3D Boundary Element Method for the Laplace and screened Poisson equations

Giovanni Maria Bonvini, corr. author Carlo de Falco

Politecnico di Milano, Department of Mathematics, Milan, Italy

May 14, 2025

Abstract

A C++ program for the solution of the Laplace and screened Poisson equation on generic 3D domains with mixed Dirichlet-Neumann-Robin boundary conditions via Boundary Element Method is presented. The solver adopts the well-documented deal.II library for the implementation. An external file allows the user to specify the problem settings, including geometry, boundary conditions, linear solver and quadrature rules. Convergence results are provided for specific benchmark problems, with a comparison against other implementations found in literature.

This project highlights the adoption of the deal.II library for Boundary Element Method implementations and lays the foundation for an efficient and generic Boundary Element Method solver.

Keywords: deal.II; Laplace equation; screened Poisson; mixed boundary conditions; general geometry; Green functions

Contents

1 Introduction

The Boundary Element Method (BEM) represents a class of efficient numerical techniques for solving partial differential equations (PDEs), particularly advantageous for problems involving infinite domains or complex geometries. Through reformulation of the PDE into a boundary integral equation (BIE), BEM reduces the spatial dimension of the problem by one, eliminating the need for volume discretization. In the collocation BEM approach – the focus of this work – the boundary is divided into elements, the solution is approximated using basis functions (typically piecewise constants or linears), and boundary conditions are enforced at discrete collocation points. This methodology proves particularly effective for exterior problems where domain-based methods like the Finite Element Method (FEM) become computationally prohibitive.

This work extends the deal.II tutorial step-34 (3), which implements a BEM solver for the *exterior* Laplace equation with fully Neumann boundary conditions on a spherical domain. While this implementation provides a valuable foundation, its limitations to spherical geometries and single-boundary-condition type restrict broader applicability. Our contributions address five fundamental extensions:

- **General Geometries**: Support for arbitrary 3D surface meshes, enabling analysis of non-spherical domains
- **Interior/Exterior Formulations**: Handling of both interior and exterior Laplace problems through adaptive normal direction selection
- **Mixed Boundary Conditions**: Flexible specification of Dirichlet (prescribed potential), Neumann (prescribed flux) or Robin conditions on distinct boundary regions
- **General Green functions**: Extension of the solver to address different differential problems by specifying the Green function at runtime. We show here the application to a screened Poisson problem by changing the Green function to a Yukawa potential
- **General quadrature rule**: We allow the user to choose the preferred quadrature rule, for both singular and non-singular integrals

These enhancements significantly expand the solver's applicability to real-world engineering scenarios, such as electrostatic shielding, potential flows in cavities, mixed thermal boundary value problems.

2 Theoretical framework

The mathematical formulation of the Laplace problem that we intend to solve is the following:

$$\Delta\phi(\mathbf{x}) = 0, \quad \mathbf{x} \in \Omega \tag{1}$$

with $\Omega \subset \mathbb{R}^3$.

$$\phi(\mathbf{x}) = g(\mathbf{x}), \quad g(\mathbf{x}) \in C^1(\Gamma_D), \quad \mathbf{x} \in \Gamma_D \tag{2}$$

$$\frac{\partial\phi}{\partial\mathbf{n}} = h(\mathbf{x}), \quad h(\mathbf{x}) \in C^0(\Gamma_N), \quad \mathbf{x} \in \Gamma_N \tag{3}$$

where Γ_N and Γ_D are the boundaries of the domain where we prescribe, respectively, the Neumann and Dirichlet boundary conditions.

It can be proven that the following function, called Fundamental Solution,

$$G(\mathbf{y}-\mathbf{x}) = \frac{1}{4\pi}\frac{1}{|\mathbf{y}-\mathbf{x}|} \tag{4}$$

satisfies in a distributional sense the equation

$$\Delta_y G(\mathbf{y}-\mathbf{x}) = \delta(\mathbf{y}-\mathbf{x}) \tag{5}$$

where the derivative is done with respect to the variable $\mathbf{y}$.

By using the usual Green identities, our problem can be written on the boundary $\partial\Omega = \Gamma$ only. We recall the general definition of the second Green identity:

$$\int_\Omega (-\Delta u)v\,dx + \int_{\partial\Omega}\frac{\partial u}{\partial\mathbf{n}}v\,ds = \int_\Omega (-\Delta v)u\,dx + \int_{\partial\Omega} u\frac{\partial v}{\partial\mathbf{n}}\,ds \tag{6}$$

$$1) \quad \int_\Omega (-\Delta\phi)G(\mathbf{y}-\mathbf{x})\,dx = 0 \qquad \text{since} \quad \Delta\phi = 0 \tag{7}$$

with $\mathbf{n}$ being the normal to the surface $\partial\Omega$ pointing outward from the domain of integration Ω. It is important to note that the domain of integration Ω can be either exterior to the surface $\partial\Omega$ or interior. In any case, the normal direction should be pointing in the opposite direction of the domain of integration, as shown in Figure 1 for a 2D case.

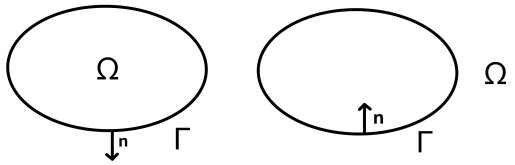

Figure 1: Normals orientation: internal (left) external (right)

From now on, we will carry on a parallel description for both internal and external problems, as they will have slightly different treatments.

If we substitute u and v in the Green identity with the solution ϕ of the Laplace problem (1) and the fundamental solution (4) respectively, for a fixed point $\mathbf{x}$ in the domain Ω, we obtain:

$$\begin{aligned}&\int_\Omega (-\Delta_y\phi)G(\mathbf{y}-\mathbf{x})\,dy + \int_{\partial\Omega}\frac{\partial\phi}{\partial\mathbf{n}}(\mathbf{y})G(\mathbf{y}-\mathbf{x})\,ds_y \\ &= \int_\Omega (-\Delta_y G(\mathbf{y}-\mathbf{x}))\phi(\mathbf{y})\,dy + \int_{\partial\Omega}\phi(\mathbf{y})\frac{\partial G(\mathbf{y}-\mathbf{x})}{\partial\mathbf{n}}\,ds_y\end{aligned} \tag{8}$$

$$\Rightarrow \qquad \int_{\partial\Omega}\frac{\partial\phi}{\partial\mathbf{n}}(\mathbf{y})G(\mathbf{y}-\mathbf{x})\,ds_y = -\phi(\mathbf{x}) + \int_{\partial\Omega}\phi(\mathbf{y})\frac{\partial G(\mathbf{y}-\mathbf{x})}{\partial\mathbf{n}}\,ds_y \tag{9}$$

since

$$\Delta\phi = 0 \quad \forall x \in \Omega$$

and

$$\int_\Omega (-\Delta_y G(\mathbf{y}-\mathbf{x}))\phi(\mathbf{y})\,dy = \int_\Omega -\delta(\mathbf{y}-\mathbf{x})\phi(\mathbf{y})dy = -\phi(\mathbf{x})$$

We can now define the so called Single and Double Layer Potential operators

$$(S\frac{\partial\phi}{\partial\mathbf{n}})(\mathbf{x}) = \int_{\partial\Omega} G(\mathbf{y}-\mathbf{x})\frac{\partial\phi}{\partial\mathbf{n}}(\mathbf{y})\,ds_y \tag{10}$$

$$(D\phi)(\mathbf{x}) = \int_{\partial\Omega} \frac{\partial G(\mathbf{y}-\mathbf{x})}{\partial\mathbf{n}}\phi(\mathbf{y})\,ds_y \tag{11}$$

and rewrite Equation (9) in the form:

$$\phi(\mathbf{x}) - (D\phi)(\mathbf{x}) = -(S\frac{\partial\phi}{\partial\mathbf{n}})(\mathbf{x}) \quad \forall\mathbf{x}\in\Omega \tag{12}$$

It is important to highlight what happens to the Single and Double Layer Potential operators when we want to solve an *external* problem. In this case, the domain Ω will have boundary $\partial\Omega = \Gamma\cup\Gamma_\infty$, where we define the "boundary at infinity" as:

$$\Gamma_\infty := \lim_{r\to\infty} \partial B_r(0) \tag{13}$$

The boundary integrals for the Single and Double layer operators will therefore have to be split into an integral over Γ (which is what we actually define as our operators) and an integral over Γ_∞. For most cases, we can enforce a boundary condition at infinity and set $\nabla\phi = 0$ at Γ_∞ and choose ϕ_∞ as constant value for $\phi(\mathbf{x})$ when $\mathbf{x}\to\infty$. This result in the following:

$$\int_{\Gamma\cup\Gamma_\infty} G(\mathbf{y}-\mathbf{x})\frac{\partial\phi}{\partial\mathbf{n}}(\mathbf{y})\,ds_y = \int_\Gamma G(\mathbf{y}-\mathbf{x})\frac{\partial\phi}{\partial\mathbf{n}}(\mathbf{y})\,ds_y \tag{14}$$

since ϕ is constant at infinity, and

$$\int_{\Gamma\cup\Gamma_\infty} \frac{\partial G(\mathbf{y}-\mathbf{x})}{\partial\mathbf{n}}\phi(\mathbf{y})\,ds_y = \phi_\infty + \int_\Gamma \frac{\partial G(\mathbf{y}-\mathbf{x})}{\partial\mathbf{n}}\phi(\mathbf{y})\,ds_y \tag{15}$$

since

$$\int_{\Gamma_\infty} \frac{\partial G(\mathbf{y}-\mathbf{x})}{\partial\mathbf{n}}\,\phi_\infty\,ds_y = -\lim_{r\to\infty}\int_{\partial B_r(0)} \frac{\mathbf{r}}{r}\cdot\nabla G(\mathbf{y}-\mathbf{x})\,\phi_\infty\,ds_y = \phi_\infty \tag{16}$$

So Equation (12), for an external problem, becomes

$$\phi(\mathbf{x}) - (D\phi)(\mathbf{x}) = \phi_\infty - (S\frac{\partial\phi}{\partial\mathbf{n}})(\mathbf{x}) \quad \forall\mathbf{x}\in\Omega \tag{17}$$

