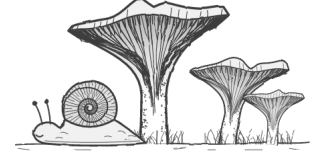


The Boundary Element Method

Mikkel Paltorp (mpasc@dtu.dk), 12 May 2026

Department of Applied Mathematics and Computer Science, Technical University of Denmark



In the previous post¹ the Helmholtz equation was solved using the Finite Element Method (FEM), where the whole domain Ω is divided into elements. The Boundary Element Method (BEM) instead first rewrites the Helmholtz equation as an integral equation that lives only on the boundary $\Gamma = \partial\Omega$ [1]. As a result, only the boundary has to be divided into elements, which is particularly attractive for exterior problems, such as the sound radiated from a vibrating object into free space, where the domain Ω in theory is infinite.

The building blocks are the same as in the previous post: the unknown functions are approximated as $p(\mathbf{x}) \approx \mathbf{T}(\mathbf{x})\mathbf{p}$, the elements are parameterized by local coordinates $\mathbf{u} \in \mathcal{L}^e$, the element localization matrices $\mathbf{L}^e$ pick out the local values $\mathbf{p}^e = \mathbf{L}^e\mathbf{p}$, and integrals over the elements are approximated by a quadrature rule. The only difference in these building blocks is that the elements are now surface elements Γ^e instead of volume elements Ω^e . What changes is the equation being discretized and the approach used to discretize it.

1. The Kirchhoff–Helmholtz integral equation

For time-harmonic problems the pressure can be written as $p(\mathbf{x}, t) = \text{Re}\{p(\mathbf{x})e^{\alpha i\omega t}\}$, where $\alpha = \pm 1$ is the sign of the chosen time dependency. Inserting this into the wave equation results in the Helmholtz equation

$$\Delta p(\mathbf{x}) + k^2 p(\mathbf{x}) = 0, \quad \mathbf{x} \in \Omega, \quad (1)$$

which looks independent of α . This can cause confusion when the time dependency is not stated explicitly, as the relation between the normal derivative of the pressure and the normal velocity v_n does depend on α

$$\frac{\partial p}{\partial \mathbf{n}}(\mathbf{x}) = s(\alpha)k v_n(\mathbf{x}), \quad s(\alpha) = -\alpha i \rho_0 c_0, \quad \mathbf{x} \in \Gamma, \quad (2)$$

where the normal $\mathbf{n}$ points into the domain Ω , i.e. out of a vibrating object, and $s(\alpha)$ is introduced for simplicity, similarly to [2]. Note that this is opposite to the previous post, where the boundary term of the integration by parts used the outward normal of Ω .

In three dimensions the Green's function, which will come in handy soon enough, is given by

$$G(\mathbf{x}, \mathbf{y}) = \frac{\exp(-\alpha i k \|\mathbf{x} - \mathbf{y}\|_2)}{4\pi \|\mathbf{x} - \mathbf{y}\|_2}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^3. \quad (3)$$

It is the solution of the Helmholtz equation with a point source

$$\Delta G(\mathbf{x}, \mathbf{y}) + k^2 G(\mathbf{x}, \mathbf{y}) = -\delta(\mathbf{x} - \mathbf{y}), \quad (4)$$

where δ is the Dirac delta function, while the sign α makes the waves of G travel outwards from the source. The Green's function takes two inputs, $\mathbf{x}$ and $\mathbf{y}$,

Contents

1. The Kirchhoff–Helmholtz integral equation
2. Collocation
3. Computing the matrices
4. Acceleration methods
 - 4.1. The fast multipole method
 - 4.2. Hierarchical matrices
 - 4.3. The interpolated factored Green function
- Bibliography

¹ Earlier post: *The Finite Element Method*.
mipals.github.io/blog/2026/05-11-fem-introduction/

commonly referred to as the target location ($\mathbf{x}$) and the source location ($\mathbf{y}$), or simply the target(s) and source(s).

The derivation of the integral equation starts in the same way as the FEM: we multiply the Helmholtz equation with a test function q and integrate over the domain

$$\int_{\Omega} q(\mathbf{y})(\Delta p(\mathbf{y}) + k^2 p(\mathbf{y})) d\mathbf{y} = 0. \quad (5)$$

