \simeq \sum_m T_{mk} G(x, y_m^B), \qquad k \geq 1. \tag{3.11}$$

Substituting in (3.10), we recognize a projection rule

$$u(x_i) = \mathrm{Re}\, G(x_i, 0) \sum_j Q_{0,j} q_j + \mathrm{Re} \sum_m G(x_i, y_m^B) \sum_j Q_m^B(y_j) q_j,$$

with projection functions

$$Q_{0,j} = 1, \qquad Q_m^B(y_j) = -\sum_k \frac{T_{mk}}{k!} z_{y_j}^k. \tag{3.12}$$

- The corresponding (transpose) interpolation rule is tied to the Taylor-harmonic series expansion in z_x rather than z_y. For a single charge at y, let the center of the evaluation box be x^A, so we perform a Taylor expansion in z_x about z_{x^A}. This gives

$$G(x_i, y) = G(x^A, y) + \sum_{k \geq 1} z_{x_i - x^A}^k \frac{1}{k!}\left(\frac{\partial}{\partial z_x}\right)^k G(x, y)|_{x=x^A}. \tag{3.13}$$

Unlike in the previous bullet point (but along the lines of the interpretation of multipoles discussed above equation (3.10)), we are now in presence of partials with respect to z_x, not z_y. Approximate the derivative by a finite difference formula over points x_n^A clustered about x_A, as

$$\left(\frac{\partial}{\partial z_x}\right)^k \log(z_x - z_y)|_{z_x=z_{x^A}} \simeq -\sum_n T_{nk} G(x_n^A, y)$$

The minus sign is necessary to identify the T_{nk} above with the T_{mk} in equation (3.11), again because the derivative is in z_x rather than in z_y. Substituting in (3.13), we recognize an interpolation rule

$$G(x_i, y) \simeq P_{0,i} G(x^A, y) + \sum_n P_n^A(x_i) G(x_n^A, y),$$

with interpolation functions

$$P_{0,i} = 1, \qquad P_n^A(x_i) = -\sum \frac{T_{nk}}{k!} z^k_{x_i - x^A}.$$

These expressions are the same as in equation (3.12), after relabeling $P \to Q$, $x_i \to y_j$, $x^A \to 0$.

It should be empashized that the multipole expansion in the x variable, defined in (3.10), can also give rise to an interpolation scheme, but that this scheme is *not* dual to the projection scheme that (3.10) defines when seen as a Taylor expansion in the y variable. Instead, a "multipole" interpolation rule could be obtained by viewing (3.10) as a Laurent expansion in z_x, and discretize the contour integral for its coefficients with a quadrature. We leave the details to the reader.

There exists a 3D version of the multipole expansion that uses spherical harmonics. We also leave this topic out.

3.3 The fast multipole method

In this section we present the interpolative fast multipole method (FMM), where the kernel is assumed to have similar smoothness properties as $1/|x-y|$ or $\log(|x-y|)$. We continue to assume that G is symmetric, and we assume two spatial dimensions for the remainder of this section.

3.3.1 Setup

Consider a dyadic tree partitioning of the domain $[0,1]^2$, say, into target boxes A and sources boxes B, of sidelength $2^{-\ell}$ with $\ell = 0, 1, \ldots, L$. This collection of boxes comes with a quadtree structure: call A_p the parent of a box A, and call B_c the four children of a box B. The highest level of the tree ($\ell = 0$) is called root, while the lowest-level boxes are called leaves. The root is usually visualized at the top. The tree may not be "perfect", in the sense that not all leaf boxes occur at the same level. We also allow for the situation where the tree is cut before reaching the unique root, allowing instead for several root boxes (a case in which we should strictly speaking no longer be referring to the structure as a tree, though we will continue with the abuse of vocabulary.)

Two boxes at the same scale are said to be *well-separated* if they are not adjacent to one another, i.e., if B is neither A nor any of its eight nearest neighbors. In that case we say that the boxes are in the far-field of one another: $B \in \text{far(A)}$, or equivalently $A \in \text{far(B)}$.

In general, what "sufficiently far" means is kernel-dependent, and is first and foremost a linear-algebraic notion rather than a geometrical one: A and B are well-separated when the restriction of $G(x, y)$ to $x \in A$ and $y \in B$ has a low numerical rank. In the Laplace of low-frequency Hemholtz case, this is precisely the case when A is not adjacent to B (as we proved for the log kernel in the previous section.)

Next, we define the *interaction list* of a box A to be the set of boxes B, at the same scale as A, in the far-field of A, and such that B_p is not in the far field of A_p. In other words, the interaction list is the set of boxes which are "far but not too far". We denote it as $IL(A)$. In 2D, the interaction list has 27 boxes when the far field is defined as earlier. (This number becomes 189 in 3D.) Correspondingly, we denote by $NL(A)$ the neighbor list of A, consisting of A and its 8 nearest neighbors at the same scale.

We will consider the potentials resulting from separated interactions. For each source box B, we let the partial potential from B be

$$u^B(x) = \int_B G(x, y)q(y)dy, \qquad x \in \text{far(B)}. \tag{3.14}$$

For each evaluation box A, we let the partial potential to A be

$$u^{\text{far(A)}}(x) = \int_{\text{far(A)}} G(x, y)q(y)dy, \qquad x \in A. \tag{3.15}$$

Finally, we need a projection rule

$$G(x, y) = \sum_m G(x, y_m^B)P_m^B(y), \qquad x \in \text{far(B)}, y \in B, \tag{3.16}$$

and an interpolation rule

$$G(x, y) = \sum_n P_n^A(x)G(x_n^A, y), \qquad x \in A, y \in \text{far(A)}. \tag{3.17}$$

In the sequel we do not keep track of the various truncation errors.

