ASC Report No. 44/2009

HILBERT – A MATLAB Implementation of Adaptive BEM

Markus Aurada, Michael Ebner, Samuel Ferraz-Leite, Petra Goldonits, Michael Karkulik, Markus Mayr, Dirk Praetorius

Institute for Analysis and Scientific Computing
Vienna University of Technology — TU Wien
www.asc.tuwien.ac.at ISBN 978-3-902627-02-5

ISBN 978-3-902627-02-5

HILBERT — A MATLAB IMPLEMENTATION OF ADAPTIVE BEM

MARKUS AURADA, MICHAEL EBNER, SAMUEL FERRAZ-LEITE, PETRA GOLDENITS, MICHAEL KARKULIK, MARKUS MAYR, AND DIRK PRAETORIUS

Abstract. The Matlab BEM library `HILBERT` allows the numerical solution of the 2D Laplace equation on some bounded Lipschitz domain with mixed boundary conditions by use of an adaptive Galerkin boundary element method (BEM). This paper provides a documentation of `HILBERT`. The reader will be introduced to the data structures of `HILBERT` and mesh-refinement strategies. We discuss our approach of solving the Dirichlet problem (Section 5), the Neumann problem (Section 6), and the mixed boundary value problem with Dirichlet and Neumann boundary conditions (Section 7). Besides a brief introduction to these problems, their equivalent integral formulations, and the corresponding BEM discretizations, we put an emphasis on possible strategies to steer an adaptive mesh-refining algorithm. In particular, various error estimators are discussed. Another notable feature is a complete and detailed description of our Matlab implementation which enhances the reader's understanding of how to use the `HILBERT` program package.

1. Introduction

The boundary element method is a discretization scheme for the numerical solution of elliptic differential equations. On an analytical level, the differential equation, stated in the domain, is reformulated in terms of a certain integral equation called *representation theorem* or *third Green's formula*. For the Laplace equation on some bounded Lipschitz domain $\Omega \subset \mathbb{R}^2$, each solution of

$$-\Delta u = f \quad \text{in } \Omega \tag{1.1}$$

can explicitly be written in the form

$$u(x) = \widetilde{N}f(x) + \widetilde{V}\phi(x) - \widetilde{K}g(x) \quad \text{for all } x \in \Omega, \tag{1.2}$$

where $\phi := \partial_n u$ is the normal derivative and $g := u|_\Gamma$ is the trace of u on $\Gamma := \partial\Omega$. The involved linear integral operators read

$$\widetilde{N}f(x) := -\frac{1}{2\pi}\int_\Omega \log|x-y|\, f(y)\, dy, \tag{1.3}$$

$$\widetilde{V}\phi(x) := -\frac{1}{2\pi}\int_\Gamma \log|x-y|\, \phi(y)\, d\Gamma(y), \tag{1.4}$$

$$\widetilde{K}g(x) := -\frac{1}{2\pi}\int_\Gamma \frac{(y-x)\cdot n_y}{|y-x|^2}\, g(y)\, d\Gamma(y), \tag{1.5}$$

where n_y denotes the outer unit vector of Ω at some point $y \in \Gamma$. Put differently, the solution u of (1.1) is known as soon as the Cauchy data $(\partial_n u, u|_\Gamma)$ are known on the entire boundary Γ.

If one considers the trace of u, the representation formula (1.2) becomes

$$g = u|_\Gamma = N_0 f + V\phi - (K - 1/2)g. \tag{1.6}$$

If one considers the normal derivative of u, the representation formula (1.2) becomes

$$\phi = \partial_n u = N_1 f + (K' + 1/2)\phi + Wg. \tag{1.7}$$

The two linear equations (1.6)–(1.7) are known as *Calderón system*. It involves six linear integral operators acting only on Γ: the simple-layer potential V, the double-layer potential

Date: **29.01.2010**, 20.11.2009.
`HILBERT` — `http://www.asc.tuwien.ac.at/abem/hilbert/`.

K with adjoint operator K', the hypersingular integral operator W, and the trace N_0 and the normal derivative N_1 of the Newton potential $\widetilde{N}$.

For the boundary element method, the Laplace equation with given boundary data is equivalently stated in terms of the Calderón system (1.6)–(1.7). This leads to a boundary integral equation formulated on Γ. This integral equation is solved numerically to obtain (approximations of) the missing Cauchy data. In a postprocessing step, the computed Cauchy data are then plugged into the representation formula (1.2) to obtain an approximation of the solution u of the differential equation.

Examples for this approach are given in the subsequent sections: In Section 5, we consider the Dirichlet problem, where $g = u|_\Gamma$ is known and where the unknown normal derivative $\phi = \partial_n u$ has to be computed. More precisely, we consider the weakly-singular integral formulation of

$$\begin{aligned} -\Delta u &= 0 \quad \text{in } \Omega, \\ u &= g \quad \text{on } \Gamma. \end{aligned} \tag{1.8}$$

In Section 6, we consider the Neumann problem, where the normal derivative ϕ is known and where the unknown trace g has to be computed. More precisely, we consider the hypersingular integral formulation of

$$\begin{aligned} -\Delta u &= 0 \quad \text{in } \Omega, \\ \partial_n u &= \phi \quad \text{on } \Gamma. \end{aligned} \tag{1.9}$$

Finally, in Section 7, we consider a mixed boundary value problem, where Γ is split into two disjoint parts Γ_D and Γ_N and where g is known only on $\Gamma_D \subset \Gamma$, whereas ϕ is known only on $\Gamma_N \subset \Gamma$. More precisely, we consider the so-called symmetric integral formulation of

$$\begin{aligned} -\Delta u &= 0 \quad \text{in } \Omega, \\ u &= g \quad \text{on } \Gamma_D, \\ \partial_n u &= \phi \quad \text{on } \Gamma_N. \end{aligned} \tag{1.10}$$

All of these integral formulations lead to first-kind integral equations with elliptic integral operators so that the Lax-Milgram lemma applies and provides existence and uniqueness of discrete solutions.

Whereas this documentation focusses on the implementation of the adaptive lowest-order BEM only, we refer to the literature for details on the analysis and the numerics of BEM: For instance, the analysis of boundary integral equations is completely presented in the monograph [Mc]. For the state of the art in numerical analysis, we refer to [SS]. Fast BEM is discussed in [RS]. These algorithms are, however, beyond the scope of Matlab and consequently beyond the scope of `HILBERT`. Finally, an introductory overview on the analysis of elliptic boundary integral equations and the boundary element method is best found in [S].

1.1. What is `HILBERT`. Throughout, $\Gamma = \partial\Omega$ is the piecewise affine boundary of a polygonal Lipschitz domain $\Omega \subset \mathbb{R}^2$. Sometimes, Γ is partitioned into finitely many relatively open and disjoint boundary pieces, e.g. in a Dirichlet boundary Γ_D and a Neumann boundary Γ_N, i.e., $\Gamma = \overline{\Gamma}_D \cup \overline{\Gamma}_N$ and $\Gamma_D \cap \Gamma_N = \emptyset$.

Let $\mathcal{E}_\ell = \{E_1, \ldots, E_N\}$ be a finite set of affine line segments $E_j \in \mathcal{E}_\ell$, i.e., there holds

$$E_j = [a_j, b_j] := \operatorname{conv}\{a_j, b_j\} \tag{1.11}$$

with certain $a_j, b_j \in \mathbb{R}^2$ with $a_j \neq b_j$. We say that $\mathcal{E}_\ell$ is a mesh (or triangulation) of Γ provided that $\Gamma = \bigcup_{j=1}^N E_j$ and $|E_j \cap E_k| = 0$ for all $E_j, E_k \in \mathcal{E}_\ell$ with $E_j \neq E_k$. If Γ is partitioned into Γ_D and Γ_N, we assume that this partition is resolved by $\mathcal{E}_\ell$, i.e., $E_j \in \mathcal{E}_\ell$ satisfies either $E_j \subseteq \overline{\Gamma}_D$ or $E_j \subseteq \overline{\Gamma}_N$. Finally, $\mathcal{K}_\ell = \{z_1, \ldots, z_N\}$ denotes the set of all nodes of the mesh $\mathcal{E}_\ell$, and we note that there holds $\#\mathcal{E}_\ell = \#\mathcal{K}_\ell$ for the closed boundary Γ.

`HILBERT` [AEFMGKP] is a Matlab library for the solution of (1.8)–(1.10) by use of h-adaptive lowest-order Galerkin BEM. In particular, missing Neumann data are approximated by an $\mathcal{E}_\ell$-piecewise constant function $\Phi_\ell \approx \phi$, and missing Dirichlet data are approximated by an $\mathcal{E}_\ell$-piecewise affine and continuous function $G_\ell \approx g$. Given an initial coarse mesh $\mathcal{E}_0$ of Γ, the adaptive loop generates a sequence of improved meshes $\mathcal{E}_\ell$ by iterative local mesh-refinement. Throughout, HILBERT uses the canonical bases, i.e., characteristic functions χ_j associated with elements $E_j \in \mathcal{E}_\ell$ to represent discrete fluxes Φ_ℓ and nodal hat functions ζ_k associated with nodes $z_k \in \mathcal{K}_\ell$ to represent discrete stresses G_ℓ.

1.2. Outline of Documentation. Section 2 recalls some analytical preliminaries like the definition of boundary integrals and the arclength derivative. Moreover, some notation is introduced which is used throughout the entire document. In Section 3, we give a concise overview on all functions and functionality provided by HILBERT. Section 4 discusses our implementation of the local mesh-refinement and the marking strategy used in the adaptive mesh-refining algorithms. The Dirichlet problem (1.8) and its numerical solution by use of HILBERT is discussed in Section 5, whereas Section 6 is concerned with the Neumann problem (1.9). Finally, Section 7 treats the use of HILBERT for the numerical solution of the mixed boundary value problem (1.10).

2. Preliminaries

2.1. Functions on the Boundary. With each element $E_j = [a_j, b_j] \in \mathcal{E}_\ell$, we associate the affine mapping

$$\gamma_j : [-1,1] \to E_j, \quad \gamma_j(s) = \frac{1}{2}\big(a_j + b_j + s(b_j - a_j)\big) \tag{2.1}$$

which maps the reference element $[-1,1] \subset \mathbb{R}$ bijectively onto E_j.

Let $\mathcal{P}^p(\mathcal{E}_\ell)$ be the space of all $\mathcal{E}_\ell$-piecewise polynomials of degree $p \in \mathbb{N}_0$ with respect to the arclength. By definition, this means that for all $f_\ell \in \mathcal{P}^p(\mathcal{E}_\ell)$ and all elements $E_j \in \mathcal{E}_\ell$, the function $f_\ell \circ \gamma_j : [-1,1] \to \mathbb{R}$ satisfies

$$f_\ell \circ \gamma_j \in \mathcal{P}^p[-1,1], \tag{2.2}$$

i.e., $f_\ell \circ \gamma_j$ is a usual polynomial of degree (at most) p. Note that functions $f_\ell \in \mathcal{P}^p(\mathcal{E}_\ell)$ are, in general, not continuous, but have jumps at the nodes of $\mathcal{E}_\ell$.

In particular, $\mathcal{P}^0(\mathcal{E}_\ell)$ is the space of all $\mathcal{E}_\ell$-piecewise constant functions. If $\chi_j \in \mathcal{P}^0(\mathcal{E}_\ell)$ denotes the characteristic function of $E_j \in \mathcal{E}_\ell$, the set $\{\chi_1, \dots, \chi_N\}$ is a basis of $\mathcal{P}^0(\mathcal{E}_\ell)$.

One particular example for a function in $\mathcal{P}^0(\mathcal{E}_\ell)$ is the local mesh-width $h_\ell \in \mathcal{P}^0(\mathcal{E}_\ell)$ which is defined $\mathcal{E}_\ell$-elementwise by

$$h_\ell|_E := \mathrm{length}(E) = |b - a| \quad \text{for all } E = [a,b] \in \mathcal{E}_\ell. \tag{2.3}$$

Let $\mathcal{S}^1(\mathcal{E}_\ell) := \mathcal{P}^1(\mathcal{E}_\ell) \cap C(\Gamma)$ denote the set of all continuous and (with respect to the arclength) $\mathcal{E}_\ell$-piecewise affine functions. For each node $z_j \in \mathcal{K}_\ell$ of $\mathcal{E}_\ell$, let $\zeta_j \in \mathcal{S}^1(\mathcal{E}_\ell)$ be the associated hat function, i.e., $\zeta_j(z_k) = \delta_{jk}$. Then, the set $\{\zeta_1, \dots, \zeta_N\}$ is a basis of $\mathcal{S}^1(\mathcal{E}_\ell)$.

In HILBERT, we only consider the lowest-order Galerkin BEM, and the spaces $\mathcal{P}^0(\mathcal{E}_\ell)$ and $\mathcal{S}^1(\mathcal{E}_\ell)$ will be of major interest.

2.2. Boundary Integrals. Let $I \subset \mathbb{R}$ be a compact interval in $\mathbb{R}$. For $E_j \in \mathcal{E}_\ell$, let $\pi_j : I \to E_j$ be a continuously differentiable and bijective mapping. For any function $f : E_j \to \mathbb{R}$, the boundary integral is then defined via

$$\int_{E_j} f \, d\Gamma = \int_{E_j} f(x) \, d\Gamma(x) := \int_I (f \circ \pi_j)(t) \, |\pi_j'(t)| \, dt. \tag{2.4}$$

One can prove that this definition is independent of the parametrization π_j. For the reference parametrization γ_j, there holds

$$\int_{E_j} f \, d\Gamma = \frac{\mathrm{length}(E_j)}{2} \int_{-1}^{1} f \circ \gamma_j \, dt,$$

where $\mathrm{length}(E_j) := |b_j - a_j|$ denotes the Euclidean length of $E_j = [a_j, b_j] \subset \mathbb{R}^2$. For the arclength parametrization

$$\beta_j : [0, \mathrm{length}(E_j)] \to E_j, \quad \beta_j(t) := a_j + \frac{t}{\mathrm{length}(E_j)}\,(b_j - a_j)$$

holds

$$\int_{E_j} f\, d\Gamma = \int_0^{\mathrm{length}(E_j)} f \circ \beta_j\, dt.$$

2.3. Arclength Derivative. Let $I \subset \mathbb{R}$ be a compact interval in $\mathbb{R}$. For $E_j \in \mathcal{E}_\ell$, let $\pi_j : I \to E_j$ be a continuously differentiable and bijective mapping with $|\pi_j'(t)| > 0$ for all $t \in I$. For any function $f : E_j \to \mathbb{R}$, the arclength derivative f' is then defined by

$$(f' \circ \pi_j)(t) = \frac{1}{|\pi_j'(t)|}\,(f \circ \pi_j)'(t) \quad \text{for all } x = \pi_j(t) \in E_j. \tag{2.5}$$

Again, one can show that this definition is independent of the chosen parametrization. For the reference parametrization γ_j, we obtain

$$(f' \circ \gamma_j)(t) = \frac{2}{\mathrm{length}(E_j)}\,(f \circ \gamma_j)'(t) \quad \text{for all } x = \gamma_j(t) \in E_j,$$

whereas the arclength parametrization β_j leads to

$$f' \circ \beta_j = (f \circ \beta_j)'.$$

3. Overview on Hilbert

3.1. Tree Structure and Installation. The BEM library `HILBERT` is contained in a zip-file `hilbert.zip`. Unzipping this zip-archive, you obtain the following tree structure

```
hilbert/
   examples/
      example01/
      example02/
      example03/
      example04/
      general/
   lib/
   source/
   visualization/
```

The root directory `hilbert/` contains this documentation `documentation.pdf` as well as the licence agreement `license.pdf`. Besides this, it contains installation files `Makefile`, `Makefile.m`, and `make.m` to build the integral operators from the UNIX shell or the Matlab command line.

The m-files of `HILBERT` are contained in the folder `lib/`, whereas the C-source codes of the integral operators are contained in `source/`. The folder `examples/` contains various examples and demo files which demonstrate the use of `HILBERT`. The discrete solutions obtained can be visualized via functions from the folder `visualization/`.

To install `HILBERT`, unpack the zip-archive, change to the root folder, start Matlab, and type `make` at the Matlab command line. In UNIX, you may type `make` in the shell and start Matlab afterwards. Both will create certain MEX-files in the folder `lib/`. You may then test the installation by running some example from the `examples/` folder.

3.2. Data Structure. The set of nodes $\mathcal{K}_\ell = \{z_1, \dots, z_N\}$ of the mesh $\mathcal{E}_\ell = \{E_1, \dots, E_N\}$ is represented by the $(N \times 2)$-array `coordinates`. The j-th row of `coordinates` stores the coordinates of the j-th node $z_j = (x_j, y_j) \in \mathbb{R}^2$ as

$$\texttt{coordinates}(j,:) = [\ x_j \ \ y_j\ ].$$

If Γ is not split into several parts, the mesh $\mathcal{E}_\ell$ is represented by the $(N \times 2)$-(formally integer) array `elements`. The i-th boundary element $E_i = [z_j, z_k]$ with nodes $z_j, z_k \in \mathcal{K}_\ell$ is stored as

$$\texttt{elements}(i,:) = [\ j \quad k\ ],$$

where the nodes are given in counterclockwise order, i.e., the parametrization of the boundary element $E_i \subset \Gamma$ is mathematically positive. Put differently, the outer normal vector $\mathbf{n}_i \in \mathbb{R}^2$ of Γ on a boundary element $E_i = [z_j, z_k]$ reads

$$\mathbf{n}_i = \frac{1}{|z_k - z_j|} \begin{pmatrix} y_k - y_j \\ x_j - x_k \end{pmatrix} \quad \text{with} \quad z_j = (x_j, y_j), z_k = (x_k, y_k).$$

If Γ is split into Dirichlet boundary Γ_D and Neumann boundary Γ_N, the mesh $\mathcal{E}_\ell$ is represented by the $(N_D \times 2)$-array `dirichlet` and the $(N_N \times 2)$-array `neumann` which describe the elements $E_j \subseteq \overline{\Gamma}_D$ and $E_k \subseteq \overline{\Gamma}_N$ as before. Then, there formally holds

```
elements = [dirichlet;neumann]
```

with $N = N_D + N_N$. The array `elements`, however, is not explicitly built or stored in this case.

3.3. Overview on Functions and Functionality. In this section, we list all functions provided by HILBERT, describe their input and output parameters, provide a short overview on their functionality, and give links to a detailed description within this documentation.

Throughout, let $\mathcal{E}_\ell = \{E_1, \dots, E_N\}$ be a given mesh of Γ with nodes $\mathcal{K}_\ell = \{z_1, \dots, z_N\}$ described in terms of `coordinates` and `elements`. Recall that χ_j denotes the characteristic function associated with $E_j \in \mathcal{E}_\ell$ and that ζ_k denotes the hat function associated with $z_k \in \mathcal{K}_\ell$.

3.3.1. General Functions. For marking elements in an adaptive mesh-refining strategy, we use the Dörfler marking introduced in [D]. This is realized in a generalized way by the function

```
[marked [,marked2,..]] = markElements(theta [,rho], indicator1 [,indicator2,..]);
```

see Section 4.1 for details.

For the local mesh-refinement of the boundary mesh, we realize an Algorithm from [AGP] which is proven to be optimal with respect to the number of generated elements. For a mesh $\mathcal{E}_\ell$ described in terms of `coordinates` and `elements` and a vector `marked` containing the indices of elements $E_j \in \mathcal{T}_\ell$ to be refined, the function call reads

```
[coord,elem,father2son] = refineBoundaryMesh(coordinates,elements,marked);
```

where the generated mesh $\mathcal{E}_{\ell+1}$ is described by `coord` and `elem`. Moreover, `father2son` returns a link between the meshes $\mathcal{E}_{\ell+1}$ and $\mathcal{E}_\ell$. Further optional arguments of the function are discussed in Section 4.2.

3.3.2. Discrete Integral Operators. The discrete simple-layer potential matrix

$$\mathbf{V} \in \mathbb{R}^{N \times N}_{sym}, \quad \mathbf{V}_{jk} = \langle V\chi_k\,,\,\chi_j \rangle_{L^2(\Gamma)} \tag{3.1}$$

is returned by call of

```
V = buildV(coordinates,elements [,eta]);
```

Note that $\mathbf{V}$ is a dense matrix. Since MATLAB does not easily allow matrix compression techniques like hierarchical matrices [H] or the fast multipole method, the assembly of $\mathbf{V}$ (and the other discrete integral operators below) as well as the storage is of quadratic complexity. To lower the computational time in MATLAB, the computation is done in C via the MATLAB-MEX interface. We stress that all matrix entries can be computed analytically by use of analytic anti-derivatives derived, e.g., in [M]. If numerical stability is concerned, it is, however, better to compute certain entries by use of numerical quadrature instead, see [Ma]. The optional admissibility parameter `eta` determines whether certain entries $\mathbf{V}_{jk}$ are computed by numerical quadrature instead of analytic integration. Details are found in Section 5.1.

The discrete double-layer potential matrix

$$\mathbf{K} \in \mathbb{R}^{N\times N}, \quad \mathbf{K}_{jk} = \langle K\zeta_k \,,\, \chi_j\rangle_{L^2(\Gamma)} \tag{3.2}$$

is returned by call of

```
K = buildK(coordinates,elements [,eta]);
```

see Section 5.1. The discrete hypersingular integral operator matrix

$$\mathbf{W} \in \mathbb{R}^{N\times N}_{sym}, \quad \mathbf{W}_{jk} = \langle W\zeta_k \,,\, \zeta_j\rangle_{L^2(\Gamma)} \tag{3.3}$$

is returned by call of

```
W = buildW(coordinates,elements [,eta]);
```

see Section 6.1. As above, the optional parameter `eta` in both functions determines whether all matrix entries are computed analytically via anti-derivatives from [M] or whether certain entries are computed by numerical quadrature.

3.3.3. Numerical Solution of Dirichlet Problem.

The Laplace problem with Dirichlet boundary condition (1.8) is equivalently recast in Symm's integral equation

$$V\phi = (K + 1/2)g$$

with g being the known Dirichlet data and ϕ being the unknown Neumann data. We refer to Section 5 for details. In the Galerkin formulation, we replace the Dirichlet data g by its nodal interpolant G_ℓ. The right-hand side vector then takes the form

$$\mathbf{b} \in \mathbb{R}^N \quad \text{with} \quad \mathbf{b}_j = \langle (K + 1/2)G_\ell \,,\, \chi_j\rangle_{L^2(\Gamma)} \tag{3.4}$$

and is computed by

```
b = buildSymmRHS(coordinates,elements,g);
```

where `g` is a function handle for the given Dirichlet data, see Section 5.2.

By approximation of g, we introduce an additional error which is controlled by the so-called Dirichlet data oscillations

$$\mathrm{osc}_\ell^2 = \sum_{E\in\mathcal{E}_\ell} \mathrm{osc}_\ell(E)^2 \quad \text{with} \quad \mathrm{osc}_\ell(E)^2 = \mathrm{length}(E)\|(g - G_\ell)'\|^2_{L^2(E)}, \tag{3.5}$$

cf. [AGP]. The function call

```
osc = computeOscDirichlet(coordinates,elements,g);
```

returns a column vector with `osc(j)` $= \mathrm{osc}_\ell(E_j)^2$, cf. Section 5.3.

In academic experiments, the exact solution $\phi \in L^2(\Gamma)$ is known, and

$$\mathrm{err}_\ell^2 + \mathrm{osc}_\ell^2 = \sum_{E\in\mathcal{E}_\ell} \big(\mathrm{err}_\ell(E)^2 + \mathrm{osc}_\ell(E)^2\big) \quad \text{with} \quad \mathrm{err}_\ell(E)^2 = \mathrm{length}(E)\|\phi - \Phi_\ell\|^2_{L^2(E)} \tag{3.6}$$

is an upper bound for the Galerkin error $|\!|\!|\phi - \Phi_\ell|\!|\!|_V^2$ with respect to the energy norm $|\!|\!|\cdot|\!|\!|_V$. The function call

```
err = computeErrNeumann(coordinates,elements,x,phi);
```

returns a column vector with `err(j)` $= \mathrm{err}_\ell(E_j)^2$. Here, `phi` is a function handle for the known exact solution and `x` is the coefficient vector of the Galerkin solution $\Phi_\ell = \sum_{j=1}^N x_j\chi_j$. We refer to Section 5.4 for details.

For a posteriori error estimation and to steer an adaptive mesh-refinement, `HILBERT` includes four $(h-h/2)$-based error estimators from [FP, EFFP], discussed in Section 5.5 in detail: With $\widehat{\Phi}_\ell$ an improved Galerkin solution with respect to a uniformly refined mesh $\widehat{\mathcal{E}}_\ell$, the a posteriori error estimators

$$\eta_\ell = |\!|\!|\widehat{\Phi}_\ell - \Phi_\ell|\!|\!|_V \text{ and } \mu_\ell^2 = \sum_{E\in\mathcal{E}_\ell} \mu_\ell(E)^2 \text{ with } \mu_\ell(E)^2 = \mathrm{length}(E)\,\|\widehat{\Phi}_\ell - \Phi_\ell\|^2_{L^2(E)} \tag{3.7}$$

can be computed by

```
eta = computeEstSlpEta(father2son,V_fine,x_fine,x_coarse);
```

and

```
mu = computeEstSlpMu(coordinates,elements,father2son,x_fine,x_coarse);
```

respectively. Here, `V_fine` is the Galerkin matrix for the uniformly refined mesh $\widehat{\mathcal{E}}_\ell$, and `x` and `x_fine` are the coefficient vectors for Φ_ℓ and $\widehat{\Phi}_\ell$, respectively. Then, `eta` $= \eta_\ell^2$ and `mu` is a column vector with `mu(j)` $= \mu_\ell(E_j)^2$. The additional field `father2son` describes how to obtain $\widehat{\mathcal{E}}_\ell$ from the given mesh $\mathcal{E}_\ell$, cf. Section 4.2.

The computation of Φ_ℓ can be avoided by taking the L^2-projection onto $\mathcal{P}^0(\mathcal{E}_\ell)$. This leads to a posteriori error estimators

$$(3.8)\quad \widetilde{\eta}_\ell = |||(1-\Pi_\ell)\widehat{\Phi}_\ell|||_V \text{ and } \widetilde{\mu}_\ell^2 = \sum_{E\in\mathcal{E}_\ell} \widetilde{\mu}_\ell(E)^2 \text{ with } \widetilde{\mu}_\ell(E)^2 = \text{length}(E)\,\|(1-\Pi_\ell)\widehat{\Phi}_\ell\|_{L^2(E)}^2$$

which are computed by

```
eta_tilde = computeEstSlpEtaTilde(father2son,V_fine,x_fine);
```

and

```
mu_tilde = computeEstSlpMuTilde(coordinates,elements,father2son,x_fine);
```

Then, `eta_tilde` $= \widetilde{\eta}_\ell^2$ and `mu_tilde` is a column vector with `mu_tilde(j)` $= \widetilde{\mu}_\ell(E_j)^2$.

An adaptive algorithm based on $\widetilde{\mu}_\ell$ is realized in the MATLAB script **`adaptiveSymm`** found in the folder `examples/`. Details are given in Section 5.6.1.

Finally, certain examples are contained in the folders `examples/exampleXX/`, e.g., in `examples/example01/` the function

```
exampleSymm01(Nmax)
```

computes sequences of discrete solutions based on uniform and adaptive mesh-refinement until in both cases meshes $\mathcal{E}_\ell$ with $\#\mathcal{E}_\ell >$ `Nmax` elements have been created. On-the-fly, the decay of the error bound $(\text{err}_\ell^2 + \text{osc}_\ell^2)^{1/2}$ as well as the decay of all error estimators η_ℓ, $\widetilde{\eta}_\ell$, μ_ℓ, and $\widetilde{\mu}_\ell$ is plotted over the number of elements as well as over the runtime of the adaptive algorithm realized in **`adaptiveSymm`**.

3.3.4. Numerical Solution of Neumann Problem. The Laplace problem with Neumann boundary condition (1.9) is equivalently recast in the hypersingular integral equation

$$Wg = (1/2 - K')\phi$$

with g being the unknown Dirichlet data and ϕ being the known Neumann data. We refer to Section 6 for details. Since the hypersingular integral operator W is only semi-elliptic with kernel being the constant functions, we use the Galerkin matrix

$$\mathbf{W} + \mathbf{S} \in \mathbb{R}^{N\times N}_{sym} \quad\text{with}\quad \mathbf{S}_{jk} = \Big(\int_\Gamma \zeta_j\,d\Gamma\Big)\Big(\int_\Gamma \zeta_k\,d\Gamma\Big).$$

The stabilization matrix $\mathbf{S}$ is provided by

```
S = buildHypsingStabilization(coordinates,elements);
```

cf. Section 6.2. In the Galerkin formulation, we further replace the Neumann data ϕ by its L^2-projection Φ_ℓ. The right-hand side vector then takes the form

$$(3.9)\qquad \mathbf{b} \in \mathbb{R}^N \quad\text{with}\quad \mathbf{b}_j = \langle (1/2 - K')\Phi_\ell\,,\,\zeta_j\rangle_{L^2(\Gamma)}$$

and is computed by

```
b = buildHypsingRHS(coordinates,elements,phi);
```

where `phi` is a function handle for the given Neumann data, see Section 6.3.

Similar to above, the additional approximation error is controlled by the so-called Neumann data oscillations

$$\text{osc}_\ell^2 = \sum_{E\in\mathcal{E}_\ell} \text{osc}_\ell(E)^2 \quad \text{with} \quad \text{osc}_\ell(E)^2 = \text{length}(E)\|\phi - \Phi_\ell\|_{L^2(E)}^2. \tag{3.10}$$

The function call

```
osc = computeOscNeumann(coordinates,elements,phi);
```

returns a column vector with `osc(j)` $= \text{osc}_\ell(E_j)^2$, cf. Section 6.4.