The difference is that we now integrate by parts *twice*, moving both derivatives from p onto q . Writing the normal derivative of p in terms of the normal velocity, it follows that

$$\begin{aligned} \int_{\Omega} p(\mathbf{y})(\Delta q(\mathbf{y}) + k^2 q(\mathbf{y})) d\mathbf{y} + \int_{\Gamma} \frac{\partial q}{\partial \mathbf{n}}(\mathbf{y}) p(\mathbf{y}) dS_{\mathbf{y}} \\ - s(\alpha) k \int_{\Gamma} q(\mathbf{y}) v_{\mathbf{n}}(\mathbf{y}) dS_{\mathbf{y}} = 0, \end{aligned}$$

where the signs of the boundary terms are due to $\mathbf{n}$ pointing into Ω . For exterior problems the boundary also includes a sphere at infinity, whose integral vanishes when p satisfies the Sommerfeld radiation condition below. Choosing the test function as the Green's function, $q(\mathbf{y}) = G(\mathbf{x}, \mathbf{y})$, the volume integral collapses due to Equation 4, and we are left with the Kirchhoff-Helmholtz integral equation

$$\zeta(\mathbf{x}) p(\mathbf{x}) - \int_{\Gamma} \frac{\partial G(\mathbf{x}, \mathbf{y})}{\partial \mathbf{n}(\mathbf{y})} p(\mathbf{y}) dS_{\mathbf{y}} + s(\alpha) k \int_{\Gamma} G(\mathbf{x}, \mathbf{y}) v_{\mathbf{n}}(\mathbf{y}) dS_{\mathbf{y}} = 0, \quad (6)$$

where $\zeta(\mathbf{x})$ is the so-called integral free term depending on the geometry at $\mathbf{x}$. For $\mathbf{x}$ in the interior of Ω it is equal to one, in which case Equation 6 gives the pressure at $\mathbf{x}$ from the values of p and $v_{\mathbf{n}}$ on the boundary. For $\mathbf{x}$ on a smooth part of the boundary it is equal to $1/2$, while at corners and edges it takes values between 0 and 1. Setting $\mathbf{x} \in \Gamma$ makes Equation 6 an equation involving only functions on the boundary: a boundary integral equation.

The time dependency still enters Equation 6 through G and $s(\alpha)$. Changing α conjugates the Green's function and flips the sign of $s(\alpha)$, so a sign error in either of them results in a solution of the wrong problem. Finally, the derivation assumes that p satisfies the Sommerfeld radiation condition

$$\lim_{r \rightarrow \infty} r^{(d-1)/2} \left(\frac{\partial p}{\partial r} + \alpha i k p \right) = 0, \quad (7)$$

where d is the dimension, meaning that no waves come in from infinity. In the FEM this condition has to be imposed by truncating the domain with an artificial boundary, whereas in the BEM it is built into the Green's function, so that the solution of Equation 6 satisfies it automatically.

2. Collocation

As in the FEM, we multiply Equation 6 with a test function φ , this time integrating over the boundary,

$$\int_{\Gamma} \varphi(\mathbf{x}) \left(\zeta(\mathbf{x}) p(\mathbf{x}) - \int_{\Gamma} \frac{\partial G(\mathbf{x}, \mathbf{y})}{\partial \mathbf{n}(\mathbf{y})} p(\mathbf{y}) dS_{\mathbf{y}} + s(\alpha) k \int_{\Gamma} G(\mathbf{x}, \mathbf{y}) v_{\mathbf{n}}(\mathbf{y}) dS_{\mathbf{y}} \right) dS_{\mathbf{x}} = 0. \quad (8)$$

Choosing $\varphi(\mathbf{x}) \approx \mathbf{a}^{\top} \mathbf{T}(\mathbf{x})^{\top}$ would result in a Galerkin BEM. The integrals of the Galerkin BEM are, however, double surface integrals with a singular integrand, which makes them expensive to compute. Instead, the most common approach is the so-called collocation approach, where the test function is a sum of Dirac delta functions placed at the collocation points $\mathbf{t}_1, \dots, \mathbf{t}_n \in \Gamma$

$$\varphi(\mathbf{x}) = \mathbf{a}^{\top} \begin{bmatrix} \delta(\mathbf{x} - \mathbf{t}_1) \\ \delta(\mathbf{x} - \mathbf{t}_2) \\ \vdots \\ \delta(\mathbf{x} - \mathbf{t}_n) \end{bmatrix}, \quad (9)$$

where $\mathbf{a} \in \mathbb{C}^n$ is again a vector of arbitrary coefficients. The collocation points are in most cases chosen as the interpolation nodes of the basis functions, so that the cardinal property $\mathbf{T}(\mathbf{t}_j) \mathbf{p} = p_j$ can be used. Inserting φ into Equation 8, the sifting property of the Dirac delta function evaluates the outer integral at the collocation points, so that Equation 8 becomes $\mathbf{a}^{\top} \mathbf{r} = 0$ with

$$\mathbf{r} = \begin{bmatrix} \zeta(\mathbf{t}_1) p(\mathbf{t}_1) \\ \vdots \\ \zeta(\mathbf{t}_n) p(\mathbf{t}_n) \end{bmatrix} - \begin{bmatrix} \int_{\Gamma} \frac{\partial G(\mathbf{t}_1, \mathbf{y})}{\partial \mathbf{n}(\mathbf{y})} p(\mathbf{y}) dS_{\mathbf{y}} \\ \vdots \\ \int_{\Gamma} \frac{\partial G(\mathbf{t}_n, \mathbf{y})}{\partial \mathbf{n}(\mathbf{y})} p(\mathbf{y}) dS_{\mathbf{y}} \end{bmatrix} + s(\alpha) k \begin{bmatrix} \int_{\Gamma} G(\mathbf{t}_1, \mathbf{y}) v_{\mathbf{n}}(\mathbf{y}) dS_{\mathbf{y}} \\ \vdots \\ \int_{\Gamma} G(\mathbf{t}_n, \mathbf{y}) v_{\mathbf{n}}(\mathbf{y}) dS_{\mathbf{y}} \end{bmatrix}.$$