3.3.2 Basic architecture

The three main steps of the fast multipole method (FMM) are that 1) the projection rule is successively applied from finer to coarser scales to create canonical charges in sources boxes at all the levels of the tree; 2) the kernel G is used to compute potentials, at selected locations, from these canonical charges; and 3) the interpolation rule is successively applied from coarser to finer scales to compute canonical potentials in evaluation boxes at all levels of the tree.

More precisely,

- Consider canonical charges q_m^B at nodes y_m^B, expected to obey

$$u^B(x) = \sum_m G(x, y_m^B) q_m^B, \qquad x \in \text{far(B)}. \tag{3.18}$$

  The projection rule (3.16) gives a relation between the charges in B and the canonical charges q_m^B: combine (3.14), (3.16), and compare to (3.18) to get

$$q_m^B = \int_B P_m^B(y) q(y) dy. \tag{3.19}$$

  Apply this rule in the case when $q(y)$ are canonical charges in the children boxes:

$$q_m^B = \sum_c \sum_{m'} P_m^B(y_{m'}^{B_c}) q_{m'}^{B_c}. \tag{3.20}$$

  Because (3.18) is sometimes called a multipole expansion (even when the projection rule is not generated by multipoles), the relationship (3.20) is called a multipole-to multipole (M2M) operation, or M2M translation. The first step of the FMM is the cascade of M2M translations in an upward pass of the quadtree.

- At every scale, for every box A, and every box B in the interaction list of A, the canonical charges are converted into potentials via the so-called multipole-to-local (M2L) conversion rule

$$u_n^{A,M2L} = \sum_{B \in IL(A)} \sum_m G(x_n^A, y_m^B) q_m^B. \tag{3.21}$$

- Consider canonical potentials u_n^A at the nodes x_n^A, expected to obey

$$u_n^A = \int_{\text{far(A)}} G(x_n^A, y) q(y) dy. \tag{3.22}$$

The interpolation rule (3.17) gives a relation between the potentials in A and the canonical potentials u_n^A: combine (3.15), (3.17), and compare to (3.22) to get

$$u^{\text{far(A)}}(x) = \sum_n P_n^A(x) u_n^A, \qquad x \in A. \tag{3.23}$$

Apply this rule in the case when the canonical potentials are those of the parent box A_p:

$$u_n^{A,L2L} = \sum_{n'} P_{n'}^{A_p}(x_n^A) u_{n'}^{A_p}. \tag{3.24}$$

Because (3.23) is sometimes called a local expansion, the relationship (3.24) is called a local-to-local (L2L) operation, or L2L translation.

- Since

$$\text{far}(A) = \text{far}(A_p) \cup IL(A),$$

it remains to add the M2L and L2L potentials to obtain the canonical potentials for box A:

$$u_n^A = u_n^{A,M2L} + u_n^{A,L2L}.$$

The last step of the FMM is the cascade of L2L translations, added to the M2L conversions, in a downward pass of the quadtree.

3.3.3 Algorithm and complexity

Let N be the maximum of the number of charges and number of evaluation points. We build the quadtree adaptively so that the leaf boxes contain no more than s charges and evaluation points, with s to be determined later. Assume that the projection and interpolation rule both involve (no more than) p canonical charges and canonical potentials per box, i.e., both m and n run from 1 to p.

The FMM algorithm is as follows. Assume that the projection/interpolation rules, and the interaction lists, are precomputed (they don't depend on the particular charge distribution.)

1. *Initialization*

   Bin the y_j in boxes B at all scales, and the x_i in boxes A at all scales.

   Let L be the level of the finest leaf box.

   For B leaf boxes

   Source-to-multipole: $q_m^B = \sum_{y_j \in B} P_m^B(y_j) q_j$.

   End

   For A root box(es)

   $u_N^A = 0$

   End

2. *Upward pass*

   For $\ell = L-1, \ldots, 1$

   For B in tree at level ℓ

   M2M operation from B_c to B:

   $q_m^B = \sum_c \sum_{m'} P_m^B(y_{m'}^{B_c}) q_{m'}^{B_c}$.

   End

   End

3. *Downward pass*

   For $\ell = 2, \ldots, L$

   For A in tree at level ℓ

   L2L operation from A_p to A, and M2L conversion:

   $u_n^A = \sum_{n'} P_{n'}^{A_p}(x_n^A) u_{n'}^{A_p} + \sum_{B \in IL(A)} \sum_m G(x_n^A, y_m^B) q_m^B$

   End

   End

4. *Termination*

   For A leaf boxes

   Local-to-evaluation and diagonal interactions:

   $u_i = \sum_n P_n^A(x_i) u_n^A + \sum_{B \in NL(A)} \sum_{y_j \in B} G(x_i, y_j) q_j$.

   End

Let us now analyze the complexity of the algorithm.