We can now apply the boundary conditions (2), (3) at the border Γ, to obtain:

$$(S\frac{\partial\phi}{\partial\mathbf{n}})(\mathbf{x}) = \int_\Gamma G(\mathbf{y}-\mathbf{x})\frac{\partial\phi}{\partial\mathbf{n}}(\mathbf{y})\,ds_y = \int_{\Gamma_D} G(\mathbf{y}-\mathbf{x})\frac{\partial\phi}{\partial\mathbf{n}}(\mathbf{y})\,ds_y + \int_{\Gamma_N} G(\mathbf{y}-\mathbf{x})h(\mathbf{y})(\mathbf{y})\,ds_y \tag{18}$$

$$(D\phi)(\mathbf{x}) = \int_\Gamma \frac{\partial G(\mathbf{y}-\mathbf{x})}{\partial\mathbf{n}}\phi(\mathbf{y})\,ds_y = \int_{\Gamma_D} \frac{\partial G(\mathbf{y}-\mathbf{x})}{\partial\mathbf{n}}g(\mathbf{y})\,ds_y + \int_{\Gamma_N} \frac{\partial G(\mathbf{y}-\mathbf{x})}{\partial\mathbf{n}}\phi(\mathbf{y})\,ds_y \tag{19}$$

valid $\forall\mathbf{x}\in\Omega$. Now taking the limit for $\mathbf{x}\to\Gamma$, using the well-known properties of the single and double layer operators - continuity of S and jump of D, see (2) - we obtain an equation for ϕ just on the boundary Γ of Ω, usually referred as the Boundary Integral Equation (BIE):

$$\alpha(\mathbf{x})\phi(\mathbf{x}) - (D\phi)(\mathbf{x}) = -(S\frac{\partial\phi}{\partial\mathbf{n}})(\mathbf{x}) \quad \forall\mathbf{x}\in\Gamma \tag{20}$$

for an interior problem, and

$$\alpha(\mathbf{x})\phi(\mathbf{x}) - (D\phi)(\mathbf{x}) = \phi_\infty - (S\frac{\partial\phi}{\partial\mathbf{n}})(\mathbf{x}) \quad \forall\mathbf{x}\in\Gamma \tag{21}$$

for an exterior problem.
From now on, we will assume a value for $\phi_\infty = 0$ so that the formulations for the interior and exterior problems coincide.
The quantity $\alpha(\mathbf{x})$ is the fraction of solid angle by which the point $\mathbf{x}$ sees the domain of integration

Ω. In particular, at points $\mathbf{x}$ where the boundary is differentiable (i.e. smooth) we have $\alpha(\mathbf{x}) = \frac{1}{2}$, but the value may be smaller or larger at points where the boundary has a corner or an edge.
To find the value of $\alpha(\mathbf{x})$ at a given point in the border $\partial\Omega$ we can use the double layer potential itself:
This is possible because for the Laplace equation the solution of a *fully Neumann* problem is known up to an arbitrary constant c, which means that, if we set the Neumann data to be zero, then any constant $\phi = \phi_\infty$ will be a solution. Inserting the constant solution and the Neumann boundary condition in the boundary integral equation for an exterior problem (21), we have

$$\alpha(\mathbf{x})\phi_\infty - \int_{\partial\Omega} \frac{\partial G(\mathbf{y}-\mathbf{x})}{\partial \mathbf{n}} \phi_\infty \, ds_y = \phi_\infty \tag{22}$$

which gives

$$\alpha(\mathbf{x}) = 1 + \int_{\partial\Omega} \frac{\partial G(\mathbf{y}-\mathbf{x})}{\partial \mathbf{n}} \, ds_y = 1 + (D[1])(\mathbf{x}) \tag{23}$$

2.1 Exterior solution

Once the BIE is resolved - i.e. the value for ϕ and ϕ_n on the whole boundary is found - we can finally compute the solution for $\phi(\mathbf{x})$ in all $\mathbf{x} \in \Omega$. To this end, we recall, from Equation (12)

$$\phi(\mathbf{x}) = (D\phi)(\mathbf{x}) - (S\frac{\partial \phi}{\partial \mathbf{n}})(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega \tag{24}$$

where now we have everything that is on the right hand side since S and D are integrals we can evaluate and ϕ and ϕ_n on the boundary we have just computed.

3 Numerical discretization

The numerical approximation of the BIE are commonly referred as boundary element methods (BEM). The whole idea behind BEM is to find a numerical solution of the BIE and then use it to retrive the solution in the domain Ω with the Equations (12) or (17).
Let $\mathcal{T}_h = \bigcup_i K_i$ be a subdivision of the manifold Γ into M quadrilaterals. We will call each individual quadrilateral an *element*. We define a finite dimensional space V_h as

$$V_h := \left\{ v \in C^0(\Gamma) \quad s.t. \quad v|_{K_i} \in Q^1(K_i), \ \forall i \right\} \tag{25}$$

with basis functions $\psi_i(\mathbf{x})$. An element ϕ_h of V_h is uniquely identified by the vector $\boldsymbol{\phi}$ of its coefficients ϕ_i, that is:

$$\phi_h := \sum_j \phi_j \psi_j(\mathbf{x}), \quad \boldsymbol{\phi} := \{\phi_j\} \tag{26}$$

The most common approximation of boundary integral equation is by use of the collocation based boundary element method. This method requires the evaluation of the boundary integral equation at a number of collocation points which is equal to the number of unknowns of the system. We call these points $\mathbf{x}_i$ with $i = 1, ..., N$. The problem therefore becomes the following. Given $g(\mathbf{x}), h(\mathbf{x})$ Dirichlet and Neumann data on the boundary, find a function $\phi_h \in V_h$ that satify

$$\alpha(\mathbf{x}_i)\phi_h(\mathbf{x}_i) - \int_\Gamma \frac{\partial G(\mathbf{y}-\mathbf{x}_i)}{\partial \mathbf{n}} \phi_h(\mathbf{y}) \, ds_y = -\int_\Gamma G(\mathbf{y}-\mathbf{x}_i) \frac{\partial \phi_h}{\partial \mathbf{n}}(\mathbf{y}) \, ds_y \quad \forall i = 1, \ldots, N \tag{27}$$

Using the definition for the functions in the space V_h, we can write $\phi_h = \phi_j \psi_j(\mathbf{x})$ and $\frac{\partial \phi_h}{\partial n} = \phi_{n,h} = \phi_{n,j}\psi_j(\mathbf{x})$ we can rewrite the problem as

$$\sum_j [\alpha(\mathbf{x}_i)\psi_j(\mathbf{x}_i) - \int_\Gamma \frac{\partial G(\mathbf{y}-\mathbf{x}_i)}{\partial \mathbf{n}} \psi_j(\mathbf{y}) \, ds_y]\phi_j = \sum_j [-\int_\Gamma G(\mathbf{y}-\mathbf{x}_i)\psi_j(\mathbf{y}) \, ds_y]\phi_{n,j} \tag{28}$$

thus leading to the algebraic form

$$\boldsymbol{H\phi = G\phi_n} \tag{29}$$

with the following definitions:

$$\boldsymbol{H}_{ij} = \alpha(\mathbf{x}_i)\psi_j(\mathbf{x}_i) - \int_\Gamma \frac{\partial G(\mathbf{y}-\mathbf{x}_i)}{\partial \mathbf{n}} \psi_j(\mathbf{y}) \, ds_y \tag{30}$$

$$\boldsymbol{G}_{ij} = -\int_{\Gamma} G(\mathbf{y}-\mathbf{x}_i)\psi_j(\mathbf{y})\, ds_y \tag{31}$$

$$\{\boldsymbol{\phi}\}_j = \phi_j \tag{32}$$

$$\{\boldsymbol{\phi_n}\}_j = \phi_{n,j} \tag{33}$$

In order to solve the algebraic problem we need to state which is the matrix and which is the right hand side so that we can write a formulation in the form $\boldsymbol{Ax} = \boldsymbol{b}$. For a mixed boundary conditions problem we need to rearrange the matrices $\boldsymbol{H}$ and $\boldsymbol{G}$, exploiting the fact that we have known values for $\boldsymbol{\phi}$ and $\boldsymbol{\phi_n}$, where they are prescribed by $g(\mathbf{x})$ and $h(\mathbf{x})$.
We can separate the vectors $\boldsymbol{\phi}$ and $\boldsymbol{\phi_n}$ as:

$$\boldsymbol{\phi} = \begin{pmatrix} \hat{\boldsymbol{\phi}} \\ \boldsymbol{\phi} \end{pmatrix} \quad \boldsymbol{\phi_n} = \begin{pmatrix} \boldsymbol{\phi_n} \\ \hat{\boldsymbol{\phi_n}} \end{pmatrix} \tag{34}$$

where $\hat{\boldsymbol{\phi}}$ and $\hat{\boldsymbol{\phi_n}}$ are the known values. Note that the order of the known and unknown values in the two vectors could be very different from case (34), often with alternating values of knowns and unknowns inside each vector. Nonetheless, since $\Gamma_D \cap \Gamma_n = \emptyset$ we will always have that at a given position j we will either have $\{\boldsymbol{\phi}\}_j$ known and $\{\boldsymbol{\phi_n}\}_j$ unknown or vice-versa.
We can, therefore, write (29) as

$$[\boldsymbol{H_1} : \boldsymbol{H_2}] \begin{pmatrix} \hat{\boldsymbol{\phi}} \\ \boldsymbol{\phi} \end{pmatrix} = [\boldsymbol{G_1} : \boldsymbol{G_2}] \begin{pmatrix} \boldsymbol{\phi_n} \\ \hat{\boldsymbol{\phi_n}} \end{pmatrix} \tag{35}$$

and rearrange it in

$$[\boldsymbol{H_2} : -\boldsymbol{G_1}] \begin{pmatrix} \boldsymbol{\phi} \\ \boldsymbol{\phi_n} \end{pmatrix} = [\boldsymbol{G_2} : -\boldsymbol{H_1}] \begin{pmatrix} \hat{\boldsymbol{\phi_n}} \\ \hat{\boldsymbol{\phi}} \end{pmatrix} \tag{36}$$

to find, in conclusion, the algebraic system

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{37}$$

with

$$\boldsymbol{A} = [\boldsymbol{H_2} : -\boldsymbol{G_1}], \quad \boldsymbol{b} = \boldsymbol{G_2}\hat{\boldsymbol{\phi_n}} - \boldsymbol{H_1}\hat{\boldsymbol{\phi}}, \quad \boldsymbol{x} = \begin{pmatrix} \boldsymbol{\phi} \\ \boldsymbol{\phi_n} \end{pmatrix} \tag{38}$$

3.1 Screened Poisson equation

Even if the program was initially built to solve the Laplace problem, we decided to expand its range of application to a more general differential equation: the screened Poisson equation. Its most elementary application is the problem of a point charge in a spherical cavity of radius a inside a conducting medium.