In its current form the unknowns are still *functions*, which a computer cannot be tasked to find. Instead, we insert the approximations $p(\mathbf{y}) \approx \mathbf{T}(\mathbf{y}) \mathbf{p}$ and $v_{\mathbf{n}}(\mathbf{y}) \approx \mathbf{T}(\mathbf{y}) \mathbf{v}_{\mathbf{n}}$ and handle the three vectors one at a time. For the first vector, the cardinal property of the basis functions gives

$$\begin{bmatrix} \zeta(\mathbf{t}_1) p(\mathbf{t}_1) \\ \vdots \\ \zeta(\mathbf{t}_n) p(\mathbf{t}_n) \end{bmatrix} \approx \begin{bmatrix} \zeta(\mathbf{t}_1) \mathbf{T}(\mathbf{t}_1) \mathbf{p} \\ \vdots \\ \zeta(\mathbf{t}_n) \mathbf{T}(\mathbf{t}_n) \mathbf{p} \end{bmatrix} = \begin{bmatrix} \zeta(\mathbf{t}_1) p_1 \\ \vdots \\ \zeta(\mathbf{t}_n) p_n \end{bmatrix} = \text{diag}(\zeta) \mathbf{p}, \quad (10)$$

where $\zeta = [\zeta(\mathbf{t}_1) \dots \zeta(\mathbf{t}_n)]^{\top}$. For the two integrals, the coefficients $\mathbf{p}$ and $\mathbf{v}_{\mathbf{n}}$ do not depend on $\mathbf{y}$ and can be moved outside of the integrals

$$\begin{bmatrix} \int_{\Gamma} \frac{\partial G(\mathbf{t}_1, \mathbf{y})}{\partial \mathbf{n}(\mathbf{y})} p(\mathbf{y}) dS_{\mathbf{y}} \\ \vdots \\ \int_{\Gamma} \frac{\partial G(\mathbf{t}_n, \mathbf{y})}{\partial \mathbf{n}(\mathbf{y})} p(\mathbf{y}) dS_{\mathbf{y}} \end{bmatrix} \approx \underbrace{\begin{bmatrix} \int_{\Gamma} \frac{\partial G(\mathbf{t}_1, \mathbf{y})}{\partial \mathbf{n}(\mathbf{y})} \mathbf{T}(\mathbf{y}) dS_{\mathbf{y}} \\ \vdots \\ \int_{\Gamma} \frac{\partial G(\mathbf{t}_n, \mathbf{y})}{\partial \mathbf{n}(\mathbf{y})} \mathbf{T}(\mathbf{y}) dS_{\mathbf{y}} \end{bmatrix}}_{\mathbf{F}} \mathbf{p}, \quad (11)$$

$$\begin{bmatrix} \int_{\Gamma} G(\mathbf{t}_1, \mathbf{y}) v_{\mathbf{n}}(\mathbf{y}) dS_{\mathbf{y}} \\ \vdots \\ \int_{\Gamma} G(\mathbf{t}_n, \mathbf{y}) v_{\mathbf{n}}(\mathbf{y}) dS_{\mathbf{y}} \end{bmatrix} \approx \underbrace{\begin{bmatrix} \int_{\Gamma} G(\mathbf{t}_1, \mathbf{y}) \mathbf{T}(\mathbf{y}) dS_{\mathbf{y}} \\ \vdots \\ \int_{\Gamma} G(\mathbf{t}_n, \mathbf{y}) \mathbf{T}(\mathbf{y}) dS_{\mathbf{y}} \end{bmatrix}}_{\mathbf{G}} \mathbf{v}_{\mathbf{n}}, \quad (12)$$

where each entry of the vectors is a row vector, as $\mathbf{T}(\mathbf{y})$ is, so that $\mathbf{F}$ and $\mathbf{G}$ are $n \times n$ matrices. Collecting the terms, $\mathbf{a}^\top \mathbf{r} = 0$ becomes

$$\mathbf{a}^\top ((\text{diag}(\boldsymbol{\zeta}) - \mathbf{F})\mathbf{p} + s(\alpha)k\mathbf{G}\mathbf{v}_n) \approx 0. \quad (13)$$

As this has to hold for all $\mathbf{a} \in \mathbb{C}^n$, the term inside the parentheses must vanish, and the collocation BEM therefore reads

$$(\text{diag}(\boldsymbol{\zeta}) - \mathbf{F})\mathbf{p} + s(\alpha)k\mathbf{G}\mathbf{v}_n = \mathbf{H}\mathbf{p} + s(\alpha)k\mathbf{G}\mathbf{v}_n \approx \mathbf{0}, \quad (14)$$

where we simply defined $\mathbf{H} = \text{diag}(\boldsymbol{\zeta}) - \mathbf{F}$ for the sake of convenience. In a typical problem $\mathbf{v}_n$ is known (e.g. from the vibration of a structure) and Equation 14 is solved for $\mathbf{p}$.