Claim 1. *If we take $s = p$, the complexity of the 2D FMM is $O(pN)$.*

Proof. From our definition of s, we see that the number of leaf boxes is $O(N/s)$. The total number of boxes is at most twice the number of leaf boxes, regardless of the tree, hence it is also a $O(N/s)$. The complexity of one M2M, or one M2L, or one L2L operation is a small p-by-p mat-vec, hence p^2 operations.

For the initialization step, there exists an efficient way of binning that takes $O(N)$ operations[3]. The source-to-multipole step involves mapping every one of the N charge to p canonical charges, hence has complexity $O(pN)$.

In the upward pass, every one of the $O(N/s)$ source boxes is visited once, with an M2M that costs p^2 operations, for a total of $O(p^2N/s)$. In the downward pass, every one of the $O(N/s)$ evaluation boxes is visited once, with an M2L and an L2L that both cost p^2 operations, for a total of $O(p^2N/s)$ as well. Notice that the constant in this latter O is at least 27, the size of the interaction list.

For the termination, the local-to-evaluation step involves mapping p canonical potentials to every one of the N evaluation points, hence has complexity $O(pN)$. The diagonal term is a sum over $O(s)$ sources for each of the N evaluation points, hence has complexity $O(sN)$.

The overall operation count is $O(pN + p^2N/s + sN)$, and is minimized provided we take s on the order of p. This shows that the complexity is $O(pN)$ in 2D. □

Though we do not prove this here, the level of accuracy is inherited from the truncation error of the multipole expansion. We saw in the previous section that we can expect the error to decay geometrically, as $O((r/d)^p)$ where r and d are the separation parameters of any two boxes in the far field. Another way of linking accuracy to the overall complexity count is to let ϵ be a bound on the error, of the form $(r/d)^p$, and write the total complexity as $O(\log(1/\epsilon)N)$. This leads to a very favorable dependence of the complexity on the desired accuracy level: by doubling the number of operations, the number of correct digits essentially doubles as well.

[3]It is not trivial to show that the complexity of binning is $O(N)$ rather than $O(N \log N)$ regardless of the tree. We will not prove this result here.

3.4 Exercises

1. In this question, label by $\mathbf{x}_j$, $j = 1, 2, 3, 4$, the four corners of the square $[-1, 1]^2$, and by $\mathbf{y_k}, k = 1, 2, 3, 4$, the four corners of the square $[-5, 5]^2$. You may take the ordering to be the top-left, top-right, bottom-left, and bottom-right corners respectively, although any other choice is acceptable. Let $\phi(\mathbf{x}) = \frac{1}{|\mathbf{x}|}$.

   (a) Consider a unit charge at $\mathbf{x}_0$, generating a potential $u(\mathbf{x}) = \phi(\mathbf{x} - \mathbf{x}_0)$. We say that canonical charges q_j placed at $\mathbf{x}_j, j = 1, 2, 3, 4$ are the *projections* by collocation of the unit charge at $\mathbf{x}_0$ if the combination $\sum_{j=1}^{4} q_j \phi(\mathbf{x} - \mathbf{x}_j)$ matches $\phi(\mathbf{x} - \mathbf{x}_0)$ at the four points $\mathbf{x} = \mathbf{y}_k, k = 1, 2, 3, 4$.

      i. Find those four canonical charges $q_j, j = 1, 2, 3, 4$, in the case when $\mathbf{x}_0 = (0.1, 0.2)$.

   (b) (Interpolation via projection) We now reverse the roles of $\mathbf{x}_0$ and $\mathbf{y}_1$. Consider a unit charge at the point $\mathbf{y}_1$, generating a potential $v(\mathbf{x}) = \phi(\mathbf{x} - \mathbf{y}_1)$. Assume that you have at your disposal the evaluations of this potential at each of the four points $\mathbf{x} = \mathbf{x}_j, j = 1, 2, 3, 4$.

      i. How can you use the knowledge of these four evaluations, and of the four canonical charges q_j of question 2.1, to perform *interpolation* to find an approximation of the value of the potential $v(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_0$?

      ii. Find this numerical value in the case when $\mathbf{x}_0 = (0.1, 0.2)$, and compare it to the true value $v(\mathbf{x}_0)$. Justify your finding.

   (c) Consider again the function $v(\mathbf{x}) = \phi(\mathbf{x} - \mathbf{y}_1)$. We say that weights w_j placed at nodes $\mathbf{x}_j, j = 1, 2, 3, 4$, define an *interpolation* scheme for $v(\mathbf{x})$ if the combination $\sum_{j=1}^{4} w_j v(\mathbf{x}_j)$ is a good approximation of $v(\mathbf{x})$ for $\mathbf{x}$ in a neighborhood of the $\mathbf{x}_j$.

      i. Find those four weights w_j in the case of bilinear interpolation at $\mathbf{x} = \mathbf{x}_0 = (0.1, 0.2)$.

      ii. Find the numerical value of the bilinear interpolant of $v(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_0$, and compare it to the true value $v(\mathbf{x}_0)$.