In academic experiments, the exact solution $g \in H^1(\Gamma)$ is known, and

$$\text{err}_\ell^2 + \text{osc}_\ell^2 = \sum_{E\in\mathcal{E}_\ell} \text{err}_\ell(E)^2 + \text{osc}_\ell(E)^2 \quad \text{with} \quad \text{err}_\ell(E)^2 = \text{length}(E)\|(g - G_\ell)'\|_{L^2(E)}^2 \tag{3.11}$$

is an upper bound for the Galerkin error $\|\!|g - G_\ell|\!\|_{W+S}^2$ with respect to the energy norm $\|\!|\cdot|\!\|_{W+S}$. The function call

```
err = computeErrDirichlet(coordinates,elements,x,g);
```

returns a column vector with `err(j)` $= \text{err}_\ell(E_j)^2$. Here, `g` is a function handle for the known exact solution and `x` is the nodal vector of the Galerkin solution $G_\ell = \sum_{j=1}^N x_j\zeta_j$. We refer to Section 6.5 for details.

For a posteriori error estimation and to steer an adaptive mesh-refinement, `HILBERT` includes four $(h - h/2)$-based error estimators from [EFGP], discussed in Section 5.5 in detail: These read

$$\eta_\ell = \|\!|\widehat{\Phi}_\ell - \Phi_\ell|\!\|_{W+S} \qquad \text{and} \quad \mu_\ell^2 = \sum_{E\in\mathcal{E}_\ell} \mu_\ell(E)^2 \text{ with } \mu_\ell(E)^2 = \text{length}(E)\,\|\widehat{\Phi}_\ell - \Phi_\ell\|_{L^2(E)}^2$$

$$\widetilde{\eta}_\ell = \|\!|(1 - I_\ell)\widehat{\Phi}_\ell|\!\|_{W+S} \quad \text{and} \quad \widetilde{\mu}_\ell^2 = \sum_{E\in\mathcal{E}_\ell} \widetilde{\mu}_\ell(E)^2 \text{ with } \widetilde{\mu}_\ell(E)^2 = \text{length}(E)\,\|((1 - I_\ell)\widehat{\Phi}_\ell)'\|_{L^2(E)}^2$$

with I_ℓ the nodal interpolation onto $\mathcal{S}^1(\mathcal{T}_\ell)$. These are computed by

```
eta = computeEstHypEta(elem_fine,elements,father2son,WS_fine,x_fine,x);
eta_tilde = computeEstHypEtaTilde(elem_fine,elements,father2son,WS_fine,x_fine);
```

and

```
mu = computeEstHypMu(elem_fine,elements,father2son,x_fine,x);
mu_tilde = computeEstHypMuTilde(elem_fine,elements,father2son,x_fine);
```

Then, `eta` $= \eta_\ell^2$ as well as `eta_tilde` $= \widetilde{\eta}_\ell^2$, and `mu` as well as `mu_tilde` are column vectors with `mu(j)` $= \mu_\ell(E_j)^2$ and `mu_tilde(j)` $= \widetilde{\mu}_\ell(E_j)^2$, respectively. As input, these functions take `elements` and `elem_fine` which describe $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$, respectively, as well as the link between both meshes given by `father2son`. `WS_fine` is the Galerkin matrix for $\widehat{\mathcal{E}}_\ell$. The vectors `x` and `x_fine` are the nodal vectors of the Galerkin solutions G_ℓ and $\widehat{G}_\ell$, respectively.

An adaptive algorithm based on $\widetilde{\mu}_\ell$ is implemented in the MATLAB script **`adaptiveHypsing`** in the folder `examples/`, cf. Section 6.7.1. Moreover, the folders `examples/exampleXX/` contain various examples like **`exampleHypsing01`**`(Nmax)` in `examples/example01/`.

3.3.5. Numerical Solution of Mixed Boundary Value Problem. The Laplace problem with mixed boundary condition (1.10) is equivalently recast in an integral equation which involves the Calderón projector, see Section 7. Our implementation is based on the functions provided for the Dirichlet and Neumann problem. Note that $\mathcal{E}_\ell$ is now given in terms of `coordinates`, `dirichlet`, and `neumann`.

In the problem formulation, the Dirichlet data g are only known on Γ_D. For the integral formulation, one has to fix some extension $\overline{g}$ to Γ. We replace $\overline{g}$ by its nodal interpolation $\overline{G}_\ell$.

Since $\overline{g}$ is only implicitly built on the initial mesh $\mathcal{E}_0$, we need to guarantee that $\overline{G}_\ell|_{\Gamma_N} = \overline{g}|_{\Gamma_N}$. This is done by

```
gh = buildMixedDirichlet(coordinates,dirichlet,neumann, ...
                         father2neumann,neumann_old,gh_old,g);
```

where `gh` is the nodal vector of $\overline{G}_\ell$, `gh_old` is the nodal vector of $\overline{G}_{\ell-1}$, and g is a function handle for the Dirichlet data on Γ_D. Details are found in Section 7.2

To re-use the functions implemented for the hypersingular integral equation from the previous section, we have to guarantee that the first n nodes of $\mathcal{K}_\ell$ belong to $\overline{\Gamma}_N$. This needs some possible reordering of `coordinates` as well as some corresponding update of `dirichlet` and `neumann` which is done by

```
[coordinates,dirichlet,neumann] = buildMixedElements(coordinates,dirichlet,neumann);
```

We stress that the ordering of `dirichlet` and `neumann`, i.e., the numbering of the elements $\mathcal{E}_\ell = \{E_1, \dots, E_N\}$ is not affected. Details are found in Section 7.3.

The right-hand side vector **b** for the Galerkin formulation, split into contributions on Γ_N and contributions on Γ_D, is computed by

```
[bN,bD] = buildMixedRHS(coordinates,dirichlet,neumann,gh,V,K,W,g,phi);
```

Here, `gh` is the nodal vector of the extended Dirichlet data and `g` is a function handle for the Dirichlet data on Γ_D. The function handle `phi` provides the Neumann data on Γ_N. The matrices `V`, `K`, and `W` are the discrete integral operators associated with $\mathcal{E}_\ell$.

An adaptive algorithm from [AGKP] based on the $\widetilde{\mu}_\ell$-estimators from the previous sections, is provided by **`adaptiveMixed`** in the folder `examples/`. Various examples are found in the folders `examples/exampleXX/`, e.g., **`exampleMixed01a(`**`Nmax`**`)`** in `examples/example01/`.

3.4. Visualization of Discrete Solutions.

Provided that Γ is connected, the function

```
plotArclengthP0(coordinates,elements,phih [,phi], [,figure])
```

plots a discrete solution $\Phi_\ell \in \mathcal{P}^0(\mathcal{E}_\ell)$ over the arclength. The elementwise values of Φ_ℓ are provided by the column vector `phih`. With the optional function handle `phi` the exact solution ϕ can be plotted into the same plot for comparison. With the function

```
plotArclengthS1(coordinates,elements,gh [,g], [,figure])
```

one can plot a discrete solution $G_\ell \in \mathcal{S}^1(\mathcal{E}_\ell)$ over the arclength. The nodal values of G_ℓ are provided by the column vector `gh`. With the optional function handle `g` the exact solution g can be plotted into the same plot for comparison.

For both functions, the optional parameter `figure` prescribes the figure number for the plot.

3.5. Numerical Examples.

So far, `HILBERT` contains four different examples, where the exact solution u of

$$-\Delta u = 0 \quad \text{in } \Omega \tag{3.12}$$

is prescribed.

In `examples/example01/`, the exact solution reads in polar coordinates

$$u(r,\varphi) = r^{2/3}\cos(2\varphi/3), \tag{3.13}$$

and Ω is a rotated L-shaped domain with $\operatorname{diam}(\Omega) < 1$ and reentrant corner with angle $3\pi/2$. The rotation is done in a way that the Dirichlet data $g = u|_\Gamma$ are smooth, whereas the Neumann data $\phi = \partial_n u$ have a generic singularity at the reentrant corner.

In `examples/example02/`, the exact solution reads in polar coordinates

$$u(r,\varphi) = r^{4/7}\cos(4\varphi/7), \tag{3.14}$$

and Ω is a Z-shaped domain with $\mathrm{diam}(\Omega) < 1$ and reentrant corner with angle $7\pi/4$. Then, the Dirichlet data g as well as the Neumann data ϕ have generic singularities at the reentrant corner.

In `examples/example03/`, the exact solution reads

$$u(x,y) = \sinh(2\pi\, x) \cdot \cos(2\pi\, y) \tag{3.15}$$

on the cube $\Omega = (0, 1/2)^2$. Now, the Dirichlet data g as well as the Neumann data ϕ are smooth so that uniform mesh-refinement is theoretically predicted to be optimal.

In `examples/example04/` taken from [AGP], we consider the rotated L-shaped domain from the first example. The exact solution is chosen in a way such that the Dirichlet data g have a weak singularity at the uppermost corner, whereas the Neumann data ϕ have a generic singularity at the reentrant corner. Therefore, the adaptive algorithm has to resolve two singularities.

For Symm's integral equation, the hypersingular integral equation, and the symmetric formulation of the mixed boundary value problem, the example files, e.g. for `example/example01/`, are run by

```
exampleSymm01(Nmax)
exampleHypsing01(Nmax)
exampleMixed01a(Nmax)
```

where `Nmax` defines the stopping criterion for the adaptive and uniform mesh-refinement. More precisely, the functions compute sequences of discrete solutions based on uniform and adaptive mesh-refinement until in both cases meshes $\mathcal{E}_\ell$ with $\#\mathcal{E}_\ell >$ `Nmax` elements have been created. On-the-fly, the decay of the error bounds $(\mathrm{err}_\ell^2 + \mathrm{osc}_\ell^2)^{1/2}$ as well as the decay of all error estimators η_ℓ, $\widetilde{\eta}_\ell$, μ_ℓ, and $\widetilde{\mu}_\ell$ is plotted over the number of elements as well as over the runtime of the corresponding $\widetilde{\mu}_\ell$-steered adaptive algorithm.

Listing 1

```
function varargout = markElements(theta,varargin)
%*** check whether optional parameter rho is given or not
if nargin == nargout +1
    rho = 0;
else
    rho = varargin{1};
    varargin = varargin(2:end);
end

%*** enforce input parameters to be column vectors and count their length
nE = zeros(1,nargout+1);
for j = 1:nargout
    nE(j+1) = length(varargin{j});
    varargin{j} = reshape(varargin{j},nE(j+1),1);
end

%*** generate set of all indicators
indicators = cat(1,varargin{:});
nE = cumsum(nE);

%*** realization of Doerfler marking
[indicators,idx] = sort(indicators,'descend');
sum_indicators = cumsum(indicators);
ell = max(ceil(rho*nE(end)),find(sum_indicators>=sum_indicators(end)*theta,1));
marked = idx(1:ell);

%*** split subset marked into subsets with respect to input vectors
for j = 1:nargout
    varargout{j} = marked( marked>nE(j) & marked<=nE(j+1) ) - nE(j);
end
```

4.1. Dörfler Marking (5.28) for Local Mesh-Refinement (Listing 1). We realize the marking criterion proposed by Dörfler [D] in a generalized form which is suitable even for mixed boundary value problems or the FEM-BEM coupling. Suppose that $\mathcal{E}_\ell^{(1)}, \dots, \mathcal{E}_\ell^{(n)}$ are pairwise disjoint meshes which provide indicators $\varrho_\ell^{(k)}(E)$ for all $E \in \mathcal{E}_\ell^{(k)}$. We formally define $\mathcal{E}_\ell := \bigcup_{j=1}^n \mathcal{E}_\ell^{(j)}$ and $\varrho_\ell(E) := \varrho_\ell^{(k)}(E)$ for all $E \in \mathcal{E}_\ell^{(k)}$ and $k = 1, \dots, n$. For given $0 < \theta < 1$, we then aim to find the minimal set $\mathcal{M}_\ell \subseteq \mathcal{E}_\ell$ such that

$$\theta \sum_{E \in \mathcal{E}_\ell} \varrho_\ell(E) \le \sum_{E \in \mathcal{M}_\ell} \varrho_\ell(E). \tag{4.1}$$

Finally, we define and return $\mathcal{M}_\ell^{(k)} := \mathcal{M}_\ell \cap \mathcal{E}_\ell^{(k)}$ for all $k = 1, \dots, n$.

A second generalization is concerned with the minimal cardinality of $\mathcal{M}_\ell$. For analytical convergence results, the minimal set $\mathcal{M}_\ell$ with (4.1) is sufficient. However, small sets $\mathcal{M}_\ell$ lead to many iterations in the adaptive loop and may thus lead to a large runtime. With an additional parameter $0 < \rho < 1$, one remedy for this drawback can be to determine the minimal superset $\mathcal{M}_\ell \subseteq \overline{\mathcal{M}}_\ell \subseteq \mathcal{E}_\ell$ with

$$\frac{\#\overline{\mathcal{M}}_\ell}{\#\mathcal{E}_\ell} \ge \rho \quad \text{and} \quad \varrho_\ell(E) \ge \varrho_\ell(E') \quad \text{for all } E \in \overline{\mathcal{M}}_\ell \text{ and } E' \in \mathcal{E}_\ell \backslash \overline{\mathcal{M}}_\ell. \tag{4.2}$$

From an analytical point of view, any superset $\overline{\mathcal{M}}_\ell$ of $\mathcal{M}_\ell$ also leads to a convergent adaptive algorithm. Our definition of $\overline{\mathcal{M}}_\ell$ guarantees that at least a fixed percentage of elements is refined and that these elements have the largest associated refinement indicators. Note that the parameter ρ gives a lower bound for the percentage of elements which are refined.

Our implementation of the marking criterion includes (optionally) the generalizations (4.1)–(4.2) of the original strategy from [D]:

- In the simplest case, the function **`markElements`** is called by
  `marked = `**`markElements`**`(theta,indicator)`
  where `indicator` is a column vector, where `indicator(j)` corresponds to some element $E_j \in \mathcal{E}_\ell$. The function **`markElements`** then returns the indices corresponding to the minimal set $\mathcal{M}_\ell$.
- Alternatively, the function can be called by
  `marked = `**`markElements`**`(theta,rho,indicator)`
  and returns the indices corresponding to the minimal set $\overline{\mathcal{M}}_\ell \supseteq \mathcal{M}_\ell$ with (4.2).
- For the general formulation described above, the function is called by
  `[marked1,marked2,...] = `**`markElements`**`(theta [,rho], ind1,ind2,...)`
  where, e.g., `ind1` is the vector of indicators $\varrho_\ell(E_j^{(1)})$ for all $E_j^{(1)} \in \mathcal{E}_\ell^{(1)} = \{E_1^{(1)}, \dots, E_{N^{(1)}}^{(1)}\}$. The function returns the indices corresponding to the sets $\mathcal{M}_\ell^{(k)} \subseteq \mathcal{E}_\ell^{(k)}$ (or $\overline{\mathcal{M}}_\ell^{(k)}$ if ρ is given).
- First, we check whether the parameter ρ is given. If not, it is set to 0 (Line 3–8).
- The given indicator vectors are reshaped into column vectors, and their length is stored in the vector `nE` (Line 11–15).
- We build the vector of all indicators $\varrho_\ell(E_j)$ (Line 18) which corresponds to the ordered set $\mathcal{E}_\ell = \{E_1, \dots, E_N\}$. Moreover, `nE` becomes a vector of pointers so that `nE(`j`)+1` and `nE(`$j+1$`)` give the start and the end of $\mathcal{E}_\ell^{(j)}$ with respect to `indicators` (Line 19).
- To determine the minimal set $\mathcal{M}_\ell$ we sort the vector `indicators` (Line 22). Mathematically, this corresponds to finding a permutation π such that $\varrho_\ell(E_{\pi(j)}) \geq \varrho_\ell(E_{\pi(j+1)})$. We then compute the vector `sum_indicators` of sums $\sum_{j=1}^k \varrho_\ell(E_{\pi(j)})$ (Line 23). Note that `sum_indicators(end)` contains $\sum_{j=1}^N \varrho_\ell(E_{\pi(j)}) = \sum_{j=1}^N \varrho_\ell(E_j)$. Finding the minimal set $\mathcal{M}_\ell$ is thus equivalent to finding the minimal index k with $\theta \sum_{j=1}^N \varrho_\ell(E_{\pi(j)}) \leq \sum_{j=1}^k \varrho_\ell(E_{\pi(j)})$, and there holds $\mathcal{M}_\ell = \{E_{\pi(1)}, \dots, E_{\pi(k)}\}$. If ρ is specified, we choose the minimal index $\overline{k} \geq k$ with $\overline{k} \geq \rho N$. Altogether, Line 24–25 thus determines the indices of elements in $\mathcal{M}_\ell$ and $\overline{\mathcal{M}}_\ell$, respectively.
- Finally, we use the pointer vector `nE` to determine the indices of $\mathcal{M}_\ell^{(k)}$ with respect to $\mathcal{E}_\ell^{(k)} = \{E_1^{(k)}, ..., E_{N^{(k)}}^{(k)}\}$ (Line 28–30).

Listing 2

```
function [coordinates,varargout] = refineBoundaryMesh(coordinates,varargin)
%*** fix the blow-up factor for the K-mesh constant,
%*** where we assume C(Mesh_0) = 1, i.e., the initial mesh is uniform
kappa = 2;

%*** count number of boundary parts from input
%*** nB will hold this number
nB = 0;

for iter = 1 : (nargin - 1)

    if size(varargin{iter},2) == 2
        nB = nB + 1;
        nE_boundary(iter) = size(varargin{iter},1);
    else
        break;
    end

end

%*** check the correct number of input parameters
if ~( (nargin == (nB+1)) || (nargin == (2*nB+1)) )
    error('refineBoundaryMesh: Wrong number of input arguments!');
end

%*** check the correct number of output parameters
if ~( (nargout == (nB+1)) || (nargout == (2*nB+1)) )
    error('refineBoundaryMesh: Wrong number of output arguments!');
end

%*** check, if user asks for father2son fields in output
if nargout == (2*nB+1)
    output_father2son = true;
else
    output_father2son = false;
end

%*** obtain set of all elements of the boundary partition
elements = cat(1,varargin{1 : nB});

%*** indices of a boundary part w.r.t. entire field elements
ptr_boundary = cumsum([0,nE_boundary]);

%*** 1. determine whether uniform or adaptive mesh-refinement
%*** 2. in case of adaptive mesh-refinement compute vector marked
%***    of marked elements w.r.t. entire field elements
if (nB+1) == nargin
    refinement = 'uniform';
else
    refinement = 'adaptive';
    marked = zeros(0,1); % marked elements w.r.t. entire field elements

    for iter = 1 : nB
        marked = [marked; varargin{iter + nB} + ptr_boundary(iter)];
    end

end

nC = size(coordinates,1); % number of coordinates
nE = size(elements,1);    % number of elements

if strcmp(refinement,'adaptive')

    %*** if element Ej is marked and if its neighbour Ek satisfies
    %*** hk >= kappa*hj, we (recursively) mark Ek for refinement as well

    %*** marked elements Ej will be refined, i.e., flag(j) = 1
    flag = zeros(nE,1);
    flag(marked) = 1;

```

```
%*** determine neighbouring elements
node2element = zeros(nC,2);
node2element(elements(:,1),2) = (1:nE)';
node2element(elements(:,2),1) = (1:nE)';
element2neighbour = [ node2element(elements(:,1),1), ...
                      node2element(elements(:,2),2) ];

%*** compute (squared) local mesh-size
h = sum((coordinates(elements(:,1),:)-coordinates(elements(:,2),:)).^2,2)';

%*** the formal recursion is avoided by sorting elements by mesh-size
[tmp,sorted_elements] = sort(h);
for j = sorted_elements
    if flag(j)
        neighbours = element2neighbour(j,:);
        neighbours = neighbours( find(neighbours) );
        flag( neighbours(h(neighbours) >= kappa*h(j)) ) = 1;
    end
end

%*** obtain vector of marked elements
marked = find(flag);
nM = length(marked);

%*** compute and add new nodes
coordinates = [coordinates;zeros(nM,2)];
coordinates((1:nM)+nC,:) = ( coordinates(elements(marked,1),:) ...
                           + coordinates(elements(marked,2),:) )*0.5;

%*** refinement of mesh iterates over each boundary part
for iter = 1:nB

    %*** determine which marked elements belong to boundary part
    idx = find( (ptr_boundary(iter) < marked) ...
                & (marked <= ptr_boundary(iter+1)) );
    nM_boundary = length(idx);

    %*** allocate new elements
    new_elements = [varargin{iter};zeros(nM_boundary,2)];

    %*** generate new elements
    new_elements((1:nM_boundary)+nE_boundary(iter),:) ...
        = [ nC + idx, elements(marked(idx),2) ];
    new_elements( marked(idx) - ptr_boundary(iter),2 ) = nC + idx;

    %*** add new_elements and father2son to output
    varargout{iter} = new_elements;

    %*** compute father2son only if desired
    if output_father2son == true

        %*** generate father2son
        father2son = repmat((1:nE_boundary(iter))',1,2);
        father2son( marked(idx) - ptr_boundary(iter),2 ) ...
            = (1:nM_boundary)' + nE_boundary(iter);

        %*** add new_elements and father2son to output
```

```
            varargout{nB+iter} = father2son;
        end

    end

elseif strcmp(refinement,'uniform')

    %*** compute and add new nodes
    coordinates = [coordinates;zeros(nE,2)];
    coordinates((nC+1):end,:) = ( coordinates(elements(:,1),:) ...
                                + coordinates(elements(:,2),:) )*0.5;

    %*** uniform refinement of mesh iterates over each boundary part
    for iter = 1:nB

        %*** generate new elements
        idx = (ptr_boundary(iter)+1):ptr_boundary(iter+1);
        varargout{iter} = [ varargin{iter}(:,1),nC + idx' ; ...
                            nC + idx', varargin{iter}(:,2) ];

        %*** compute father2son only if desired
        if output_father2son == true

            %*** build father2son
            varargout{nB+iter} ...
                = [(1:nE_boundary(iter))', ...
                   (1:nE_boundary(iter))' + nE_boundary(iter)];
        end

    end
end
```

4.2. Local Refinement of Boundary Element Mesh (Listing 2). In many cases, one is not interested in computing only one approximation U with respect to a fixed given boundary element mesh $\mathcal{E}$, but in computing a sequence of more and more accurate approximations U_ℓ corresponding to a sequence $\mathcal{E}_\ell$ of boundary element meshes with decreasing mesh-sizes. To that end, our software package HILBERT provides an efficient mesh-refinement **refineBoundaryMesh** for boundary element meshes, which covers the following tasks:

- uniform refinement of a given mesh
- refinement of certain marked elements, specified by the user
- linkage between elements of the input mesh with elements of the refined mesh
- handling of meshes that are split into finitely many distinct parts, e.g., $\Gamma = \overline{\Gamma}_D \cup \overline{\Gamma}_N$
- guaranteed boundedness of the K-mesh constant

Throughout, refinement of an element means that E_i is bisected into two elements e_j, e_k of half length. We now discuss certain aspects of our implementation from Listing 2, where the data structure of `coordinates`, `elements`, `dirichlet`, and `neumann` is described in Section 3.2 above. The main focus is, however, on the practical use of the function.

- **Input/Output Parameters:** To allow a partition of Γ into finitely many parts (e.g., a Dirichlet and a Neumann boundary), the formal signature reads
  ```
      [coordinates,varargout] = refineBoundaryMesh(coordinates,varargin)
  ```
  To explain the variable input/output parameters, we consider certain examples.
- Suppose that $\mathcal{E}_\ell = \{E_1, \ldots, E_N\}$ is described by `coordinates` and `elements`. Then,
  ```
      [coordinates_fine,elements_fine,father2son] ...
          = refineBoundaryMesh(coordinates,elements)
  ```

provides the uniformly refined mesh $\widehat{\mathcal{E}}_\ell = \{e_1, \dots, e_{2N}\}$, where each element $E_i \in \mathcal{E}_\ell$ is bisected in certain sons $e_j, e_k \in \widehat{\mathcal{E}}_\ell$. The $(N \times 2)$-matrix `father2son` provides a link between the element indices in the sense that

`father2son`$(i,:) = [\,j,\,k\,]$ for $E_i = e_j \cup e_k$.

The output parameter `father2son` is optional and can be omitted.

- Suppose that $\mathcal{M}_\ell \subseteq \mathcal{E}_\ell$ is a set of elements which are marked for refinement. Let `marked` be an $(M \times 1)$-column vector containing the indices of the elements in $\mathcal{M}_\ell$. Then,
```
[coordinates_fine,elements_fine,father2son] ...
          = refineBoundaryMesh(coordinates,elements,marked)
```
provides a mesh $\mathcal{E}_{\ell+1}$ which is only refined locally in the sense that all elements of $\mathcal{M}_\ell$ are refined. If an element $E_i \in \mathcal{E}_\ell$ is not refined, there holds $E_i = e_j \in \mathcal{E}_{\ell+1}$, where the link between these indices are given by
`father2son`$(i,:) = [\,j,\,j\,]$ for $E_i = e_j$.
Again, the output parameter `father2son` is optional and can be omitted.
- Suppose that Γ is split into a Dirichlet boundary Γ_D and a Neumann boundary Γ_N. In this case, the mesh $\mathcal{E}_\ell = \{E_1, \dots, E_N\}$ is described in terms of `coordinates`, `dirichlet`, and `neumann`, cf. Section 3.2. Then,
```
[coordinates_fine,dirichlet_fine,neumann_fine,dir2son,neu2son] ...
          = refineBoundaryMesh(coordinates,dirichlet,neumann)
```
provides the uniformly refined mesh $\widehat{\mathcal{E}}_\ell$. As `father2son` in the previous cases with a single boundary part, now the arrays `dir2son` and `neu2son` provide the link between the coarse mesh parts and the refined ones, e.g., `dirichlet` and `dirichlet_fine`. For instance, suppose that $\mathcal{E}_\ell|_{\Gamma_D} = \{E_1^D, \dots, E_{N_D}^D\}$ and $\widehat{\mathcal{E}}_\ell|_{\Gamma_D} = \{e_1^D, \dots, e_{2N_D}^D\}$. Then, there holds
`dir2son`$(i,:) = [\,j,\,k\,]$ for $E_i^D = e_j^D \cup e_k^D$.
Finally, the fields `dir2son` and `neu2son` are optional in the sense that they can either both be asked for or both be omitted.
- Suppose that $\mathcal{M}_\ell^D \subseteq \mathcal{E}_\ell|_{\Gamma_D}$ and $\mathcal{M}_\ell^N \subseteq \mathcal{E}_\ell|_{\Gamma_N}$ are sets of marked elements. Let `marked_dirichlet` and `marked_neumann` be $(M_D \times 1)$- and $(M_N \times 1)$-column vectors containing the indices of the elements in $\mathcal{M}_\ell^D$ and $\mathcal{M}_\ell^N$, respectively. Then,
```
[coordinates_fine,dirichlet_fine,neumann_fine,dir2son,neu2son] ...
          = refineBoundaryMesh(coordinates,dirichlet,neumann, ...
                                         marked_dirichlet,marked_neumann)
```
provides a mesh $\mathcal{E}_{\ell+1}$ which is only refined locally in the sense that all elements of $\mathcal{M}_\ell^D \cup \mathcal{M}_\ell^N$ are refined. We stress that the optional input `marked_dirichlet` and `marked_neumann` can either both be given or both be omitted. The optional output has already been described before.
- If Γ is split into more than two boundary parts, described by, e.g., `dirichlet`, `neumann`, and `robin`, the function **`refineBoundaryMesh`** can be used accordingly.
- **Refinement of an Element:** Suppose that element $E_i = [a, b] \in \mathcal{E}_\ell$ is bisected into two sons $e_j, e_k \in \mathcal{E}_{\ell+1}$ (or $\widehat{\mathcal{E}}_\ell$). Then, there holds $e_j = [a, m]$ and $e_k = [m, b]$, where $m = (a+b)/2$ denotes the midpoint of E_i. Note that `elements(i,:)` returns the indices of the nodes $a, b \in \mathcal{K}_\ell$. Clearly, $\mathcal{K}_\ell \subseteq \mathcal{K}_{\ell+1}$ and, e.g., the index of $a = z_p \in \mathcal{K}_{\ell+1}$ is obtained by $p =$ `elements_fine(father2son(`i`,1),1)`.
- **Boundedness of K-Mesh Constant:** Many estimates in numerical analysis depend on local quantities of the mesh, e.g., on an upper bound of the K-mesh constant

$$\kappa(\mathcal{E}_\ell) := \sup\big\{\mathrm{length}(E_j)/\mathrm{length}(E_k) \,:\, E_j, E_k \in \mathcal{E}_\ell \text{ with } E_j \cap E_k \neq \emptyset\big\} \geq 1 \tag{4.3}$$

which is the maximal ratio of the element widths of neighbouring elements. Let $\mathcal{E}_0$ be a given initial mesh. Let $\mathcal{E}_\ell$ be inductively obtained by refinement of arbitrary sets of marked elements $\mathcal{M}_j \subseteq \mathcal{E}_j$ with $0 \leq j \leq \ell - 1$. To avoid the blow-up of the K-mesh constant, one thus aims for a refinement rule which guarantees $\sup_{\ell \in \mathbb{N}} \kappa(\mathcal{E}_\ell) < \infty$. Our refinement rule, proposed and analyzed in [AGP, Section 2.2], guarantees

$$\sup_{\ell \in \mathbb{N}} \kappa(\mathcal{E}_\ell) \le 2\,\kappa(\mathcal{E}_0) \tag{4.4}$$

by refinement of all elements in a certain superset $\overline{\mathcal{M}}_\ell \supseteq \mathcal{M}_\ell$. If the initial mesh is uniform, one can prove that our refinement rule guarantees

$$\#\mathcal{E}_\ell - \#\mathcal{E}_0 \lesssim \sum_{j=0}^{\ell-1} \#\mathcal{M}_j, \tag{4.5}$$

i.e. the set $\overline{\mathcal{M}}_j$ is generically of the same size as $\mathcal{M}_j$, cf. [AGP, Theorem 2.5]. The constant hidden in the symbol $\lesssim$ only depends on the initial mesh $\mathcal{E}_0$.