3.1.1 Analytical formulation

The problem is formulated with a Laplace problem at the interior of the sphere and a screened Poisson problem for the exterior:

$$\nabla^2 \phi_{\text{in}}(\mathbf{r}) = 0, \qquad 0 \le r < a, \tag{39}$$

$$\left(\nabla^2 - \kappa^2\right) \phi_{\text{out}}(\mathbf{r}) = 0, \qquad r > a, \qquad \phi_{\text{out}}(\mathbf{r}) \xrightarrow{r\to\infty} 0. \tag{40}$$

with the interface conditions for the electric potential

$$\phi_{\text{in}} = \phi_{\text{out}}, \qquad \varepsilon_i\, \partial_r \phi_{\text{in}} = \varepsilon_o\, \partial_r \phi_{\text{out}} \qquad \text{at } r = a. \tag{41}$$

where ε_i and ε_o are the dielectric constants for the interior and exterior domain.
Analytical solutions for this coupled problem can be found with a series expansion (see (11)):

$$\phi_{\text{in}}(r,\theta) = \sum_{n=0}^{\infty} \Big[C_n\, r^n + D_n\, r^{-(n+1)} \Big] P_n(\cos\theta), \tag{42}$$

$$\phi_{\text{out}}(r,\theta) = \sum_{n=0}^{\infty} B_n\, k_n\big(\kappa r\big)\, P_n(\cos\theta), \tag{43}$$

with k_n the modified spherical Bessel function of the third kind (also known as modified spherical Hankel function) and n=0,1,2,... labels the degree of the Legendre polynomial $P_n(\cos\theta)$. For each degree n define $u = \kappa a$. Applying the two interface conditions yields

$$\begin{aligned} C_n a^{\,n} + D_n a^{-(n+1)} &= B_n\, k_n(u), \\ \varepsilon_i \Big[n C_n a^{\,n-1} - (n+1) D_n a^{-(n+2)} \Big] &= \varepsilon_o\, \kappa\, B_n k_n'(u). \end{aligned} \tag{44}$$

Solve the first continuity equation for C_n, insert into the flux equation, and divide by $k_n(u)$ to find the *Robin transmission condition*:

$$\partial_r \phi_{\text{out}}(a,\theta) + \beta_n(u)\, \phi_{\text{out}}(a,\theta) = g_n(\theta), \tag{45}$$

where

$$\beta_n(u) = -\frac{\varepsilon_i}{\varepsilon_o} \frac{n}{a} \tag{46}$$

$$g_n(\theta) = -\frac{\varepsilon_i}{\varepsilon_o} \frac{(2n+1) D_n}{a^{\,n+2}}\, P_n(\cos\theta) \tag{47}$$

This is the general analytical Robin boundary operator that couples the Laplace interior to the screened-Poisson exterior.
Thanks to this, the initial interior-exterior coupled problem becomes just an exterior problem with Robin boundary conditions:

$$\big(\nabla^2 - \kappa^2\big)\, \phi(\mathbf{r}) = 0, \qquad r > a, \qquad \phi(\mathbf{r}) \xrightarrow{r\to\infty} 0. \tag{48}$$

$$\partial_r \phi(a,\theta) + \beta_n(u)\, \phi(a,\theta) = g_n(\theta), \tag{49}$$

The implementation of this problem into our program is rather simple as the Green function (i.e. fundamental solution) for the screened Poisson - also known as Yukawa potential - is little different to the Laplace one:

$$G_\kappa(\mathbf{y}-\mathbf{x}) = \frac{1}{4\pi} \frac{e^{-\kappa|\mathbf{y}-\mathbf{x}|}}{|\mathbf{y}-\mathbf{x}|} \tag{50}$$

which is solution, in a distributional sense, to

$$[\Delta_y - \kappa^2] G_\kappa(\mathbf{y}-\mathbf{x}) = \delta(\mathbf{y}-\mathbf{x}) \tag{51}$$

Notice that for $\mathbf{y} \to \mathbf{x}$ the Yukawa potential shows the same $\frac{1}{R}$ behaviour of the Laplace Green function. This allows us to maintain the same quadrature rules adopted for the Laplace problem. Additionally, if we set the screening factor $\kappa = 0$ we correctly retrieve the Laplace Green function. The definitions for the single and double layer operators hold, with the only modification that instead of G we use G_κ. We recall Equation (20)

$$\alpha(\mathbf{x})\phi(\mathbf{x}) - (D_\kappa \phi)(\mathbf{x}) = -(S_\kappa \frac{\partial \phi}{\partial \mathbf{n}})(\mathbf{x}) \quad \forall \mathbf{x} \in \Gamma \tag{52}$$

where

$$(S_\kappa \phi_n)(\mathbf{x}) = \int_\Gamma G_\kappa(\mathbf{y}-\mathbf{x})\, \phi_n(\mathbf{y})\, ds_y, \quad (D_\kappa \phi)(\mathbf{x}) = \int_\Gamma \partial_{n_y} G_\kappa(\mathbf{y}-\mathbf{x})\, \phi(\mathbf{y})\, ds_y. \tag{53}$$

Now insert the Robin conditions (49) into (52) to get

$$\alpha\phi - D_\kappa \phi = -S_\kappa\big(g - \beta\phi\big) = -S_\kappa g + S_\kappa(\beta\phi), \qquad \mathbf{x} \in \Gamma. \tag{54}$$

Re-arrange to obtain the boundary integral equation for the unknown surface potential ϕ:

$$\big(\alpha I - D_\kappa\big)\phi \;-\; S_\kappa(\beta\phi) \;=\; -\,S_\kappa g. \tag{55}$$

3.1.2 Numerical discretization

Following the same approach as for the Laplace problem, we can define $\{\psi_j\}_{j=1}^N$ to be a continuous nodal basis functions on the triangulated surface Γ. Then, let $\mathbf{x}_i$ $(i = 1, \ldots, N)$ be the node (collocation) points. Therefore, we can write

$$\phi_h(\mathbf{y}) = \sum_{j=1}^{N} \phi_j \, \psi_j(\mathbf{y}), \qquad g_h(\mathbf{y}) = \sum_{j=1}^{N} g_j \, \psi_j(\mathbf{y}). \tag{56}$$

Now evaluate (55) at $\mathbf{x}_i$ and replace ϕ, g by the discrete expansions:

$$\sum_{j=1}^{N} \Big[\alpha_i \delta_{ij} - D_{ij}^{(\kappa)} - B_{ij}^{(\kappa)} \Big] \, \phi_j = - \sum_{j=1}^{N} S_{ij}^{(\kappa)} \, g_j, \qquad i = 1, \ldots, N. \tag{57}$$

Matrix entries:

$$D_{ij}^{(\kappa)} = \int_\Gamma \partial_{n_y} G_\kappa(\mathbf{y} - \mathbf{x}_i) \, \psi_j(\mathbf{y}) \, ds_y \quad \text{(double layer)}, \tag{58}$$

$$S_{ij}^{(\kappa)} = \int_\Gamma G_\kappa(\mathbf{y} - \mathbf{x}_i) \, \psi_j(\mathbf{y}) \, ds_y \quad \text{(single layer)}, \tag{59}$$

$$B_{ij}^{(\kappa)} = \int_\Gamma G_\kappa(\mathbf{y} - \mathbf{x}_i) \, \beta(\mathbf{y}) \, \psi_j(\mathbf{y}) \, ds_y. \tag{60}$$

yielding the following algebraic system

$$\boxed{\big[\, \alpha I - D^{(\kappa)} - B^{(\kappa)} \, \big] \, \boldsymbol{\phi} \;=\; - S^{(\kappa)} \, \mathbf{g}} \tag{61}$$

Where $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_N)^\mathsf{T}$ are the unknown nodal values of the surface potential, $\mathbf{g} = (g_1, \ldots, g_N)^\mathsf{T}$ is the (known) vector obtained by sampling g at the nodes, and the matrices $H^{(\kappa)}, S^{(\kappa)}, B^{(\kappa)}$ are defined by the integrals above. Once $\boldsymbol{\phi}$ is found we can retrieve the normal flux at the boundary ϕ_n using the Robin condition:

$$\phi_{n,i} = g_i - \beta_i \phi_i \tag{62}$$

3.2 Exterior solution

Once the Boundary Integral Equation has been solved on Γ and we know both the boundary potential ϕ and its normal derivative $\partial\phi/\partial\mathbf{n}$ at every collocation point, the value of ϕ at any field point $\mathbf{x} \in \Omega$ (either *outside* or *inside* the surface, depending on the sign convention chosen in Section 2) is recovered by directly evaluating Equations (12) and (17). In practice we assemble

$$\phi_h(\mathbf{x}) = D_h[\phi](\mathbf{x}) \;-\; S_h\big[\phi_n\big](\mathbf{x}), \tag{63}$$

where D_h and S_h denote the same discrete double- and single-layer operators already used for the system matrices (Equations (30) and (31)), but now *with the collocation point* $\mathbf{x}$ *located in the domain rather than on the boundary.* Numerically the evaluation reuses the existing quadrature code:

1. For every element K_e we compute the elementary contributions

$$\int_{K_e} \partial_{\mathbf{n}_y} G(\mathbf{x} - \mathbf{y}) \, \phi_h(\mathbf{y}) \, dS_\mathbf{y}, \qquad \int_{K_e} G(\mathbf{x} - \mathbf{y}) \, \phi_{n,h}(\mathbf{y}) \, dS_\mathbf{y}.$$

2. If $\mathbf{x}$ happens to be very close to K_e, the same singular-aware quadrature (`QGaussOneOverR`) is employed; in the far field ordinary Gauss rules suffice.