3. Computing the matrices

While Equation 14 is easily stated, it does not mention how to actually compute $\mathbf{F}$ and $\mathbf{G}$. The procedure is the same as for the mass and stiffness matrices in the previous post²: the boundary integral is split into a sum over the elements, each element integral is moved to local coordinates, and the result is approximated by a quadrature rule. On element e the pressure is interpolated as $p(\mathbf{y}^e(\mathbf{u})) \approx \mathbf{T}^e(\mathbf{u})\mathbf{L}^e\mathbf{p}$, so that the k th row of $\mathbf{F}$ becomes

$$\begin{aligned} \int_{\Gamma} \frac{\partial G(\mathbf{t}_k, \mathbf{y})}{\partial \mathbf{n}(\mathbf{y})} \mathbf{T}(\mathbf{y}) dS_y &\approx \sum_{e=1}^N \left(\int_{\mathcal{L}^e} \frac{\partial G(\mathbf{t}_k, \mathbf{y}^e(\mathbf{u}))}{\partial \mathbf{n}(\mathbf{y}^e(\mathbf{u}))} \mathbf{T}^e(\mathbf{u}) \text{jacobian}(\Gamma^e, \mathbf{u}) d\mathbf{u} \right) \mathbf{L}^e \\ &\approx \sum_{e=1}^N \left(\sum_{i=1}^{Q(\mathbf{t}_k, e)} \frac{\partial G(\mathbf{t}_k, \mathbf{y}^e(\mathbf{u}_i))}{\partial \mathbf{n}(\mathbf{y}^e(\mathbf{u}_i))} \mathbf{T}^e(\mathbf{u}_i) \text{jacobian}(\Gamma^e, \mathbf{u}_i) w_i \right) \mathbf{L}^e, \end{aligned} \quad (15)$$

where the first approximation is due to the geometry being approximated by the elements, and the second is due to the quadrature. The k th row of $\mathbf{G}$ is computed in the same way, with $G(\mathbf{t}_k, \mathbf{y})$ in place of its normal derivative. The jacobian describes the local deformation of the map from local to global coordinates. For the curve elements of 2D problems this is a change of length, while for the surface elements of 3D problems it is a change of area

$$\begin{aligned} \text{2D: } \text{jacobian}(\Gamma^e, \mathbf{u}) &= \left\| \mathbf{X}^e \frac{d\mathbf{N}^e(\mathbf{u})}{du} \right\|_2 && (\text{length deformation}), \\ \text{3D: } \text{jacobian}(\Gamma^e, \mathbf{u}) &= \left\| \left(\mathbf{X}^e \frac{\partial \mathbf{N}^e(\mathbf{u})}{\partial u_1} \right) \times \left(\mathbf{X}^e \frac{\partial \mathbf{N}^e(\mathbf{u})}{\partial u_2} \right) \right\|_2 && (\text{area deformation}). \end{aligned} \quad (16)$$

The new difficulty compared to the FEM is that the Green's function is singular at $\mathbf{y} = \mathbf{t}_k$, so a standard quadrature rule is inaccurate on the elements that contain, or lie close to, the collocation point. This is why Equation 15 highlights that the number of quadrature points $Q(\mathbf{t}_k, e)$ can depend on both the collocation point and the element. For the elements containing $\mathbf{t}_k$, special techniques are needed, such as splitting the element at $\mathbf{t}_k$ and using a transformation whose jacobian cancels the $1/r$ singularity.

² mipals.github.io/blog/2026/05-11-fem-introduction/

Example 1 (Pulsating sphere) A sphere of radius a whose surface moves uniformly with the normal velocity v_0 radiates the pressure $p(r) = A \exp(-\alpha i k r)/r$. Matching the radial velocity at $r = a$ gives the analytical surface pressure

$$p(a) = \rho_0 c_0 v_0 \frac{\alpha i k a}{1 + \alpha i k a}. \quad (17)$$

To compute it with the BEM we use `BoundaryIntegralEquations.jl`, which assembles $\mathbf{F}$ and $\mathbf{G}$ as in Equation 15, including the special quadrature for the elements close to the collocation points. The sphere is meshed with continuous quadratic triangles, both for the geometry ($\mathbf{N}^e$) and for the pressure and normal velocity ($\mathbf{T}^e$), and the collocation points are the interpolation nodes. The package uses the time dependency $\exp(-i\omega t)$, i.e. $\alpha = -1$, and the normals of the mesh point into the sphere. The latter flips the sign of the matrix returned as $\mathbf{F}$, and the package returns $1 - \zeta$ instead of ζ .