Finally, we give a rough overview on the code:

- Variable input-/output parameters are treated in Line 8–60.
- The case of adaptive refinement is treated in Line 62–131.
- In order to ensure the boundedness of the K-mesh constant, the refinement algorithm checks the mesh-size ratio of neighbouring elements: If $E_i \in \mathcal{E}_\ell$ is marked for refinement, any neighbour E_j with
$$h_\ell|_{E_j} / h_\ell|_{E_i} \ge 2$$
is recursively marked for refinement as well (Line 68–92). This guarantees $\kappa(\mathcal{E}_\ell) \le 2\kappa(\mathcal{E}_0)$ for all generated meshes $\mathcal{E}_\ell$.
- For all refined elements the coordinates of the midpoints of these elements are computed as new nodes for the refined mesh (Line 96–98).
- We loop over each boundary part (Line 101–131), generate new elements as result of bisecting the respective coarse mesh elements (Line 109–114) and build the linkage arrays in Line 123–125.
- The case of uniform refinement (Line 133–158) is a straight forward implementation.

5. Symm's Integral Equation

Continuous Model Problem. In the entire section, we consider Symm's integral equation

$$V\phi = (K + 1/2)g \quad \text{on } \Gamma \tag{5.1}$$

with V the simple-layer potential and K the double-layer potential, where $\Gamma = \partial\Omega$ is the piecewise-affine boundary of a polygonal Lipschitz domain $\Omega \subset \mathbb{R}^2$. This integral equation is an equivalent formulation of the Dirichlet problem

$$-\Delta u = 0 \text{ in } \Omega \quad \text{with} \quad u = g \text{ on } \Gamma. \tag{5.2}$$

Formally, the Dirichlet data satisfy $g \in H^{1/2}(\Gamma)$. We will, however, assume additional regularity $g \in H^1(\Gamma) \subset H^{1/2}(\Gamma)$ so that g is, in particular, continuous. The exact solution $\phi \in H^{-1/2}(\Gamma)$ of (5.1) is the normal derivative $\phi = \partial_n u$ of the solution $u \in H^1(\Omega)$ of (5.2).

Note that (5.1) can equivalently be written in variational form

$$\langle V\phi\,,\,\psi\rangle_\Gamma = \langle (K+1/2)g\,,\,\psi\rangle_\Gamma \quad \text{for all } \psi \in H^{-1/2}(\Gamma), \tag{5.3}$$

where $\langle\cdot\,,\,\cdot\rangle_\Gamma$ denotes the extended $L^2(\Gamma)$-scalar product, i.e. $\langle\phi\,,\,\psi\rangle_\Gamma = \int_\Gamma \phi\psi\,d\Gamma$ for $\phi, \psi \in L^2(\Gamma)$ and with $\int_\Gamma d\Gamma$ integration along the boundary. Provided that $\operatorname{diam}(\Omega) < 1$, one can show that the left-hand side

$$\langle\!\langle \phi\,,\psi\rangle\!\rangle_V := \langle V\phi\,,\,\psi\rangle_\Gamma \quad \text{for } \phi, \psi \in H^{-1/2}(\Gamma) \tag{5.4}$$

of (5.3) defines a scalar product on $H^{-1/2}(\Gamma)$, and the induced norm $|\!|\!|\phi|\!|\!|_V := \langle\!\langle \phi\,,\phi\rangle\!\rangle_V^{1/2}$ is an equivalent norm on $H^{-1/2}(\Gamma)$. In particular, the variational form (5.3) has a unique solution $\phi \in H^{-1/2}(\Gamma)$ which depends continuously on the data g with respect to the $H^{1/2}(\Gamma)$-norm.

Galerkin Discretization. To discretize (5.3), we first replace the continuous Dirichlet data $g \in H^1(\Gamma)$ by its nodal interpolant

$$G_\ell := \sum_{j=1}^N g(z_j)\zeta_j \in \mathcal{S}^1(\mathcal{E}_\ell) \subset H^1(\Gamma) \tag{5.5}$$

where $\{\zeta_1, \dots, \zeta_N\}$ denotes the set of canonical basis functions of $\mathcal{S}^1(\mathcal{E}_\ell)$. Second, we replace the entire function space $H^{-1/2}(\Gamma)$ in (5.3) by the finite-dimensional space $\mathcal{P}^0(\mathcal{E}_\ell)$. Since the discrete space $\mathcal{P}^0(\mathcal{E}_\ell)$ is a subspace of $H^{-1/2}(\Gamma)$, $\langle\!\langle\cdot\,,\cdot\rangle\!\rangle_V$ from (5.4) is also a scalar product on $\mathcal{P}^0(\mathcal{E}_\ell)$. Consequently, there is a unique Galerkin solution $\Phi_\ell \in \mathcal{P}^0(\mathcal{E}_\ell)$ of

$$\langle V\Phi_\ell\,,\,\Psi_\ell\rangle_\Gamma = \langle (K+1/2)G_\ell\,,\,\Psi_\ell\rangle_\Gamma \quad \text{for all } \Psi_\ell \in \mathcal{P}^0(\mathcal{E}_\ell). \tag{5.6}$$

According to Linear Algebra, (5.6) holds for all $\Psi_\ell \in \mathcal{P}^0(\mathcal{E}_\ell)$ if and only if it holds for all (canonical) basis functions $\chi_k \in \mathcal{B}_\ell = \{\chi_1, \dots, \chi_N\}$ of $\mathcal{P}^0(\mathcal{E}_\ell)$. With the coefficient vector $\mathbf{x} \in \mathbb{R}^N$ of the ansatz

$$\Phi_\ell = \sum_{j=1}^N \mathbf{x}_j\chi_j \tag{5.7}$$

and the vector $\mathbf{g} \in \mathbb{R}^N$ defined by $\mathbf{g}_j := g(z_j)$ for all $z_j \in \mathcal{K}_\ell$, the Galerkin formulation (5.6) is thus equivalent to

$$\sum_{j=1}^N \mathbf{x}_j\langle V\chi_j\,,\,\chi_k\rangle_\Gamma = \langle V\Phi_\ell\,,\,\chi_k\rangle_\Gamma = \langle (K+1/2)G_\ell\,,\,\chi_k\rangle_\Gamma = \sum_{j=1}^N \mathbf{g}_j\langle (K+1/2)\zeta_j\,,\,\chi_k\rangle_\Gamma$$

for all $k = 1, \dots, N$. If we define matrices $\mathbf{V}, \mathbf{K}, \mathbf{M} \in \mathbb{R}^{N\times N}$ by

$$\mathbf{V}_{kj} = \langle V\chi_j\,,\,\chi_k\rangle_\Gamma, \quad \mathbf{K}_{kj} = \langle K\zeta_j\,,\,\chi_k\rangle_\Gamma, \quad \mathbf{M}_{kj} = \langle \zeta_j\,,\,\chi_k\rangle_\Gamma \quad \text{for all } j, k = 1, \dots, N, \tag{5.8}$$

the last equation becomes

$$(\mathbf{V}\mathbf{x})_k = \sum_{j=1}^N \mathbf{x}_j \mathbf{V}_{kj} = \sum_{j=1}^N \mathbf{g}_j \Big(\mathbf{K}_{kj} + \frac{1}{2}\mathbf{M}_{kj}\Big) = \Big(\mathbf{K}\mathbf{g} + \frac{1}{2}\mathbf{M}\mathbf{g}\Big)_k \quad \text{for all } k = 1, \dots, N.$$

Altogether, the Galerkin formulation (5.6) is thus equivalent to the linear system

$$\mathbf{V}\mathbf{x} = \mathbf{K}\mathbf{g} + \frac{1}{2}\mathbf{M}\mathbf{g}. \tag{5.9}$$

We stress that $\mathbf{V}$ is symmetric and positive definite since it stems from a scalar product. In particular, the linear system (5.9) has a unique solution $\mathbf{x} \in \mathbb{R}^N$.

5.1. Computation of Discrete Integral Operators V and K. The matrices $\mathbf{V}, \mathbf{K} \in \mathbb{R}^{N\times N}$ defined in (5.8) are implemented in the programming language C via the Matlab-Mex-Interface. The simple-layer potential matrix $\mathbf{V}$ is returned by call of

```
V = buildV(coordinates,elements [,eta]);
```

In general, all matrix entries of $\mathbf{V}$ can be computed analytically by use of anti-derivatives found in [M]. However, analytic integration leads to cancellation effects if the integration domain is small, i.e. $\int_a^b \cdot \, dx$ with $a \approx b$. In this case, the (continuous) integrand is generically of one sign so that Gaussian quadrature (with positive weights) appears to be more stable.

Let $\eta \geq 0$ be given. Recall that

$$\mathbf{V}_{kj} = -\frac{1}{2\pi} \int_{E_k} \int_{E_j} \log|x-y| \, d\Gamma(y) \, d\Gamma(x).$$

A pair of elements (E_j, E_k) is called *admissible* provided that

$$\min\{\text{length}(E_j), \text{length}(E_k)\} \leq \eta \, \text{dist}(E_j, E_k) \tag{5.10}$$

with $\text{dist}(\cdot,\cdot)$ the distance of E_j and E_k. Otherwise, the pair (E_j, E_k) is called inadmissible. Note that for $\mathbf{V}_{kj}$, the Fubini theorem applies and proves that one can assume w.l.o.g. that $\text{length}(E_k) \leq \text{length}(E_j)$. Note that the cancellation effects from the outer integration are thus generically higher than those of the inner integration. For fixed $x \in E_k$, the inner integral

$$\int_{E_j} \log|x-y| \, d\Gamma(y)$$

is computed analytically [M]. If the pair (E_j, E_k) is admissible, we parametrize E_k and approximate

$$\begin{aligned}\int_{E_k}\int_{E_j} \log|x-y| \, d\Gamma(y)\, d\Gamma(x) &= \int_{-1}^1 \int_{E_j} \log|\gamma_k(s) - y| \, d\Gamma(y) \, ds \\ &\approx \sum_{m=1}^p \omega_m \int_{E_j} \log|\gamma_k(s_m) - y| \, d\Gamma(y)\end{aligned}$$

with a Gaussian quadrature on $[-1,1]$ of length p.

For fixed $\eta > 0$, the described procedure leads to some approximate matrix $\mathbf{V}_p \approx \mathbf{V}$. It is proven in [Ma, Satz 3.13] that $\mathbf{V}_p$ converges exponentially to $\mathbf{V}$ with respect to the Frobenius norm (and hence the ℓ_2-operator norm) as $p \to \infty$.

In `HILBERT`, we choose $\eta = 1/2$, if the optional parameter `eta` is not specified. If `eta` is given by the user, we set $\eta =$ `eta`. Note that for given `eta` ≤ 0 all entries of $\mathbf{V}$ are inadmissible and thus computed analytically. For `eta` > 0 or non-specified, certain entries are computed semi-analytically as described before, where we use a Gaussian quadrature of length $p = 16$. Different values of p can be chosen by modification of the file `source/geometry.h` and by re-building the integral operators, see Section 3.1.

The double-layer potential matrix $\mathbf{K}$ is obtained by call of

```
K = buildK(coordinates,elements [,eta]);
```

Note that the entries of $\mathbf{K}$ read

$$\mathbf{K}_{kj} = -\frac{1}{2\pi}\int_{E_k}\int_{\mathrm{supp}(\zeta_j)} \frac{(y-x)\cdot\mathbf{n}_j}{|x-y|^2}\,\zeta_j(y)\,d\Gamma(y)\,d\Gamma(x),$$

where $\mathrm{supp}(\zeta_j)$ denotes the support of ζ_j and where $\mathbf{n}_j \in \mathbb{R}^2$ denotes the (constant) outer normal vector on E_j. For a mesh of a closed boundary Γ, $\mathrm{supp}(\zeta_j)$ is the union of precisely two elements $E_i \in \mathcal{E}_\ell$. Therefore, the computation of $\mathbf{K}_{kj}$ needs the computation of double integrals of the type

$$\int_{E_k}\int_{E_i} \frac{(y-x)\cdot\mathbf{n}_j}{|x-y|^2}\,\zeta_j(y)\,d\Gamma(y)\,d\Gamma(x).$$

These can be computed analytically by use of anti-derivatives from [M]. For admissible pairs (E_i, E_k), we may proceed as described for $\mathbf{V}$. More precisely, we change the order of integration so that the smaller element corresponds to the outer integration, and we use numerical quadrature to compute the outer integral. As for $\mathbf{V}$, this provides an approximation $\mathbf{K}_p \approx \mathbf{K}$ which converges exponentially to $\mathbf{K}$ as $p \to \infty$.

LISTING 3

```
function b = buildSymmRHS(coordinates,elements,uD)
%*** nodal evaluation of Dirichlet data
uDh = uD(coordinates);

%*** compute DLP-matrix for P0 x S1
K = buildK(coordinates,elements);

%*** compute mass-type matrix for P0 x S1
nE = size(elements,1);
h = sqrt(sum((coordinates(elements(:,1),:)-coordinates(elements(:,2),:)).^2,2));
I = reshape(repmat(1:nE,2,1),2*nE,1);
J = reshape(elements',2*nE,1);
A = reshape(repmat(0.5*h,1,2)',2*nE,1);
M = sparse(I,J,A);

%*** build right-hand side vector
b = K*uDh + M*uDh*0.5;
```

5.2. Building of Right-Hand Side Vector (Listing 3). To compute the vector

$$\mathbf{b} := \mathbf{K}\mathbf{g} + \frac{1}{2}\mathbf{M}\mathbf{g} \in \mathbb{R}^N \tag{5.11}$$

from (5.9), it essentially remains to build the matrix $\mathbf{M} \in \mathbb{R}^{N\times N}$. Let $z_m, z_n \in \mathcal{K}_\ell$ denote the nodes of an element $E_k \in \mathcal{E}_\ell$, i.e., $E_k = [z_m, z_n]$. Note that the entry $\mathbf{M}_{kj} = \langle \zeta_j\,,\,\chi_k\rangle_\Gamma = \int_{E_k} \zeta_j\,ds$ satisfies

$$\mathbf{M}_{kj} = \begin{cases} 0 & \text{if } z_j \notin \{z_m, z_n\}, \\ \mathrm{length}(E_k)/2 & \text{if } z_j \in \{z_m, z_n\}. \end{cases}$$

We thus may assemble the matrix $\mathbf{M}$ in the following way:

```
nE = size(elements,1);
M = sparse(nE,nE);
for k = 1:nE
    a = coordinates(elements(k,1),:);
    b = coordinates(elements(k,2),:);
    h = norm(b-a);
    M(k,elements(k,:)) = h/2;
```

```
end
```

We stress, however, that this implementation will lead to (at least) quadratic runtime with respect to the number N of elements. The reason for this is the internal storage of sparse matrices in MATLAB by use of the CCS format. This requires to sort the corresponding memory with every update of the sparse matrix and thus leads to a complexity $\mathcal{O}(k \log k)$ for k non-zero entries. Since this is done for $k = 1, \dots, N$, one consequently expects a computational complexity of order $\mathcal{O}(N^2 \log N)$ which can even be observed experimentally [FPW].

Building sparse matrices in MATLAB is efficiently done via the built-in function `sparse` which takes the coordinate format $I, J, A \in \mathbb{R}^n$, where $\mathbf{M}_{ij} = A_k$ for $i = I_k$ and $j = J_k$. Here, the matrix $\mathbf{M}$ has $n = 2N$ non-zero entries.

Altogether, the documentation of Listing 3 reads as follows:

- The function takes the mesh $\mathcal{E}_\ell$ described in terms of `coordinates` and `elements` as well as a function handle `uD` for the Dirichlet data g.
- We compute the vector $\mathbf{g} \in \mathbb{R}^N$ of nodal values of g (Line 3) and build the double-layer potential matrix $\mathbf{K} \in \mathbb{R}^{N\times N}$ (Line 6).
- The column vector $h \in \mathbb{R}^N$ contains $h_j = \mathrm{length}(E_j)$ (Line 10). We stress that the Euclidean length `h = norm(b-a)` can also be computed via `h = sqrt(sum((b-a).^2,2))` if $a, b \in \mathbb{R}^2$ are row-vectors. Then, the vectors $I, J, A \in \mathbb{R}^{2N}$ of the coordinate format of $\mathbf{M}$ are computed (Line 11–13), and the matrix $\mathbf{M}$ is built (Line 14).
- Finally, we build the right-hand side vector $\mathbf{b}$ (Line 17).

LISTING 4

```
function osc = computeOscDirichlet(coordinates,elements,uD)
%*** compute midpoints of all elements
midpoints = 0.5*( coordinates(elements(:,1),:) + coordinates(elements(:,2),:) );

%*** evaluate Dirichlet data at element midpoints
uD_midpoints = uD(midpoints);

%*** evaluate Dirichlet data at all nodes
uD_nodes = uD(coordinates);

%*** compute oscillations of Dirichlet data via adapted Newton-Cotes formula
osc = 4/3*( uD_nodes(elements(:,1))+uD_nodes(elements(:,2))-2*uD_midpoints ).^2;
```

5.3. Computation of Data Oscillations for Dirichlet Data (Listing 4). Instead of solving the correct variational form (5.3), we solve

$$\langle V\phi_\ell\,,\,\psi\rangle_\Gamma = \langle (K+1/2)G_\ell\,,\,\psi\rangle_\Gamma \quad \text{for all } \psi \in H^{-1/2}(\Gamma) \tag{5.12}$$

with perturbed right-hand side, where we use the approximation $G_\ell \approx g$. It is an analytical observation that the error between the exact solution $\phi \in H^{-1/2}(\Gamma)$ of (5.3) and the exact solution $\phi_\ell \in H^{-1/2}(\Gamma)$ of the perturbed formulation (5.12) is controlled by

$$\|\phi - \phi_\ell\|_V \lesssim \|h_\ell^{1/2}(g - G_\ell)'\|_{L^2(\Gamma)} =: \mathrm{osc}_{D,\ell}, \tag{5.13}$$

where $(\cdot)'$ denotes the arclength-derivative, cf. [AGP].

We now aim for a numerical approximation of the local contributions

$$\mathrm{osc}_{D,\ell}(E_j) = \|h_\ell^{1/2}(g - G_\ell)'\|_{L^2(E_j)} = \mathrm{length}(E_j)^{1/2}\|(g - G_\ell)'\|_{L^2(E_j)} \quad \text{for all } E_j \in \mathcal{E}_\ell.$$

For $E_j = [a_j, b_j] \in \mathcal{E}_\ell$ and $h := \mathrm{length}(E_j) = |b_j - a_j|$, let $\gamma_j : [-1, 1] \to E_j$ denote the reference parametrization from (2.1). Recall that $|\gamma_j'| = h/2$. With the definition of a boundary integral from Section 2.2 and the definition of the arclength derivative from Section 2.3, we obtain

$$\|v'\|_{L^2(E_j)}^2 = \int_{E_j} (v')^2 \, d\Gamma \overset{\text{Def}}{=} \frac{h}{2}\int_{-1}^{1} \big((v'\circ\gamma_j)(s)\big)^2 \, ds \overset{\text{Def}}{=} \frac{2}{h}\int_{-1}^{1} \big((v\circ\gamma_j)'(s)\big)^2 \, ds. \tag{5.14}$$

We now approximate $w := v\circ\gamma_j : [-1,1] \to \mathbb{R}$ by some polynomial $p_j \in \mathcal{P}^2[-1,1]$ with

$$p_j(-1) = w(-1) = v(a_j), \quad p_j(0) = w(0) = v(m_j), \quad p_j(1) = w(1) = v(b_j),$$

where $m_j = (a_j + b_j)/2$ denotes the midpoint of E_j. Note that $p_j' \in \mathcal{P}^1[-1,1]$ and $(p_j')^2 \in \mathcal{P}^2[-1,1]$ so that

$$\|v'\|_{L^2(E_j)}^2 = \frac{2}{h}\int_{-1}^{1} \big((v\circ\gamma_j)'(s)\big)^2 \, ds \approx \frac{2}{h}\int_{-1}^{1} (p_j')^2 \, ds = \frac{2}{h}\,\mathrm{quad}_2\big((p_j')^2\big),$$

where $\mathrm{quad}_2(\cdot)$ is a quadrature rule on $[-1,1]$ which is exact on $\mathcal{P}^2[-1,1]$. We use a 3-point Newton-Côtes formula with nodes $s_1 = -1$, $s_2 = 0$, and $s_3 = 1$, which is exact on $\mathcal{P}^3[-1,1]$. It thus remains to evaluate $p_j'(s_k)$ by use of $p_j(-1)$, $p_j(0)$, and $p_j(1)$. To that end, we write p_j in terms of the Lagrangian basis

$$p_j = v(a_j)L_1 + v(m_j)L_2 + v(b_j)L_3, \quad \text{whence} \quad p_j' = v(a_j)L_1' + v(m_j)L_2' + v(b_j)L_3'.$$

The Lagrange polynomials L_k associated with $s_k = -1, 0, 1$ read

$$L_1(s) = s(s-1)/2, \quad L_2(s) = 1 - s^2, \quad L_3(s) = s(s+1)/2,$$

and their derivatives are

$$L_1'(s) = (2s-1)/2, \quad L_2'(s) = -2s, \quad L_3'(s) = (2s+1)/2.$$

With the matrix $(L_k'(s_j))_{j,k=1}^3$, $p_j'(s_k)$ is thus obtained from a matrix-vector multiplication

$$\begin{pmatrix} p_j'(-1) \\ p_j'(0) \\ p_j'(+1) \end{pmatrix} = \begin{pmatrix} L_1'(-1) & L_2'(-1) & L_3'(-1) \\ L_1'(0) & L_2'(0) & L_3'(0) \\ L_1'(+1) & L_2'(+1) & L_3'(+1) \end{pmatrix} \begin{pmatrix} v(a_j) \\ v(m_j) \\ v(b_j) \end{pmatrix} = \begin{pmatrix} -3/2 & +2 & -1/2 \\ -1/2 & 0 & +1/2 \\ +1/2 & -2 & +3/2 \end{pmatrix} \begin{pmatrix} v(a_j) \\ v(m_j) \\ v(b_j) \end{pmatrix}.$$

For the computation of the local Dirichlet data oscillations

$$\mathrm{osc}_{D,\ell}(E_j)^2 = h\,\|(g - G_\ell)'\|_{L^2(E_j)}^2 = 2\int_{-1}^{1} \big((g - G_\ell)\circ\gamma_j)'(s)\big)^2 \, ds,$$

we have $v = g - G_\ell$. This results in $(g - G_\ell)(a_j) = 0 = (g - G_\ell)(b_j)$ by definition of the nodal interpolant G_ℓ. Consequently, everything simplifies to

$$\begin{pmatrix} p_j'(-1) \\ p_j'(0) \\ p_j'(+1) \end{pmatrix} = \begin{pmatrix} 2\,v(m_j) \\ 0 \\ -2\,v(m_j) \end{pmatrix} = \Big(g(m_j) - \frac{g(a_j) + g(b_j)}{2}\Big) \begin{pmatrix} +2 \\ 0 \\ -2 \end{pmatrix} = \big(g(a_j) + g(b_j) - 2\,g(m_j)\big) \begin{pmatrix} -1 \\ 0 \\ +1 \end{pmatrix}.$$

Note that the weights of the Newton-Côtes formula read

$$\omega_k = \int_{-1}^{1} L_k(t)\, dt, \quad \text{whence} \quad \omega_1 = 1/3, \quad \omega_2 = 4/3, \quad \omega_3 = 1/3.$$

Therefore,

$$\begin{aligned} \mathrm{osc}_{D,\ell}(E_j)^2 \approx \widetilde{\mathrm{osc}}_{D,\ell}(E_j)^2 := 2\,\mathrm{quad}_2\big((p_j')^2\big) &= 2\sum_{k=1}^{3} \omega_k \big(p_j'(s_k)\big)^2 \\ &= \frac{4}{3}\big(g(a_j) + g(b_j) - 2\,g(m_j)\big)^2. \end{aligned} \tag{5.15}$$

Altogether, the documentation of Listing 4 now reads as follows:

- The function takes the mesh $\mathcal{E}_\ell$ in terms of `coordinates` and `elements` as well as a function handle `uD` for the Dirichlet data g (Line 1).
- We first compute all element midpoints (Line 3) and evaluate the Dirichlet data g at all midpoints (Line 6) and all nodes (Line 9).

- Finally, Formula (5.15) is realized (Line 12) simultaneously for all elements $E_j \in \mathcal{E}_\ell$.
- The function returns the column vector of elementwise Dirichlet data oscillations
$$\mathbf{v} := \big(\widetilde{\mathrm{osc}}_{D,\ell}(E_1)^2, \dots, \widetilde{\mathrm{osc}}_{D,\ell}(E_N)^2\big) \in \mathbb{R}^N$$
so that $\mathrm{osc}_{D,\ell} \approx \big(\sum_{j=1}^N \mathbf{v}_j\big)^{1/2}$.

Remark 5.1. *For smooth Dirichlet data g and uniform meshes with mesh-size h, there holds*
$$\mathrm{osc}_{D,\ell} = \mathcal{O}(h^{3/2}) \quad \textit{and} \quad \Big|\mathrm{osc}_{D,\ell} - \Big(\sum_{j=1}^N \mathbf{v}_j\Big)^{1/2}\Big| = \mathcal{O}(h^{5/2}).$$
Therefore, the quadrature error is of higher order when compared to the discretization order.

Listing 5

```
function err = computeErrNeumann(coordinates,elements,p,phi)
%*** arbitrary quadrature on [-1,1] with exactness n >= 2, e.g., gauss(2)
quad_nodes = [-1 1]/sqrt(3);
quad_weights = [1;1];

%*** the remaining code is independent of the chosen quadrature rule
nE = size(elements,1);
nQ = length(quad_nodes);

%*** build vector of evaluations points as (nQ*nE x 2)-matrix
a = coordinates(elements(:,1),:);
b = coordinates(elements(:,2),:);
sx = reshape(a,2*nE,1)*(1-quad_nodes) + reshape(b,2*nE,1)*(1+quad_nodes);
sx = 0.5*reshape(sx',nQ*nE,2);

%*** phi(sx) usually depends on the normal vector, whence phi takes sx and the
%*** nodes of the respective element to compute the normal
a_sx = reshape(repmat(reshape(a,2*nE,1),1,nQ)',nE*nQ,2);
b_sx = reshape(repmat(reshape(b,2*nE,1),1,nQ)',nE*nQ,2);

%*** perform all necessary evaluations of phi as (nE x nQ)-matrix
phi_sx = reshape(phi(sx,a_sx,b_sx),nQ,nE)';

%*** compute vector of (squared) element-widths
h = sum((a-b).^2,2);

%*** compute Neumann error simultaneously for all elements
err_sx = (phi_sx - repmat(reshape(p,nE,1),1,nQ)).^2;
err = 0.5*h.*(err_sx*quad_weights);
```

5.4. Computation of Reliable Error Bound for $\|\phi - \Phi_\ell\|_V$ (Listing 5).

We assume that the exact Neumann data satisfy $\phi \in L^2(\Gamma)$. Let $\Phi_\ell^* \in \mathcal{P}^0(\mathcal{E}_\ell)$ be the (only theoretically computed) Galerkin solution with respect to the non-perturbed right-hand side $(K + 1/2)g$ instead of $(K + 1/2)G_\ell$. Let Π_ℓ denote the L^2-orthogonal projection onto $\mathcal{P}^0(\mathcal{E}_\ell)$. With the technique from [FP, AGP], we obtain
$$\|\phi - \Phi_\ell^*\|_V \le \|\phi - \Pi_\ell\phi\|_V \lesssim \|h_\ell^{1/2}(\phi - \Pi_\ell\phi)\|_{L^2(\Gamma)}$$
as well as
$$\|\Phi_\ell^* - \Phi_\ell\|_V \lesssim \mathrm{osc}_{D,\ell},$$

where $\mathrm{osc}_{D,\ell}$ denote the Dirichlet data oscillations from Section 5.3. Note that Π_ℓ is even the $\mathcal{E}_\ell$-elementwise best approximation operator. With the triangle inequality, we therefore obtain

$$|||\phi - \Phi_\ell|||_V \lesssim \|h_\ell^{1/2}(\phi - \Phi_\ell)\|_{L^2(\Gamma)} + \mathrm{osc}_{D,\ell} =: \mathrm{err}_{N,\ell} + \mathrm{osc}_{D,\ell}.$$

In this section, we aim to numerically compute

$$\mathrm{err}_{N,\ell} = \Big(\sum_{j=1}^N \mathrm{err}_{N,\ell}(E_j)^2\Big)^{1/2}, \quad \text{where} \quad \mathrm{err}_{N,\ell}(E_j)^2 = \mathrm{length}(E_j)\,\|\phi - \Phi_\ell\|_{L^2(E_j)}^2.$$

With $\mathbf{x} \in \mathbb{R}^N$ the coefficient vector of

$$\Phi_\ell = \sum_{j=1}^N \mathbf{x}_j \chi_j,$$

there holds

$$\begin{aligned}\mathrm{err}_{N,\ell}(E_j)^2 = \mathrm{length}(E_j)\int_{E_j} |\phi - \mathbf{x}_j|^2\,d\Gamma &= \frac{\mathrm{length}(E_j)^2}{2}\int_{-1}^1 |\phi\circ\gamma_j(s) - \mathbf{x}_j|^2\,ds \\ &\approx \frac{\mathrm{length}(E_j)^2}{2}\,\mathrm{quad_n}\big((\phi\circ\gamma_j - \mathbf{x}_j)^2\big) =: \widetilde{\mathrm{err}}_{N,\ell}(E_j)^2, \qquad (5.16)\end{aligned}$$

where $\mathrm{quad}_n(\cdot)$ denotes a quadrature rule on $[-1,1]$ which is exact for polynomials of degree n, i.e. $\mathrm{quad}_n(p) = \int_{-1}^1 p\,ds$ for all $p \in \mathcal{P}^n[-1,1]$. With the definition $\widetilde{\mathrm{err}}_{N,\ell} := \big(\sum_{j=1}^N \widetilde{\mathrm{err}}_{N,\ell}(E_j)^2\big)^{1/2}$, one can then prove that

$$|\mathrm{err}_{N,\ell} - \widetilde{\mathrm{err}}_{N,\ell}| = \mathcal{O}(h^{n/2+1}).$$

For smooth ϕ, there holds $\mathrm{err}_{N,\ell} = \mathcal{O}(h^{3/2})$. For our implementation, we thus choose the Gauss quadrature with two nodes, which is exact for polynomials of degree $n = 3$. As for the Dirichlet data oscillations, this choice then leads to

$$|\mathrm{err}_{N,\ell} - \widetilde{\mathrm{err}}_{N,\ell}| = \mathcal{O}(h^{5/2}), \quad \text{whereas at most} \quad \mathrm{err}_{N,\ell} = \mathcal{O}(h^{3/2}),$$

i.e. our implementation is accurate up to higher-order terms. The documentation of Listing 5 now simply reads as follows:

- The function takes the given mesh $\mathcal{E}_\ell$ in form of the arrays `coordinates` and `elements`, the coefficient vector $\mathtt{p} = \mathbf{x}$ as well as a function handle `phi` for the Neumann data. The function `phi` is called by
  ```
      y = phi(x,a,b)
  ```
  with $(n \times 2)$-arrays `x`, `a`, and `b`. The j-th rows $\mathtt{x}(j,:)$, $\mathtt{a}(j,:)$, and $\mathtt{b}(j,:)$ correspond to a point $x_j \in [a_j, b_j] \subset \mathbb{R}^2$. The entry $\mathtt{y}(j)$ of the column vector `y` then contains $\phi(x_j)$.
- As stated above, we use the Gauss quadrature with two nodes (Line 3–4).
- If $s_k \in [-1,1]$ is a quadrature node and $E_j = [a_j, b_j] \in \mathcal{E}_\ell = \{E_1, \dots, E_N\}$ is an element, the function ϕ has to be evaluated at

  $$\gamma_j(s_k) = \frac{1}{2}\big(a_j + b_j + s_k(b_j - a_j)\big) = \frac{1}{2}\big(a_j(1 - s_k) + b_j(1 + s_k)\big).$$

  In Line 11–14, we build the $(2N \times 2)$-array `sx` which contains all necessary evaluation points. Note that the two evaluation points at E_j are stored in $\mathtt{sx}(2j-1,:)$ and $\mathtt{sx}(2j,:)$.
- In Line 18–19, we compute the $(2N\times 2)$-arrays `a_sx` and `b_sx` such that, e.g., $\mathtt{a\_sx}(2j-1,:)$ and $\mathtt{a\_sx}(2j,:)$ contain the first node $a_j \in \mathbb{R}^2$ of the boundary element $E_j = [a_j, b_j]$.
- We then evaluate the Neumann data ϕ simultaneously in all evaluation points and we reshape this $(2N \times 1)$-array into a $(N \times 2)$-array `phi_sx` such that $\mathtt{phi\_sx}(j,:)$ contains all ϕ-values related to E_j (Line 22).