3. The loop over all elements yields the two scalar sums forming Equation (63).

Because no algebraic system is solved in this post-processing stage, the operation scales linearly with the number of elements and can be carried out at an arbitrary set of evaluation points (grid, probe line, or mesh of an enclosing surface) without modifying the solver's core.

The identical procedure applies to an *interior* evaluation: the only difference is the sign of the normal vector used in $\partial_{\mathbf{n}_y} G$ (cf. the normal-orientation discussion in Sections 1 and 2). Hence the same implementation delivers both interior and exterior potential maps once the boundary data are known.

4 Numerical integration

In order to compute the elements of the matrices $\mathbf{H}$ and $\mathbf{G}$ we need to find the values of the integrals in (30) and (31). The numerical integration scheme is here presented for $\mathbf{G}_{ij}$, but the same treatment applies to $\mathbf{H}_{ij}$. We can divide the boundary Γ into a number $e = 1, ..., N$ elements and have

$$\int_\Gamma G(\mathbf{y} - \mathbf{x}_i)\psi_j(\mathbf{y})\, ds_y = \sum_e \int_{K_e} G(\mathbf{y} - \mathbf{x}_i)\psi_j(\mathbf{y})\, ds_y \tag{64}$$

The integral over the element the general K_e is mapped to a reference square element upon which the numerical integration with q quadrature points is performed.

$$\int_{K_e} G(\mathbf{y} - \mathbf{x}_i)\psi_j(\mathbf{y})\, ds_y = \sum_{h=1}^{q} G(\hat{\mathbf{y}} - \hat{\mathbf{x}}_i)\psi_j(\hat{\mathbf{y}}) J_{K_e} w_h \tag{65}$$

where J_{K_e} is the Jacobian of the transformation and w_h the weight of each quadrature point.

4.1 Treating singular integrals

Boundary-element formulations collapse a volumetric PDE onto the boundary Γ; the price to pay is that the resulting integral operators contain *singular kernels*. Accurate and efficient evaluation of these integrals is therefore one of the core technical challenges of every BEM implementation.

$$G(\mathbf{y} - \mathbf{x}) = \frac{1}{4\pi} \frac{1}{|\mathbf{y} - \mathbf{x}|} \sim \frac{1}{R}, \qquad R = |\mathbf{y} - \mathbf{x}|, \tag{66}$$

$$\partial_{\mathbf{n}} G(\mathbf{y} - \mathbf{x}) = -\frac{(\mathbf{y} - \mathbf{x}) \cdot \mathbf{n}_y}{4\pi\, |\mathbf{y} - \mathbf{x}|^3} \sim \frac{1}{R^2}, \tag{67}$$

describe, respectively, the *single-layer* and *double-layer* (Laplace) kernels. Equation (66) is *weakly singular* ($1/R$ is integrable on a surface), whereas (67) appears *hypersingular*.

Handling the $1/R$ singularity For the element that contains the collocation point $\mathbf{x}$ we switch to local polar coordinates centred at that point. In the reference square $\left[0, 1\right]^2$ a *Duffy transform* $(u, v) = (\rho \cos\theta,\ \rho \sin\theta)$ gives a Jacobian factor $\rho = R$, exactly cancelling the $1/R$ in (66). The integrand becomes smooth and is integrated with an ordinary Gauss rule. deal.II packs this procedure in the class `QGaussOneOverR(<2>)` , so the code merely swaps quadrature objects when the panel is "singular".

The remaining, direction-dependent part yields the well-known $\frac{1}{2}\,\phi(\mathbf{x})$ "jump" term; the program adds this analytic contribution to the matrix diagonal after numerical assembly. Thus both kinds of singularities are treated with a combination of *(i)* analytic regularisation (for the jump) and *(ii)* a polar/Duffy transformation embedded in a purpose-built quadrature rule (7).

Apparent $1/R^2$ singularity of the double layer For constant boundary element (in our case linear quads) the scalar product $(\mathbf{y} - \mathbf{x}) \cdot \mathbf{n}_y = 0$ every time because both $\mathbf{x}$ and $\mathbf{y}$ lie on the same element and since the element is flat this means that $(\mathbf{y} - \mathbf{x}) \perp \mathbf{n}_y$. So the integral yields 0 and no special treatment is required.

5 Implementation

5.1 Introduction to deal.ii

Deal.II is an open-source, dimension-independent C++ library originally designed for finite element methods but extended to boundary-element formulations via codimension templates (e.g. `Triangulation<dim-1,dim>`, `FE_Q<dim-1,dim>`). Its modular architecture, extensive tutorial suite (nearly 100 step-by-step examples), and comprehensive Doxygen documentation make it an ideal foundation for developing a collocation BEM solver for the 3D Laplace equation. In particular, tutorial step-34—which implements a fully Neumann collocation BEM for Laplace's equation—provides a ready-to-use starting point that demonstrates the library's boundary-element capabilities. Nonetheless, other libraries where considered for the implementation, such as Bembel (6), BemTool (1) and BETL (10). A brief comparison is here reported:

- **Documentation & Community:** Deal.II offers nearly 100 tutorials with in-depth commentary, a fully indexed Doxygen reference, a responsive user forum and mailing list, and regular release cycles, whereas Bembel, BemTool and BETL provide sparse examples and limited online support.
- **Solver Interfaces:** Native wrappers for PETSc and Trilinos allow access to scalable Krylov methods (e.g. GMRES) and advanced preconditioners (AMG, block preconditioners), features absent or rudimentary in the other three libraries.
- **Parallel Scalability:** Hybrid parallelism (Intel TBB for shared memory, MPI for distributed memory) in deal.II scales across hundreds to thousands of cores, essential for dense BEM matrix assembly; in contrast, Bembel uses OpenMP only for local assembly, and BemTool/BETL have no distributed parallel support.
- **Extensibility & Maintenance:** A large developer base and continuous integration pipelines ensure new features, bug fixes and performance optimisations are rapidly incorporated, whereas the other libraries see infrequent updates and narrower contributor pools.

Thanks to its complete end-to-end support for codimension-based boundary elements, industrial-strength solver backends, proven parallel scalability and unrivalled documentation/community ecosystem, deal.II was chosen as the core library for the 3D Laplace BEM code in this project.

5.2 Use of the program

5.2.1 Mesh requirements

The deal.ii library, through the `Triangulation<dim - 1, dim>` class, allows to create a mesh based on standard manifold geometries such as sphere, cylinder, torus and others. Nonetheless, we wanted a program able to run simulations on generic geometries and, therefore, we chose to import meshes generated with external software, such as Gmsh.
To be able to correctly run a simulation the mesh must satisfy some requirements:

1. The geometry must be a 3D manifold;
2. The mesh must be the surface mesh of the manifold;
3. The mesh must be made of quadrilateral elements;
4. `.msh` and `.vtk` formats are accepted;
5. The mesh must be counter-clockwise oriented so that normals to the surface are pointing outward (a python script `check_normals.py` is provided to check the correct orientation of the mesh);

Python scripts to create the mesh used in the benchmark problems are provided for reference, as well as the mesh files used.

5.2.2 Note on spherical mesh: *cubed sphere*

It is important to highlight that the surface mesh of a sphere - which was widely used in this program - using only quads element is no trivial task. To this end, we suggest the user to follow the procedure that was here adopted:

1. Build the surface mesh of a cube of length $2R$ (where R is the desired radius of the sphere) using uniform quad element and with the center of the cube lying on the center of the desired sphere;
2. Project the nodal points with a linear transformation onto a spherical surface;

This method effectively creates the so called *cubed sphere* mesh, which is necessary if we adopt quads element and we want to avoid singularities at the poles. A python script with Gmsh API is provided in the repository for this purpose.

5.2.3 User specifications: parameter file

The user interface is a file called `parameters.prm`. This file allow the user to define the problem specifications, which are passed at runtime to the program.
The possible specifications are the following:

- **Geometry**: the user can specify the mesh filenames that will be used to build the triangulation. If multiple filenames are provided, the solver will perform multiple simulations (over the different triangulations) and then return a convergence table
- **Differential problem**: the user can specify if the problem is *interior* or *exterior* and can specify the Green functions of the differential problem of choice. In this project we study Laplace and screened Poisson problems, but, with the implementation of a suitable quadrature rule for singular integrals, other Green functions may be chosen.
- **Exact solution**: the user can specify the boundary value of the analytical solution for both ϕ and ϕ_n for convergence analysis
- **Quadrature rules**: the user can specify the quadrature rule of choice for both singular and non-singular integrals, choosing from deal.II wide variety of options, as well as the quadrature order
- **Solver options**: the user can specify GMRES settings such as tolerance and maximum iterations
- **Boundary conditions**: the user can specify the boundary conditions for the problem choosing between Dirichlet, Neumann, Robin or mixed

Notice that to define an analytical function (for the exact solution, the Green functions and boundary data) in the `parameters.prm` file the user must follow a specific syntax, which is provided in the deal.II documentation in (4).

5.2.4 Run

Once the mesh is created and the parameter file suitably modified, the user is ready to launch the simulation with the following command from the terminal:

```
cmake .
make run
```

Follow the instructions in Section (7) to run a test case.