julia

```
1 using BoundaryIntegralEquations, LinearAlgebra
2 ## Physical parameters. The sphere pulsates with the normal velocity v_0.
3 a = 1.0           # Radius of the sphere
4 ρ = 1.21          # Density of air
5 c = 343.0         # Speed of sound
6 v_0 = 1.0         # Normal velocity of the surface
7 s = im*ρ*c        # s(α) for the time dependency exp(-iωt), i.e. α = -1
8 ## Mesh: 464 quadratic triangles with 930 nodes
9 pkg_dir = pkgdir(BoundaryIntegralEquations)
10 mesh_file = joinpath(pkg_dir, "examples", "meshes", "sphere_1m_coarse")
11 mesh = load3dTriangularComsolMesh(mesh_file)
12 v_n = fill(v_0, size(mesh.sources, 2))
13 ## Solving (diag(ζ) - F)p = -s k G v_n for each wavenumber.
14 # The normals of the mesh point into the sphere, so the F of the package
15 # is -F, and the integral free term of the exterior problem is ζ = 1 - C.
16 ka = range(0.05, 5, length=100)
17 p_bem = map(ka) do kai
18     k = kai/a
19     F, G, C = assemble_parallel!(mesh, k, mesh.sources; progress=false)
20     H = Diagonal(1 .- C) + F
21     p = H \ (-s*k*G*v_n)
22     sum(p)/length(p) # The surface pressure is constant; average it.
23 end
24 ## The analytical surface pressure of the pulsating sphere.
25 p_analytical = @. ρ*c*v_0*(-im*ka)/(1 - im*ka)
```

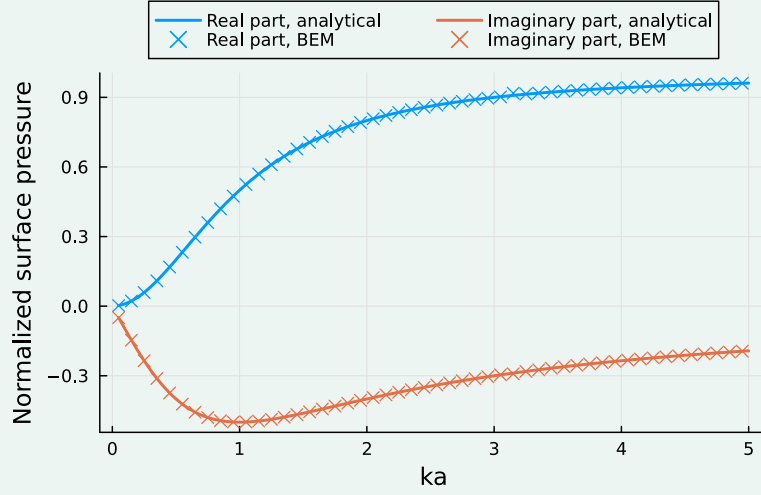


Figure 1: The surface pressure of the pulsating sphere normalized by $\rho_0 c_0 v_0$, analytical and computed with the BEM using 464 quadratic elements.

4. Acceleration methods

A considerable downside of the BEM is that $\mathbf{H}, \mathbf{G} \in \mathbb{C}^{n \times n}$ are dense, unlike the sparse mass and stiffness matrices of the FEM. The memory used to store them therefore scales as $\mathcal{O}(n^2)$, and a direct solve of Equation 14 is $\mathcal{O}(n^3)$, rendering the direct application of the method unusable for large n . This is somewhat offset by the fact that only the boundary is discretized, but for large problems it is a serious limitation. However, various methods have been developed that reduce the memory and the cost of a matrix-vector product to $\mathcal{O}(n \log n)$ or even $\mathcal{O}(n)$ by exploiting the rank structure of the problem. As these methods only define the matrix-vector products, Equation 14 must then be solved by an iterative method, the most common being the generalized minimal residual method (GMRES) [3], [4], [5].

The aim of this section is to show how the matrix-vector products with $\mathbf{G}$ and $\mathbf{F}$ can be computed in $\mathcal{O}(n)$ (or $\mathcal{O}(n \log n)$) time using the fast multipole method (FMM) or hierarchical matrices ($\mathcal{H}$ -matrices). To do so, we start by assuming that all elements are integrated using the same Q quadrature points.³ Multiplying the k th row of $\mathbf{G}$ with a vector $\mathbf{z}$ it follows that

$$\begin{aligned} \left(\int_{\Gamma} G(\mathbf{t}_k, \mathbf{y}) \mathbf{T}(\mathbf{y}) dS_{\mathbf{y}} \right) \mathbf{z} &\approx \left(\sum_{e=1}^N \sum_{i=1}^Q G(\mathbf{t}_k, \mathbf{y}^e(\mathbf{u}_i)) \text{jacobian}(\Gamma^e, \mathbf{u}_i) w_i \mathbf{T}^e(\mathbf{u}_i) \mathbf{L}^e \right) \mathbf{z} \\ &= \left(\sum_{j=1}^{NQ} G(\mathbf{t}_k, \mathbf{y}_j) \underbrace{\text{jacobian}(\Gamma^{e(j)}, \mathbf{u}_j) w_j \mathbf{T}^{e(j)}(\mathbf{u}_j) \mathbf{L}^{e(j)}}_{j\text{th row of } \mathbf{C}} \right) \mathbf{z} \quad (18) \\ &= [G(\mathbf{t}_k, \mathbf{y}_1) \ G(\mathbf{t}_k, \mathbf{y}_2) \ \dots \ G(\mathbf{t}_k, \mathbf{y}_{NQ})] \mathbf{C} \mathbf{z}, \end{aligned}$$