- We realize Equation (5.16). We first derive the necessary evaluations of $(\phi - \mathbf{x}_j)^2$ in Line 28. Multiplication with the quadrature weights and coefficient-wise weighting with $\text{length}(E_j)^2/2$ provides the $(N\times 1)$-array `err` such that `err`$(j) \approx \text{length}(E_j)\|\phi-\Phi_\ell\|_{L^2(E_j)}^2$. More precisely, there holds $\text{err}_{N,\ell}^2 \approx \sum_{j=1}^N$ `err`$(j) = \widetilde{\text{err}}_{N,\ell}^2$.

Remark 5.2. *In academic experiments, the exact solution ϕ is usually known and has certain regularity $\phi \in L^2(\Gamma)$ which only depends on the geometry of Γ. As explained before, there holds*

$$|||\phi - \Phi_\ell|||_V \lesssim \text{err}_{N,\ell} + \text{osc}_{D,\ell}$$

so that we can control the error reliably. Moreover, the convergence $\text{err}_{N,\ell} \to 0$ as $\ell \to \infty$ might indicate that there are no major bugs in the implementation — since we compare the Galerkin solution with the exact solution. □

5.5. Computation of $(h - h/2)$-Based A Posteriori Error Estimators. In this section, we discuss the implementation of four error estimators introduced and analyzed in [FP]. Let $\widehat{\mathcal{E}}_\ell = \{e_1, \dots, e_{2N}\}$ be the uniform refinement of the mesh $\mathcal{E}_\ell = \{E_1, \dots, E_N\}$. Let $\Phi_\ell \in \mathcal{P}^0(\mathcal{E}_\ell)$ and $\widehat{\Phi}_\ell \in \mathcal{P}^0(\widehat{\mathcal{E}}_\ell)$ be the Galerkin solutions (5.6) with respect to $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$ and the same approximate Dirichlet data G_ℓ, i.e., there holds

$$\langle V\Phi_\ell\,,\,\Psi_\ell\rangle_V = \langle (K+1/2)G_\ell\,,\,\Psi_\ell\rangle_\Gamma \quad \text{for all } \Psi_\ell \in \mathcal{P}^0(\mathcal{E}_\ell)$$

as well as

$$\langle V\widehat{\Phi}_\ell\,,\,\widehat{\Psi}_\ell\rangle_V = \langle (K+1/2)G_\ell\,,\,\widehat{\Psi}_\ell\rangle_\Gamma \quad \text{for all } \widehat{\Psi}_\ell \in \mathcal{P}^0(\widehat{\mathcal{E}}_\ell).$$

With $\phi_\ell \in H^{-1/2}(\Gamma)$ the exact solution of (5.12), one can expect

$$|||\phi_\ell - \Phi_\ell|||_V \approx |||\widehat{\Phi}_\ell - \Phi_\ell|||_V =: \eta_\ell, \tag{5.17}$$

which results in

$$|||\phi - \Phi_\ell|||_V \le |||\phi - \phi_\ell|||_V + |||\phi_\ell - \Phi_\ell|||_V \lesssim \text{osc}_{D,\ell} + \eta_\ell \tag{5.18}$$

according to the triangle inequality and (5.13).

Clearly, the Galerkin solution $\widehat{\Phi}_\ell$ with respect to the uniformly refined mesh $\widehat{\mathcal{E}}_\ell$ is more accurate than Φ_ℓ. Consequently, any algorithm will return $\widehat{\Phi}_\ell$ instead of Φ_ℓ if $\widehat{\Phi}_\ell$ has been computed. From this point of view, Φ_ℓ then becomes a side result and leads to unnecessary computational effort. One can prove that one may replace Φ_ℓ by a cheap (but appropriate) postprocessing $\Pi_\ell\widehat{\Phi}_\ell$ of $\widehat{\Phi}_\ell$. This leads to some error estimator

$$\eta_\ell \sim |||\widehat{\Phi}_\ell - \Pi_\ell\widehat{\Phi}_\ell|||_V =: \widetilde{\eta}_\ell \tag{5.19}$$

which always stays proportional to η_ℓ, indicated by $\eta_\ell \sim \widetilde{\eta}_\ell$. To be more precise, Π_ℓ denotes the L^2-orthogonal projection onto $\mathcal{P}^0(\mathcal{E}_\ell)$, which simply reads

$$(\Pi_\ell\widehat{\Phi}_\ell)|_{E_i} = \frac{1}{\text{length}(E_i)}\int_{E_i} \widehat{\Phi}_\ell\,d\Gamma \quad \text{for all } E_i \in \mathcal{E}_\ell \tag{5.20}$$

in case of the lowest-order discretization, i.e. piecewise constant ansatz and test functions.

However, one essential drawback of the error estimators η_ℓ and $\widetilde{\eta}_\ell$ is that they do not provide an additional information on the local errors, i.e., the error $|||\phi - \Phi_\ell|||_V$ related to some element $E_i \in \mathcal{E}_\ell$. This is different for the error estimators μ_ℓ and $\widetilde{\mu}_\ell$ discussed in the following. For instance, one can prove that

$$\eta_\ell \sim \mu_\ell := \|h_\ell^{1/2}(\widehat{\Phi}_\ell - \Phi_\ell)\|_{L^2(\Gamma)} = \Big(\sum_{i=1}^N \text{length}(E_i)\|\widehat{\Phi}_\ell - \Phi_\ell\|_{L^2(E_i)}^2\Big)^{1/2}. \tag{5.21}$$

Then, the local contributions

$$\mu_\ell(E_i) := \text{length}(E_i)^{1/2}\|\widehat{\Phi}_\ell - \Phi_\ell\|_{L^2(E_i)} \quad \text{for all } E_i \in \mathcal{E}_\ell \tag{5.22}$$

give some measure for the error on E_i.

As the computation of the error estimator η_ℓ, the computation of μ_ℓ needs the computation of two Galerkin solutions Φ_ℓ and $\widehat{\Phi}_\ell$. As before, the computation of the coarse-mesh solution Φ_ℓ can be avoided by use of the projected fine mesh solution $\Pi_\ell\widehat{\Phi}_\ell$. One can mathematically prove that

$$\eta_\ell \sim \widetilde{\mu}_\ell := \|h_\ell^{1/2}(\widehat{\Phi}_\ell - \Pi_\ell\widehat{\Phi}_\ell)\|_{L^2(\Gamma)}. \tag{5.23}$$

In the following subsections, we first discuss the computation of the *global* error estimators η_ℓ and $\widetilde{\eta}_\ell$ from (5.17) and (5.19). Then, we give an implementation of the *local* error estimators μ_ℓ and $\widetilde{\mu}_\ell$ from (5.21) and (5.23), where our functions return the local contributions, see e.g. (5.22), to steer an adaptive mesh-refinement.

Remark 5.3. *If we plot the error estimators η_ℓ, $\widetilde{\eta}_\ell$, μ_ℓ and $\widetilde{\mu}_\ell$ over the number of elements, one can mathematically predict that the corresponding curves, for a sequence of arbitrarily refined meshes, are parallel. In mathematical terms, this reads*

$$\eta_\ell \le \widetilde{\eta}_\ell \lesssim \widetilde{\mu}_\ell \le \mu_\ell \lesssim \eta_\ell, \tag{5.24}$$

cf. [EFFP, FP]. *Empirically, one observes a very good coincidence of η_ℓ and $\widetilde{\eta}_\ell$ in the sense that the corresponding curves almost coincide. The same is observed for the curves of μ_ℓ and $\widetilde{\mu}_\ell$.* □

Remark 5.4. *Mathematically, the error estimate* (5.18) *involves the so-called saturation assumption: Assume that we could compute the Galerkin solutions Φ_ℓ^* and $\widehat{\Phi}_\ell^*$ with respect to $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$ for the non-perturbed variational formulation* (5.3)*, i.e., we formally use the exact Dirichlet data g instead of the interpolated data G_ℓ — although the right-hand side is, in fact, non-computable because of Kg. Then, the saturation assumption states that*

$$\|\phi - \widehat{\Phi}_\ell^*\|_V \le q\,\|\phi - \Phi_\ell^*\|_V \tag{5.25}$$

with some uniform and ℓ-independent constant $q \in (0,1)$. —Put differently, uniform mesh-refinement leads to a uniform improvement of the discretization error.— Provided (5.25)*, one can prove that*

$$\eta_\ell \le \|\phi_\ell - \Phi_\ell\|_V \le (1-q^2)^{-1/2}\,\eta_\ell \tag{5.26}$$

which is the mathematical basis of (5.17)*, cf.* [FP].

We stress that this assumption is somewhat natural and can, for instance, be proven for the finite element method [DN]*, see also* [FOP, Section 2.3]*. For the boundary element method, however,* (5.25) *still remains open.*

Finally, one can prove that (5.25) *is sufficient and in some sense even necessary to guarantee* (5.18)*.* □

Remark 5.5. *In academic experiments, the exact solution ϕ is usually known and has certain regularity $\phi \in L^2(\Gamma)$ which only depends on the geometry of Γ. In this case, one can experimentally verify the saturation assumption as follows: In Section 5.4, we derived*

$$\|\phi - \Phi_\ell\|_V \lesssim \mathrm{err}_{N,\ell} + \mathrm{osc}_{D,\ell}.$$

If the right-hand side has the same convergence behaviour as the error estimator $\eta_\ell + \mathrm{osc}_{D,\ell}$, this proves empirically

$$\|\phi - \Phi_\ell\|_V \lesssim \eta_\ell + \mathrm{osc}_{D,\ell}$$

and confirms the saturation assumption. □

Listing 6

```
function est = computeEstSlpEta(father2son,V_fine,x_fine,x_coarse)
```

```
2 %*** compute coefficient vector of (phi_fine - phi_coarse) w.r.t. to fine mesh
3 x_fine(father2son(:,1)) = x_fine(father2son(:,1)) - x_coarse;
4 x_fine(father2son(:,2)) = x_fine(father2son(:,2)) - x_coarse;
5
6 %*** compute energy ||| phi_fine - phi_coarse |||^2
7 est = x_fine'*(V_fine*x_fine);
```

5.5.1. Computation of Error Estimator η_ℓ (Listing 6). In this section, we aim to compute the error estimator $\eta_\ell = \|\widehat{\Phi}_\ell - \Phi_\ell\|_V$ from (5.17). Let $\widehat{\chi}_j$ denote the characteristic function associated with some fine-mesh element $e_j \in \widehat{\mathcal{E}}_\ell$. Let $\mathbf{x} \in \mathbb{R}^N$ and $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ be the coefficient vectors of Φ_ℓ and $\widehat{\Phi}_\ell$ with respect to the canonical bases of $\mathcal{P}^0(\mathcal{E}_\ell)$ and $\mathcal{P}^0(\widehat{\mathcal{E}}_\ell)$, i.e.

$$\Phi_\ell = \sum_{j=1}^{N} \mathbf{x}_j \chi_j \quad \text{and} \quad \widehat{\Phi}_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{x}}_j \widehat{\chi}_j.$$

Because of $\mathcal{P}^0(\mathcal{E}_\ell) \subset \mathcal{P}^0(\widehat{\mathcal{E}}_\ell)$, there is a unique vector $\widehat{\mathbf{y}} \in \mathbb{R}^{2N}$ such that

$$\Phi_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{y}}_j \widehat{\chi}_j.$$

With the vectors $\widehat{\mathbf{x}}, \widehat{\mathbf{y}} \in \mathbb{R}^{2N}$, there holds

$$\begin{aligned}\eta_\ell^2 = \|\widehat{\Phi}_\ell - \Phi_\ell\|_V^2 = \langle\!\langle \widehat{\Phi}_\ell - \Phi_\ell \,, \widehat{\Phi}_\ell - \Phi_\ell \rangle\!\rangle_V &= \sum_{j,k=1}^{2N} (\widehat{\mathbf{x}}_j - \widehat{\mathbf{y}}_j)(\widehat{\mathbf{x}}_k - \widehat{\mathbf{y}}_k) \langle\!\langle \widehat{\chi}_j \,, \widehat{\chi}_k \rangle\!\rangle_V \\ &= (\widehat{\mathbf{x}} - \widehat{\mathbf{y}}) \cdot \widehat{\mathbf{V}} (\widehat{\mathbf{x}} - \widehat{\mathbf{y}}),\end{aligned}$$

where $\widehat{\mathbf{V}}$ is the matrix for the simple-layer potential (5.8) with respect to the fine mesh $\widehat{\mathcal{E}}_\ell$. With these observations, the documentation of Listing 6 reads as follows:

- The function takes the coefficient vectors $\mathbf{x} \in \mathbb{R}^N$ and $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ of the Galerkin solutions Φ_ℓ and $\widehat{\Phi}_\ell$ as well as the simple-layer potential matrix $\widehat{\mathbf{V}}$ for the fine mesh $\widehat{\mathcal{E}}_\ell$. Besides this, the $(N \times 2)$-array `father2son` links the indices of elements $E_i \in \mathcal{E}_\ell$ with the indices of the sons $e_j, e_k \in \widehat{\mathcal{E}}_\ell$ in the sense that `father2son(`i`,:) = [`j`,`k`]` for $E_i = e_j \cup e_k$ and consequently $\widehat{\mathbf{y}}_j = \widehat{\mathbf{y}}_k = \mathbf{x}_i$.
- We overwrite the vector $\widehat{\mathbf{x}}$ by the coefficient vector $\widehat{\mathbf{x}} - \widehat{\mathbf{y}}$ of $\widehat{\Phi}_\ell - \Phi_\ell$ (Line 3–4).
- Finally, the function returns $\eta_\ell^2 = \|\widehat{\Phi}_\ell - \Phi_\ell\|_V^2$ (Line 7).

Listing 7

```
function est = computeEstSlpEtaTilde(father2son,V_fine,x_fine)
%*** compute L2-projection Pi_coarse*phi_fine onto coarse mesh
pi_x_fine = 0.5*( x_fine(father2son(:,1)) + x_fine(father2son(:,2)) );

%*** compute coefficient vector of (1-Pi_coarse)*phi_fine
x_fine(father2son(:,1)) = x_fine(father2son(:,1)) - pi_x_fine;
x_fine(father2son(:,2)) = x_fine(father2son(:,2)) - pi_x_fine;

%*** compute energy ||| (1-Pi_coarse)*phi_fine |||^2
est = x_fine'*(V_fine*x_fine);
```

5.5.2. Computation of Error Estimator $\widetilde{\eta}_\ell$ (Listing 7). We adopt the notation of Section 5.5.1 for the computation of η_ℓ, namely $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ with

$$\widehat{\Phi}_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{x}}_j \widehat{\chi}_j.$$

Let $e_j, e_k \in \widehat{\mathcal{E}}_\ell$ be the sons of $E_i \in \mathcal{E}_\ell$, i.e. $E_i = e_j \cup e_k$. Then,

$$\int_{E_i} \widehat{\Phi}_\ell \, d\Gamma = \int_{e_j} \widehat{\Phi}_\ell \, d\Gamma + \int_{e_k} \widehat{\Phi}_\ell \, d\Gamma = \text{length}(e_j)\, \widehat{\mathbf{x}}_j + \text{length}(e_k)\, \widehat{\mathbf{x}}_k = \text{length}(E_i)\, \frac{\widehat{\mathbf{x}}_j + \widehat{\mathbf{x}}_k}{2}.$$

Put differently, there holds

$$(\Pi_\ell \widehat{\Phi}_\ell)|_{E_i} = \frac{\widehat{\mathbf{x}}_j + \widehat{\mathbf{x}}_k}{2} \quad \text{for all } E_i \in \mathcal{E}_\ell \quad \text{with} \quad E_i = e_j \cup e_k \text{ and } e_j, e_k \in \widehat{\mathcal{E}}_\ell$$

for the L^2-projection Π_ℓ defined in (5.20). Representing $\Pi_\ell \widehat{\Phi}_\ell \in \mathcal{P}^0(\mathcal{E}_\ell)$ with respect to the fine-mesh $\widehat{\mathcal{E}}_\ell$, we obtain

$$\Pi_\ell \widehat{\Phi}_\ell = \sum_{n=1}^{2N} \widehat{\mathbf{z}}_n \widehat{\chi}_n,$$

where the vector $\widehat{\mathbf{z}} \in \mathbb{R}^{2N}$ satisfies $\widehat{\mathbf{z}}_j = \widehat{\mathbf{z}}_k = \frac{\widehat{\mathbf{x}}_j + \widehat{\mathbf{x}}_k}{2}$ provided that $e_j, e_k \in \widehat{\mathcal{E}}_\ell$ are the sons of some element $E_i \in \mathcal{E}_\ell$. As in Section 5.5.1, there holds

$$\widetilde{\eta}_\ell^2 = \|\widehat{\Phi}_\ell - \Pi_\ell \widehat{\Phi}_\ell\|_V^2 = (\widehat{\mathbf{x}} - \widehat{\mathbf{z}}) \cdot \widehat{\mathbf{V}}(\widehat{\mathbf{x}} - \widehat{\mathbf{z}}).$$

Therefore, the documentation of Listing 7 reads as follows:

- The function takes the simple-layer potential matrix $\widehat{\mathbf{V}}$ for the fine mesh $\widehat{\mathcal{E}}_\ell$ and the coefficient vector $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ of $\widehat{\Phi}_\ell$. Moreover, the link between $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$ is provided by means of `father2son`.
- We first compute the coefficient vector of $\Pi_\ell \widehat{\Phi}_\ell$ with respect to the coarse mesh $\mathcal{E}_\ell$ (Line 3).
- We then overwrite $\widehat{\mathbf{x}}$ by the coefficient vector $\widehat{\mathbf{x}} - \widehat{\mathbf{z}} \in \mathbb{R}^{2N}$ of $\widehat{\Phi}_\ell - \Pi_\ell \widehat{\Phi}_\ell$ (Line 6–7).
- Finally, the function returns $\widetilde{\eta}_\ell^2 = \|\widehat{\Phi}_\ell - \Pi_\ell \widehat{\Phi}_\ell\|_V^2$ (Line 10).

Listing 8

```
function ind = computeEstSlpMu(coordinates,elements,father2son,x_fine,x_coarse)
%*** compute (squared) local mesh-size
h = sum((coordinates(elements(:,1),:) - coordinates(elements(:,2),:)).^2,2);

%*** compute coefficient vector of (phi_fine - phi_coarse) w.r.t. to fine mesh
x_fine(father2son(:,1)) = x_fine(father2son(:,1)) - x_coarse;
x_fine(father2son(:,2)) = x_fine(father2son(:,2)) - x_coarse;

%*** compute ind(j) = diam(Ej)*|| phi_fine - phi_coarse ||_{L2(Ej)}^2
ind = 0.5*h.*( x_fine(father2son(:,1)).^2 + x_fine(father2son(:,2)).^2 );
```

5.5.3. Computation of Error Estimator μ_ℓ (Listing 8). In this section, we discuss the implementation of

$$\mu_\ell^2 = \sum_{i=1}^{N} \mu_\ell(E_i)^2, \quad \text{where} \quad \mu_\ell(E_i)^2 = \text{length}(E_i)\, \|\widehat{\Phi}_\ell - \Phi_\ell\|_{L^2(E_i)}^2.$$

We adopt the notation of Section 5.5.1, namely $\widehat{\mathbf{x}}, \widehat{\mathbf{y}} \in \mathbb{R}^{2N}$ with

$$\widehat{\Phi}_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{x}}_j \widehat{\chi}_j \quad \text{and} \quad \Phi_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{y}}_j \widehat{\chi}_j.$$

For fixed $E_i \in \mathcal{E}_\ell$ and sons $e_j, e_k \in \widehat{\mathcal{E}}_\ell$ with $E_i = e_j \cup e_k$, we obtain

$$\|\widehat{\Phi}_\ell - \Phi_\ell\|_{L^2(E_i)}^2 = \int_{e_j} (\widehat{\Phi}_\ell - \Phi_\ell)^2 \, d\Gamma + \int_{e_k} (\widehat{\Phi}_\ell - \Phi_\ell)^2 \, d\Gamma = \frac{\text{length}(E_i)}{2} \big((\widehat{\mathbf{x}}_j - \widehat{\mathbf{y}}_j)^2 + (\widehat{\mathbf{x}}_k - \widehat{\mathbf{y}}_k)^2 \big).$$

This implies

$$\mu_\ell(E_i)^2 = \frac{\text{length}(E_i)^2}{2} \big((\widehat{\mathbf{x}}_j - \widehat{\mathbf{y}}_j)^2 + (\widehat{\mathbf{x}}_k - \widehat{\mathbf{y}}_k)^2 \big). \tag{5.27}$$

Altogether, the documentation of Listing 8 reads as follows:

- As input arguments, the function takes the mesh $\mathcal{E}_\ell$, the link between $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$, and the coefficient vectors $\mathbf{x} \in \mathbb{R}^N$ and $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ of the Galerkin solutions Φ_ℓ and $\widehat{\Phi}_\ell$ (Line 1).
- We compute the vector of all squared element-sizes (Line 3).
- We overwrite the coefficient vector $\widehat{\mathbf{x}}$ of $\widehat{\Phi}_\ell$ by the coefficient vector $\widehat{\mathbf{x}} - \widehat{\mathbf{y}}$ of $\widehat{\Phi}_\ell - \Phi_\ell$ (Line 6–7).
- Finally (Line 10), the function realizes (5.27) and returns the vector

  $$\mathbf{v} := (\mu_\ell(E_1)^2, \dots, \mu_\ell(E_N)^2) \in \mathbb{R}^N$$

  so that $\mu_\ell = \big(\sum_{i=1}^N \mathbf{v}_i \big)^{1/2}$.

Listing 9

```
 1 function ind = computeEstSlpMuTilde(coordinates,elements,father2son,x_fine)
 2 %*** compute (squared) local mesh-size
 3 h = sum((coordinates(elements(:,1),:) - coordinates(elements(:,2),:)).^2,2);
 4
 5 %*** compute L2-projection Pi_coarse*phi_fine onto coarse mesh
 6 pi_x_fine = 0.5*( x_fine(father2son(:,1)) + x_fine(father2son(:,2)) );
 7
 8 %*** compute coefficient vector of (1-Pi_coarse)*phi_fine
 9 x_fine(father2son(:,1)) = x_fine(father2son(:,1)) - pi_x_fine;
10 x_fine(father2son(:,2)) = x_fine(father2son(:,2)) - pi_x_fine;
11
12 %*** compute ind(j) = diam(Ej)*|| (1-Pi_coarse)*phi_fine ||_{L2(Ej)}^2
13 ind = 0.5*h.*( x_fine(father2son(:,1)).^2 + x_fine(father2son(:,2)).^2 );
```

5.5.4. Computation of Error Estimator $\widetilde{\mu}_\ell$ (Listing 9).

In this section, we finally aim to compute

$$\widetilde{\mu}_\ell^2 = \sum_{i=1}^N \widetilde{\mu}_\ell(E_i)^2, \quad \text{where} \quad \widetilde{\mu}_\ell(E_i)^2 = \text{length}(E_i) \, \|\widehat{\Phi}_\ell - \Pi_\ell \widehat{\Phi}_\ell\|_{L^2(E_i)}^2.$$

We adopt the notation of the preceding Sections 5.5.1–5.5.3, namely $\widehat{\mathbf{x}}, \widehat{\mathbf{z}} \in \mathbb{R}^{2N}$ with

$$\widehat{\Phi}_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{x}}_j \widehat{\chi}_j \quad \text{and} \quad \Pi_\ell \widehat{\Phi}_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{z}}_j \widehat{\chi}_j.$$

Based on this, the abbreviate documentation of Listing 9 reads as follows:

- The function takes the mesh $\mathcal{E}_\ell$, the link between $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$, and the coefficient vectors $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ of $\widehat{\Phi}_\ell$ (Line 1). It overwrites $\widehat{\mathbf{x}}$ by the coefficient vector $\widehat{\mathbf{x}} - \widehat{\mathbf{z}}$ of $\widehat{\Phi}_\ell - \Pi_\ell \widehat{\Phi}_\ell$ (Line 6–10).

- Finally (Line 13), the function returns the vector
$$\mathbf{v} := (\widetilde{\mu}_\ell(E_1)^2, \dots, \widetilde{\mu}_\ell(E_N)^2) \in \mathbb{R}^N.$$
In particular, there holds $\widetilde{\mu}_\ell = \big(\sum_{i=1}^N \mathbf{v}_i\big)^{1/2}$.

5.6. Adaptive Mesh-Refinement. Usually computing time and memory requirements are limiting quantities for numerical simulations. Therefore, one aims to choose the mesh such that it is coarse, where the (unknown) solution is smooth, and fine, where the (unknown) solution is singular. Based on a local error estimator, e.g. $\widetilde{\mu}_\ell$, such meshes are constructed in an iterative way. In each step, one refines the mesh only locally, i.e. one refines elements E_j, where the error appears to be large, namely, where the local contributions $\widetilde{\mu}_\ell(E_j)$ are large. For the error estimator $\widetilde{\mu}_\ell$ from Section 5.5.4, a possible adaptive algorithm reads as follows:

Input: Initial mesh $\mathcal{E}_0$, Dirichlet data g, adaptivity parameter $0 < \theta < 1$, maximal number $N_{\max} \in \mathbb{N}$ of elements, and counter $\ell = 0$.

(i) Build uniformly refined mesh $\widehat{\mathcal{E}}_\ell$.
(ii) Compute Galerkin solution $\widehat{\Phi}_\ell \in \mathcal{P}^0(\widehat{\mathcal{E}}_\ell)$.
(iii) Compute refinement indicators $\widetilde{\mu}_\ell(E)^2$ and oscillation terms $\mathrm{osc}_{D,\ell}(E)^2$ for all $E \in \mathcal{E}_\ell$.
(iv) Find minimal set $\mathcal{M}_\ell \subseteq \mathcal{E}_\ell$ such that

$$\theta\,(\widetilde{\mu}_\ell^2 + \mathrm{osc}_{D,\ell}^2) = \theta \sum_{E\in\mathcal{E}_\ell} \widetilde{\mu}_\ell(E)^2 + \mathrm{osc}_{D,\ell}(E)^2 \le \sum_{E\in\mathcal{M}_\ell} \widetilde{\mu}_\ell(E)^2 + \mathrm{osc}_{D,\ell}(E)^2. \tag{5.28}$$

(v) Refine at least marked elements $E \in \mathcal{M}_\ell$ and obtain mesh $\mathcal{E}_{\ell+1}$ with $\kappa(\mathcal{E}_{\ell+1}) \le 2\kappa(\mathcal{E}_0)$.
(vi) Stop provided that $\#\mathcal{E}_{\ell+1} \ge N_{\max}$; otherwise, increase counter $\ell \mapsto \ell+1$ and go to (i).