(d) (Projection via interpolation) We now revert the roles of $\mathbf{x}_0$ and $\mathbf{y}_1$ to what they were in question 2.1. Consider a unit charge at $\mathbf{x}_0$ generating a potential $u(\mathbf{x}) = \phi(\mathbf{x} - \mathbf{x}_0)$.

i. How can you use the knowledge of the bilinear interpolation weights w_j of question 2.3, to find the *projections* of the unit charge at $\mathbf{x}_0$ as canonical charges $\tilde{q}_j$ at each of the four points $\mathbf{x} = \mathbf{x}_j$, $j = 1, 2, 3, 4$?

ii. Find the numerical values of these canonical charges when $\mathbf{x}_0 = (0.1, 0.2)$, and check their accuracy by comparing the true potential $u(\mathbf{x})$ to its approximation $\tilde{u}(\mathbf{x}) = \sum_{j=1}^{4} \tilde{q}_j \phi(\mathbf{x} - \mathbf{x}_j)$, when $\mathbf{x} = \mathbf{y}_1$. Argue the expected accuracy of this projection scheme.

2. Consider N source points $\mathbf{y}_j$ drawn uniformly at random in the unit square $[0, 1]^2$, and N target points $\mathbf{x}_i$ drawn uniformly at random in some square to be specified. Consider the kernel $G(\mathbf{x}, \mathbf{y}) = \log(\|\mathbf{x} - \mathbf{y}\|)$. Compute the ϵ-rank of the interactions[4], with $\epsilon = 10^{-5}$, and for the following scenarios:

(a) $N = 100$ and the target square is $[2, 3] \times [0, 1]$.

(b) $N = 1000$ and the target square is $[2, 3] \times [0, 1]$. How does your answer compare to that obtained in point (a), and why?

(c) $N = 100$ and the target square is $[1.25, 1.5] \times [0, 0.25]$. How does your answer compare to that obtained in point (a), and why?

(d) Again, $N = 100$ and the target square is $[2, 3] \times [0, 1]$, but use the explicit 2D multipole expansion formula seen earlier in this section. Find p such that the resulting error $\|G - UV^T\|_2 / \|G\|_2 \leq \epsilon$. How does your answer compare to that obtained in point (a), and why?

[4] I.e., the minimum rank r of a decomposition $G \simeq UV^T$, so that $\|G - UV^T\|_2 / \|G\|_2 \leq \epsilon$. It is obtained as the smallest number r such that $\sigma_{r+1}/\sigma_1 \leq \epsilon$, where σ_j are the singular values of G.

Chapter 4

Hierarchical matrices

This chapter details the *linear algebraic* point of view behind partitioned low-rank matrices, the fast multipole method, and the calculus of hierarchial matrices.

4.1 Low-rank decompositions

In this section we discuss the linear algebraic context of the projection and interpolation operations, and their generalizations.

Let us return to the case of $x \in A$ and $y \in B$, with A and B well-separated, so we are in effect considering an off-diagonal block of G (but still continue to write it G ouf of convenience). The variable x is viewed as a (possibly continuous) row index, while y is a (possibly continuous) column index. A low-rank expansion of G is any expression where x vs. y appear in separated factors, of the form

$$G(x,y) = \sum_{m,n} U_n(x) S_{nm} V_m(y), \qquad G = USV^T$$

It is often useful to limit the ranges of m and n to $1 \leq m, n \leq p$, which results in a numerical error ϵ:

$$\|G(x,y) - \sum_{m,n=1}^{P} U_n(x) S_{nm} V_m(y)\| \leq \epsilon. \tag{4.1}$$

The ϵ-rank of G is the smallest p for which (4.1) holds with the spectral (induced ℓ_2) norm. The optimal factors U, S, and V, for which the separation error is smallest, are given by the singular value decomposition.

There are, however, other ways to perform low-rank decompositions that have different redeeming features. Consider the single-charge projection relation (3.2), namely

$$G(x, y_j) \simeq \sum_m G(x, y_m^B) Q_m^B(y_j), \qquad x \in A,\ y_j \in B,$$

In the case when the y_m^B are taken as a subset of the original points y_j, the relation can be read as a low-rank decomposition

$$G \simeq G_c Q^T,$$

where G_c is a tall-and-thin column restriction of G, and Q is the matrix generated by the projection rule.

Conversely, when x_n^A form a subset of the x_i, the single-charge interpolation condition (3.5) can be read as the low-rank decomposition

$$G \simeq P G_r,$$

where G_r is now a short-and-wide row restriction of G, and P is the matrix generated by the interpolation rule. Recall that (i) G is symmetic, (ii) we may take $P = Q$, and (iii) the same subset of points can be used to generate the rows of G_r and the columns of G_c, hence the projection and interpolation decompositions are truly transposes of one another.

The low-rank decompositions generated from projection and interpolation are simply called *interpolative decompositions*. They interesting in their own right, for two reasons: 1) the factors G_c and G_r only require evaluations of the kernel: if an explicit formula or an on-demand evaluation function is available for that kernel, the storage requirement for P and/or Q is much lighter than that of the three factors U, S, and V^T; and 2) the matrix P is independent of the box B that defines the column-restriction of G, and similarly Q^T is independent of the box A that defines the row restriction of G, hence such interpolative decompositions can often be re-used across boxes.

We have already seen examples of Q that can be used in a projection decomposition $G \simeq G_c Q^T$. The linear algebra point of view invites to choose Q as the solution of an overdetermined least-squares problem of the form

$$Q^T = G_c^+ G,$$

but only after the G_c and G matrices are properly restricted in x, their *row* index. One could for instance let x run over the sequence x_i: this results in a projection rule that is often at best valid near the convex hull of the points x_i. In particular, the projection rule would need to change if the evaluation box A is changed. A much better choice is the collocation rule seen previously, which uses check points $\check{x}_i$ on a check surface surrounding B: the resulting projection rule is now independent of the evaluation box and will work as soon as the evaluation point is outside the check surface. But notice that we had to lift the restriction $x \in A$ in the collocation scheme, hence we had to look outside of the linear algebra framework to find the better solution!