where the subscript j refers to an ordering of the quadrature points of all elements, and $e(j)$ returns the element that quadrature point j lies on. The matrix $\mathbf{C} \in \mathbb{R}^{NQ \times n}$ is *extremely* sparse and can be thought of as a map from the values $\mathbf{z}$ at the collocation points to the coefficients $\mathbf{c} = \mathbf{C} \mathbf{z}$ at the quadrature points

³ This results in inaccurate integrals for the elements close to the collocation point, which are fixed by a near-field correction below.

$\mathbf{y}_j$. The same approach applies to $\mathbf{F}$, where the normal derivative is written as $\mathbf{n}(\mathbf{y}_j)^\top \nabla_{\mathbf{y}} G(\mathbf{t}_k, \mathbf{y}_j)$

$$\begin{aligned} & \left(\int_{\Gamma} \frac{\partial G(\mathbf{t}_k, \mathbf{y})}{\partial \mathbf{n}(\mathbf{y})} \mathbf{T}(\mathbf{y}) dS_{\mathbf{y}} \right) \mathbf{z} \\ & \approx \left[\mathbf{n}(\mathbf{y}_1)^\top \nabla_{\mathbf{y}} G(\mathbf{t}_k, \mathbf{y}_1) \dots \mathbf{n}(\mathbf{y}_{NQ})^\top \nabla_{\mathbf{y}} G(\mathbf{t}_k, \mathbf{y}_{NQ}) \right] \mathbf{C} \mathbf{z}. \end{aligned} \quad (19)$$

Applying Equation 18 or Equation 19 directly for all n rows results in products that scale as $\mathcal{O}(NQn)$, which could hurt more than we gain, as $NQ \gtrsim n$. The point is that the dense part of the product, a sum of Green's functions over the points $\mathbf{y}_j$, is exactly what the FMM and $\mathcal{H}$ -matrices accelerate. Doing so blindly, however, keeps the inaccurate integration of the elements close to the collocation points. To fix this, a sparse near-field correction $\mathbf{S}$ is added, which subtracts the inaccurate integration using only Q quadrature points and adds the accurate one. Applying Equation 18 or Equation 19 for all rows at the same time can therefore be thought of as

$$\mathbf{A} = \mathbf{B}\mathbf{C} + \mathbf{S}, \quad \mathbf{A}, \mathbf{S} \in \mathbb{C}^{n \times n}, \quad \mathbf{B} \in \mathbb{C}^{n \times NQ}, \quad \mathbf{C} \in \mathbb{R}^{NQ \times n}, \quad (20)$$

where $\mathbf{B}$ is the dense part approximated by either the FMM or an $\mathcal{H}$ -matrix (see Equation 22 and Equation 23 below), $\mathbf{C}$ is the coefficient map, and $\mathbf{S}$ is the near-field correction. As $\mathbf{S}$ and $\mathbf{C}$ are highly sparse, both their assembly and their products scale as $\mathcal{O}(n\tau)$ with $\tau \ll n$. Using an approximation of $\mathbf{B}$ whose product scales linearly in time and memory therefore results in a representation of $\mathbf{A}$ that scales similarly.

To compute $\mathbf{S}$ efficiently, its sparsity pattern must be known in advance. For collocation this could, for example, be the elements neighboring each interpolation node, as shown in Figure 2. For discontinuous elements, whose nodes lie inside the elements, the support of each basis function is a single element (panel (b)). While this works well in many cases, it fails when elements that are not neighbors are geometrically close, as is the case for e.g. thin structures.

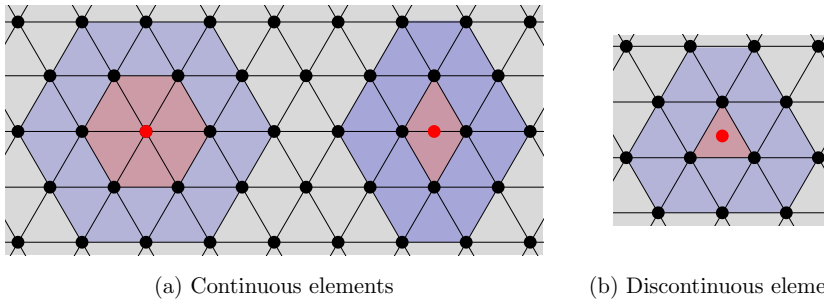


Figure 2: The support of the basis function of a node (red point) is shown in red. In some cases the near-field correction is extended to also include the blue elements.