5.2.5 Build system with CMake

The solver keeps the deal.II tutorial layout: a concise `CMakeLists.txt` locates an installed deal.II (`find_package(deal.II 9.6 REQUIRED)`), checks that muParser support is enabled, and then creates one single target containing `step-34-Bonvini.cc`.
During the configuration step

```
$ cmake .
```

queries `deal.IIConfig.cmake`, propagates the correct compiler flags, MPI settings, and link libraries, and generates the platform-specific `Makefile`. The tutorial template also adds a `run` target, so compiling and executing in the chosen build-type (Debug by default) reduces to

```
$ make run
```

All optional features (e. g. `-DCMAKE_BUILD_TYPE=Release` or static linking) can still be passed to the initial `cmake` command, but the two-line workflow above is sufficient for every system supported by deal.II.

5.3 Code architecture

The code follows an object-oriented structure centered around the **BEMProblem** class template. The templated design - typical of the deal.ii library scripts - allow for compile time specialization into 2D or 3D problems. Even if the project is focused on 3D problems, the templatization of the class is maintained to allow for any further implementations.
The **BEMProblem** class encapsulates data (mesh, dof, matrices, vectors) and operational methods, making the code maintainable and organised. Each method of the class corresponds to a

logical task (read input, assemble, solve, output), facilitating both reading and possible modification or extension.
The methods, following the workflow order, are here briefly described:

read_parameters() This method reads the user's `parameters.prm` file and populates all solver options—geometry filenames, Green functions, boundary flags and data, quadrature choices, and solver tolerances—into class members. Internally, it uses deal.II's `ParameterHandler`, which maintains a map from key names to `std::string` values. Each `declare_entry()` or `enter_subsection()` call builds the expected key tree, and `parse_input()` reads and validates the file (with the user-supplied data) against the declared patterns.
Constant values are retrieved via `get_double()`, `get_bool()`, etc., while analytic boundary values, exact solution and Green functions use `Functions::ParsedFunction<dim>`. The latter's `declare_parameters()` registers three string fields ("Variable names", "Function constants", "Function expression"), and `parse_parameters()` hands them to muParser to compile into a fast internal AST (abstract-syntax-tree). Because all string-to-AST conversion happens exactly once in `parse_parameters()`, and every subsequent `value()` call is a straight numeric tree-evaluate, this design cleanly separates file I/O, parsing, and fast runtime evaluation. No further dependencies beyond **muParser** are needed, and every `ParsedFunction` holds its own independent AST and variable map. This design centralizes all user-configurable options in one place, ensures type-safe validation, and keeps run-time evaluations of parsed expressions efficient.

read_mesh() This method is responsible for accessing the mesh file (defined with a `.msh` file for example) and loading it onto a `Triangulation<dim-1, dim>` structure of deal.II.

init_structures() Once the triangulation dimension is defined, this method is called to initialize all the structures necessary for the BEM calculations: `system_matrix`, `system_rhs`, matrix `H`, matrix `G` and vectors `phi`, `phi_n`, `alpha`, and the linear system solution vector `ls_solution`. Boundary Finite Elements are also associated to the triangulation at this stage, and degrees of freedom are distributed to the collocation points.

set_boundary_flags() Evaluates the type of boundary condition for each support point (degree of freedom on the boundary) using a function parsed from the input file. Based on the value returned (0 = Neumann, 1 = Dirichlet, 2 = Robin), the code activates boolean flags, which are stored in three `std::vector<bool>(n_dof)` named `assign_dirichlet`, `assign_neumann` and `assign_robin`. The method also populates the vectors $\boldsymbol{\phi}$ and $\boldsymbol{\phi_n}$ where known values - coming from boundary conditions - are available. This approach with flag vectors for boundary conditions makes it possible to uniformly handle mixtures of different conditions and to identify knowns and unknowns of the system.

assemble_system() This function contains the core of the BEM calculations and is responsible for the assembly of the matrices **H** and **G**.
The first thing that the method does is to create an object of the deal.II class `FEValues<dim - 1, dim>`, which will handle the finite element evaluations and the quadrature integrations.
After this, an outer loop over all the *elements* is started. Now we can define the finite elements for the given element as well as its quadrature points and normal direction vectors for each quadrature point (that in our program will be the same for all quadrature points since we adopt only 1st order elements). It is important to note that, due to the counter-clockwise mesh orientation, the normals to the surface calculated at this stage will always be pointing in the outward direction with respect to the mesh interior. For this reason, if the problem that needs to be solved is an exterior problem, the normals are flipped in the other direction to comply with the Green identity.
Once this is done, a secondary loop over all the points (DOFs) of the triangulation is started. Inside this loop, a distinction between singular and non-singular integrals is made. For non-singular integrals (i.e. when the given point lies outside the given element) the standard quadrature formula (see Section (4)) is adopted.
A local loop over the quadrature points of the given element is started and the integral in (65) is resolved. We note that both the quadrature weights and the jacobian of the transformation, as well as the finite element values, are handled by an object of the class `FEValues<dim - 1, dim>` with the methods `shape_value(j,q)` and `JxW(q)`. The single and double layer operators are instead retrived from the `Kernel namespace`.
Special care is needed, instead, when dealing with singular integrals (i.e. when the given point lies

inside the given element). For these cases, a special quadrature formula (explained in Subsection (4.1) is applied by means of the `get_singular_quadrature()` method of the class `Quadrature<dim - 1>`.
Once the integrals are evaluated the elements of the matrices **H** and **G** are populated.
Now the elements of **G** are computed. The elements of **H**, instead, need an additional term that comes from $\alpha(\mathbf{x}_i)\psi(\mathbf{x}_i)$, as shown in (30). Since $\phi_j(\mathbf{x}_i) = \delta_{ij}$ the corresponding matrix is a diagonal one with entries equal to $\alpha(\mathbf{x}_i)$. The solid angles $\alpha(\mathbf{x}_i)$, as described in Equation (23), are calculated multiplying the matrix **H** with a vector of elements all equal to -1. Then, the result is added back to **H** to yield its final form.

recombine_matrices() This method contains the core difference between a mixed problem and one with fully Dirichlet or fully Neumann boundary conditions. For a fully Dirichlet problem, for example, the linear system $\mathbf{Ax} = \mathbf{b}$ that needs to be solved is easily assembled with $\mathbf{A} = \mathbf{G}$ and $\mathbf{b} = \mathbf{H}\boldsymbol{\phi}$, analogously for a fully Neumann problem we have $\mathbf{A} = \mathbf{H}$ and $\mathbf{b} = \mathbf{G}\boldsymbol{\phi_n}$. Special care, instead, is needed to assemble the system matrix and right hand side for a mixed boundary problem. This method is responsible for recombining the matrices, as shown in Equation (38). When dealing with a mixed Dirichlet-Neumann-Robin problem, a special treatment is needed and we explain it in Appendix (A).

solve_system() This method is straightforward as it consists of just few lines where an object of the class `SolverGMRES` is created and its method `solve(system_matrix, ls_solution, system_rhs, PreconditionIdentity())` is adopted to find the solution of the linear system. Note that the configurations for convergence and max iterations of the GMRES are defined in the input parameter file. Note that for simplicity, we chose to not use any preconditioner, hence the parameter `PreconditionIdentity()` is passed. This is because for our test cases the GMRES was already able to converge at 10^{-10} within less then 10 iterations. For larger problems ($10 - 20.000$ dofs) it is possible to add any preconditioner with just one line of code. Additionally, when the problem has boundary conditions of just one type (eg. fully Dirichlet) then the system matrix becomes symmetric and CG can be adopted to further speed up the computations. It is important to note that the critical bottleneck of a BEM implementation is not the solution of the linear system but the assembly phase, where expensive integrals are evaluated in two nested loops.

retrieve_solution() For a general mixed problem, the solution of the linear system found with `solve_system()` contains the solution for both $\boldsymbol{\phi}$ and $\boldsymbol{\phi_n}$. The method `retrieve_solution()` assigns the correct values of the linear system solution to $\boldsymbol{\phi}$ and $\boldsymbol{\phi_n}$, using the boundary flags `assign_neumann`, `assign_dirichlet` and `assign_robin`.

find_mesh_size() This intermediate step is adopted to find the size of the given mesh, defined as the mean dimension of all the cells, where the dimension of a single cell is defined as its maximum diagonal. The mesh size is saved for convergence analysis.

compute_errors(cycle) This method is responsible for evaluating the L_2 and L_∞ errors of the solution against the analytical exact solution, if provided. The L_2 error is computed with the `compute_global_error()` method of the deal.II's `VectorTools` namespace, while the L_∞ error is computed by finding the maximum absolute difference between the numerical and analytical solution in all the triangulation points.
Furthermore, we estimate the error on the jump distribution α (which should be 0.5 for smooth surfaces) by simply calculating the deviation of α from 0.5 in norm L^∞. All these indicators are added to a `convergence_table`, together with the mesh data (number of elements, degrees of freedom, mesh size), so that the convergence rates can be derived in post-processing.

compute_exterior_solution() If the exterior solution is required - passed as a flag by the user in the `parameters.prm` file - then this method is called to compute the exterior/interior solution, as explained in Subsection (2.1).

output_results(cycle) This method adopts the deal.II class `DataOut` to store the simulation results into output files. The solution for $\boldsymbol{\phi}$ and $\boldsymbol{\phi_n}$, with their analytical counterpart (if provided), $\boldsymbol{\alpha}$ and the boundary conditions are saved in a `.vtk` file easily accessible with Paraview. The convergence table is also saved and provided in a `.txt` file.

release_memory() This method is responsible for deallocating the memory used by all the previously adopted structures (matrices and vectors) to avoid undesired overlap between consecutive simulations.

run() This is the last method of the template class `BEMProblem<dim>` and is responsible for calling all the other methods, in the order that they are here presented. This method is also responsible for looping over different simulations if multiple mesh filenames are provided by the user that wants to perform a convergence study.

main() At this stage, the `main()` function is extremely slim as it only needs to:

1. Parse the input parameter filename: `parameters.prm`;
2. Create an object of the class `BEMProblem`;
3. Call the `run()` method to start the simulation;

5.3.1 Quadrature rule sulection

In the original step-34 tutorial, non-singular boundary integrals could already be handled by deal.II's built-in `QuadratureSelector`, which reads a user–specified quadrature name from the parameter file and returns the corresponding `Quadrature<dim>` object for regular integrals. The quadrature rules provided by deal.II with `QuadratureSelector` are specified at (5).