As for the *column* restriction that takes G to G_c, many choices are possible. Linear algebra offers an interesting solution: the columns (at y_m^B) that take part in G_c can be picked from the columns (at y_j) of G by a rank-revealing QR decomposition[1]. Cheaper methods exist for selecting these columns, such as random sampling, or adaptive cross-approximation (ACA). We also saw in the previous section that we can pick the y_m^B as the nodes of a polynomial interpolation scheme, though this results in points that may be outside of the original collection y_j – a solution which again operates outside a strict linear algebra framework.

Another useful low-rank decomposition of G is the so-called CUR form, or skeleton form, written as

$$G \simeq G_c Z G_r,$$

where the middle matrix Z is chosen for the approximation to be good. The best choice in the least-squares (Frobenius) sense is $Z = G_c^+ G G_r^+$, which again requires collocation to properly row-restrict G and G_c; and column-restrict G and G_r. A convenient but suboptimal choice is $Z = G_{cr}^+$ where G_{cr} is the row and column restriction of G corresponding to the same index sets that define G_c and G_r. We can compare the different factorizations of G to conclude that we may define "skeleton" projection and interpolation rules via

$$Q^T = Z G_r, \qquad P = G_c Z.$$

[1]The letter Q is already taken, so write $G = UR$ for the thin QR decomposition. Because of pivoting, R is not necessarily lower triangular. Now extract from G the exact same columns that were orthogonalized in the process of forming U, and call the result G_c. Write $G_c = UR_c$ with R_c a square, invertible matrix. Then $G = G_c R_c^{-1} R = G_c Q$ with $Q = R_c^{-1} R$.

The most useful aspect of the low rank point of view is that it is *universal*, i.e., it informs the feasibility of projection and interpolation regardless of the particular method used.

1. If there exists a rank-r factorization of G into UV^T, then there must exist a factorization $G = G_c Q^T$ using r columns of G. In particular, a projection rule incurs no penalty[2] if the y_m^B are chosen among the y_j points (the interpolative case) rather than outside of this collection (the extrapolative case).

2. The availabilty of efficient projection and interpolation rules crucially hinges on the separability of the kernel G restricted to source and evaluation boxes. If the ϵ-rank of a block of G is greater than r, then there does not exist a projection rule with r canonical charges, or an interpolation rule with r canonical potentials, that could ever reach an error smaller than ϵ (in an ℓ_2 sense).

We already saw a justification of the separability of the $A \times B$ block of G in the multipole section. Let us now argue, from an algebraic argument in 1D, that we can expect low rank for off-diagonal blocks of inverse of banded matrices. Consider a finite difference discretization of a boundary-value ODE, leading to

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \gamma_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \gamma_{n-1} & \alpha_n \end{bmatrix}$$

Let $G = T^{-1}$, i.e., G is the so-called Green's matrix corresponding to a 3-point finite difference stencil.

Theorem 2. *Assume T is tridiagonal and invertible. Then $G = T^{-1}$ has rank-1 off-diagonal blocks.*

[2]There is truly no penalty if the rank is exactly r, and in exact arithmetic. There is a minor factor $\sqrt{rn}$ in loss of accuracy for truncation to r separated components by a QR decomposition, or in our notations $G_c Q^T$, over an SVD if the rank is not exactly r, however.

Proof. Fix j, and consider that

$$T = U + \beta_j e_j e_j^T,$$

where U is T with its β_j element put to zero, and e_j is the j-th canonical basis vector in $\mathbb{R}^n$. Notice that U is upper block-triangular with a block of zeros in positions $[1:j] \times [(j+1):n]$. Since the inverse of a block-triangular matrix is also block-triangular, U^{-1} also has a block of zeros in positions $[1:j] \times [(j+1):n]$. To obtain T^{-1}, apply the Woodbury formula, which shows that T^{-1} differs from U^{-1} by a rank-1 matrix:

$$T^{-1} = U^{-1} - \frac{(U^{-1}e_j)(e_{j+1}^T U^{-1})}{1 + \beta_j e_{j+1}^T U^{-1} e_j}.$$

(The scalar denominator is nonzero because T^{-1} is invertible.) Hence the block $[1:j] \times [(j+1):n]$ of T^{-1} has rank 1. An analogous argument shows that the block $[(j+1):n] \times [1:j]$ also has rank 1. □

More generally, if T has band width $2p+1$, i.e., p upper diagonals and p lower diagonals, then the rank of the off-diagonal blocks of T^{-1} is p.

A matrix whose off-diagonal blocks have rank p is called p-semiseparable. For each p, tshe set of invertible p-semiseparable matrices is closed under inversion (unlike the set of tridiagonal matrices, for instance).