4.1. The fast multipole method

The FMM can be used to accelerate sums of the Green's function ($\mathbf{G}$) as well as of its normal derivative ($\mathbf{F}$) [6], [7], [8]. Throughout the years, many good resources explaining the intricacies of the FMM have been written. For a gentle introduction I recommend the short course by Greengard and the pedestrian introduction by Ying [9], while the books by Gumerov and Duraiswami [10] and Liu [11] give a more detailed look with an emphasis on the BEM. Ying also has a wonderful introductory

talk. Given these resources, the details of the FMM are left out here, and we focus on the connection to the BEM. For this, it is enough to think of the FMM as a way of speeding up sums of the form

$$u(\mathbf{t}_k) = \sum_{j=1}^{NQ} G(\mathbf{t}_k, \mathbf{y}_j) c_j, \quad u(\mathbf{t}_k) = \sum_{j=1}^{NQ} \mathbf{n}(\mathbf{y}_j)^\top \nabla_{\mathbf{y}} G(\mathbf{t}_k, \mathbf{y}_j) c_j, \quad (21)$$

where the j th term is left out when $\mathbf{t}_k = \mathbf{y}_j$. Looking back at Equation 18 and Equation 19, this is exactly what is computed for each row of $\mathbf{G}$ and $\mathbf{F}$, with the coefficients $\mathbf{c} = \mathbf{C}\mathbf{z}$. For a fixed precision, applying the FMM to all n collocation points scales as $\mathcal{O}(n + NQ)$ at low frequencies, and as $\mathcal{O}((n + NQ) \log(n + NQ))$ for the wideband methods needed at high frequencies. Many FMM libraries exist, such as ExaFMM, ScalFMM and FMM3D, the last of which has a Julia interface. For 2D problems, the companion library FMM2D can be used from Julia through FMM2D.jl.⁴

⁴ I maintain this Julia package.

4.2. Hierarchical matrices

The $\mathcal{H}$ -matrix approach differs from the FMM in that it does not accelerate the sums directly [12], [13]. Instead, it approximates the matrix that corresponds to computing the sums for all n collocation points at the same time. For $\mathbf{G}$ this is the matrix

$$\mathbf{B} = \begin{bmatrix} G(\mathbf{t}_1, \mathbf{y}_1) & G(\mathbf{t}_1, \mathbf{y}_2) & \dots & G(\mathbf{t}_1, \mathbf{y}_{NQ}) \\ G(\mathbf{t}_2, \mathbf{y}_1) & G(\mathbf{t}_2, \mathbf{y}_2) & \dots & G(\mathbf{t}_2, \mathbf{y}_{NQ}) \\ \vdots & \vdots & \ddots & \vdots \\ G(\mathbf{t}_n, \mathbf{y}_1) & G(\mathbf{t}_n, \mathbf{y}_2) & \dots & G(\mathbf{t}_n, \mathbf{y}_{NQ}) \end{bmatrix}, \quad (22)$$

where the approximation is based on the fact that subblocks of the matrix corresponding to points far from each other are well approximated by low-rank matrices. For $\mathbf{F}$ the matrix is

$$\mathbf{B} = \begin{bmatrix} \mathbf{n}(\mathbf{y}_1)^\top \nabla_{\mathbf{y}} G(\mathbf{t}_1, \mathbf{y}_1) & \dots & \mathbf{n}(\mathbf{y}_{NQ})^\top \nabla_{\mathbf{y}} G(\mathbf{t}_1, \mathbf{y}_{NQ}) \\ \vdots & \ddots & \vdots \\ \mathbf{n}(\mathbf{y}_1)^\top \nabla_{\mathbf{y}} G(\mathbf{t}_n, \mathbf{y}_1) & \dots & \mathbf{n}(\mathbf{y}_{NQ})^\top \nabla_{\mathbf{y}} G(\mathbf{t}_n, \mathbf{y}_{NQ}) \end{bmatrix}. \quad (23)$$

How the hierarchical low-rank approximation is computed is outside the scope of this post, but the interested reader can look at [14], [15], [16], of which [14] is my preferred introduction. Furthermore, a wonderful lecture series by Alex Barnett, Adrianna Gillman and Per-Gunnar Martinsson from the 2014 CBMS-NSF Conference: Fast Direct Solvers for Elliptic PDEs can be found online. There are many implementations of hierarchical matrices, such as HLIBpro, HLib and H2Lib, and in Julia KernelMatrices.jl [17], HssMatrices.jl and HMatrices.jl implement various formats with different properties. A longer list can be found here. An often mentioned strength of $\mathcal{H}$ -matrices is that their LU factorization can be computed efficiently, which is useful when iterative solvers converge slowly, e.g. near resonances, although the ranks, and thereby the cost, grow with the frequency. However, an LU factorization cannot be applied directly to the form of Equation 20.