However, singular integrals require specialized rules—such as a 1/R-weighted formula for 3D Laplace or Yukawa potentials—that are not provided by `QuadratureSelector`. To address this, we implement a custom factory within the member function
`BEMProblem<dim>::get_singular_quadrature(...)`:

- We define a builder type

```
using SQBuilder = std::function<Quadrature<dim-1>
    (unsigned, const Point<dim-1>&, double, bool)>;
```

  and declare a *static* map `builder_map` from *string identifiers* (parsed in `parameters.prm` as `singular_quadrature_type`) to these builder lambdas.

- Each entry constructs the appropriate singular quadrature rule for 2D surface cells:
  - `"one_over_r"` ↦ `QGaussOneOverR<dim-1>(order, pt, sym)`
  - `"telles"` ↦ `QTelles<dim-1>(order, pt)`
  - `"duffy"` ↦ `QDuffy(order, 1.0)`
  - `"triangle_polar"` ↦ `QTrianglePolar(order)`
- We compute a global `scale` = 1/`cell->measure()` (which might be necessary for some quadrature rules) and obtain the finite-element support points via `fe.get_unit_support_points()`.
- On first invocation, we invoke the chosen builder for each support point (passing the order `singular_quadrature_order`, the point, `scale`, and a symmetry flag) and cache the resulting `Quadrature<dim-1>` objects in a *static* vector.
- Subsequent calls simply return the prebuilt quadrature at the requested `index`.

In particular, the `"one_over_r"` rule is the natural choice for the 3D Laplace/Yukawa Green's function, but any other deal.II quadrature (including newly implemented ones) can be registered in `builder_map` and selected at runtime by its string name.

5.4 Python helper files

The repository contains many python files that were used to work on this project. The majority of these files were used to create the desired geometries and use the Gmsh API. They are all designed to create meshes that respect the requirements in Subsection (5.2.1).
Another important python file that you will find in the repository is the `check_normals.py`. To use this you need to run the following command from the terminal:
`python check_normals.py mesh_filename output_normals.vtk`
This will provide the user with the `output_normals.vtk` file that will contain a collection of points found in the following way:

1. Find the centers of each element of the mesh;
2. Calculate the normals to each element as the normalized cross product between the edges `p1 - p0` and `p3 - p0`, where p_0, p_1, p_3 are 3 of the 4 vertices of the element, rigorously in the order in which the triangulation is described;
3. Find the collection of points given by $c_i + \mathbf{n}_i$, where c_i are the centers and $\mathbf{n}_i$ are the normals;

Now this collection of points will be saved to view in a `.vtk` file. Now the user should open the `output_normals.vtk` along with the mesh and check that the collection of points all lie outside the geometry, as shown in Figure (2). If every point lies outside the given geometry then the triangulation correctly describes a counter-clockwise orientation.
This step is necessary because a wrongful orientation of the mesh surface will result in a different sign for the normals at the assembly phase of the matrices $\mathbf{H}$ and $\mathbf{G}$.

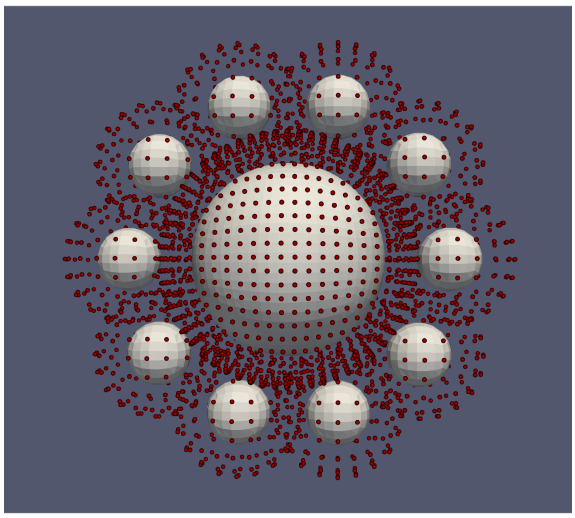

Figure 2: Check normals on multi-spheres mesh

6 Benchmark problems

In order to evaluate the program and check its accuracy some benchmark problems are proposed and convergence results are compared with other implementations found in literature. Convergence results are then reported, where we define:

$$E_{2,\phi} = \frac{\| \phi_h - \phi \|_{L^2(\Gamma)}}{\| \phi \|_{L^2(\Gamma)}}, \tag{68}$$

$$E_{\infty,\phi} = \frac{\| \phi_h - \phi \|_{L^\infty(\Gamma)}}{\| \phi \|_{L^\infty(\Gamma)}}, \tag{69}$$

$$E_{2,\partial_n\phi} = \frac{\| \partial_n\phi_h - \partial_n\phi \|_{L^2(\Gamma)}}{\| \partial_n\phi \|_{L^2(\Gamma)}}, \tag{70}$$

$$E_{\infty,\partial_n\phi} = \frac{\| \partial_n\phi_h - \partial_n\phi \|_{L^\infty(\Gamma)}}{\| \partial_n\phi \|_{L^\infty(\Gamma)}}, \tag{71}$$

$$E_{\infty,\alpha} = \| \alpha_h - \tfrac{1}{2} \|_{L^\infty(\Gamma)}. \tag{72}$$

with the only note that the L_∞ norm is evaluated only on the collocation nodes, which is correct as long as we adopt linear shape functions, since the maximum of the function will be found at the nodal points. This no longer holds for higher order elements where the maximum could be found at interior points. In those cases, an over-sampling of the element area would be required.

6.1 Test case 1: Interior fully Dirichlet on L-shaped domain

The interior fully Dirichlet Laplace problem represents the behaviour of a scalar field - temperature, electric potential, pollutant concentration - inside a domain with no sources and imposed values at the border.

6.1.1 Problem formulation

Consider the interior Laplace problem with fully Dirichlet boundary conditions on an L-shaped domain $\Omega = \Omega_1 \cup \Omega_2$, with $\Omega_1 = \{(x, y, z) \in \mathbb{R}^3 : 0 < x, y < 2, 0 < z < 6\}$ and $\Omega_2 = \{(x, y, z) \in \mathbb{R}^3 : 0 < x, z < 2, 0 < y < 4\}$, as shown in Figure (3).

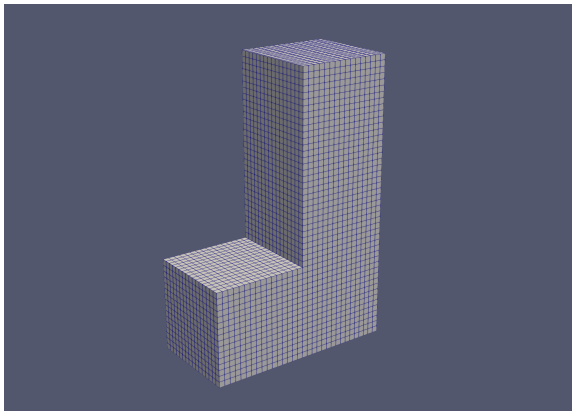

Figure 3: L-shaped domain

The problem formulation is the following:

$$\begin{cases} \Delta\phi = 0 \quad \mathbf{x} \in \Omega \\ \phi(\mathbf{x}) = e^x \sin(z) + e^z \cos(y) \quad \mathbf{x} \in \partial\Omega \end{cases} \tag{73}$$

The numerical solution for the normal flux ϕ_n on the boundary of this problem can be seen in Figure (4).

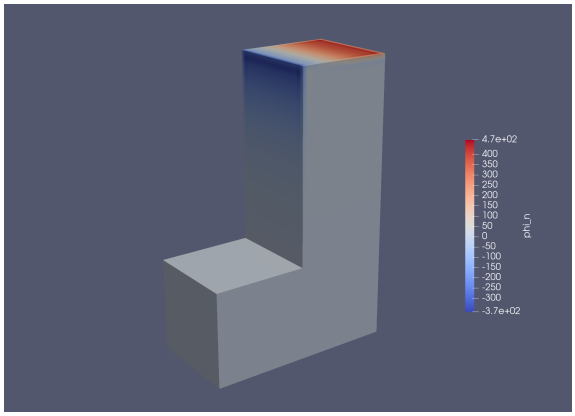

Figure 4: L-shaped: ϕ_n on $\partial\Omega$

6.1.2 Convergence results

We can easily find the analytical form of ϕ_n on the boundary $\partial\Omega$ by taking the gradient of the Dirichlet function $e^x \sin(z) + e^z \cos(y)$ and making the scalar product with the normal direction to the surface $\mathbf{n}(\mathbf{x})$. Some care is needed only to be sure that the normals are pointing outward - since this is an *interior* problem.

$$\frac{\partial \phi}{\partial n}(x,y,z) = \begin{cases} -\sin z, & x = 0, \\ e^2 \sin z, & x = 2, \\ 0, & y = 0, \\ -e^z \sin 2, & y = 2 \text{ and } z \geq 2, \\ -e^z \sin 4, & y = 4, \\ -\big(e^x + \cos y\big), & z = 0, \\ e^x \cos 6 + e^6 \cos y, & z = 6, \\ e^x \cos 2 + e^2 \cos y, & z = 2 \text{ and } y \geq 2, \\ 0, & \text{otherwise.} \end{cases} \tag{74}$$

Note that z is the axis along the *longer leg* of the domain, y along the *shorter leg* and x spans the depth of the L.