[Extension to blocks]

4.2 Calculus of hierarchical matrices

4.3 Hierarchical matrices meet the FMM

Chapter 5

Butterfly algorithms

Butterfly algorithms are helpful for computing oscillatory sums arising in the high-frequency wave propagation context. We now consider a large wave number k, and kernels such as the fundamental solution of the Helmholtz equation in 3D,

$$G(x,y) = \frac{e^{ik|x-y|}}{4\pi|x-y|},$$

or its counterpart in 2D, $G(x,y) = \frac{i}{4}H_0^{(1)}(k|x-y|)$. The wave number is related to the angular frequency ω and the wave speed c via the dispersion relation $\omega = kc$.

5.1 Separation in the high-frequency regime

For high-frequency scattering and other oscillatory kernels, we have good separation if the boxes A and B

$$\operatorname{diam}(A) \times \operatorname{diam}(B) \lesssim d(A,B) \times \lambda,$$

where $d(A,B)$ is the distance between box centers, and $\lambda = 2\pi/k$ is the wavelength.

5.2 Architecture of the butterfly algorithm

The separation condition is much more restrictive in the high-frequency case, prompting the introduction of a different "butterfly" algorithm:

- The check potentials are split into different contributions coming from different sources boxes, and denoted u_n^{AB}; and
- The equivalent densities are split into different contributions generating potentials in different target boxes, and denoted q_m^{AB}.

The interpolation rules are now valid only for $x \in A$ and $y \in B$. The interpolation basis functions depend both on A and on B, and are denoted $P_n^{AB}(x)$.

- An interpolation rule in the x variable,

$$G(x,y) = \sum_n P_n^{AB}(x) G(x_n^A, y), \qquad x \in A, y \in B, \tag{5.1}$$

generates a notion of local expansion. Consider check potentials u_n^{AB} at the nodes x_n^A,

$$u_n^{AB} = \int_B G(x_n^A, y) q(y) dy. \tag{5.2}$$

The interpolation rule in x allows to switch from potentials at x_n^A to potentials everywhere in A: combine (3.14), (5.1), (5.2) to get

$$u^B(x) = \sum_n P_n^{AB}(x) u_n^{AB}, \qquad x \in A. \tag{5.3}$$

- An interpolation rule in the y variable,

$$G(x,y) = \sum_m G(x, y_m^B) P_m^{AB}(y), \qquad x \in A, y \in B, \tag{5.4}$$

generates a notion of multipole (interpolative) expansion. Consider equivalent densities q_m^{AB} at the nodes y_m^B, so that

$$u^B(x) = \sum_m G(x, y_m^B) q_m^{AB}, \qquad x \in A. \tag{5.5}$$

The interpolation rule in y allows to switch from densities in B to equivalent densities q_m^B: combine (3.14), (5.4), (5.5) to get

$$q_m^{AB} = \int_B P_m^{AB}(y) q(y) dy. \tag{5.6}$$

We say that a couple (A, B) is admissible when the boxes are well-separated in the sense explained in the previous section.

The condition on the admissibility of couples of boxes determines the form of an L2L operation:

$$u_n^{AB} \mathrel{+}= \sum_c \sum_{n'} P_{n'}^{A_p B_c}(x_n^A) u_{n'}^{A_p B_c}.$$

An M2M operation is

$$q_m^{AB} = \sum_c \sum_{m'} P_m^{AB}(y_{m'}^{B_c}) q_{m'}^{A_p B_c}.$$

An M2L conversion is

$$u_n^{AB^\uparrow} \mathrel{+}= \sum_m G(x_n^A, y_m^B) q_m^{A^\uparrow B},$$

where B is in the interaction list of A. There is no sum to perform over the interaction list. The notation $A^\uparrow$ refers to some ancestor of A; this precaution arises from the fact that we want A and B on the same level for the M2L translation, but the u_n^{AB} and q_m^{AB} are usually available only for boxes at different levels (B at a higher level than A for u and B at a lower level than A for q).

The choice of interpolation scheme may be dictated by the particular kernel. An all-purpose choice is to use translated copies of the kernel itself:

$$P_n^{AB}(x) = \sum_m G(x, y_m^B) d_{mn}^{AB}.$$

A substitution in (5.1) reveals that the d coefficients are obtained as the middle factor of a skeleton decomposition of the (A, B) block of G,

$$G(x, y) = \sum_{m,n} G(x, y_m^B) d_{mn}^{AB} G(x_n^A, y).$$

Appendix A

The Fast Fourier transform (FFT)

Appendix B

Linear and cyclic convolutions

In this section we expand on the link between the different kinds of convolution, and how they can all be computed using the FFT. The convolution of two vectors u and v is always a vector of the form $w_j = \sum_k u_{j-k} v_k$, but the bound on k, and the way $j - k$ is understood, differ from one kind of convolution to the next.

B.1 Cyclic convolution

The most algorithmically favorable situation is that of the cyclic (or periodic, or circular) convolution

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{N-1} \end{pmatrix} = \begin{pmatrix} u_0 & u_{N-1} & u_{N-2} & \cdots & u_1 \\ u_1 & u_0 & u_{N-1} & \cdots & u_2 \\ u_2 & u_1 & u_0 & \cdots & u_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{N-1} & u_{N-2} & u_{N-3} & \cdots & u_0 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix}$$

All three vectors u, v, and w have indices ranging from 0 to $N - 1$. The convolution is called cyclic, or circular, or periodic, because we use u_{N-1} in place of u_{-1} when the logical progression of terms calls for negative indices. The matrix of u is called circulant: its columns are cyclic shifts of one another.