4.3. The interpolated factored Green function

A more recent approach is the interpolated factored Green function (IFGF) method [18]. An advantage of the IFGF is that it does not need the split into low and high frequencies required by the FMM, while at the same time being easily parallelized, as it does not rely on the fast Fourier transform [19]. However, as opposed to the purely algebraic $\mathcal{H}$ -matrices, the IFGF is based on interpolation. It therefore requires that the matrix is generated by a function that can be evaluated at any pair of points $(\mathbf{x}, \mathbf{y})$. Unfortunately, this is not the case for $\mathbf{F}$, as its entries depend on the normal, which is computed from the local coordinates $\mathbf{u}$ of the elements. A workaround is to approximate the full gradient of the Green's function with the IFGF, as it can be evaluated at any $(\mathbf{x}, \mathbf{y})$, and then compute the normal derivative from it. A simple implementation of the method can be found in IFGF.jl.

Bibliography

- [1] M. Paltorp, "Computational Improvements in the Boundary Element Method for Acoustics including Viscothermal Dissipation," Doctoral dissertation, 2023. [Online]. Available: https://orbit.dtu.dk/files/349939384/PhD_thesis_-_162_sider_-_41_farvede.pdf
- [2] S. Marburg, "The Burton and Miller Method: Unlocking Another Mystery of its Coupling Parameter," *Journal of Computational Acoustics*, vol. 24, no. 1, p. 1550016, 2016, doi: 10.1142/S0218396X15500162.
- [3] Y. Saad and M. H. Schultz, "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986, doi: 10.1137/0907058.
- [4] S. Marburg and S. Schneider, "Performance of iterative solvers for acoustic problems. Part I. Solvers and effect of diagonal preconditioning," *Engineering Analysis with Boundary Elements*, vol. 27, no. 7, pp. 727–750, 2003, doi: 10.1016/s0955-7997(03)00025-0.
- [5] S. Schneider and S. Marburg, "Performance of iterative solvers for acoustic problems. Part II. Acceleration by ILU-type preconditioner," *Engineering Analysis with Boundary Elements*, vol. 27, no. 7, pp. 751–757, 2003, doi: 10.1016/s0955-7997(03)00016-x.
- [6] L. Greengard and V. Rokhlin, "A Fast Algorithm for Particle Simulations," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, 1987, doi: 10.1016/0021-9991(87)90140-9.
- [7] L. Greengard and V. Rokhlin, "A new version of the Fast Multipole Method for the Laplace equation in three dimensions," *Acta Numerica*, vol. 6, pp. 229–269, 1997, doi: 10.1017/S0962492900002725.
- [8] H. Cheng, L. Greengard, and V. Rokhlin, "A Fast Adaptive Multipole Algorithm in Three Dimensions," *Journal of Computational Physics*, vol. 155, no. 2, pp. 468–498, 1999, doi: 10.1006/jcph.1999.6355.

- [9] L. Ying, “A pedestrian introduction to fast multipole methods,” *Science China Mathematics*, vol. 55, no. 5, pp. 1043–1051, 2012, doi: 10.1007/s11425-012-4392-0.
- [10] N. A. Gumerov and R. Duraiswami, *Fast Multipole Methods for the Helmholtz Equation in Three Dimensions*. Elsevier, 2005. doi: 10.1016/B978-0-08-044371-3.X5000-5.
- [11] Y. Liu, *Fast Multipole Boundary Element Method: Theory and Applications in Engineering*. Cambridge University Press, 2009. doi: 10.1017/CBO9780511605345.
- [12] P.-G. Martinsson, *Fast Direct Solvers for Elliptic PDEs*. Society for Industrial, Applied Mathematics, 2019. doi: 10.1137/1.9781611976045.
- [13] P.-G. Martinsson and V. Rokhlin, “A fast direct solver for boundary integral equations in two dimensions,” *Journal of Computational Physics*, vol. 205, no. 1, pp. 1–23, 2005, doi: 10.1016/j.jcp.2004.10.033.
- [14] J. Ballani and D. Kressner, “Matrices with Hierarchical Low-Rank Structures,” *Lecture Notes in Mathematics*, vol. 2173, pp. 161–209, 2016.
- [15] M. Bebendorf, *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*. Springer, 2008.
- [16] W. Hackbusch, *Hierarchical Matrices: Algorithms and Analysis*. Springer, 2015.
- [17] C. J. Geoga, M. Anitescu, and M. L. Stein, “Scalable Gaussian Process Computations Using Hierarchical Matrices,” *Journal of Computational and Graphical Statistics*, vol. 29, no. 2, pp. 227–237, 2020, doi: 10.1080/10618600.2019.1652616.
- [18] C. Bauinger and O. P. Bruno, ““Interpolated Factored Green Function” Method for accelerated solution of Scattering Problems,” *Journal of Computational Physics*, vol. 430, p. 110095, 2021, doi: 10.1016/j.jcp.2020.110095.
- [19] C. Bauinger and O. P. Bruno, “Massively parallelized interpolated factored Green function method,” *Journal of Computational Physics*, vol. 475, p. 111837, 2023, doi: 10.1017/CBO9780511605345.