Table I: Convergence table for ϕ_n

N_{dof}	L_2-error(ϕ_n)	β
650	$3,109 \cdot 10^{-01}$	-
1 154	$2,699 \cdot 10^{-01}$	0.49
2 291	$2,247 \cdot 10^{-01}$	0.53
4 610	$1,922 \cdot 10^{-01}$	0.44
10 368	$1,572 \cdot 10^{-01}$	0.49

With two error estimates E_1 and E_2 between two consecutive discretization of $\partial\Omega$, one can also compute an approximate rate of convergence $\beta = 2\frac{\ln(E_2/E_1)}{\ln(N_1/N_2)}$, with N_1 and N_2 being the number of collocation points.
The results show slow but steady convergence and are comparable with results presented in (8) at page 45, and also report in Table (II)

Table II: Convergence table for ϕ_n (S. Nintcheu Fata ((8)))

N_{dof}	L_2-error(ϕ_n)	β
578	$1,413 \cdot 10^{-01}$	-
1 236	$1,011 \cdot 10^{-01}$	0,881
2 494	$8,199 \cdot 10^{-02}$	0,597
5 642	$6,304 \cdot 10^{-02}$	0,644
7 426	$5,586 \cdot 10^{-02}$	0,880
11 242	$4,959 \cdot 10^{-02}$	0,574
16 144	$4,584 \cdot 10^{-02}$	0,435

6.2 Test case 2: Exterior fully Dirichlet on multi-spheres domain

We now cover the problem of an electrostatic field outside 11 perfectly conducting spheres.

6.2.1 Problem formulation

Consider the exterior Laplace problem with fully Dirichlet boundary conditions on the following domain: $\Omega = \Omega_1 \cup \Omega_2$ with $\Omega_1 = \{\mathbf{x} \in \mathbb{R}^3 : |\mathbf{x}| < 3\}$ and Ω_2 being the space covered by 10 identical spheres of radius 1 evenly distributed on a circe of radius 5 centered in the center of Ω_1, as shown in Figure (5).

The formulation is the following:

$$\begin{cases} \Delta\phi = 0 \quad \mathbf{x} \in \Omega \\ \phi(\mathbf{x}) = 5 \quad \mathbf{x} \in \partial\Omega_{Red} \\ \phi(\mathbf{x}) = -5 \quad \mathbf{x} \in \partial\Omega_{Blue} \end{cases} \tag{75}$$

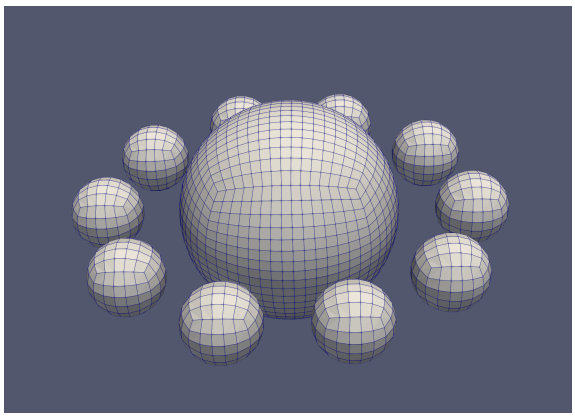

Figure 5: Multi-spheres domain

where we define Ω_{Red} as the union between the large sphere (Ω_1) with 5 alternating small spheres and Ω_{Blue} as the remaining 5 spheres, as shown in Figure (6).

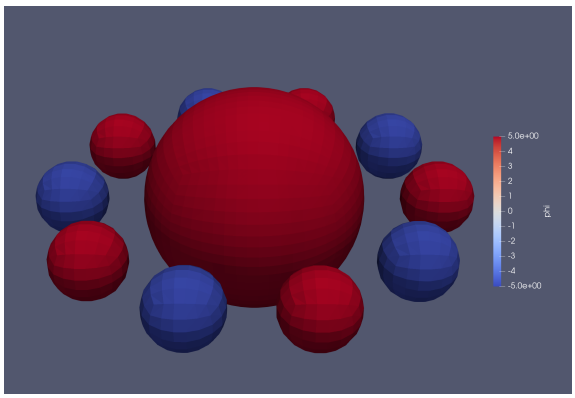

Figure 6: Multi-spheres Dirichlet BC

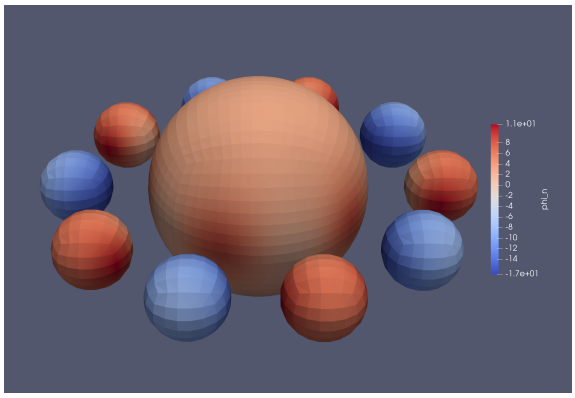

Figure 7: Multi-spheres: ϕ_n on $\partial\Omega$

6.2.2 Convergence results

For the exterior problem on the multi-spheres domain we cannot find an analytical solution for the ϕ_n on the boundary, but we can still perform a convergence test - as proposed by (9) at page 79, 80 - by looking at the minimum and maximum charge densities (i.e. ϕ_n) on the spheres.

The results are in perfect agreement with the ones provided by (9) at page 80, that we report also in Table IV.

Table III: Charge densities on the spheres

N_{dof}	$\min(\phi_n)$	$\max(\phi_n)$
646	$-19,1868$	$13,1873$
2 850	$-16,8488$	$11,4937$
11 422	$-16,8294$	$11,3976$

Table IV: Charge densities on the spheres (Yijun Liu (9))

N_{dof}	$\min(\phi_n)$	$\max(\phi_n)$
8 448	$-16,4905$	$11,1837$
13 200	$-16,5363$	$11,2218$
19 008	$-16,6322$	$11,2558$
25 872	$-16,6436$	$11,2746$
33 792	$-16,6733$	$11,3792$
42 768	$-16,6648$	$11,3810$
52 800	$-16,7435$	$11,3787$
82 500	$-16,7068$	$11,2964$
118 800	$-17,1157$	$12,6511$

6.3 Test case 3: Exterior screened Poisson with Robin boundary conditions on a sphere

The problem is to find the electric potential outside of a spherical cavity inside a conducting medium when a point charge is placed in the middle of the cavity. As explained in Subsection (3.1.1), the initial coupled interior Laplace-exterior screened Poisson is transformed into an only exterior screened Poisson problem where the coupling is held with Robin boundary conditions.

6.3.1 Problem formulation

Consider the exterior screened Poisson problem on a sphere of radius $a = 1$ with screening parameter $\kappa = 1.5$ and Robin boundary conditions:

$$\begin{cases} \left(\nabla^2 - \kappa^2\right)\phi(\mathbf{r}) = 0, \qquad r > a, \qquad \phi(\mathbf{r}) \xrightarrow{r\to\infty} 0 \\ \partial_r\phi(a,\theta) + \beta_n(u)\,\phi(a,\theta) = g_n(\theta), \end{cases} \tag{76}$$

with the following definitions from Subsection (3.1.1):

$$\beta_n(u) = -\frac{\varepsilon_i}{\varepsilon_o}\frac{n}{a} \tag{77}$$

$$g_n(\theta) = -\frac{\varepsilon_i}{\varepsilon_o}\,\frac{(2n+1)D_n}{a^{\,n+2}}\,P_n(\cos\theta) \tag{78}$$

If we consider the easiest case where 1 point-like charge with $q = 1$ is placed in the middle of the sphere then only the monopole terms of β_n and g_n will survive (i.e. for $n = 0$). We solve for a problem with $\varepsilon_i = \varepsilon_o$ and we use for the monopole the term $D_0 = e^{-\kappa a}$ so that

$$\beta_n(\kappa) = \beta_0(\kappa) = \kappa, \qquad g_n = g_0 = -\frac{e^{-\kappa}}{a^2} = -e^{-\kappa}, \tag{79}$$

and the Robin condition becomes

$$\boxed{\partial_r\phi + \kappa\,\phi = -e^{-\kappa}} \tag{80}$$

6.3.2 Convergence results

The exact solution for the potential and the normal flux on the surface of the sphere for this problem is

$$\phi = \frac{e^{-\kappa a}}{a}, \qquad \phi_n = -(\kappa a + 1)\frac{e^{-\kappa a}}{a^2} \tag{81}$$

The convergence results are shown in the table and in the convergence plot below.

The results show very good converge for the given problem, with both L_2 and L_∞ errors showing an order of 2 for both ϕ and ϕ_n.

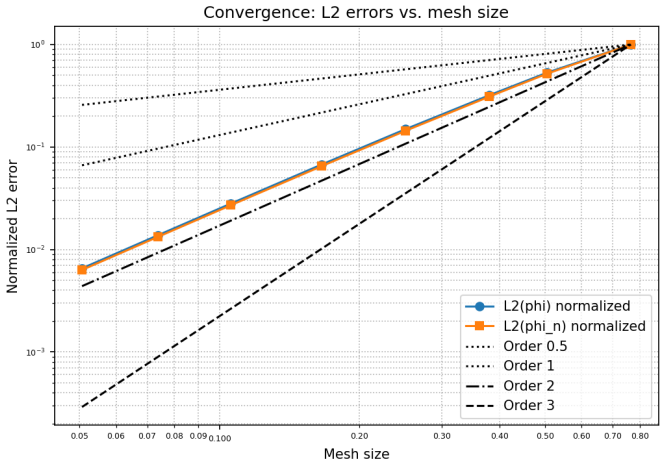

Figure 8: L_2 convergence plot

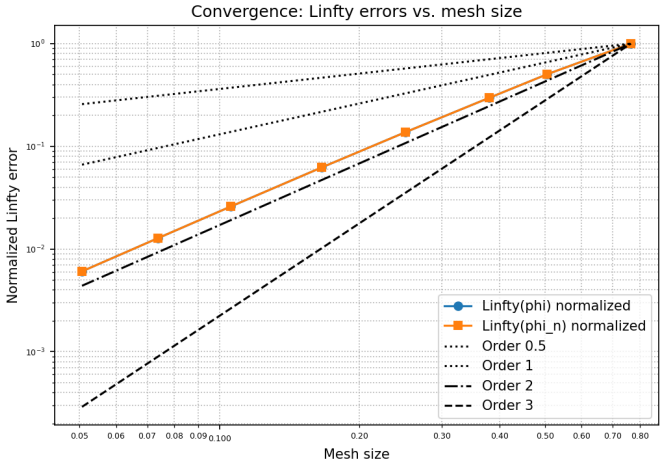

Figure 9: L_∞ convergence plot

Table V: Convergence table

N_{cells}	N_{dof}	mesh size	L_2-error(ϕ)	L_∞-error(ϕ)	L_2-error(ϕ_n)	L_∞-error(ϕ_n)
24	26	0.7654	$4.104e-01$	$2.721e-01$	$3.542e-01$	$1.633e-01$
54	56	0.5061	$2.184e-01$	$1.370e-01$	$1.841e-01$	$8.219e-02$
96	98	0.3802	$1.314e-01$	$8.083e-02$	$1.099e-01$	$4.850e-02$
216	218	0.2510	$6.119e-02$	$3.735e-02$	$5.094e-02$	$2.241e-02$
486	488	0.1659	$2.770e-02$	$1.695e-02$	$2.303e-02$	$1.017e-02$
1176	1178	0.1059	$1.152e-02$	$7.068e-03$	$9.581e-03$	$4.241e-03$
2400	2402	0.0738	$5.651e-03$	$3.469e-03$	$4.701e-03$	$2.082e-03$
5046	5048	0.0507	$2.687e-03$	$1.650e-03$	$2.236e-03$	$9.897e-04$