Output: Adaptively generated mesh $\widehat{\mathcal{E}}_\ell$ and corresponding discrete solution $\widehat{\Phi}_\ell \in \mathcal{P}^0(\widehat{\mathcal{E}}_\ell)$.

The marking criterion (5.28) has been proposed in the context of adaptive finite element methods [D]. Let formally $N_{\max} = \infty$ so that the adaptive algorithm computes a sequence of discrete solutions $\widehat{\Phi}_\ell$ (or even Φ_ℓ, although this is not computed). In [FOP, Section 3], we prove that the saturation assumption (5.25) implies convergence of $\widehat{\Phi}_\ell$ and Φ_ℓ to ϕ, provided that the right-hand side g is not disturbed, i.e., $g = G_\ell$. The same result also holds for $\widetilde{\mu}_\ell$ replaced by μ_ℓ.

In [AFP], we changed the notion of convergence and proved that for certain error estimators —amongst them are $\widetilde{\mu}_\ell$ and μ_ℓ— the adaptive algorithm guarantees $\lim_\ell \widetilde{\mu}_\ell = 0$. This concept is followed in [AGP] to prove that the adaptive algorithm stated above, yields $\lim_\ell(\mu_\ell^2 + \mathrm{osc}_{D,\ell}^2) = 0$. If the saturation assumption (5.25) holds (at least in infinitely many steps), we obtain convergence of Φ_ℓ to ϕ due to $|||\phi - \Phi_\ell|||_V^2 \lesssim \mu_\ell^2 + \mathrm{osc}_{D,\ell}^2$.

For adaptive finite element schemes, it could recently be proven that adaptive algorithms of this type even lead to quasi-optimal meshes [CKNS]. For adaptive BEM, such a result is completely open although numerical experiments give evidence for such an optimality result.

Listing 10

```
% adaptiveSymm provides the implementation of an adaptive mesh-refining
% algorithm for Symm's integral equation.

%*** maximal number of elements
nEmax = 100;

%*** adaptivity parameter
theta = 0.25;
rho = 0.25;

%*** adaptive mesh-refining algorithm
```

```
12 while size(elements,1) < nEmax
13 
14     %*** build uniformly refined mesh
15     [coordinates_fine,elements_fine,father2son] ...
16         = refineBoundaryMesh(coordinates,elements);
17 
18     %*** compute fine-mesh solution
19     V_fine = buildV(coordinates_fine,elements_fine);
20     b_fine = buildSymmRHS(coordinates_fine,elements_fine,@g);
21     x_fine = V_fine\b_fine;
22 
23     %*** compute (h-h/2)-error estimator tilde-mu
24     mu_tilde = computeEstSlpMuTilde(coordinates,elements,father2son, ...
25                                            x_fine);
26     %*** compute data oscillations
27     osc_fine = computeOscDirichlet(coordinates_fine,elements_fine,@g);
28     osc = osc_fine(father2son(:,1)) + osc_fine(father2son(:,2));
29 
30     %*** mark elements for refinement
31     marked = markElements(theta,rho,mu_tilde + osc);
32 
33     %*** generate new mesh
34     [coordinates,elements] = refineBoundaryMesh(coordinates,elements,marked);
35 end
```

5.6.1. Implementation of Adaptive Algorithm (Listing 10). The MATLAB script of Listing 10 realizes the adaptive algorithm from the beginning of this section.

- We use the adaptivity parameter $\theta = 1/4$ in (5.28) and mark at least the 25% of elements with the largest indicators (Line 8–9).
- Recall that the function **computeEstSlpMuTilde** as well as **computeOscDirichlet** return vectors of quadratic terms $\widetilde{\mu}_\ell(E)^2$ and $\mathrm{osc}_\ell(E)^2$, respectively. Note that (5.28) corresponds to the choice $\varrho_\ell(E) := \widetilde{\mu}_\ell(E)^2 + \mathrm{osc}_\ell(E)^2$ in (4.1). Therefore, the marking criterion (4.1) is provided by means of the function **markElements** (Line 31).

6. Hypersingular Integral Equation

Continuous Model Problem. In the entire section, we consider the hypersingular integral equation

$$Wu = (1/2 - K')\phi \quad \text{on } \Gamma \tag{6.1}$$

with W the hypersingular integral operator and K' the adjoint double-layer potential, where $\Gamma = \partial\Omega$ is the piecewise-affine boundary of a polygonal Lipschitz domain $\Omega \subset \mathbb{R}^2$. This integral equation is an equivalent formulation of the Neumann problem

$$-\Delta u = 0 \text{ in } \Omega \quad \text{with} \quad \partial_n u = \phi \text{ on } \Gamma. \tag{6.2}$$

Note that due to the Gauss Divergence Theorem there holds

$$\int_\Gamma \phi \, d\Gamma = \int_{\partial\Omega} \partial_n u \, d\Gamma = \int_\Omega \Delta u \, dx = 0.$$

Formally, the Neumann data satisfy $\phi \in H_*^{-1/2}(\Gamma)$, where the subscript abbreviates the constraint $\langle \phi \,,\, 1\rangle_\Gamma = 0$. We will, however, assume additional regularity $\phi \in C(\mathcal{E}_\ell) \subset L^2(\Gamma) \subset H^{-1/2}(\Gamma)$. The exact solution $u \in H^{1/2}(\Gamma)$ of the integral formulation (6.1) is just the Dirichlet data $u|_\Gamma$ of the solution $u \in H^1(\Omega)$ of (6.2).

Due to the fact that there holds $Wc = 0$ for all constant functions $c \in \mathbb{R}$, the solutions of (6.1) and (6.2) are only unique up to additive constants. To fix the additive constant, one usually assumes integral mean zero for the respective solutions. In this sense, (6.1) can equivalently be formulated in variational form: Find $u_* \in H_*^{1/2}(\Gamma) := \big\{ v \in H^{1/2}(\Gamma) \,:\, \int_\Gamma v \, d\Gamma = 0 \big\}$ such that

$$\langle W u_* \,,\, v_* \rangle_\Gamma = \langle (1/2 - K')\phi \,,\, v_* \rangle_\Gamma \quad \text{for all } v_* \in H_*^{1/2}(\Gamma). \tag{6.3}$$

One can prove that this formulation has a unique solution, since the left-hand side defines a scalar product on $H_*^{1/2}(\Gamma)$ even with equivalent norms.

From another point of view, one can consider the bilinear form

$$\langle\!\langle u \,, v \rangle\!\rangle_{W+S} := \langle W u \,,\, v \rangle_\Gamma + \Big(\int_\Gamma u \, d\Gamma \Big) \Big(\int_\Gamma v \, d\Gamma \Big) \quad \text{for all } u, v \in H^{1/2}(\Gamma), \tag{6.4}$$

which leads to the following modified variational form: Find $u \in H^{1/2}(\Gamma)$ such that

$$\langle\!\langle u \,, v \rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi \,,\, v \rangle_\Gamma \quad \text{for all } v \in H^{1/2}(\Gamma). \tag{6.5}$$

One can prove that $\langle\!\langle \cdot \,, \cdot \rangle\!\rangle_{W+S}$ from (6.4) defines a scalar product such that the induced norm $|\!|\!| u |\!|\!|_{W+S} := \langle\!\langle u \,, u \rangle\!\rangle_{W+S}^{1/2}$ is an equivalent norm on $H^{1/2}(\Gamma)$. Consequently, (6.5) has a unique solution u which depends continuously on the Neumann data ϕ. Moreover, one can prove that

$$\langle (1/2 - K')\psi \,,\, 1 \rangle_\Gamma = \frac{1}{2} \langle \psi \,,\, 1 \rangle_\Gamma - \langle \psi \,,\, K1 \rangle_\Gamma = \langle \psi \,,\, 1 \rangle_\Gamma = 0 \quad \text{for all } \psi \in H_*^{-1/2}(\Gamma).$$

If we plug in $v = 1$ in (6.5), we thus obtain

$$\begin{aligned} \Big(\int_\Gamma u \, d\Gamma \Big) \operatorname{length}(\Gamma) &= \langle W u \,,\, 1 \rangle_\Gamma + \Big(\int_\Gamma u \, d\Gamma \Big) \Big(\int_\Gamma 1 \, d\Gamma \Big) \\ &= \langle\!\langle u \,, 1 \rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi \,,\, 1 \rangle_\Gamma = 0 \end{aligned}$$

according to the fact that the kernel of the hypersingular integral operator W consists of constant functions. This implies $u \in H_*^{1/2}(\Gamma)$. For a test function $v_* \in H_*^{1/2}(\Gamma)$, the variational formulation (6.5) thus becomes

$$\langle W u \,,\, v_* \rangle_\Gamma = \langle\!\langle u \,, v_* \rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi \,,\, v_* \rangle_\Gamma,$$

i.e. (6.5) reduces to (6.3). Altogether we obtain that the unique solution u of (6.5) is also the unique solution of (6.3), i.e., (6.5) is an equivalent formulation of (6.3).

Galerkin Discretization. To discretize (6.5), we first replace the Neumann data $\phi \in H_*^{-1/2}(\Gamma) \cap L^2(\Gamma)$ by its L^2-projection $\Phi_\ell \in \mathcal{P}^0(\mathcal{E}_\ell)$,

$$\Phi_\ell|_{E_j} = \frac{1}{\mathrm{length}(E_j)} \int_{E_j} \phi \, d\Gamma =: \mathbf{p}_j \quad \text{for all } E_j \in \mathcal{E}_\ell. \tag{6.6}$$

According to this definition, there holds

$$\int_\Gamma \Phi_\ell \, d\Gamma = \sum_{E \in \mathcal{E}_\ell} \int_E \Phi_\ell \, d\Gamma = \sum_{E \in \mathcal{E}_\ell} \int_E \phi \, d\Gamma = \int_\Gamma \phi \, d\Gamma = 0,$$

i.e. there holds $\Phi_\ell \in H_*^{-1/2}(\Gamma)$, too. Second, we replace the function space $H^{1/2}(\Gamma)$ in (6.5) by the finite-dimensional space $\mathcal{S}^1(\mathcal{E}_\ell)$. Since $\mathcal{S}^1(\mathcal{E}_\ell)$ is a subspace of $H^{1/2}(\Gamma)$, $\langle\!\langle \cdot \, , \cdot \rangle\!\rangle_{W+S}$ from (6.4) is also a scalar product on $\mathcal{S}^1(\mathcal{E}_\ell)$. Consequently, there exists a unique Galerkin solution $U_\ell \in \mathcal{S}^1(\mathcal{E}_\ell)$ of the discretized problem

$$\langle\!\langle U_\ell \, , V_\ell \rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell \, , V_\ell \rangle_\Gamma \quad \text{for all } V_\ell \in \mathcal{S}^1(\mathcal{E}_\ell). \tag{6.7}$$

As in the continuous case, the discrete solution U_ℓ automatically satisfies $\int_\Gamma U_\ell \, d\Gamma = 0$ which follows from $1 = \sum_{j=1}^N \zeta_j \in \mathcal{S}^1(\mathcal{E}_\ell)$, which allows us to plug in $V_\ell = 1$ in (6.7). Indeed,

$$\begin{aligned}\Big(\int_\Gamma U_\ell \, d\Gamma\Big) \mathrm{length}(\Gamma) &= \langle W U_\ell \, , 1 \rangle_\Gamma + \Big(\int_\Gamma U_\ell \, d\Gamma\Big)\Big(\int_\Gamma 1 \, d\Gamma\Big) \\ &= \langle\!\langle U_\ell \, , 1 \rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell \, , 1 \rangle_\Gamma = 0.\end{aligned}$$

According to Linear Algebra, (6.7) holds for all $V_\ell \in \mathcal{S}^1(\mathcal{E}_\ell)$ if and only if it holds for all basis functions $\zeta_k \in \{\zeta_1, \dots, \zeta_N\}$ of $\mathcal{S}^1(\mathcal{E}_\ell)$. With $\mathbf{p} \in \mathbb{R}^N$ from (6.6) and the coefficient vector $\mathbf{x} \in \mathbb{R}^N$ of the ansatz

$$U_\ell = \sum_{j=1}^N \mathbf{x}_j \zeta_j, \tag{6.8}$$

the Galerkin formulation (6.7) is thus equivalent to

$$\sum_{j=1}^N \mathbf{x}_j \langle\!\langle \zeta_j \, , \zeta_k \rangle\!\rangle_{W+S} = \langle\!\langle U_\ell \, , \zeta_k \rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell \, , \zeta_k \rangle_\Gamma = \sum_{j=1}^N \mathbf{p}_j \langle (1/2 - K')\chi_j \, , \zeta_k \rangle_\Gamma \tag{6.9}$$

for all $k = 1, \dots, N$. In the context of Symm's integral equation of Section 5, we have already defined the matrices $\mathbf{K}, \mathbf{M} \in \mathbb{R}^{N \times N}$ by

$$\mathbf{K}_{jk} = \langle K\zeta_k \, , \chi_j \rangle_\Gamma \quad \text{and} \quad \mathbf{M}_{jk} = \langle \zeta_k \, , \chi_j \rangle_\Gamma,$$

cf. (5.8). The right-hand side of the last equation thus reads

$$\sum_{j=1}^N \mathbf{p}_j \langle (1/2 - K')\chi_j \, , \zeta_k \rangle_\Gamma = \frac{1}{2} \sum_{j=1}^N \mathbf{p}_j \langle \chi_j \, , \zeta_k \rangle_\Gamma - \sum_{j=1}^N \mathbf{p}_j \langle \chi_j \, , K\zeta_k \rangle_\Gamma = \frac{1}{2} (\mathbf{M}^T \mathbf{p})_k - (\mathbf{K}^T \mathbf{p})_k.$$

To compute the left-hand side of (6.9), we define matrices $\mathbf{W}, \mathbf{S} \in \mathbb{R}^{N \times N}$ by

$$\mathbf{W}_{kj} = \langle W\zeta_j \, , \zeta_k \rangle_\Gamma, \quad \mathbf{S}_{kj} = \Big(\int_\Gamma \zeta_j \, d\Gamma\Big)\Big(\int_\Gamma \zeta_k \, d\Gamma\Big) \quad \text{for all } j, k = 1, \dots, N. \tag{6.10}$$

Then there holds

$$\big((\mathbf{W} + \mathbf{S})\mathbf{x}\big)_k = \sum_{j=1}^N \mathbf{x}_j (\mathbf{W}_{kj} + \mathbf{S}_{kj}) = \langle W U_\ell \, , \zeta_k \rangle_\Gamma + \Big(\int_\Gamma U_\ell \, d\Gamma\Big)\Big(\int_\Gamma \zeta_k \, d\Gamma\Big) = \langle\!\langle U_\ell \, , \zeta_k \rangle\!\rangle_{W+S}.$$

Altogether, the Galerkin system (6.7) is equivalently stated by

$$(\mathbf{W} + \mathbf{S})\mathbf{x} = \frac{1}{2}\mathbf{M}^T \mathbf{p} - \mathbf{K}^T \mathbf{p}. \tag{6.11}$$

Note that the matrix $\mathbf{S}$ has rank 1 since it can be written in the form

$$\mathbf{S} = \mathbf{c}\mathbf{c}^T \quad \text{with the column vector} \quad \mathbf{c} \in \mathbb{R}^N \text{ with } \mathbf{c}_j := \int_\Gamma \zeta_j \, d\Gamma. \tag{6.12}$$

Finally, we stress that the matrix $\mathbf{W} + \mathbf{S}$ from (6.11) is symmetric and positive definite since it stems from a scalar product. Consequently, the linear system (6.11) has a unique solution $\mathbf{x} \in \mathbb{R}^N$.

6.1. Computation of Discrete Integral Operator W. The matrix $\mathbf{W} \in \mathbb{R}^{N\times N}_{sym}$ defined in (6.10) is implemented in the programming language C via the MATLAB-MEX-Interface. It is returned by call of

```
W = buildW(coordinates,elements [,eta]);
```

The entries of the matrix $\mathbf{W}$ are computed with the help of Nédélec's formula which is presented in the following identity

$$\langle Wu \, , \, v\rangle_\Gamma = \langle Vu' \, , \, v'\rangle_\Gamma \quad \text{for all } u, v \in H^1(\Gamma). \tag{6.13}$$

Since $\zeta_j' \in \mathcal{P}^0(\mathcal{E}_\ell)$, this gives a direct link between the matrices $\mathbf{W}$ and $\mathbf{V}$, namely, each entry of $\mathbf{W}$ is the weighted sum of four entries of $\mathbf{V}$. The optional parameter `eta` decides whether all entries of $\mathbf{W}$ are computed analytically or if certain double integrals are computed by numerical quadrature. We refer to Section 5.1 for details.

LISTING 11

```matlab
function S = buildHypsingStabilization(coordinates,elements)
nE = size(elements,1);

%*** compute local mesh-size
h = sqrt(sum((coordinates(elements(:,1),:)-coordinates(elements(:,2),:)).^2,2));

%*** build vector with entries c(j) = int_Gamma hatfunction(j) ds
c = 0.5*accumarray(reshape(elements,2*nE,1),[h;h]);

%*** build stabilization matrix
S = c*c';
```

6.2. Compute Stabilization for Hypersingular Integral Equation (Listing 11). The kernel of the hypersingular integral operator W is the space of constant functions. Since $1 = \sum_{j=1}^N \zeta_j \in \mathcal{S}^1(\mathcal{E}_\ell)$, the corresponding matrix $\mathbf{W}$ defined by $\mathbf{W}_{kj} = \langle W\zeta_j \, , \, \zeta_k\rangle_\Gamma$ for all $j, k \in \{1, \dots, N\}$ cannot be regular. One can prove, however, that it is semi-positive definite. As we have figured out in the introduction, one remedy is to consider the extended bilinear form $\langle\!\langle \cdot \, , \cdot \rangle\!\rangle_{W+S}$ from (6.4). It thus remains to assemble the rank-1-matrix $\mathbf{S} = \mathbf{c}\mathbf{c}^T \in \mathbb{R}^{N\times N}$ from (6.12). For building the vector $\mathbf{c}$ with

$$\mathbf{c}_k := \int_\Gamma \zeta_k \, d\Gamma = \sum_{i=1}^N \int_{E_i} \zeta_k \, d\Gamma,$$

note that the support of ζ_k consists precisely of the elements $E_i \in \mathcal{E}_\ell$ which include $z_k \in \mathcal{K}_\ell$ as a node. The vector $\mathbf{c}$ can be assembled $\mathcal{E}_\ell$-elementwise, and for each element E_i two entries of $\mathbf{c}$ are updated. Moreover, there holds

$$\int_{E_i} \zeta_k \, d\Gamma = \begin{cases} 0, & \text{if } z_k \notin E_i, \\ \text{length}(E_i)/2, & \text{else.} \end{cases}$$

Consequently, the assembly of the vector $\mathbf{c}$ can be done as follows, where $\mathsf{h}(j)$ contains the element-width $\text{length}(E_j)$.

```matlab
nE = size(elements,1);
h = sqrt(sum((coordinates(elements(:,1),:)-coordinates(elements(:,2),:)).^2,2));
c = zeros(nE,1);
for j = 1:nE
    nodes = elements(j,:);
    c(nodes) = c(nodes) + 0.5*h(j);
end
```

For the final implementation of **`buildHypsingStabilization`** in Listing 11, the `for`-loop is eliminated by use of `accumarray`:

- The function takes the mesh $\mathcal{E}_\ell$ described by the arrays `coordinates` and `elements`.
- We compute the vector of all element-widths (Line 5).
- The former `for`-loop is written in compact form (Line 8).
- Finally, the function builds and returns the stabilization matrix $\mathbf{S}$ (Line 11).

Listing 12

```matlab
function b = buildHypsingRHS(coordinates,elements,phi)
%*** arbitrary quadrature on [-1,1] with exactness n >= 2, e.g., gauss(2)
quad_nodes = [-1 1]/sqrt(3);
quad_weights = [1;1];

%*** the remaining code is independent of the chosen quadrature rule
nE = size(elements,1);
nQ = length(quad_nodes);

%*** build vector of evaluations points as (nQ*nE x 2)-matrix
a = coordinates(elements(:,1),:);
b = coordinates(elements(:,2),:);
sx = reshape(a,2*nE,1)*(1-quad_nodes) + reshape(b,2*nE,1)*(1+quad_nodes);
sx = 0.5*reshape(sx',nQ*nE,2);

%*** phi(sx) usually depends on the normal vector, whence phi takes sx and the
%*** nodes of the respective element to compute the normal
a_sx = reshape(repmat(reshape(a,2*nE,1),1,nQ)',nE*nQ,2);
b_sx = reshape(repmat(reshape(b,2*nE,1),1,nQ)',nE*nQ,2);

%*** perform all necessary evaluations of phi as (nE x nQ)-matrix
phi_sx = reshape(phi(sx,a_sx,b_sx),nQ,nE)';

%*** compute elementwise integral mean of phi
phi_mean = (phi_sx*quad_weights*0.5)';

%*** compute DLP-matrix for P0 x S1
K = buildK(coordinates,elements);

%*** compute mass-type matrix for P0 x S1
nE = size(elements,1);
h = sqrt(sum((coordinates(elements(:,1),:)-coordinates(elements(:,2),:)).^2,2));
I = reshape(repmat(1:nE,2,1),2*nE,1);
J = reshape(elements',2*nE,1);
A = reshape(repmat(0.5*h,1,2)',2*nE,1);
M = sparse(I,J,A);

%*** build right-hand side vector
b =  (phi_mean*M*0.5 - phi_mean*K)';
```

6.3. Build Right-Hand Side for Hypersingular Integral Equation (Listing 12). With the representation

$$\Phi_\ell = \sum_{j=1}^{N} \mathbf{p}_j \chi_j$$

and the transposed matrices of $\mathbf{K}$ and $\mathbf{M}$, the right-hand side vector for (6.11) reads

$$\mathbf{b} := \frac{1}{2}\mathbf{M}^T\mathbf{p} - \mathbf{K}^T\mathbf{p} = \Big(\frac{1}{2}\mathbf{p}^T\mathbf{M} - \mathbf{p}^T\mathbf{K}\Big)^T, \tag{6.14}$$

where we identify the vector $\mathbf{p} \in \mathbb{R}^N$ with a matrix $\mathbf{p} \in \mathbb{R}^{N\times 1}$.

For the implementation, we compute the elementwise integral mean Φ_ℓ by use of numerical quadrature:

$$\mathbf{p}_j := \frac{1}{\mathrm{length}(E_j)} \int_{E_j} \phi \, d\Gamma = \frac{1}{2}\int_{-1}^{1} \phi\circ\gamma_j \, ds \approx \frac{1}{2}\,\mathrm{quad}_n(\phi\circ\gamma_j) =: \widetilde{\mathbf{p}}_j \quad \text{for all } E_j \in \mathcal{E}_\ell. \tag{6.15}$$

Here, we use the parametrization $\gamma_j : [-1,1] \to E_j$ from (2.1). Moreover, $\mathrm{quad}_n(\cdot)$ denotes a quadrature rule which is exact of order $n \in \mathbb{N}$, i.e., $\mathrm{quad}_n(p) = \int_{-1}^{1} p\, ds$ for all $p \in \mathcal{P}^n[-1,1]$. In our realization, we use a Gauss quadrature with two nodes. Note that this provides exactness $n = 3$ and leads to an approximation error of order $\mathcal{O}(h^{5/2})$, cf. Section 6.4.

With these preparations, the documentation of Listing 12 reads as follows:

- The function takes as input the given mesh $\mathcal{E}_\ell$ in form of the arrays `coordinates` and `elements` as well as a function handle `phi` for the Neumann data. A call of the function `phi` is done by
  `y = phi(x,a,b)`
  with $(n \times 2)$-arrays `x`, `a`, and `b`. The j-th rows `x`$(j,:)$, `a`$(j,:)$, and `b`$(j,:)$ correspond to a point $x_j \in [a_j, b_j] \subset \mathbb{R}^2$. The entry `y`$(j)$ of the column vector `y` then contains $\phi(x_j)$.
- As stated above, we use the Gauss quadrature with two nodes (Line 3–4).
- If $s_k \in [-1,1]$ is a quadrature node and $E_j = [a_j, b_j] \in \mathcal{E}_\ell = \{E_1, \dots, E_N\}$ is an element, the function ϕ has to be evaluated at
  $$\gamma_j(s_k) = \frac{1}{2}\big(a_j + b_j + s_k(b_j - a_j)\big) = \frac{1}{2}\big(a_j(1-s_k) + b_j(1+s_k)\big).$$
  In Line 11–14, we build the $(2N \times 2)$-array `sx` which contains all necessary evaluation points. Note that the two evaluation points at E_j are stored in `sx`$(2j-1,:)$ and `sx`$(2j,:)$.
- In Line 18–19, we compute the $(2N \times 2)$-arrays `a_sx` and `b_sx` such that, e.g., `a_sx`$(2j-1,:)$ and `a_sx`$(2j,:)$ contain the first node $a_j \in \mathbb{R}^2$ of the boundary element $E_j = [a_j, b_j]$.
- We then evaluate the Neumann data ϕ simultaneously in all evaluation points and we reshape this $(2N \times 1)$-array into an $(N \times 2)$-array `phi_sx` such that `phi_sx`$(j,:)$ contains all ϕ-values related to E_j (Line 22).
- As a next step, we compute the $(N \times 1)$-array `phi_mean` of all integral means along the lines of (6.15), namely `phi_mean`$(j) = \mathrm{quad}_n(\phi\circ\gamma_j)/2$ (Line 25).
- Next we build $\mathbf{K}$ (Line 28) and $\mathbf{M}$ (Line 31–36), cf. Section 5.2 above.
- Finally (Line 39), the function computes and returns the vector $\mathbf{b}$ as described in (6.14).

Listing 13

```
function osc = computeOscNeumann(coordinates,elements,phi)
%*** arbitrary quadrature on [-1,1] with exactness n >= 2, e.g., gauss(2)
quad_nodes = [-1 1]/sqrt(3);
quad_weights = [1;1];

%*** the remaining code is independent of the chosen quadrature rule
nE = size(elements,1);
```

```
nQ = length(quad_nodes);

%*** build vector of evaluations points as (nQ*nE x 2)-matrix
a = coordinates(elements(:,1),:);
b = coordinates(elements(:,2),:);
sx = reshape(a,2*nE,1)*(1-quad_nodes) + reshape(b,2*nE,1)*(1+quad_nodes);
sx = 0.5*reshape(sx',nQ*nE,2);

%*** phi(sx) usually depends on the normal vector, whence phi takes sx and the
%*** nodes of the respective element to compute the normal
a_sx = reshape(repmat(reshape(a,2*nE,1),1,nQ)',nE*nQ,2);
b_sx = reshape(repmat(reshape(b,2*nE,1),1,nQ)',nE*nQ,2);

%*** perform all necessary evaluations of phi as (nE x nQ)-matrix
phi_sx = reshape(phi(sx,a_sx,b_sx),nQ,nE)';

%*** compute elementwise integral mean of phi
phi_mean = phi_sx*quad_weights*0.5;

%*** compute vector of (squared) element-widths
h = sum((a-b).^2,2);

%*** compute oscillation terms
osc_sx = (phi_sx - repmat(phi_mean,1,nQ)).^2;
osc = 0.5*h.*(osc_sx*quad_weights);
```

6.4. Computation of Data Oscillations for Neumann Data (Listing 13). Instead of solving the correct variational form (6.5), we solve

$$\langle\!\langle u_\ell \,, v\rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell \,, v\rangle_\Gamma \quad \text{for all } v \in H^{1/2}(\Gamma) \tag{6.16}$$

with perturbed right-hand side, where we use the approximation $\Phi_\ell \approx \phi$. Analytically, the error between the exact solution $u \in H^{1/2}(\Gamma)$ of (6.5) and the exact solution $u_\ell \in H^{1/2}(\Gamma)$ of the perturbed formulation (6.16) is controlled by

$$|\!|\!| u - u_\ell |\!|\!|_{W+S} \lesssim \|h_\ell^{1/2}(\phi - \Phi_\ell)\|_{L^2(\Gamma)} =: \mathrm{osc}_{N,\ell}, \tag{6.17}$$

see [AGP]. We now aim for a numerical approximation of the local contributions

$$\mathrm{osc}_{N,\ell}(E_j) := \|h_\ell^{1/2}(\phi - \Phi_\ell)\|_{L^2(E_j)} = \mathrm{length}(E_j)^{1/2}\,\|\phi - \mathbf{p}_j\|_{L^2(E_j)} \quad \text{for all } E_j \in \mathcal{E}_\ell,$$

where —as for the computation of the right-hand side vector $\mathbf{b}$ in Section 6.3— $\mathbf{p}_j$ abbreviates the integral mean

$$\mathbf{p}_j := \frac{1}{\mathrm{length}(E_j)} \int_{E_j} \phi \, d\Gamma = \frac{1}{2}\int_{-1}^{1} \phi\circ\gamma_j \, ds \approx \frac{1}{2}\,\mathrm{quad}_n(\phi\circ\gamma_j) =: \widetilde{\mathbf{p}}_j. \tag{6.18}$$

Here, we use the parametrization $\gamma_j : [-1,1] \to E_j$ from (2.1). Moreover, $\mathrm{quad}_n(\cdot)$ denotes the same quadrature rule as for the computation of the right-hand side vector $\mathbf{b}$ which is exact of order $n \in \mathbb{N}$, i.e., $\mathrm{quad}_n(p) = \int_{-1}^{1} p\, ds$ for all $p \in \mathcal{P}^n[-1,1]$. With this quadrature rule, the local Neumann oscillations are approximated by

$$\begin{aligned}\mathrm{osc}_{N,\ell}(E_j)^2 = \mathrm{length}(E_j)\int_{E_j} |\phi - \mathbf{p}_j|^2 \, d\Gamma &= \frac{\mathrm{length}(E_j)^2}{2}\int_{-1}^{1} |\phi\circ\gamma_j(s) - \mathbf{p}_j|^2 \, ds \\ &\approx \frac{\mathrm{length}(E_j)^2}{2}\,\mathrm{quad_n}\big((\phi\circ\gamma_j - \widetilde{\mathbf{p}}_j)^2\big) =: \widetilde{\mathrm{osc}}_{N,\ell}(E_j)^2. \end{aligned} \tag{6.19}$$

With the definition $\widetilde{\mathrm{osc}}_{N,\ell} := \big(\sum_{j=1}^N \widetilde{\mathrm{osc}}_{N,\ell}(E_j)^2\big)^{1/2}$, one can then prove that

$$|\mathrm{osc}_{N,\ell} - \widetilde{\mathrm{osc}}_{N,\ell}| = \mathcal{O}(h^{n/2+1}).$$

Since $\mathrm{osc}_{N,\ell} = \mathcal{O}(h^{3/2})$, we should thus choose $n \geq 2$. For our implementation, we use the Gauss quadrature rule with two nodes, which is exact for polynomials of degree $n = 3$. As for the Dirichlet data oscillations, this choice leads to

$$|\mathrm{osc}_{N,\ell} - \widetilde{\mathrm{osc}}_{N,\ell}| = \mathcal{O}(h^{5/2}), \quad \text{whereas} \quad \mathrm{osc}_{N,\ell} = \mathcal{O}(h^{3/2}).$$

The documentation of Listing 13 simply reads as follows:

- The function takes the given mesh $\mathcal{E}_\ell$ in form of the arrays `coordinates` and `elements` as well as a function handle `phi` for the Neumann data.
- The Lines 2–25 are identical with those of Listing 12, cf. Section 6.3.
- We realize Equation (6.19). Since we are using the same quadrature rule as for the computation of the integral mean, all necessary evaluations of ϕ have already been computed. Therefore, we derive the necessary evaluations of $(\phi - \widetilde{\mathbf{p}}_j)^2$ in Line 31. Multiplication with the quadrature weights and coefficient-wise weighting with $\mathrm{length}(E_j)^2/2$ provides the $(N \times 1)$-array `osc` such that `osc`$(j) \approx \mathrm{length}(E_j)\|\phi - \Phi_\ell\|^2_{L^2(E_j)}$. More precisely, there holds $\mathrm{osc}^2_{N,\ell} \approx \widetilde{\mathrm{osc}}^2_{N,\ell} = \sum_{j=1}^N$ `osc`(j).