Mathematically, the "wrap-around" can also be realized with the modulo operation, which computes the remainder of the integer division by N. For instance,

$$-1 \bmod N = N - 1,$$

$$0 \bmod N = 0$$
$$(N-1) \bmod N = N-1,$$
$$N \bmod N = 0,$$
$$(N+1) \bmod N = 1,$$

etc. The cyclic convolution is then written compactly as

$$w_j = \sum_{k=0}^{N-1} u_{(j-k) \bmod N} \, v_k.$$

As explained in the main text, the cyclic convolution can be computed fast by multiplication in the discrete Fourier domain.

B.2 Linear convolution

Linear convolutions are more often encountered in practice and do not involve wraparound:

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{N-1} \\ \vdots \\ w_{2N-4} \\ w_{2N-3} \\ w_{2N-2} \end{pmatrix} = \begin{pmatrix} u_0 & & & & \\ u_1 & u_0 & & & \\ u_2 & u_1 & u_0 & & \\ \vdots & \vdots & \ddots & \ddots & \\ u_{N-1} & u_{N-2} & \cdots & u_1 & u_0 \\ & \ddots & \ddots & \vdots & \vdots \\ & & u_{N-1} & u_{N-2} & u_{N-3} \\ & & & u_{N-1} & u_{N-2} \\ & & & & u_{N-1} \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix}$$

More concisely,

$$w_j = \sum_{k=0}^{N-1} u_{j-k} v_k,$$

with $0 \le j \le 2N-2$, and the convention that negative indices give rise to zero terms. Note that the output has size $2N-1$ rather than N.

The matrix involved has Toeplitz structure (constant along diagonals), and is still circulant (the columns are cyclic shifts of one another), but it is rectangular hence not diagonalizable by the Fourier transform. It can be turned into a square, circulant matrix by the following extension trick:

1) zeropad v with $N-1$ zeros, and 2) extend the matrix by considering the complete set of cyclic shifts of its first column, to obtain the equivalent expression

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{N-1} \\ \vdots \\ w_{2N-4} \\ w_{2N-3} \\ w_{2N-2} \end{pmatrix} = \left(\begin{array}{ccccc|ccccc} u_0 & & & & & u_{N-1} & u_{N-2} & \cdots & u_1 \\ u_1 & u_0 & & & & & u_{N-1} & \cdots & u_2 \\ u_2 & u_1 & u_0 & & & & & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & & & & & u_{N-1} \\ u_{N-1} & u_{N-2} & \cdots & u_1 & u_0 & & & & \\ & \ddots & \ddots & \vdots & \vdots & u_0 & & & \\ & & u_{N-1} & u_{N-2} & u_{N-3} & \vdots & u_0 & & \\ & & & u_{N-1} & u_{N-2} & u_{N-3} & \vdots & \ddots & \\ & & & & u_{N-1} & u_{N-2} & u_{N-3} & \cdots & u_0 \end{array}\right) \left(\begin{array}{c} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \\ \hline 0 \\ 0 \\ \vdots \\ 0 \end{array}\right)$$

This expression can now be computed fast with the FFT. The extended matrix implementing the cyclic convolution has size $2N-1$ by $2N-1$.

B.3 Partial linear convolution

Convolutions arising from integral equations often involve Toeplitz non-circulant matrices of the form

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{N-1} \end{pmatrix} = \begin{pmatrix} u_0 & u_{-1} & u_{-2} & \cdots & u_{-N+1} \\ u_1 & u_0 & u_{-1} & \cdots & u_{-N+2} \\ u_2 & u_1 & u_0 & \cdots & u_{-N+3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{N-1} & u_{N-2} & u_{N-3} & \cdots & u_0 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix}$$

The columns are no longer cyclic shifts of one another. We still recognize a restriction of a linear convolution; namely we can write

$$\begin{pmatrix} \times \\ \times \\ \vdots \\ \times \\ \hline w_0 \\ w_1 \\ \vdots \\ w_{N-2} \\ w_{N-1} \\ \hline \times \\ \vdots \\ \times \\ \times \end{pmatrix} = \left(\begin{array}{ccccc} u_{-N+1} & & & & \\ u_{-N+2} & u_{-N+1} & & & \\ \vdots & \vdots & \ddots & & \\ u_{-1} & u_{-2} & \cdots & u_{-N+1} & \\ \hline u_0 & u_{-1} & \cdots & u_{-N+2} & u_{-N+1} \\ u_1 & u_0 & \cdots & u_{-N+3} & u_{-N+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{N-2} & u_{N-3} & \cdots & u_0 & u_{-1} \\ u_{N-1} & u_{N-2} & \cdots & u_1 & u_0 \\ \hline & u_{N-1} & \cdots & u_2 & u_1 \\ & & \ddots & \vdots & \vdots \\ & & & u_{N-1} & u_{N-2} \\ & & & & u_{N-1} \end{array}\right) \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix}$$

The crosses indicate "spurious" components whose value is often not of interest. The extended matrix implementing the linear convolution has size $3N-2$ by N. The same trick as earlier can be used to embed this linear convolution into a cyclic convolution of size $3N-2$, resulting in a reasonably fast algorithm.