7 Instructions to run a Test case

In order to run a test case the user should follow the instructions below:

1. Clone the repository
2. Enter the folder of the desired benchmark:
   (a) `L_shaped_benchmark` for Test case 1
   (b) `multi_spheres_benchmark` for Test case 2
   (c) `screened_Poisson_sphere_benchmark` for Test case 3

3. Copy and paste the mesh files (`.msh`) and the `parameters.prm` file found in the benchmark folder in the main folder of the repository (`step-34-Bonvini`)

4. From terminal, run the following commands:

   - `cmake .`
   - `make run`

The user will see if the simulation is running correctly from the log prints in the terminal. Once the simulation is over, the user will find in the `step-34-Bonvini` folder the following files:

- `3d_boundary_solution_n.vtk` where n will be the index of the solution number (since multiple mesh files are provided. One can access the vtk file in Paraview and see the values of:

  1. Exact solution on the boundary for ϕ and ϕ_n (provided by the parameter file)
  2. Found solution on the boundary for ϕ and ϕ_n
  3. Value of $\alpha(\mathbf{x})$ on the boundary
  4. Boundary conditions type (1 for Dirichlet and 0 for Neumann) on the surface

- `convergence_table.txt`, which appears also in the terminal at the end of the simulation

Note for convergence of Test case 2 To check the convergence in Test case 2 a python script `minMaxChargesPlanetary.py` is provided in the repository. The user should edit the file to specify the desired .vtk file that contains the solution and then run it to find the min and max values of the charges on the surface of the spheres.
The expected results for the three test cases are presented in the convergence subsections of Section (6).

8 Limitations and future developments

The limitations of this program are mainly two. First of all, since no manifold description of the domain is provided, the order of the elements is the one defined with the mesh. If a precise geometrical description of the domain was available then it would be possible to exploit deal.II higher order mapping capabilities and significantly reduce the error due to the spatial approximation. The second relevant limitation is that the program is able to solve both interior and exterior problem, but not a coupled interior-exterior one, unless one of the two is resolved analytically (eg. screened Poisson with Robin boundary conditions). A coupled solver would require the solution to fulfill continuity conditions for the potential and the normal flux at the border, but would greatly enlarge the sphere of applications of the program. Some improvements are also possible in terms of efficiency. To this matter, parallelization could be adopted to speed up the double loop over cells and DOFs at the assembly phase, since each row of the matrices is independent to each other. Note that, since the great bottleneck of a BEM implementation is the memory requirement, to be scalable the parallelization should only be considered with a shared memory approach. Another efficiency improvement, especially effective for large systems, would be to choose the linear solver based on the specific problem (eg. for a fully Dirichlet problem the matrix is SPD so the optimal choice would be CG).

9 Conclusions

The solver presented in this work handles **both Laplace and screened-Poisson equations** and supports **Dirichlet, Neumann, Robin, and mixed boundary conditions**. Moreover, enabling user selection of the Green's function and quadrature scheme confers greater generality and extensibility upon the program. Three benchmark problems confirm the accuracy of the solver, comparing it with other implementations found in literature. The implementation is intentionally **simple, modular, and easily extendable**: each phase (parsing, assembly, solve, post-processing) is encapsulated in a dedicated method, while all low-level tasks rely on the well-established deal.II library. These qualities make the program a clean starting point for further developments such as shared-memory parallelism, higher-order boundary elements or a solver for different differential problems.

A Mixed problem with Dirichlet-Neumann-Robin boundary conditions

At every collocation point $x_i \in \Gamma$ the interior Laplace limit of Green's identity is

$$\big(\alpha_i I - D\big)\,\phi(x_i) \;+\; S\,\phi_n(x_i) = 0, \tag{82}$$

with $\phi_n := \partial_n \phi$, and the discrete operators

$$D_{ij} = \int_\Gamma \partial_{n_y} G(x_i, y)\,\psi_j(y)\,dS_y, \qquad S_{ij} = \int_\Gamma G(x_i, y)\,\psi_j(y)\,dS_y. \tag{83}$$

Let $\mathcal{D}, \mathcal{N}, \mathcal{R}$ index Dirichlet, Neumann and Robin nodes. Data are $\phi = g_D$ on Γ_D, $\phi_n = g_N$ on Γ_N, and the Robin relation $\phi_n = g_R - \beta\phi$ on Γ_R. For ease of notation, call $q := \phi_n$ Collect the *unknowns* $x = [\, q_{\mathcal{D}},\, \phi_{\mathcal{N}},\, \phi_{\mathcal{R}}\,]^T$.

First block–row (collocation for $x_i \in \Gamma_D$): Equation (82) gives

$$\sum_{j\in\mathcal{D}} S_{ij}\,q_j - \sum_{j\in\mathcal{N}} D_{ij}\,\phi_j - \sum_{j\in\mathcal{R}} \big(D_{ij} - \beta_j S_{ij}\big)\phi_j \;=\; -\alpha_i g_D(i) + \sum_{j\in\mathcal{D}} D_{ij}\,g_D(j), \tag{84}$$

which directly supplies the first row of the matrix A and right–hand side b.

For $x_i \in \Gamma_N$ one sets $q_i = g_N(i)$ and keeps ϕ_i unknown; for $x_i \in \Gamma_R$ insert $q_i = g_R(i) - \beta_i \phi_i$. Analogue derivations for each boundary subset produce the other two block–rows in the same fashion as (84). The resulting block structure is therefore

$$\underbrace{\begin{bmatrix} S_{\mathcal{D}\mathcal{D}} & -D_{\mathcal{D}\mathcal{N}} & -D_{\mathcal{D}\mathcal{R}} + \beta S_{\mathcal{D}\mathcal{R}} \\ -S_{\mathcal{N}\mathcal{D}} & D_{\mathcal{N}\mathcal{N}} & D_{\mathcal{N}\mathcal{R}} - \beta S_{\mathcal{N}\mathcal{R}} \\ S_{\mathcal{R}\mathcal{D}} & -D_{\mathcal{R}\mathcal{N}} & -D_{\mathcal{R}\mathcal{R}} + \beta S_{\mathcal{R}\mathcal{R}} \end{bmatrix}}_{A} \begin{bmatrix} q_{\mathcal{D}} \\ \phi_{\mathcal{N}} \\ \phi_{\mathcal{R}} \end{bmatrix} = \underbrace{\begin{bmatrix} -(\alpha I - D)_{\mathcal{D}\mathcal{D}}\,g_D \\ S_{\mathcal{N}\mathcal{N}}\,g_N \\ -(\alpha I - D)_{\mathcal{R}\mathcal{D}}\,g_D + S_{\mathcal{R}\mathcal{R}}\,g_R \end{bmatrix}}_{b}, \tag{85}$$

where β denotes the diagonal matrix of Robin coefficients. Equation (85) is the dense linear system that the collocation BEM solves to recover the unknown Dirichlet flux $q_{\mathcal{D}}$, Neumann potential $\phi_{\mathcal{N}}$, and Robin potential $\phi_{\mathcal{R}}$; the remaining Robin flux follows from $q_{\mathcal{R}} = g_R - \beta\phi_{\mathcal{R}}$.

References

[1] Claeys, X. and contributors. *BemTool: A C++ Boundary-Element Library*, `https://github.com/xclaeys/BemTool`. 2025.

[2] Costabel, Martin. "Boundary Integral Operators on Lipschitz Domains: Elementary Results", *SIAM J. Math. Anal. 19(3): 613–626*, (1988).

[3] deal.II. URL: `https://dealii.org/current/doxygen/deal.II/step_34.html`.

[4] deal.II. URL: `https://dealii.org/current/doxygen/deal.II/classFunctionParser.html`.

[5] deal.II. URL: `https://dealii.org/current/doxygen/deal.II/quadrature__selector_8cc_source.html`.

[6] Dölz, J. *et al.*, "Bembel: The fast isogeometric boundary element C++ library for Laplace, Helmholtz, and electric wave equation", *Software X*, Vol. 11, (2020), p. 100476. ISSN: 2352-7110. DOI: `https://doi.org/10.1016/j.softx.2020.100476`.

[7] Duffy, M. G. "Quadrature over a pyramid or cube of integrands with a singularity at a vertex", *SIAM Journal on Numerical Analysis 19(6), 1260-1262*, (1982).

[8] Fata, S. Nintcheu. "Explicit expressions for 3D boundary integrals in potential theory", *Int. J. Numer. Meth. Engng 2009; 78:32–47*, (2008).

[9] Liu, Yijun. *Fast Multipole Boundary Element Method: Theory and Applications in Engineering*, Cambridge University Press, 2009.

[10] Sauter, S. A., Pechstein, B., and contributors. *BETL: Boundary Element Template Library*, `https://www.sam.math.ethz.ch/betl/`. 2025.

[11] Xu, Zhenli. "Mellin Transform and Image Charge Method for Dielectric Sphere in an Electrolyte", (2013).