Listing 14

```
function err = computeErrDirichlet(coordinates,elements,g,uD)
%*** compute midpoints of all elements
midpoints = 0.5*( coordinates(elements(:,1),:) + coordinates(elements(:,2),:) );

%*** compute p = (uD - uDh) at element midpoints
p_midpoints = uD(midpoints) - 0.5*sum(g(elements),2);

%*** compute p = (uD - uDh) at all nodes
p_nodes = uD(coordinates) - g;

%*** evaluate derivative p' at all elements (left,midpoint,right)
p_prime = [p_nodes(elements) p_midpoints] * [-3 -1 1 ; -1 1 3 ; 4 0 -4]*0.5;

%*** compute Dirichlet error simultaneously for all elements
err = 2*p_prime.^2*[1;4;1]/3;
```

6.5. Computation of Reliable Error Bound for $\|u - U_\ell\|_{W+S}$ (Listing 14). We assume that the exact Dirichlet data satisfy additional regularity $u \in H^1(\Gamma)$. Let $U_\ell^* \in \mathcal{S}^1(\mathcal{E}_\ell)$ be the (only theoretically computed) Galerkin solution with respect to the non-perturbed right-hand side $(1/2 - K')\phi$ instead of $(1/2 - K')\Phi_\ell$. Moreover, let I_ℓ denote the nodal interpolation operator onto $\mathcal{S}^1(\mathcal{E}_\ell)$. With the technique from [EFGP, AGP], we obtain

$$\|u - U_\ell^*\|_{W+S} \leq \|u - I_\ell u\|_{W+S} \lesssim \|h_\ell^{1/2}(u - I_\ell u)'\|_{L^2(\Gamma)} \leq \|h_\ell^{1/2}(u - U_\ell)'\|_{L^2(\Gamma)}$$

as well as

$$\|U_\ell^* - U_\ell\|_{W+S} \lesssim \mathrm{osc}_{N,\ell},$$

where $\mathrm{osc}_{N,\ell}$ denotes the Neumann data oscillations from Section 6.4. We therefore obtain

$$\begin{aligned}\|u - U_\ell\|_{W+S} &\leq \|u - U_\ell^*\|_{W+S} + \|U_\ell^* - U_\ell\|_{W+S} \\ &\lesssim \|h_\ell^{1/2}(u - U_\ell)'\|_{L^2(\Gamma)} + \mathrm{osc}_{N,\ell} =: \mathrm{err}_{D,\ell} + \mathrm{osc}_{N,\ell}.\end{aligned}$$

For the numerical realization of

$$\mathrm{err}_{D,\ell} = \Big(\sum_{j=1}^{N} \mathrm{err}_{D,\ell}(E_j)^2\Big)^{1/2}, \quad \text{where} \quad \mathrm{err}_{D,\ell}(E_j)^2 = \mathrm{length}(E_j)\, \|(u-U_\ell)'\|^2_{L^2(E_j)},$$

we use the same ideas as for the Dirichlet data oscillations in Section 5.3, where

$$\mathrm{err}_{D,\ell}(E_j)^2 = 2\int_{-1}^{1} \big((u-U_\ell)\circ\gamma_j\big)'(s)^2\,ds \approx \mathrm{quad}_2\big((p_j')^2\big) =: \widetilde{\mathrm{err}}_{D,\ell}(E_j). \tag{6.20}$$

Here, $p_j \in \mathcal{P}^2[-1,1]$ is the unique polynomial with $p_j(-1) = v(a_j)$, $p_j(1) = v(b_j)$, and $p_j(0) = v(m_j)$, where $v = u - U_\ell$ as well as $E_j = [a_j, b_j]$ and $m_j = (a_j + b_j)/2$. Recall that

$$\begin{pmatrix} p_j'(-1) \\ p_j'(0) \\ p_j'(+1) \end{pmatrix} = \begin{pmatrix} -3/2 & +2 & -1/2 \\ -1/2 & 0 & +1/2 \\ +1/2 & -2 & +3/2 \end{pmatrix} \begin{pmatrix} v(a_j) \\ v(m_j) \\ v(b_j) \end{pmatrix} = \begin{pmatrix} -3/2 & -1/2 & +2 \\ -1/2 & +1/2 & 0 \\ +1/2 & +3/2 & -2 \end{pmatrix} \begin{pmatrix} v(a_j) \\ v(b_j) \\ v(m_j) \end{pmatrix}.$$

As we are at last targeted on vectorization, we write the linear system row-wise as

$$\big(p_j'(-1), p_j'(0), p_j'(+1)\big) = \big(v(a_j), v(b_j), v(m_j)\big) \begin{pmatrix} -3/2 & -1/2 & +1/2 \\ -1/2 & +1/2 & +3/2 \\ +2 & 0 & -2 \end{pmatrix}. \tag{6.21}$$

For the numerical quadrature, we use a Newton-Côtes formula with three nodes $s_k \in \{-1, 0, +1\}$ and corresponding weights $\omega_k = \{1/3, 4/3, 1/3\}$. The documentation of Listing 14 now reads as follows:

- The function takes the mesh $\mathcal{E}_\ell$ in terms of `coordinates` and `elements` as well as the nodal vector $\mathbf{g} \in \mathbb{R}^N$ of $U_\ell = \sum_{j=1}^N \mathbf{g}_j \zeta_j$ and the function handle `uD` for the exact solution u (Line 1).
- We first compute all element midpoints (Line 3) and evaluate the solution $u - U_\ell$ at all midpoints (Line 6) and all nodes (Line 9).
- Using (6.21), we provide all necessary evaluations of $p_j'(s_k)$ in form of the $(N \times 3)$-array `p_prime` (Line 12).
- Finally, Line 15 realizes (6.20), and the function returns the column vector `err`, where `err`$(j) = \widetilde{\mathrm{err}}_{D,\ell}(E_j)^2$. In particular, there holds $\mathrm{err}_{D,\ell} \approx \widetilde{\mathrm{err}}_{D,\ell} := \big(\sum_{j=1}^N \texttt{err}(j)\big)^{1/2}$.

Remark 6.1. *In academic experiments, the exact solution u is usually known and has certain regularity $u \in H^1(\Gamma)$ which only depends on the geometry of Γ. As explained before, there holds*

$$|\!|\!| u - U_\ell |\!|\!|_{W+S} \lesssim \mathrm{err}_{D,\ell} + \mathrm{osc}_{N,\ell},$$

so that we can control the error reliably. Moreover, the convergence $\mathrm{err}_{D,\ell} \to 0$ as $\ell \to \infty$ might indicate that there are no major bugs in the implementation — since we compare the Galerkin solution with the exact solution. $\square$

6.6. Computation of $(h - h/2)$-Based A Posteriori Error Estimators. In this section, we discuss the implementation of four error estimators which are introduced and analyzed in [EFGP]. Let $\widehat{\mathcal{E}}_\ell = \{e_1, \dots, e_{2N}\}$ be the uniform refinement of the mesh $\mathcal{E}_\ell$. Let $U_\ell \in \mathcal{S}^1(\mathcal{E}_\ell)$ and $\widehat{U}_\ell \in \mathcal{S}^1(\widehat{\mathcal{E}}_\ell)$ be the Galerkin solutions of (6.7) with respect to $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$ and the same approximate Neumann data Φ_ℓ, i.e. there holds

$$\langle\!\langle U_\ell \,, V_\ell \rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell \,, V_\ell \rangle_\Gamma \quad \text{for all } V_\ell \in \mathcal{S}^1(\mathcal{E}_\ell)$$

and

$$\langle\!\langle \widehat{U}_\ell \,, \widehat{V}_\ell \rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell \,, \widehat{V}_\ell \rangle_\Gamma \quad \text{for all } \widehat{V}_\ell \in \mathcal{S}^1(\widehat{\mathcal{E}}_\ell).$$

As for Symm's integral equation, one can expect

$$|\!|\!| u_\ell - U_\ell |\!|\!|_{W+S} \approx |\!|\!| \widehat{U}_\ell - U_\ell |\!|\!|_{W+S} = |\!|\!| \widehat{U}_\ell - U_\ell |\!|\!|_{W} =: \eta_\ell, \tag{6.22}$$

where $u_\ell \in H^{1/2}(\Gamma)$ denotes the exact solution of

(6.23) $$\langle\!\langle u_\ell\,, v\rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell\,,\, v\rangle_\Gamma \quad \text{for all } v \in H^{1/2}(\Gamma).$$

According to (6.17), (6.22), and the triangle inequality, there holds

(6.24) $$|\!|\!| u - U_\ell |\!|\!|_{W+S} \le |\!|\!| u - u_\ell |\!|\!|_{W+S} + |\!|\!| u_\ell - U_\ell |\!|\!|_{W+S} \lesssim \mathrm{osc}_{N,\ell} + \eta_\ell.$$

Clearly, the Galerkin solution $\widehat{U}_\ell$ with respect to $\mathcal{S}^1(\widehat{\mathcal{E}}_\ell)$ is more accurate than U_ℓ. Consequently, any algorithm will return $\widehat{U}_\ell$ instead of U_ℓ if $\widehat{U}_\ell$ has been computed. From this point of view U_ℓ becomes a side result and leads to unnecessary computational effort. Similar to Section 5, one can prove that one may replace U_ℓ by a cheap (but appropriate) postprocessing $I_\ell \widehat{U}_\ell$ of $\widehat{U}_\ell$. This leads to some error estimator

(6.25) $$\eta_\ell \sim |\!|\!| \widehat{U}_\ell - I_\ell \widehat{U}_\ell |\!|\!|_{W+S} =: \widetilde{\eta}_\ell$$

which always stays proportional to η_ℓ, indicated by $\eta_\ell \sim \widetilde{\eta}_\ell$, cf. [EFGP]. To be more precise, I_ℓ denotes the nodal interpolation operator on $\mathcal{S}^1(\mathcal{E}_\ell)$, which is given by

$$I_\ell U_\ell := \sum_{z \in \mathcal{K}_\ell} U_\ell(z)\zeta_z,$$

where $\mathcal{K}_\ell$ denotes the set of all nodes of $\mathcal{E}_\ell$ and where ζ_z denotes the hat-function associated with some node $z \in \mathcal{K}_\ell$.

As a matter of fact, the error estimators η_ℓ and $\widetilde{\eta}_\ell$ do not provide any information about the local errors, i.e., the error $|\!|\!| u_\ell - U_\ell |\!|\!|_{W+S}$ related to some element $E_i \in \mathcal{E}_\ell$. This is different for the error estimators μ_ℓ and $\widetilde{\mu}_\ell$ discussed in the following. For instance, one can prove that

(6.26) $$\eta_\ell \sim \mu_\ell := \|h_\ell^{1/2}(\widehat{U}_\ell - U_\ell)'\|_{L^2(\Gamma)} = \Big(\sum_{i=1}^{N} \mathrm{length}(E_i)\|(\widehat{U}_\ell - U_\ell)'\|_{L^2(E_i)}^2\Big)^{1/2}.$$

The local contributions

(6.27) $$\mu_\ell(E_i) := \mathrm{length}(E_i)^{1/2}\|(\widehat{U}_\ell - U_\ell)'\|_{L^2(E_i)} \quad \text{for all } E_i \in \mathcal{E}_\ell$$

give some measure for the error on E_i.

As the computation of the error estimator η_ℓ, the computation of μ_ℓ requires two Galerkin solutions U_ℓ and $\widehat{U}_\ell$. As before, the computation of the coarse-mesh solution U_ℓ can be avoided by use of the nodal interpolant $I_\ell \widehat{U}_\ell$. One can mathematically prove that

(6.28) $$\eta_\ell \sim \widetilde{\mu}_\ell := \|h_\ell^{1/2}(\widehat{U}_\ell - I_\ell \widehat{U}_\ell)'\|_{L^2(\Gamma)}.$$

In the following subsections, we first discuss the computation of the *global* error estimators η_ℓ and $\widetilde{\eta}_\ell$ from (6.22) and (6.25). Then, we give an implementation of the *local* error estimators μ_ℓ and $\widetilde{\mu}_\ell$ from (6.26) and (6.28), where our functions return the local contributions, see e.g. (6.27), to steer an adaptive mesh-refinement.

Remark 6.2. *If we plot the error estimators η_ℓ, $\widetilde{\eta}_\ell$, μ_ℓ and $\widetilde{\mu}_\ell$ over the number of elements, one can mathematically predict that the corresponding curves, for a sequence of arbitrarily refined meshes, are parallel. In mathematical terms, this reads*

(6.29) $$\eta_\ell \le \widetilde{\eta}_\ell \lesssim \widetilde{\mu}_\ell \le \mu_\ell \lesssim \eta_\ell,$$

cf. [EFGP]. *Empirically, one observes a very good coincidence of η_ℓ and $\widetilde{\eta}_\ell$ in the sense that the corresponding curves almost coincide. The same is observed for the curves of μ_ℓ and $\widetilde{\mu}_\ell$.* □

Remark 6.3. *Mathematically, the error estimate* (6.22) *respectively* (6.24) *involves the so-called saturation assumption: Assume that we could compute the Galerkin solutions U_ℓ^* and $\widehat{U}_\ell^*$ with respect to $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$ for the non-perturbed variational formulation* (6.5), *i.e., we formally use the exact Neumann data ϕ instead of the interpolated data Φ_ℓ — although the right-hand* *side is, in practice, non-computable because of $K'\phi$. Then, the saturation assumption states that*

$$\|u - \widehat{U}^*_\ell\|_{W+S} \le q\,\|u - U^*_\ell\|_{W+S} \tag{6.30}$$

with some uniform and ℓ-independent constant $q \in (0,1)$. Put differently, uniform mesh-refinement leads to a uniform improvement of the discretization error. Provided (6.30), one can prove that

$$\eta_\ell \le \|u_\ell - U_\ell\|_{W+S} \le (1-q^2)^{-1/2}\,\eta_\ell. \tag{6.31}$$

We stress that this assumption is somewhat natural and can, for instance, be proven for the finite element method [DN, FOP]*. For the boundary element method, however, (6.30) still remains open.*

Finally, one can prove that (6.30) is sufficient and in some sense even necessary to guarantee (6.24). □

Remark 6.4. *In academic experiments, the exact solution u of the hypersingular integral equation is usually known and has certain regularity $u \in H^1(\Gamma)$ which only depends on the geometry of Γ. In this case, one can experimentally verify the saturation assumption as follows: In Section 6.5, we derived*

$$\|u - U_\ell\|_W \lesssim \mathrm{err}_{D,\ell} + \mathrm{osc}_{N,\ell}.$$

If the right-hand side has the same convergence behaviour as the error estimator $\eta_\ell + \mathrm{osc}_{N,\ell}$, this proves empirically

$$\|u - U_\ell\|_W \lesssim \eta_\ell + \mathrm{osc}_{N,\ell}$$

and confirms the saturation assumption. □

Listing 15

```
function est = computeEstHypEta(elements_fine,elements_coarse,father2son,...
                                                  W_fine,x_fine,x_coarse)
nC = length(x_coarse);

%*** build index field k = idx(j) such that j-th node of coarse mesh coincides
%*** with k-th node of fine mesh
idx = zeros(nC,1);
idx(elements_coarse) = [ elements_fine(father2son(:,1),1), ...
                         elements_fine(father2son(:,2),2) ];

%*** build index field k = mid(j) such that midpoint of j-th element of coarse
%*** mesh is k-th node of fine mesh
mid = elements_fine(father2son(:,1),2);

%*** compute coefficient vector of (u_fine - u_coarse) w.r.t. fine mesh
x_fine(idx) = x_fine(idx) - x_coarse;
x_fine(mid) = x_fine(mid) - 0.5*sum(x_coarse(elements_coarse),2);

%*** compute energy ||| u_fine - u_coarse |||^2
est = x_fine'*(W_fine*x_fine);
```

6.6.1. Computation of Error Estimator η_ℓ (Listing 15). In this section, we aim to compute the error estimator $\eta_\ell = \|\widehat{U}_\ell - U_\ell\|_{W+S}$ from (6.22). Let $\widehat{\zeta}_j$ denote the hat-function associated with some fine-mesh node $z_j \in \widehat{\mathcal{K}}_\ell$. Let $\mathbf{x} \in \mathbb{R}^N$ and $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ be the coefficient vectors of U_ℓ and $\widehat{U}_\ell$ with respect to the canonical bases of $\mathcal{S}^1(\mathcal{E}_\ell)$ and $\mathcal{S}^1(\widehat{\mathcal{E}}_\ell)$, i.e.

$$U_\ell = \sum_{j=1}^{N} \mathbf{x}_j \zeta_j \quad \text{and} \quad \widehat{U}_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{x}}_j \widehat{\zeta}_j.$$

Similar to Section 5.5.1, there holds $\mathcal{S}^1(\mathcal{E}_\ell) \subset \mathcal{S}^1(\widehat{\mathcal{E}}_\ell)$ which provides a unique vector $\widehat{\mathbf{y}} \in \mathbb{R}^{2N}$ such that

$$U_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{y}}_j \widehat{\zeta}_j.$$

With the vectors $\widehat{\mathbf{x}}, \widehat{\mathbf{y}} \in \mathbb{R}^{2N}$, there holds

$$\begin{aligned}\eta_\ell^2 = \|\widehat{U}_\ell - U_\ell\|_{W+S}^2 = \langle\!\langle \widehat{U}_\ell - U_\ell \,, \widehat{U}_\ell - U_\ell \rangle\!\rangle_{W+S} &= \sum_{j,k=1}^{2N} (\widehat{\mathbf{x}}_j - \widehat{\mathbf{y}}_j)(\widehat{\mathbf{x}}_k - \widehat{\mathbf{y}}_k) \langle\!\langle \widehat{\zeta}_j \,, \widehat{\zeta}_k \rangle\!\rangle_{W+S} \\ &= (\widehat{\mathbf{x}} - \widehat{\mathbf{y}}) \cdot (\widehat{\mathbf{W}} + \widehat{\mathbf{S}})(\widehat{\mathbf{x}} - \widehat{\mathbf{y}}),\end{aligned}$$

where $\widehat{\mathbf{W}}$ denotes the matrix of the hypersingular integral operator and $\widehat{\mathbf{S}}$ the matrix of the stabilization term contributions (6.10) with respect to the fine mesh, cf. Section 6. The documentation of Listing 15 now reads as follows:

- The function takes the coefficient vectors $\mathbf{x} \in \mathbb{R}^N$ and $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ of the Galerkin solutions U_ℓ and $\widehat{U}_\ell$ as well as the sum $\widehat{\mathbf{W}} + \widehat{\mathbf{S}}$ of the hypersingular operator matrix $\widehat{\mathbf{W}}$ and the stabilization term matrix $\widehat{\mathbf{S}}$ for the fine mesh $\widehat{\mathcal{E}}_\ell$ stored in `W_fine`. Besides this, the function takes the coarse mesh described by the $(N \times 2)$-array `elements_coarse` and the fine mesh described by the $(2N \times 2)$-array `elements_fine`. Finally, the $(N \times 2)$-array `father2son` links the indices of elements $E_i \in \mathcal{E}_\ell$ with the indices of its sons $e_j, e_k \in \widehat{\mathcal{E}}_\ell$ in the sense that `father2son(`i`,:) = [`j`,`k`]` for $E_i = e_j \cup e_k$.
- We build an array `k = idx(i)` such that the i-th node of the coarse mesh coincides with the k-th node of the fine mesh (Line 7–9).
- Furthermore, we build an array `k = mid(j)` such that the midpoint of the j-th element of the coarse mesh is the k-th node of the fine mesh (Line 13).
- Then we overwrite successively the vector $\widehat{\mathbf{x}}$ by the coefficient vector $\widehat{\mathbf{x}} - \widehat{\mathbf{y}}$ of $\widehat{U}_\ell - U_\ell$. We first calculate this difference for any node belonging to $\mathcal{K}_\ell$ (Line 16) and in a next step for any node occurring in $\widehat{\mathcal{K}}_\ell \backslash \mathcal{K}_\ell$ by interpolating the coarse vector (Line 17).
- Finally, the function returns $\eta_\ell^2 = \|\widehat{U}_\ell - U_\ell\|_{W+S}^2$ (Line 20).

Listing 16

```
1  function est = computeEstHypEtaTilde(elements_fine,elements_coarse,...
2                                              father2son,W_fine,x_fine)
3  nC = max(elements_coarse(:));
4
5  %*** build index field k = idx(j) such that j-th node of coarse mesh coincides
6  %*** with k-th node of fine mesh
7  idx = zeros(nC,1);
8  idx(elements_coarse) = [ elements_fine(father2son(:,1),1), ...
9                           elements_fine(father2son(:,2),2) ];
10
11 %*** build index field k = mid(j) such that midpoint of j-th element of coarse
12 %*** mesh is k-th node of fine mesh
13 mid = elements_fine(father2son(:,1),2);
14
15 %*** build index field [i j] = e2n(k) such that fine-mesh nodes zi and zj are
16 %*** the nodes of the coarse-mesh element Ek
```

```
17 e2n = [ elements_fine(father2son(:,1),1) elements_fine(father2son(:,2),2) ];
18
19 %*** compute coefficient vector of (1 - I_coarse)*u_fine w.r.t. fine mesh
20 x_fine(mid) = x_fine(mid) - 0.5*sum(x_fine(e2n),2);
21 x_fine(idx) = 0;
22
23 %*** compute energy ||| (1 - I_coarse)*u_fine |||^2
24 est = x_fine'*(W_fine*x_fine);
```

6.6.2. Computation of Error Estimator $\widetilde{\eta}_\ell$ (Listing 16). In this section, we aim to compute the error estimator $\widetilde{\eta}_\ell$ which is defined by

$$\widetilde{\eta}_\ell := \|\widehat{U}_\ell - I_\ell \widehat{U}_\ell\|_{W+S}.$$

We adopt the notation of Section 6.6.1 for the computation of η_ℓ, namely $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ with

$$\widehat{U}_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{x}}_j \widehat{\zeta}_j.$$

Let $z_i \in \widehat{\mathcal{K}}_\ell \backslash \mathcal{K}_\ell$. Then, there are two elements $e_j, e_k \in \widehat{\mathcal{E}}_\ell$ being the sons of $E_i \in \mathcal{E}_\ell$, i.e. $E_i = e_j \cup e_k$, which share z_i as a common node. Since $I_\ell \widehat{U}_\ell$ restricted to some element $E_i = e_j \cup e_k$ is affine, there holds

$$I_\ell \widehat{U}_\ell(z_i) = \frac{1}{2}\Big(I_\ell \widehat{U}_\ell(z_j) + I_\ell \widehat{U}_\ell(z_k)\Big) = \frac{1}{2}\Big(\widehat{U}_\ell(z_j) + \widehat{U}_\ell(z_k)\Big), \tag{6.32}$$

where $z_j, z_k \in \mathcal{K}_\ell$ denote the outer nodes of the elements e_j, e_k. On the other hand, there holds $I_\ell \widehat{U}_\ell(z_i) = \widehat{U}_\ell(z_i)$ provided that $z_i \in \mathcal{K}_\ell$. Altogether, representing $I_\ell \widehat{U}_\ell \in \mathcal{S}^1(\mathcal{E}_\ell)$ with respect to the fine-mesh $\widehat{\mathcal{E}}_\ell$, we obtain

$$I_\ell \widehat{U}_\ell = \sum_{n=1}^{2N} \widehat{\mathbf{z}}_n \widehat{\zeta}_n, \tag{6.33}$$

where $\widehat{\mathbf{z}} \in \mathbb{R}^{2N}$ denotes the coefficient vector. As in Section 6.6.1, there holds

$$\widetilde{\eta}_\ell^2 = \|\widehat{U}_\ell - I_\ell \widehat{U}_\ell\|_{W+S}^2 = (\widehat{\mathbf{x}} - \widehat{\mathbf{z}}) \cdot (\widehat{\mathbf{W}} + \widehat{\mathbf{S}})(\widehat{\mathbf{x}} - \widehat{\mathbf{z}}).$$

Therefore, the documentation of Listing 16 reads as follows:

- The function takes the coefficient vector $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ of the Galerkin solutions $\widehat{U}_\ell$ as well as the sum $\widehat{\mathbf{W}} + \widehat{\mathbf{S}}$ of the hypersingular operator matrix $\widehat{\mathbf{W}}$ and the stabilization term matrix $\widehat{\mathbf{S}}$ for the fine mesh $\widehat{\mathcal{E}}_\ell$ stored in `W_fine`. Besides this, the function takes the coarse mesh described by the $(N \times 2)$-array `elements_coarse` and the fine mesh described by the $(2N \times 2)$-array `elements_fine`. Moreover, the link between $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$ is provided by means of `father2son`.
- We first build an array `k = idx(i)` such that the i-th node of the coarse mesh coincides with the k-th node of the fine mesh (Line 7–9).
- Furthermore, we build an array `k = mid(j)` such that the midpoint of the j-th element of the coarse mesh is the k-th node of the fine mesh (Line 13).
- Next we build an array `[i j] = e2n(k)` such that the fine-mesh nodes z_i and z_j are the nodes of the coarse mesh elements E_k (Line 17).
- We successively overwrite $\widehat{\mathbf{x}}$ by the coefficient vector $\widehat{\mathbf{x}} - \widehat{\mathbf{z}} \in \mathbb{R}^{2N}$ of $\widehat{U}_\ell - I_\ell \widehat{U}_\ell$ (Line 20–21).
- Finally, the function returns $\widetilde{\eta}_\ell^2 = \|\widehat{U}_\ell - I_\ell \widehat{U}_\ell\|_{W+S}^2$ (Line 24).

Listing 17

```
1 function ind = computeEstHypMu(elements_fine,elements_coarse,father2son,...
2                                                        x_fine,x_coarse)
```

```
nC = length(x_coarse);

%*** build index field k = idx(j) such that j-th node of coarse mesh coincides
%*** with k-th node of fine mesh
idx = zeros(nC,1);
idx(elements_coarse) = [ elements_fine(father2son(:,1),1), ...
                         elements_fine(father2son(:,2),2) ];

%*** build index field k = mid(j) such that midpoint of j-th element of coarse
%*** mesh is k-th node of fine mesh
mid = elements_fine(father2son(:,1),2);

%*** compute coefficient vector of (u_fine - u_coarse) w.r.t. fine mesh
x_fine(idx) = x_fine(idx) - x_coarse;
x_fine(mid) = x_fine(mid) - 0.5*sum(x_coarse(elements_coarse),2);

%*** compute h^2*|(u_fine - u_coarse)'|^2 for all fine-mesh elements
%*** where h denotes the diameters of the fine-mesh elements
grad = (x_fine(elements_fine)*[-1;1]).^2;

%*** compute (squared) indicators w.r.t. coarse mesh as described above
ind = 2*( grad(father2son(:,1)) + grad(father2son(:,2)) );
```

6.6.3. Computation of Error Estimator μ_ℓ (Listing 17). In this section, we discuss the implementation of

$$\mu_\ell^2 := \sum_{i=0}^{N} \mu_\ell(E_i)^2, \quad \text{where} \quad \mu_\ell(E_i)^2 := \text{length}(E_i)\|(\widehat{U}_\ell - U_\ell)'\|_{L^2(E_i)}^2.$$

Actually we calculate the squared entries $\mu_\ell(E_i)^2$ for all $E_i \in \mathcal{E}_\ell$.