However, because the specific value of $\times$ are unimportant, we can wrap the rows around to eliminate the zeros, and write the shorter linear convolution

$$\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-2} \\ w_{N-1} \\ \hline \times \\ \times \\ \vdots \\ \times \end{pmatrix} = \begin{pmatrix} u_0 & u_{-1} & \cdots & u_{-N+2} & u_{-N+1} \\ u_1 & u_0 & \cdots & u_{-N+3} & u_{-N+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{N-2} & u_{N-3} & \cdots & u_0 & u_{-1} \\ u_{N-1} & u_{N-2} & \cdots & u_1 & u_0 \\ \hline u_{-N+1} & u_{N-1} & \cdots & u_2 & u_1 \\ u_{-N+2} & u_{-N+1} & \cdots & u_3 & u_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ u_{-1} & u_{-2} & \cdots & u_{-N+1} & u_{N-1} \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix}$$

where the elements labeled $\times$ are now different. The matrix is now $2N-1$ by N. The convolution can be made cyclic by considering the complete set of cyclic shifts of the first column, resuting in an extended, $2N-1$ by $2N-1$ circulant matrix:

$$\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-2} \\ w_{N-1} \\ \hline \times \\ \times \\ \vdots \\ \times \end{pmatrix} = \left(\begin{array}{ccccc|ccc} u_0 & u_{-1} & \cdots & u_{-N+2} & u_{-N+1} & u_{N-1} & \cdots & u_1 \\ u_1 & u_0 & \cdots & u_{-N+3} & u_{-N+2} & u_{-N+1} & \cdots & u_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{N-2} & u_{N-3} & \cdots & u_0 & u_{-1} & u_{-2} & \cdots & u_{N-1} \\ u_{N-1} & u_{N-2} & \cdots & u_1 & u_0 & u_{-1} & \cdots & u_{-N+1} \\ \hline u_{-N+1} & u_{N-1} & \cdots & u_2 & u_1 & u_0 & \cdots & u_{-N+2} \\ u_{-N+2} & u_{-N+1} & \cdots & u_3 & u_2 & u_1 & \cdots & u_{-N+3} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{-1} & u_{-2} & \cdots & u_{-N+1} & u_{N-1} & u_{N-2} & \cdots & u_0 \end{array}\right) \left(\begin{array}{c} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \\ \hline 0 \\ \vdots \\ 0 \end{array}\right)$$

The prescription for a fast algorithm is therefore to consider the first column,

$$(u_0, u_1, \ldots, u_{N-1}, u_{-N+1}, \ldots, u_{-1})^T,$$

and perform its cyclic convolution with the zeropadded vector v. This variant is more efficient by roughly 50% than the $3N-2$ by $3N-2$ formulation described earlier.

B.4 Partial, symmetric linear convolution

There is a bit more to be gained in the case when the convolution reads

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{N-1} \end{pmatrix} = \begin{pmatrix} u_0 & u_1 & u_2 & \cdots & u_{N-1} \\ u_1 & u_0 & u_1 & \cdots & u_{N-2} \\ u_2 & u_1 & u_0 & \cdots & u_{N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{N-1} & u_{N-2} & u_{N-3} & \cdots & u_0 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix},$$

namely when $u_{-j} = u_j$. The matrix is now symmetric, Toeplitz, but not circulant. The algorithm in the previous section considered the mirror extension of the first column

$$(u_0, u_1, \ldots, u_{N-2}, u_{N-1})^T$$

into the $(2N-1)$-vector

$$(u_0, u_1, \ldots, u_{N-2}, u_{N-1}, u_{N-1}, u_{N-2}, \ldots, u_1)^T.$$

However, we recognize that u_{N-1} is needlessly duplicated. This leads to considering a mirror extension of length $2N-2$:

$$(u_0, u_1, \ldots, u_{N-2}, u_{N-1}, u_{N-2}, \ldots, u_1)^T.$$

This results in a $2N-2$ by $2N-2$ circulant matrix

$$\left(\begin{array}{c} w_0 \\ w_1 \\ \vdots \\ w_{N-2} \\ w_{N-1} \\ \hline \times \\ \times \\ \vdots \\ \times \end{array}\right) = \left(\begin{array}{ccccc|ccc} u_0 & u_1 & \cdots & u_{N-2} & u_{N-1} & u_{N-2} & \cdots & u_1 \\ u_1 & u_0 & \cdots & u_{N-3} & u_{N-2} & u_{N-1} & \cdots & u_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{N-2} & u_{N-3} & \cdots & u_0 & u_1 & u_2 & \cdots & u_{N-1} \\ u_{N-1} & u_{N-2} & \cdots & u_1 & u_0 & u_1 & \cdots & u_{N-2} \\ \hline u_{N-2} & u_{N-1} & \cdots & u_2 & u_1 & u_0 & \cdots & u_{N-3} \\ u_{N-3} & u_{N-2} & \cdots & u_3 & u_2 & u_1 & \cdots & u_{N-4} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ u_1 & u_2 & \cdots & u_{N-1} & u_{N-2} & u_{N-3} & \cdots & u_0 \end{array}\right) \left(\begin{array}{c} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \\ \hline 0 \\ \vdots \\ 0 \end{array}\right).$$

Whether the FFT-based algorithm for matrix-vector multiplication is now faster than the variant seen in the previous section depends on whether $2N-2$ is "more composite" than $2N-1$ (which it stands a better chance of being, by virtue of being even). With additional zeropadding, it is simple to imagine variants with $2N$ by $2N$ or larger matrices.