We adopt the notation of Section 6.6.1, namely $\widehat{\mathbf{x}}, \widehat{\mathbf{y}} \in \mathbb{R}^{2N}$ with

$$\widehat{U}_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{x}}_j \widehat{\zeta}_j \quad \text{and} \quad U_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{y}}_j \widehat{\zeta}_j.$$

For fixed $E_i \in \mathcal{E}_\ell$ and sons $e_j, e_k \in \widehat{\mathcal{E}}_\ell$ with $E_i = e_j \cup e_k$, we obtain

$$\|(\widehat{U}_\ell - U_\ell)'\|_{L^2(E_i)}^2 = \int_{E_i} \left|(\widehat{U}_\ell - U_\ell)'\right|^2 d\Gamma = \int_{e_j} \left|(\widehat{U}_\ell - U_\ell)'\right|^2 d\Gamma + \int_{e_k} \left|(\widehat{U}_\ell - U_\ell)'\right|^2 d\Gamma.$$

As $(\widehat{U}_\ell - U_\ell) \in \mathcal{S}^1(\widehat{\mathcal{E}}_\ell)$ is piecewise affine, its arc-length derivative $(\widehat{U}_\ell - U_\ell)' \in \mathcal{P}^0(\widehat{\mathcal{E}}_\ell)$ is piecewise constant. Consequently, the above formula reduces to

$$\begin{aligned}\|(\widehat{U}_\ell - U_\ell)'\|_{L^2(E_i)}^2 &= \text{length}(e_j)\left|(\widehat{U}_\ell - U_\ell)'|_{e_j}\right|^2 + \text{length}(e_k)\left|(\widehat{U}_\ell - U_\ell)'|_{e_k}\right|^2 \\ &= \frac{\text{length}(E_i)}{2}\Big(\left|(\widehat{U}_\ell - U_\ell)'|_{e_j}\right|^2 + \left|(\widehat{U}_\ell - U_\ell)'|_{e_k}\right|^2\Big).\end{aligned}$$

With $e_j = [z_{j_1}, z_{j_2}] \in \widehat{\mathcal{E}}_\ell$, we obtain

$$\left|(\widehat{U}_\ell - U_\ell)'|_{e_j}\right|^2 = \frac{\left|(\widehat{U}_\ell - U_\ell)(z_{j_2}) - (\widehat{U}_\ell - U_\ell)(z_{j_1})\right|^2}{\text{length}(e_j)^2}.$$

This implies

$$\mu_\ell(E_i)^2 = 2\Big(\left|(\widehat{U}_\ell - U_\ell)(z_{j_2}) - (\widehat{U}_\ell - U_\ell)(z_{j_1})\right|^2 + \left|(\widehat{U}_\ell - U_\ell)(z_{k_2}) - (\widehat{U}_\ell - U_\ell)(z_{k_1})\right|^2\Big) \tag{6.34}$$

Altogether, the documentation of Listing 17 reads as follows:

- As input arguments, the function takes the mesh $\mathcal{E}_\ell$ represented by the $(N \times 2)$-array `elements_coarse`, the mesh $\widehat{\mathcal{E}}_\ell$ represented by the $(2N \times 2)$-array `elements_fine`, the link between $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$, and the coefficient vectors $\mathbf{x} \in \mathbb{R}^N$ and $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ of the Galerkin solutions U_ℓ and $\widehat{U}_\ell$ (Line 1–2).
- We overwrite the vector $\widehat{\mathbf{x}}$ by coefficient vector $\widehat{\mathbf{x}} - \widehat{\mathbf{y}}$ of $\widehat{U}_\ell - U_\ell$ in exactly the same way as we did in Section 6.6.1 for the error estimator η_ℓ (Line 7–17).
- Next (Line 21), we compute the coefficient vector of the squared arc-length derivative of $\widehat{U}_\ell - U_\ell$ multiplied by the diameter of the fine-mesh elements to avoid needless computations.
- Finally (Line 24), the function realizes (6.34) and returns the vector

$$\mathbf{v} := (\mu_\ell(E_1)^2, \dots, \mu_\ell(E_N)^2) \in \mathbb{R}^N$$

so that $\mu_\ell = \big(\sum_{i=1}^N \mathbf{v}_i\big)^{1/2}$.

Listing 18

```
function ind = computeEstHypMuTilde(elements_fine,elements_coarse,...
                                                              father2son,x_fine)
nC = max(elements_coarse(:));

%*** build index field k = idx(j) such that j-th node of coarse mesh coincides
%*** with k-th node of fine mesh
idx = zeros(nC,1);
idx(elements_coarse) = [ elements_fine(father2son(:,1),1), ...
                         elements_fine(father2son(:,2),2) ];

%*** build index field k = mid(j) such that midpoint of j-th element of coarse
%*** mesh is k-th node of fine mesh
mid = elements_fine(father2son(:,1),2);

%*** build index field [i j] = e2n(k) such that fine-mesh nodes zi and zj are
%*** the nodes of the coarse-mesh element Ek
e2n = [ elements_fine(father2son(:,1),1) elements_fine(father2son(:,2),2) ];

%*** compute coefficient vector of (1 - I_coarse)*u_fine w.r.t. fine mesh
x_fine(mid) = x_fine(mid) - 0.5*sum(x_fine(e2n),2);
x_fine(idx) = 0;

%*** compute h^2*| ((1 - I_coarse)*u_fine)' |^2 for all fine-mesh elements
%*** where h denotes the diameters of the fine-mesh elements
grad = (x_fine(elements_fine)*[-1;1]).^2;

%*** compute (squared) indicators w.r.t. coarse mesh as described above
ind = 2*( grad(father2son(:,1)) + grad(father2son(:,2)) );
```

6.6.4. Computation of Error Estimator $\widetilde{\mu}_\ell$ (Listing 18). In this section, we finally aim to compute

$$\widetilde{\mu}_\ell^2 := \sum_{i=0}^N \widetilde{\mu}_\ell(E_i)^2, \quad \text{where} \quad \widetilde{\mu}_\ell(E_i)^2 := \mathrm{length}(E_i)\|(\widehat{U}_\ell - I_\ell \widehat{U}_\ell)'\|_{L^2(E_i)}^2.$$

We adopt the notation of Section 6.6.1, namely $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ with

$$\widehat{U}_\ell = \sum_{j=1}^{2N} \widehat{\mathbf{x}}_j \widehat{\zeta}_j.$$

Based on the same ideas as for the realization of the local contributions from the preceding Sections 6.6.2 and 6.6.3, a concise documentation of Listing 18 reads as follows:

- The function takes the meshes $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$, the link between $\mathcal{E}_\ell$ and $\widehat{\mathcal{E}}_\ell$, and the coefficient vector $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ of $\widehat{U}_\ell$ (Line 1–2).
- Adopting the ideas of Section 6.6.2, we compute the coefficient vector of $\widehat{U}_\ell - I_\ell \widehat{U}_\ell$ (Line 7–21).
- According to Section 6.6.3, we compute the local contributions $\mathrm{length}(e_j)^2 |(\widehat{U}_\ell - I_\ell \widehat{U}_\ell)'|^2$ for all elements $e_j \in \widehat{\mathcal{E}}_\ell$ (Line 25).
- Finally (Line 28), the function returns the vector
$$\mathbf{v} := (\widetilde{\mu}_\ell(E_1)^2, \dots, \widetilde{\mu}_\ell(E_N)^2) \in \mathbb{R}^N.$$
In particular, there holds $\widetilde{\mu}_\ell = \big(\sum_{i=1}^N \mathbf{v}_i\big)^{1/2}$.

6.7. Adaptive Mesh-Refinement for Hypersingular Integral Equation.
Usually computing time and memory requirements are limiting quantities for numerical simulations. Therefore, one aims to choose the mesh such that it is coarse, where the (unknown) solution is smooth, and fine, where the (unknown) solution is singular. Based on a local error estimator, e.g. $\widetilde{\mu}_\ell$, such meshes are constructed in an iterative way. In each step, one refines the mesh only locally, i.e. one refines elements E_j, where the error appears to be large, namely, where the local contributions $\widetilde{\mu}_\ell(E_j)$ are large. For the error estimator $\widetilde{\mu}_\ell$ from Section 6.6.4, a possible adaptive algorithm reads as follows:

Input: Initial mesh $\mathcal{E}_0$, Neumann data ϕ, adaptivity parameter $0 < \theta < 1$, maximal number $N_{\max} \in \mathbb{N}$ of elements, and counter $\ell = 0$.

(i) Build uniformly refined mesh $\widehat{\mathcal{E}}_\ell$.
(ii) Compute Galerkin solution $\widehat{U}_\ell \in \mathcal{S}^1(\widehat{\mathcal{E}}_\ell)$.
(iii) Compute refinement indicators $\widetilde{\mu}_\ell(E)^2$ and oscillation terms $\mathrm{osc}_{N,\ell}(E)^2$ for all $E \in \mathcal{E}_\ell$.
(iv) Find minimal set $\mathcal{M}_\ell \subseteq \mathcal{E}_\ell$ such that

$$\theta\,(\widetilde{\mu}_\ell^2 + \mathrm{osc}_{N,\ell}^2) = \theta \sum_{E \in \mathcal{E}_\ell} \Big(\widetilde{\mu}_\ell(E)^2 + \mathrm{osc}_{N,\ell}(E)^2\Big) \le \sum_{E \in \mathcal{M}_\ell} \Big(\widetilde{\mu}_\ell(E)^2 + \mathrm{osc}_{N,\ell}(E)^2\Big). \tag{6.35}$$

(v) Refine at least marked elements $E \in \mathcal{M}_\ell$ and obtain mesh $\mathcal{E}_{\ell+1}$ with $\kappa(\mathcal{E}_{\ell+1}) \le 2\kappa(\mathcal{E}_0)$.
(vi) Stop provided that $\#\mathcal{E}_{\ell+1} \ge N_{\max}$; otherwise, increase counter $\ell \mapsto \ell + 1$ and go to (i).

Output: Adaptively generated mesh $\widehat{\mathcal{E}}_\ell$ and corresponding discrete solution $\widehat{U}_\ell \in \mathcal{S}^1(\widehat{\mathcal{E}}_\ell)$.

The marking criterion (6.35) has been proposed in the context of adaptive finite element methods [D]. Let formally $N_{\max} = \infty$ so that the adaptive algorithm computes a sequence of discrete solutions $\widehat{U}_\ell$ (or even U_ℓ, although this is not computed). With the same techniques as in [FOP], one can prove that the saturation assumption (6.30) implies convergence of $\widehat{U}_\ell$ and U_ℓ to u, provided that the right-hand side ϕ is not disturbed, i.e., $\phi = \Phi_\ell$. The same result also holds for $\widetilde{\mu}_\ell$ replaced by μ_ℓ.

In [AFP], we changed the notion of convergence and proved that for certain error estimators — amongst them are $\widetilde{\mu}_\ell$ and μ_ℓ for Symm's integral equation — the adaptive algorithm guarantees $\lim_\ell \widetilde{\mu}_\ell = 0$. This concept is followed in [AGP] to prove that the adaptive algorithm for Symm's integral equation stated above, yields $\lim_\ell(\widetilde{\mu}_\ell^2 + \mathrm{osc}_{D,\ell}^2) = 0$. The same ideas are applicable to the hypersingular integral equation. Therefore, if the saturation assumption (6.30) holds (at least in infinitely many steps), we obtain convergence of U_ℓ to u due to $\|u - U_\ell\|_{W+S}^2 \lesssim \widetilde{\mu}_\ell^2 + \mathrm{osc}_{N,\ell}^2$.

Listing 19

```
% adaptiveHypsing provides the implementation of an adaptive mesh-refining
% algorithm for the hypersingular integral equation.
```

```
 3
 4  %*** maximal number of elements
 5  nEmax = 100;
 6
 7  %*** adaptivity parameter
 8  theta = 0.25;
 9  rho = 0.25;
10
11  %*** adaptive mesh-refining algorithm
12  while size(elements,1) < nEmax
13
14      %*** build uniformly refined mesh
15      [coordinates_fine,elements_fine,father2son] ...
16          = refineBoundaryMesh(coordinates,elements);
17
18      %*** compute fine-mesh solution
19      W_fine = buildW(coordinates_fine,elements_fine) ...
20               + buildHypsingStabilization(coordinates_fine,elements_fine);
21      b_fine = buildHypsingRHS(coordinates_fine,elements_fine,@phi);
22      x_fine = W_fine\b_fine;
23
24      %*** compute (h-h/2)-error estimator tilde-mu
25      mu_tilde = computeEstHypMuTilde(elements_fine,elements,father2son,...
26                                      x_fine);
27      %*** compute data oscillations
28      osc_fine = computeOscNeumann(coordinates_fine,elements_fine,@phi);
29      osc = osc_fine(father2son(:,1)) + osc_fine(father2son(:,2));
30
31      %*** mark elements for refinement
32      marked = markElements(theta,rho,mu_tilde + osc);
33
34      %*** generate new mesh
35      [coordinates,elements] = refineBoundaryMesh(coordinates,elements,marked);
36  end
```

6.7.1. Implementation of Adaptive Algorithm (Listing 19). The MATLAB script of Listing 19 realizes the adaptive Algorithm from the beginning of this section.

- We use the adaptivity parameter $\theta = 1/4$ in (6.35) and mark at least the 25% of elements with the largest indicators (Line 8–9). The marking criterion is explained in Section 4.1.

7. Mixed Problem

Continuous Model Problem. Let $\Gamma = \partial\Omega$ be the piecewise affine boundary of a polygonal Lipschitz domain $\Omega \subset \mathbb{R}^2$. We assume that Γ is split into two disjoint and relatively open sets Γ_D and Γ_N with $\Gamma = \overline{\Gamma}_N \cup \overline{\Gamma}_D$. Moreover, we assume positive surface measure $|\Gamma_D| > 0$ to avoid treating the pure Neumann problem from Section 6. For given Dirichlet data $u_D \in H^{1/2}(\Gamma_D)$ and Neumann data $\phi_N \in H^{-1/2}(\Gamma_N)$, we consider the mixed boundary value problem

$$\begin{aligned} -\Delta u &= 0 && \text{in } \Omega, \\ u &= u_D && \text{on } \Gamma_D, \\ \partial_n u &= \phi_N && \text{on } \Gamma_N. \end{aligned} \tag{7.1}$$

For the equivalent integral formulation of (7.1), we choose (and fix) arbitrary extensions $\overline{u}_D \in H^{1/2}(\Gamma)$ and $\overline{\phi}_N \in H^{-1/2}(\Gamma)$ of the given data from Γ_D resp. Γ_N to the entire boundary Γ. The missing boundary data, which have to be computed, are

$$u_N := u - \overline{u}_D \quad \text{and} \quad \phi_D := \partial_n u - \overline{\phi}_N. \tag{7.2}$$

One can show that this definition yields $u_N \in \widetilde{H}^{1/2}(\Gamma_N)$ and $\phi_D \in \widetilde{H}^{-1/2}(\Gamma_D)$.

Let V denote the simple-layer potential, K the double-layer potential with adjoint K', and W the hypersingular integral operator. With the so-called Calderón projector

$$A = \begin{pmatrix} -K & V \\ W & K' \end{pmatrix}, \tag{7.3}$$

which is an operator matrix, the unknown data u_N and ϕ_D satisfy the following system of integral equations

$$A \begin{pmatrix} u_N \\ \phi_D \end{pmatrix} = (1/2 - A) \begin{pmatrix} \overline{u}_D \\ \overline{\phi}_N \end{pmatrix} =: F. \tag{7.4}$$

One can prove that (7.4) is, in fact, an equivalent formulation of the mixed boundary value problem (7.1). With the spaces

$$\mathcal{H} := \widetilde{H}^{1/2}(\Gamma_N) \times \widetilde{H}^{-1/2}(\Gamma_D) \quad \text{and} \quad \mathcal{H}^* := H^{1/2}(\Gamma_D) \times H^{-1/2}(\Gamma_N), \tag{7.5}$$

one can show that $A : \mathcal{H} \to \mathcal{H}^*$ is a linear and continuous mapping. Moreover, $\mathcal{H}^*$ is the dual space of $\mathcal{H}$ with duality understood via the formula

$$\langle (v_D, \psi_N) \, , (v_N, \psi_D) \rangle_{\mathcal{H}^* \times \mathcal{H}} := \langle \psi_N \, , \, v_N \rangle_{\Gamma_N} + \langle \psi_D \, , \, v_D \rangle_{\Gamma_D} \tag{7.6}$$

for all $(v_N, \psi_D) \in \mathcal{H}$ and $(v_D, \psi_N) \in \mathcal{H}^*$, where the duality brackets $\langle \cdot \, , \, \cdot \rangle_{\Gamma_N}$ and $\langle \cdot \, , \, \cdot \rangle_{\Gamma_D}$ on the right-hand side denote the extended L^2-scalar products. Now, the operator A induces a continuous bilinear form on $\mathcal{H}$ via

$$\begin{aligned} \langle\!\langle (u_N, \phi_D) \, , (v_N, \psi_D) \rangle\!\rangle_A &: = \langle A(u_N, \phi_D) \, , \, (v_N, \psi_D) \rangle_{\mathcal{H}^* \times \mathcal{H}} \\ &= \langle W u_N + K' \phi_D \, , \, v_N \rangle_{\Gamma_N} + \langle -K u_N + V \phi_D \, , \, \psi_D \rangle_{\Gamma_D}. \end{aligned} \tag{7.7}$$

Note that this bilinear form is non-symmetric because of the entries $-K$ and K' on the right-hand side. Nevertheless, the definition

$$|\!|\!| (u_N, \phi_D) |\!|\!|_A^2 := \langle\!\langle (u_N, \phi_D) \, , (u_N, \phi_D) \rangle\!\rangle_A = \langle W u_N \, , \, u_N \rangle_{\Gamma_N} + \langle V \phi_D \, , \, \phi_D \rangle_{\Gamma_D} \tag{7.8}$$

provides a norm on $\mathcal{H}$ which is equivalent to the usual product norm. Therefore, the bilinear form $\langle\!\langle \cdot \, , \cdot \rangle\!\rangle_A$ is uniformly elliptic, and we are in the framework of the Lax-Milgram Lemma. Consequently, the variational form of (7.4)

$$\langle\!\langle (u_N, \phi_D) \, , (v_N, \psi_D) \rangle\!\rangle_A = \langle F \, , \, (v_N, \psi_D) \rangle_{\mathcal{H}^* \times \mathcal{H}} \quad \text{for all } (v_N, \psi_D) \in \mathcal{H} \tag{7.9}$$

has a unique solution $(u_N, \phi_D) \in \mathcal{H}$. To abbreviate notation, we will now use the vector-valued unknown $\mathbf{u} := (u_N, \phi_D) \in \mathcal{H}$.

Mesh Restriction and Discrete Spaces. Let $\mathcal{E}_\ell$ be a mesh of Γ. By definition, $\mathcal{E}_\ell$ then resolves Γ_D and Γ_N, cf. Section 1.1. Consequently,

$$\mathcal{E}_\ell|_{\Gamma_D} := \big\{E \in \mathcal{E}_\ell \,:\, E \subseteq \overline{\Gamma}_D\big\} \quad \text{and} \quad \mathcal{E}_\ell|_{\Gamma_N} := \big\{E \in \mathcal{E}_\ell \,:\, E \subseteq \overline{\Gamma}_N\big\}$$

define meshes of Γ_D and Γ_N, respectively. By now, we have thus defined the discrete spaces $\mathcal{P}^0(\mathcal{E}_\ell)$, $\mathcal{P}^0(\mathcal{E}_\ell|_{\Gamma_D})$, $\mathcal{P}^0(\mathcal{E}_\ell|_{\Gamma_N})$, $\mathcal{S}^1(\mathcal{E}_\ell)$, $\mathcal{S}^1(\mathcal{E}_\ell|_{\Gamma_D})$, and $\mathcal{S}^1(\mathcal{E}_\ell|_{\Gamma_N})$. In addition, we now define the discrete space

$$\mathcal{S}^1_0(\mathcal{E}_\ell|_{\Gamma_N}) := \big\{V_\ell|_{\Gamma_N} \,:\, V_\ell \in \mathcal{S}^1(\mathcal{E}_\ell) \text{ with } V_\ell|_{\Gamma_D} = 0\big\},$$

i.e., $V_\ell \in \mathcal{S}^1_0(\mathcal{E}_\ell|_{\Gamma_N})$ is a continuous and piecewise affine function which vanishes at the tips of Γ_N. One can then show, that $\mathcal{S}^1_0(\mathcal{E}_\ell|_{\Gamma_N})$ is a discrete subspace of $\widetilde{H}^{1/2}(\Gamma_N)$, whereas $\mathcal{P}^0(\mathcal{E}_\ell|_{\Gamma_D})$ is a subspace of $\widetilde{H}^{-1/2}(\Gamma_D)$.

Extension of the Given Dirichlet and Neumann Data. By Definition (7.2), the solution $\mathbf{u} = (u_N, \phi_D)$ of (7.9) depends on the chosen extensions $\overline{u}_D$ of u_D and $\overline{\phi}_N$ of ϕ_N. We assume additional regularity

$$u_D \in H^1(\Gamma_D) \subset H^{1/2}(\Gamma_D) \quad \text{and} \quad \phi_N \in L^2(\Gamma_N) \subset H^{-1/2}(\Gamma_N). \tag{7.10}$$

Let $\mathcal{E}_0$ be the initial mesh for our numerical computation. We then define $\overline{\phi}_N \in L^2(\Gamma)$ by

$$\overline{\phi}_N|_{\Gamma_N} = \phi_N \quad \text{and} \quad \overline{\phi}_N|_{\Gamma_D} = 0 \tag{7.11}$$

as well as $\overline{u}_D \in H^1(\Gamma)$ by

$$\overline{u}_D|_{\Gamma_D} = u_D \quad \text{and} \quad \overline{u}_D|_{\Gamma_N} \in \mathcal{S}^1(\mathcal{E}_0|_{\Gamma_N}) \text{ with } \overline{u}_D(z) = 0 \text{ for all } z \in \mathcal{K}_0 \cap \Gamma_N. \tag{7.12}$$

As a consequence of the inclusion $H^1(\Gamma) \subset C(\Gamma)$, this extension is unique.

Galerkin Discretization. Let $\mathcal{E}_\ell = \{E_1, \dots, E_N\}$ be a mesh of Γ obtained by certain refinements of the initial mesh $\mathcal{E}_0$. To discretize (7.9), we replace the continuous Dirichlet data $\overline{u}_D \in H^1(\Gamma) \subset C(\Gamma)$ by the nodal interpolant

$$U_{D,\ell} := \sum_{j=1}^N \overline{u}_D(z_j)\zeta_j \in \mathcal{S}^1(\mathcal{E}_\ell) \subset H^1(\Gamma) \tag{7.13}$$

and the Neumann data by its L^2-projection

$$\Phi_{N,\ell} \in \mathcal{P}^0(\mathcal{E}_\ell), \quad \Phi_{N,\ell}|_{E_i} := \frac{1}{\text{length}(E_i)} \int_{E_i} \overline{\phi}_N \, d\Gamma =: \mathbf{p}_i. \tag{7.14}$$

With the vector $\mathbf{g}_i := \overline{u}_D(z_i)$, this leads to the representations

$$U_{D,\ell} = \sum_{i=1}^N \mathbf{g}_i\zeta_i \quad \text{and} \quad \Phi_{N,\ell} = \sum_{i=1}^N \mathbf{p}_i\chi_i = \sum_{\substack{j=1\\ E_j \subseteq \overline{\Gamma}_N}}^N \mathbf{p}_j\chi_j. \tag{7.15}$$

Here, the representation for $\Phi_{N,\ell}$ shrinks to a sum over all elements on the Neumann boundary by definition (7.11) of the extended Neumann data. The representation of $U_{D,\ell}$, however, takes into account all nodes. This is due to the fact that the extension $\overline{u}_D$ of u_D has to be continuous. This leads to $\text{supp}(\overline{u}_D) \cap \Gamma_N \neq \emptyset$ in general. Restricting the sum for $U_{D,\ell}$ to Dirichlet nodes, would thus correspond to a change of the extension $\overline{u}_D$, whence the first component u_N of the solution $\mathbf{u} \in \mathcal{H}$ in every step ℓ!

We now consider the lowest-order Galerkin scheme and replace $\mathcal{H}$ by the discrete space

$$X_\ell := \mathcal{S}^1_0(\mathcal{E}_\ell|_{\Gamma_N}) \times \mathcal{P}^0(\mathcal{E}_\ell|_{\Gamma_D}) \subset \mathcal{H}. \tag{7.16}$$

Altogether, this leads to the following discrete version of the integral equation (7.4): Find $\mathbf{U}_\ell \in X_\ell$ with

$$\langle\!\langle \mathbf{U}_\ell \,, \mathbf{V}_\ell \rangle\!\rangle_A = \langle F_\ell \,, \mathbf{V}_\ell \rangle_{\mathcal{H}^* \times \mathcal{H}} \quad \text{for all } \mathbf{V}_\ell \in X_\ell, \tag{7.17}$$

where the approximated right-hand side is given by

$$F_\ell := (1/2 - A)\begin{pmatrix} U_{D,\ell} \\ \Phi_{N,\ell} \end{pmatrix}. \tag{7.18}$$

We use (7.18) here because the right hand side of (7.4) can hardly be evaluated numerically. In order to write (7.17) as a linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{7.19}$$

we have to fix a basis of the discrete space X_ℓ:

- Let $\mathcal{E}_\ell = \{E_1, \dots, E_N\}$ and assume that $\overline{\Gamma}_D = \bigcup_{j=1}^d E_j$. Then, $\{\chi_1, \dots, \chi_N\}$ is a basis of $\mathcal{P}^0(\mathcal{E}_\ell)$ and $\{\chi_1, \dots, \chi_d\}$ is a basis of $\mathcal{P}^0(\mathcal{E}_\ell|_{\Gamma_D})$.
- Let $\mathcal{K}_\ell = \{z_1, \dots, z_N\}$ and assume that $\{z_1, \dots, z_n\} = \mathcal{K}_\ell \cap \Gamma_N$. Then, $\{\zeta_1, \dots, \zeta_N\}$ is a basis of $\mathcal{S}^1(\mathcal{E}_\ell)$ and $\{\zeta_1, \dots, \zeta_n\}$ is a basis of $\mathcal{S}^1_0(\mathcal{E}_\ell|_{\Gamma_N})$.
- In particular, $\{(\zeta_1, 0), \dots, (\zeta_n, 0), (0, \chi_1), \dots, (0, \chi_d)\}$ is a basis of X_ℓ, and we fix this ordering for the implementation.

With this basis, the assembly of the the Galerkin data $\mathbf{A} \in \mathbb{R}^{(n+d)\times(n+d)}$ and $\mathbf{b} \in \mathbb{R}^{n+d}$ from (7.19) reads as follows: According to Linear Algebra, the Galerkin system (7.17) holds for all $\mathbf{V}_\ell \in X_\ell$ if it holds for all basis functions $(\zeta_j, 0)$ and $(0, \chi_k)$ of X_ℓ. Consequently, we need to compute the vector

$$\mathbf{b} \in \mathbb{R}^{n+d}, \quad \text{where} \quad \mathbf{b}_j := \langle F_\ell\,,\,(\zeta_j, 0)\rangle_{\mathcal{H}^*\times\mathcal{H}}, \quad \mathbf{b}_{n+k} := \langle F_\ell\,,\,(0, \chi_k)\rangle_{\mathcal{H}^*\times\mathcal{H}}, \tag{7.20}$$

for all $j = 1, \dots, n$ and $k = 1, \dots, d$. Recall the matrices $\mathbf{M}, \mathbf{K} \in \mathbb{R}^{N\times N}$ defined in (5.8) and the matrix $\mathbf{W} \in \mathbb{R}^{N\times N}$ from (6.10). With the data representation (7.15), there holds

$$\begin{aligned}
\mathbf{b}_j &= \langle -WU_{D,\ell} + (1/2 - K')\Phi_{N,\ell}\,,\,\zeta_j\rangle_\Gamma \\
&= \frac{1}{2}\langle \Phi_{N,\ell}\,,\,\zeta_j\rangle_\Gamma - \langle \Phi_{N,\ell}\,,\,K\zeta_j\rangle_\Gamma - \langle WU_{D,\ell}\,,\,\zeta_j\rangle_\Gamma \\
&= \frac{1}{2}\sum_{i=1}^N \mathbf{p}_i\langle \chi_i\,,\,\zeta_j\rangle_\Gamma - \sum_{i=1}^N \mathbf{p}_i\langle \chi_i\,,\,K\zeta_j\rangle_\Gamma - \sum_{i=1}^N \mathbf{g}_i\langle W\zeta_i\,,\,\zeta_j\rangle_\Gamma \\
&= \Big(\frac{1}{2}\mathbf{M}^T\mathbf{p} - \mathbf{K}^T\mathbf{p} - \mathbf{W}\mathbf{g}\Big)_j \\
&= \Big(\frac{1}{2}\mathbf{M}^T\mathbf{p} - \mathbf{K}^T\mathbf{p} - \mathbf{W}^T\mathbf{g}\Big)_j,
\end{aligned}$$

where we have finally used the symmetry of $\mathbf{W}$. Now, also recall the matrix $\mathbf{V} \in \mathbb{R}^{N\times N}$ from (5.8). The same type of arguments leads to

$$\begin{aligned}
\mathbf{b}_{n+k} &= \langle (1/2 + K)U_{D,\ell} - V\Phi_{N,\ell}\,,\,\chi_k\rangle_\Gamma \\
&= \frac{1}{2}\langle U_{D,\ell}\,,\,\chi_k\rangle_\Gamma + \langle KU_{D,\ell}\,,\,\chi_k\rangle_\Gamma - \langle V\Phi_{N,\ell}\,,\,\chi_k\rangle_\Gamma \\
&= \frac{1}{2}\sum_{i=1}^N \mathbf{g}_i\langle \zeta_i\,,\,\chi_k\rangle_\Gamma + \sum_{i=1}^N \mathbf{g}_i\langle K\zeta_i\,,\,\chi_k\rangle_\Gamma - \sum_{i=1}^N \mathbf{p}_i\langle V\chi_i\,,\,\chi_k\rangle_\Gamma \\
&= \Big(\frac{1}{2}\mathbf{M}\mathbf{g} + \mathbf{K}\mathbf{g} - \mathbf{V}\mathbf{p}\Big)_k.
\end{aligned}$$

For the right-hand side vector $\mathbf{b}$, we thus obtain the short-hand notation

$$\mathbf{b} = \begin{pmatrix} \big(\frac{1}{2}\mathbf{p}^T\mathbf{M} - \mathbf{p}^T\mathbf{K} - \mathbf{g}^T\mathbf{W}\big)^T|_{\Gamma_N} \\ \big(\frac{1}{2}\mathbf{M}\mathbf{g} + \mathbf{K}\mathbf{g} - \mathbf{V}\mathbf{p}\big)|_{\Gamma_D} \end{pmatrix}. \tag{7.21}$$

To compute the entries of the Galerkin matrix $\mathbf{A}$, we proceed in the same way. With the coefficient vector $\mathbf{x}\in\mathbb{R}^{n+d}$ of the ansatz

$$\mathbf{U}_\ell=(U_{N,\ell},\Phi_{D,\ell})\in X_\ell,\quad U_{N,\ell}=\sum_{i=1}^n\mathbf{x}_i\zeta_i,\quad \Phi_{D,\ell}=\sum_{i=1}^d\mathbf{x}_{n+i}\chi_i,$$

it is easily seen that the entries of $\mathbf{A}$ read

$$\begin{aligned}\mathbf{A}_{ij}&=\langle\!\langle(\zeta_j,0)\,,(\zeta_i,0)\rangle\!\rangle_A, &\mathbf{A}_{i,n+k}&=\langle\!\langle(0,\chi_k)\,,(\zeta_i,0)\rangle\!\rangle_A,\\ \mathbf{A}_{n+k,i}&=\langle\!\langle(\zeta_i,0)\,,(0,\chi_k)\rangle\!\rangle_A, &\mathbf{A}_{n+k,n+m}&=\langle\!\langle(0,\chi_m)\,,(0,\chi_k)\rangle\!\rangle_A,\end{aligned}$$

for all $i,j=1,\dots,n$ and $k,m=1,\dots,d$. Now, a direct computation leads to

$$\begin{aligned}\mathbf{A}_{ij}&=\langle\!\langle(\zeta_j,0)\,,(\zeta_i,0)\rangle\!\rangle_A=\langle A(\zeta_j,0)\,,\,(\zeta_i,0)\rangle=\langle(-K\zeta_j,W\zeta_j)\,,\,(\zeta_i,0)\rangle\\&=\langle W\zeta_j\,,\,\zeta_i\rangle\\ \mathbf{A}_{i,n+k}&=\langle\!\langle(0,\chi_k)\,,(\zeta_i,0)\rangle\!\rangle_A=\langle A(0,\chi_k)\,,\,(\zeta_i,0)\rangle=\langle(V\chi_k,K'\chi_k)\,,\,(\zeta_i,0)\rangle\\&=\langle K'\chi_k\,,\,\zeta_i\rangle\\ \mathbf{A}_{n+k,i}&=\langle\!\langle(\zeta_i,0)\,,(0,\chi_k)\rangle\!\rangle_A=\langle A(\zeta_i,0)\,,\,(0,\chi_k)\rangle=\langle(-K\zeta_i,W\zeta_i)\,,\,(0,\chi_k)\rangle\\&=-\langle K\zeta_i\,,\,\chi_k\rangle\\ \mathbf{A}_{n+k,n+m}&=\langle\!\langle(0,\chi_m)\,,(0,\chi_k)\rangle\!\rangle_A=\langle A(0,\chi_m)\,,\,(0,\chi_k)\rangle=\langle(V\chi_m,K'\chi_k)\,,\,(0,\chi_k)\rangle\\&=\langle V\chi_m\,,\,\chi_k\rangle.\end{aligned}$$

Altogether, we obtain the short-hand notation

$$\begin{pmatrix}\mathbf{W}|_{\Gamma_N\times\Gamma_N} & \mathbf{K}^T|_{\Gamma_N\times\Gamma_D}\\ -\mathbf{K}|_{\Gamma_D\times\Gamma_N} & \mathbf{V}|_{\Gamma_D\times\Gamma_D}\end{pmatrix}\mathbf{x}=\begin{pmatrix}\big(\tfrac12\mathbf{p}^T\mathbf{M}-\mathbf{p}^T\mathbf{K}-\mathbf{g}^T\mathbf{W}\big)^T|_{\Gamma_N}\\ \big(\tfrac12\mathbf{Mg}+\mathbf{Kg}-\mathbf{Vp}\big)|_{\Gamma_D}\end{pmatrix}\tag{7.22}$$

for the linear system (7.19)

Listing 20

```
function [b1,b2] = buildMixedRHS(coordinates,dirichlet,neumann,g,V,K,W,uD,phiN)
nD = size(dirichlet,1);
nN = size(neumann,1);
nE = nD + nN;
elements = [dirichlet;neumann];

%*** arbitrary quadrature on [-1,1] with exactness n >= 2, e.g., gauss(2)
quad_nodes = [-1 1]/sqrt(3);
quad_weights = [1;1];

%*** the remaining code is independent of the chosen quadrature rule
nQ = length(quad_nodes);

%*** build vector of evaluation points as (nQ*nN x 2)-matrix
a = coordinates(neumann(:,1),:);
b = coordinates(neumann(:,2),:);
sx = reshape(a,2*nN,1)*(1-quad_nodes) + reshape(b,2*nN,1)*(1+quad_nodes);
sx = 0.5*reshape(sx',nQ*nN,2);

%*** phiN(sx) usually depends on the normal vector, whence phi takes sx and the
%*** nodes of the respective element to compute the normal
a_sx = reshape(repmat(reshape(a,2*nN,1),1,nQ)',nN*nQ,2);
b_sx = reshape(repmat(reshape(b,2*nN,1),1,nQ)',nN*nQ,2);

%*** perform all necessary evaluations of phi as (nE x nQ)-matrix
phi_sx = reshape(phiN(sx,a_sx,b_sx),nQ,nN)';
```

```
27
28 %*** compute vector p of elementwise integral means of Neumann data phiN
29 p = zeros(nE,1);
30 p(nD+1:nE) = (phi_sx*quad_weights*0.5)';
31
32 %*** update vector g of nodal values of Dirichlet data on Dirichlet boundary
33 g(unique(dirichlet)) = uD(coordinates(unique(dirichlet),:));
34
35 %*** compute mass-type matrix for P0 x S1
36 h = sqrt(sum((coordinates(elements(:,1),:)-coordinates(elements(:,2),:)).^2,2));
37 I = reshape([1:nE;1:nE],2*nE,1);
38 J = reshape(elements',2*nE,1);
39 A = reshape(0.5*[h h]',2*nE,1);
40 M = sparse(I,J,A);
41
42 %*** compute full right-hand side
43 b1 = (0.5*p'*M - p'*K - g'*W)';
44 b2 = M*g*0.5 + K*g - V*p;
```

7.1. Build Right-Hand Side Vector (Listing 20). To compute the vector $\mathbf{b}$ from (7.19), we first recall the representation of $\mathbf{b}$ in (7.22). In the routine for the computation of $\mathbf{b}$, we choose to compute

$$\begin{pmatrix}\mathbf{b}_1\\ \mathbf{b}_2\end{pmatrix} := \begin{pmatrix} \left(\frac{1}{2}\mathbf{p}^T\mathbf{M} - \mathbf{g}^T\mathbf{W} - \mathbf{p}^T\mathbf{K}\right)^T|_{\Gamma_N} \\ \left(\frac{1}{2}\mathbf{M}\mathbf{g} + \mathbf{K}\mathbf{g} - \mathbf{V}\mathbf{p}\right)|_{\Gamma_D}\end{pmatrix}. \tag{7.23}$$

The documentation of Listing 20 reads as follows:

- As input, the function takes the mesh $\mathcal{E}_\ell$ described in terms of `coordinates`, `dirichlet` and `neumann`, where the last two arrays represent the Dirichlet and Neumann part of the boundary. The vector `g` contains —at least for all nodes $z_j \in \mathcal{K}_\ell \cap \Gamma_N$ on the Neumann boundary— the nodal values of $\overline{u}_D$, i.e., `g(j)` $= \overline{u}_D(z_j)$. The matrices `V`, `K`, and `W` are the matrices for the simple-layer potential, the double-layer potential, and the hypersingular integral operator for the mesh $\mathcal{E}_\ell$. Finally, the function handles `uD` and `phiN` provide the Dirichlet and Neumann data with the same conventions discussed above.
- Recall that we have chosen an ordering of the elements such that elements on the Dirichlet boundary are taken into account first. This is realized in Line 5.
- First (Line 8–30), the coefficient vector `p` of the approximate Neumann data $\Phi_{N,\ell} = \Pi_\ell \overline{\phi}_N$ is computed. Details are discussed in Section 6.4.
- In Line 33, the approximate Dirichlet data are updated on all Dirichlet nodes.
- Next, the mass-type matrix `M` is computed (Line 36-40), cf. Section 5.2 for details.
- Finally (Line 43–44), the right-hand side vectors `b1` and `b2` are computed.

LISTING 21

```
function g = buildMixedDirichlet(coordinates,dirichlet,neumann, ...
                                    father2neumann,neumann_old,g_old,uD)
nC = size(coordinates,1);

%*** prolongate Dirichlet data on Neumann boundary
g = zeros(nC,1);
g(neumann(father2neumann(:,1),1)) = g_old(neumann_old(:,1));
g(neumann(father2neumann(:,2),2)) = g_old(neumann_old(:,2));
g(neumann(father2neumann(:,1),2)) = 0.5*sum(g_old(neumann_old),2);

%*** evaluate Dirichlet data on Dirichlet boundary
g(unique(dirichlet)) = uD(coordinates(unique(dirichlet),:));
```

7.2. Prolongation for Mixed Problem (Listing 21). The solution u_N depends on the chosen extension $\overline{u}_D \in H^1(\Gamma)$. We start with a coarse mesh $\mathcal{E}_0$ and choose the extended Dirichlet data $\overline{u}_D$ to satisfy $\overline{u}_D|_{\Gamma_N} \in \mathcal{S}^1(\mathcal{E}_0|_{\Gamma_N})$ with $\overline{u}_D(z_i) = 0$ for all nodes $z_i \in \mathcal{K}_\ell \cap \Gamma_N$. We stress that for all subsequent meshes, which arise by mesh refinement, this extension must not be changed! This is realized in the following way: The data $U_{D,\ell}$ is the point evaluation of u_D on the Dirichlet boundary Γ_D, whereas on the Neumann boundary Γ_N it is just the prolongation of $U_{D,\ell-1}$ to the mesh $\mathcal{E}_\ell$.

It is the purpose of the function **`buildMixedDirichlet`** in Listing 21 to perform the described prolongation. A description of this function reads as follows:

- The function **`buildMixedDirichlet`** takes the following input: The mesh $\mathcal{E}_\ell$ is described by the arrays `coordinates`, `dirichlet`, and `neumann`. The array `neumann_old` is the Neumann part of the mesh $\mathcal{E}_{\ell-1}$ and the link between $\mathcal{E}_{\ell-1}|_{\Gamma_N}$ and $\mathcal{E}_\ell|_{\Gamma_N}$ is given by the array `father2neumann`. The vector `g_old` provides the nodal values of $U_{D,\ell-1}$, and `uD` is a function handle for the Dirichlet data u_D.
- The output is the vector `g`, which contains the nodal values of $U_{D,\ell}$.
- On the Neumann part of the boundary, two cases are distinguished: If a node z_i of the mesh $\mathcal{E}_\ell$ was also a node of $\mathcal{E}_{\ell-1}$, i.e. $z_i \in \mathcal{K}_{\ell-1} \cap \mathcal{K}_\ell$, there holds $U_{D,\ell}(z_i) = U_{D,\ell-1}(z_i)$ (Line 7–8). If a node z_i of the mesh $\mathcal{E}_\ell$ is a new node, i.e. $z_i \in \mathcal{K}_\ell \backslash \mathcal{K}_{\ell-1}$, it is thus the midpoint of some element $E = [z_j, z_k]$ of $\mathcal{E}_{\ell-1}$. Then, $U_{D,\ell}(z_i) = (U_{D,\ell-1}(z_j) + U_{D,\ell-1}(z_k))/2$ (Line 9).
- On the Dirichlet part of the boundary, `g` is just the nodal evaluation of u_D (Line 12).

Listing 22

```
function [coordinates,dirichlet,neumann] = buildMixedElements(coordinates, ...
                                                           dirichlet,neumann)
%*** determine nodes on Dirichlet and Neumann boundary
nC = size(coordinates,1);
nodes_neumann = unique(neumann);
nodes_dirichlet = setdiff((1:nC)',nodes_neumann);

%*** build permutation such that Neumann nodes are first
nodes = [nodes_neumann;nodes_dirichlet];
[foo,permutation] = sort(nodes);

%*** permute indices of nodes
coordinates(permutation,:) = coordinates;
neumann = permutation(neumann);
dirichlet = permutation(dirichlet);
```

7.3. Sort Mesh for Mixed Problem (Listing 22). As described above, we order the degrees of freedom in the form

$$\mathcal{B} := \{(\zeta_1, 0), \dots, (\zeta_n, 0), (0, \chi_1), \dots, (0, \chi_d)\},$$

where $\{(\zeta_1, 0), \dots, (\zeta_n, 0)\}$ is a basis of $\mathcal{S}^1_0(\mathcal{E}_\ell|_{\Gamma_N})$ and $\{(0, \chi_1), \dots, (0, \chi_d)\}$ is a basis of $\mathcal{P}^0(\mathcal{E}_\ell|_{\Gamma_D})$.

For $\mathcal{S}^1_0(\mathcal{E}_\ell|_{\Gamma_N}) \subset \widetilde{H}^{1/2}(\Gamma_N)$ we aim to benefit from the functions already written for the hypersingular integral equation. To do so, we have to embed $\mathcal{S}^1_0(\mathcal{E}_\ell|_{\Gamma_N})$ into $\mathcal{S}^1(\mathcal{E}_\ell|_{\Gamma_N})$. We thus enforce an ordering of the nodes such that $\{z_1, \dots, z_m\} = \mathcal{K}_\ell \cap \overline{\Gamma_N} = \mathcal{K}_\ell \backslash \Gamma_D$, i.e., $\{\zeta_1, \dots, \zeta_m\}$ is a basis of $\mathcal{S}^1(\mathcal{E}_\ell|_{\Gamma_N})$ and, in particular, $n < m$. Note that for Γ_N connected, there holds $m = n + 2$.

The described ordering of the nodes enforces to do some reordering of the array `coordinates` and to adapt the indices in `dirichlet` and `neumann`. Both subjects are done by the function **`buildMixedElements`**.

7.4. Computation of Data Oscillations for Mixed Boundary Data. Instead of the correct variational form (7.9)

$$\langle\!\langle \mathbf{u}\,,\mathbf{v}\rangle\!\rangle_A = \langle F\,,\,\mathbf{v}\rangle_{\mathcal{H}^*\times\mathcal{H}} \quad \text{for all } \mathbf{v}\in\mathcal{H}$$

with solution $\mathbf{u} = (u_N, \phi_D) \in \mathcal{H}$ and test functions $\mathbf{v} = (v_N, \psi_D) \in \mathcal{H}$, we solve the perturbed formulation

$$\langle\!\langle \mathbf{u}_\ell\,,\mathbf{v}\rangle\!\rangle_A = \langle F_\ell\,,\,\mathbf{v}\rangle_{\mathcal{H}^*\times\mathcal{H}} \quad \text{for all } \mathbf{v}\in\mathcal{H}. \tag{7.24}$$

In [AGKP], we prove that the error between the continuous solutions $\mathbf{u}, \mathbf{u}_\ell \in \mathcal{H}$ is controlled by

$$\begin{aligned}
|\!|\!| \mathbf{u}-\mathbf{u}_\ell |\!|\!|_A &\lesssim \|\overline{u}_D - I_\ell\overline{u}_D\|_{H^{1/2}(\Gamma)} + \|\overline{\phi}_N - \Pi_\ell\overline{\phi}_N\|_{H^{-1/2}(\Gamma)}\\
&\lesssim \|h_\ell^{1/2}(\overline{u}_D - I_\ell\overline{u}_D)'\|_{L^2(\Gamma)} + \|h_\ell^{1/2}(\overline{\phi}_N - \Pi_\ell\overline{\phi}_N)\|_{L^2(\Gamma)}\\
&= \|h_\ell^{1/2}(\overline{u}_D - I_\ell\overline{u}_D)'\|_{L^2(\Gamma_D)} + \|h_\ell^{1/2}(\overline{\phi}_N - \Pi_\ell\overline{\phi}_N)\|_{L^2(\Gamma_N)}\\
&= \|h_\ell^{1/2}(u_D - I_\ell u_D)'\|_{L^2(\Gamma_D)} + \|h_\ell^{1/2}(\phi_N - \Pi_\ell\phi_N)\|_{L^2(\Gamma_N)}\\
&=: \mathrm{osc}_{D,\ell} + \mathrm{osc}_{N,\ell}
\end{aligned}$$

where we have used the definition of the chosen extensions $\overline{\phi}_N$ and $\overline{u}_D$. Since the Dörfler marking below uses Hilbert space structure for the indicators, we rewrite the latter estimate in the form

$$|\!|\!| \mathbf{u}-\mathbf{u}_\ell |\!|\!|_A^2 \lesssim \mathrm{osc}_{D,\ell}^2 + \mathrm{osc}_{N,\ell}^2 =: \mathrm{osc}_\ell^2.$$

Note that the right-hand side is computable, and the implementation of $\mathrm{osc}_{D,\ell}$ and $\mathrm{osc}_{N,\ell}$ has already been discussed in Section 5.3 and Section 6.4.

7.5. Computation of Reliable Error Bound for $|\!|\!|\mathbf{u}-\mathbf{U}_\ell|\!|\!|_A$. We assume that the exact solution has additional regularity $\mathbf{u} = (u_N, \phi_D) \in H^1(\Gamma_N)\times L^2(\Gamma_D)$. Let $\mathbf{U}_\ell^\star \in X_\ell$ be the (unknown, but existing) Galerkin solution with respect to the exact right-hand side F instead of F_ℓ. As in the previous section, one can prove that

$$|\!|\!| \mathbf{U}_\ell^\star - \mathbf{U}_\ell |\!|\!|_A \lesssim \mathrm{osc}_\ell.$$

Moreover, the exact Galerkin solution is quasi-optimal. Therefore,

$$|\!|\!| \mathbf{u} - \mathbf{U}_\ell^\star |\!|\!|_A \lesssim |\!|\!| u_N - I_\ell u_N |\!|\!|_{W(\Gamma_N)} + |\!|\!| \phi_D - \Pi_\ell\phi_D |\!|\!|_{V(\Gamma_D)} =: \mathrm{err}_{N,\ell} + \mathrm{err}_{D,\ell}.$$

Altogether, we obtain

$$|\!|\!| \mathbf{u} - \mathbf{U}_\ell |\!|\!|_A^2 \lesssim \mathrm{err}_\ell^2 + \mathrm{osc}_\ell^2 \quad \text{with} \quad \mathrm{err}_\ell^2 = \mathrm{err}_{D,\ell}^2 + \mathrm{err}_{N,\ell}^2.$$

Note that the computation of $\mathrm{err}_{N,\ell}$ and $\mathrm{err}_{D,\ell}$ has already been discussed in Section 5.4 and Section 6.5.

7.6. Computation of $(h-h/2)$-Based A Posteriori Error Estimators. Note that the energy norm $|\!|\!|\cdot|\!|\!|_A$ induced by the Calderón projector A can be written in terms of the energy norms $|\!|\!|\cdot|\!|\!|_{V(\Gamma_D)}$ and $|\!|\!|\cdot|\!|\!|_{W(\Gamma_N)}$ induced by the simple-layer potential $V \in L(\widetilde{H}^{-1/2}(\Gamma_D); H^{1/2}(\Gamma_D))$ and the hypersingular integral operator $W \in L(\widetilde{H}^{1/2}(\Gamma_N); H^{-1/2}(\Gamma_N))$. According to (7.8), there holds

$$|\!|\!| (u_N, \phi_D) |\!|\!|_A^2 = |\!|\!| u_N |\!|\!|_{W(\Gamma_N)}^2 + |\!|\!| \phi_D |\!|\!|_{V(\Gamma_D)}^2.$$

For a posteriori error estimation, we may therefore use the estimators introduced above. Suppose that

$$\mathbf{U}_\ell = (U_{N,\ell}, \Phi_{D,\ell}) \in X_\ell \quad \text{and} \quad \widehat{\mathbf{U}}_\ell = (\widehat{U}_{N,\ell}, \widehat{\Phi}_{D,\ell}) \in \widehat{X}_\ell$$

are Galerkin solutions with respect to the mesh $\mathcal{E}_\ell$ and its uniform refinement $\widehat{\mathcal{E}}_\ell$. As in Section 5.5, we define the following four error estimators for the part of the simple-layer potential:

$$\begin{aligned}
\eta_{D,\ell} &:= |||\widehat{\Phi}_{D,\ell} - \Phi_{D,\ell}|||_{V(\Gamma_D)}, & \widetilde{\eta}_{D,\ell} &:= |||\widehat{\Phi}_{D,\ell} - \Pi_\ell\widehat{\Phi}_{D,\ell}|||_{V(\Gamma_D)},\\
\mu_{D,\ell} &:= \|h_\ell^{1/2}(\widehat{\Phi}_{D,\ell} - \Phi_{D,\ell})\|_{L^2(\Gamma_D)}, & \widetilde{\mu}_{D,\ell} &:= \|h_\ell^{1/2}(\widehat{\Phi}_{D,\ell} - \Pi_\ell\widehat{\Phi}_{D,\ell})\|_{L^2(\Gamma_D)}.
\end{aligned}$$

In analogy to Section 6.6, we define the following four error estimators for the contribution of the hypersingular integral operator:

$$\begin{aligned}
\eta_{N,\ell} &:= |||\widehat{U}_{N,\ell} - U_{N,\ell}|||_{W(\Gamma_N)}, & \widetilde{\eta}_{N,\ell} &:= |||\widehat{U}_{N,\ell} - I_\ell\widehat{U}_{N,\ell}|||_{W(\Gamma_N)},\\
\mu_{N,\ell} &:= \|h_\ell^{1/2}(\widehat{U}_{N,\ell} - U_{N,\ell})'\|_{L^2(\Gamma_N)}, & \widetilde{\mu}_{N,\ell} &:= \|h_\ell^{1/2}(\widehat{U}_{N,\ell} - I_\ell\widehat{U}_{N,\ell})'\|_{L^2(\Gamma_N)}.
\end{aligned}$$

Consequently, we obtain (at least) four a posteriori error estimators for the mixed problem:

$$\begin{aligned}
\eta_\ell^2 &:= \eta_{D,\ell}^2 + \eta_{N,\ell}^2, & \widetilde{\eta}_\ell^2 &:= \widetilde{\eta}_{D,\ell}^2 + \widetilde{\eta}_{N,\ell}^2,\\
\mu_\ell^2 &:= \mu_{D,\ell}^2 + \mu_{N,\ell}^2, & \widetilde{\mu}_\ell{}^2 &:= \widetilde{\mu}_{D,\ell}^2 + \widetilde{\mu}_{N,\ell}^2.
\end{aligned}$$

We remark that the implementation of these error estimators has already been discussed above. With the analytical techniques from [FP, EFFP] and [EFGP], we prove in [AGKP] that there holds equivalence

$$\eta_\ell \lesssim \widetilde{\eta}_\ell \lesssim \widetilde{\mu}_\ell \le \mu_\ell \lesssim \eta_\ell.$$

Moreover, there holds efficiency in the form

$$\eta_\ell \lesssim |||\mathbf{u} - \mathbf{U}_\ell|||_A + \mathrm{osc}_\ell.$$

The constants hidden in the symbol $\lesssim$ depend only on Γ and $\kappa(\mathcal{E}_\ell)$. Under a saturation assumption for the non-perturbed problem, there holds also reliability

$$|||\mathbf{u} - \mathbf{U}_\ell|||_A \lesssim \eta_\ell + \mathrm{osc}_\ell. \tag{7.25}$$

To steer an adaptive mesh-refining algorithm, it is therefore natural to use one of the combined error estimators

$$\begin{aligned}
\varrho_\ell^2 &:= \mu_\ell^2 + \mathrm{osc}_\ell^2 = (\mu_{D,\ell}^2 + \mathrm{osc}_{D,\ell}^2) + (\mu_{N,\ell}^2 + \mathrm{osc}_{N,\ell}^2),\\
\widetilde{\varrho}_\ell^2 &:= \widetilde{\mu}_\ell^2 + \mathrm{osc}_\ell^2 = (\widetilde{\mu}_{D,\ell}^2 + \mathrm{osc}_{D,\ell}^2) + (\widetilde{\mu}_{N,\ell}^2 + \mathrm{osc}_{N,\ell}^2).
\end{aligned}$$

For the same reasons as above, the usual choice is $\widetilde{\varrho}_\ell$ since it avoids the computation of the coarse-mesh Galerkin solution $\mathbf{U}_\ell \in X_\ell$, but only relies on local postprocessing of $\widehat{\mathbf{U}}_\ell$.

7.7. Adaptive Mesh-Refinement. For $E_j \in \mathcal{E}_\ell = \{E_1, \dots, E_N\}$, we consider the refinement indicators

$$\widetilde{\varrho}_\ell(E_j)^2 := \begin{cases} \widetilde{\mu}_{D,\ell}(E_j)^2 + \mathrm{osc}_{D,\ell}(E_j)^2 & \text{if } E_j \subseteq \overline{\Gamma}_D,\\ \widetilde{\mu}_{N,\ell}(E_j)^2 + \mathrm{osc}_{N,\ell}(E_j)^2 & \text{if } E_j \subseteq \overline{\Gamma}_N. \end{cases} \tag{7.26}$$

Note that there holds

$$\widetilde{\varrho}_\ell^2 = \sum_{j=1}^N \widetilde{\varrho}_\ell(E_j)^2. \tag{7.27}$$

With this notation, the adaptive algorithm takes the same form as before:

Input: Initial mesh $\mathcal{E}_0$, Dirichlet data u_D, Neumann data ϕ_N, adaptivity parameter $0 < \theta < 1$, maximal number $N_{\max} \in \mathbb{N}$ of elements, and counter $\ell = 0$.

(i) Build uniformly refined mesh $\widehat{\mathcal{E}}_\ell$.
(ii) Compute Galerkin solution $\widehat{\mathbf{U}}_\ell \in \widehat{X}_\ell$.
(iii) Compute refinement indicators $\widetilde{\varrho}_\ell(E)^2$ for all $E \in \mathcal{E}_\ell$.

(iv) Find minimal set $\mathcal{M}_\ell \subseteq \mathcal{E}_\ell$ such that

$$\theta\, \widetilde{\varrho}_\ell^2 = \theta \sum_{E\in\mathcal{E}_\ell} \widetilde{\varrho}_\ell(E)^2 \le \sum_{E\in\mathcal{M}_\ell} \widetilde{\varrho}_\ell(E)^2. \tag{7.28}$$

(v) Refine (at least) marked elements $E \in \mathcal{M}_\ell$ and obtain mesh $\mathcal{E}_{\ell+1}$ with $\kappa(\mathcal{E}_{\ell+1}) \le 2\kappa(\mathcal{E}_0)$.

(vi) Stop provided that $\#\mathcal{E}_{\ell+1} \ge N_{\max}$; otherwise, increase counter $\ell \mapsto \ell + 1$ and go to (i).

Output: Adaptively generated mesh $\widehat{\mathcal{E}}_\ell$ and corresponding discrete solution $\widehat{\mathbf{U}}_\ell \in \widehat{X}_\ell$.

The marking criterion (7.28) has been proposed in the context of adaptive finite element methods [D]. Let formally $N_{max} = \infty$ so that the adaptive algorithm computes a sequence of discrete solutions $\widehat{\mathbf{U}}_\ell$ (or even $\mathbf{U}_\ell$, although this is not computed). Based on (7.25), which holds under a saturation assumption for the non-perturbed problem, we can show with techniques introduced in [FOP] the convergence of $\widehat{\mathbf{U}}_\ell$ and $\mathbf{U}_\ell$ to $\mathbf{u}$, provided that the right hand side (u_D, ϕ_N) is not disturbed, i.e., $(u_D, \phi_N) = (U_{D,\ell}, \Phi_{N,\ell})$.

In [AFP], we changed the notion of convergence and proved that for certain error estimators — amongst them are $\widetilde{\mu}_\ell$ and μ_ℓ for Symm's integral equation — the adaptive algorithm guarantees convergence $\lim_\ell \widetilde{\mu}_\ell = 0$. This concept is followed in [AGKP] to prove that the adaptive algorithm for the mixed problem stated above, yields $\lim_\ell \widetilde{\varrho}_\ell = 0$. Therefore, if the saturation assumption holds (at least in infinitely many steps), we obtain convergence of $\mathbf{U}_\ell$ to $\mathbf{u}$ due to $\|\mathbf{u} - \mathbf{U}_\ell\|_A^2 \lesssim \varrho_\ell^2$.

Listing 23

```
% adaptiveMixed provides the implementation of an adaptive mesh-refining
% algorithm for the symmetric integral formulation of a mixed boundary value
% problem.

%*** rearrange indices such that Neumann nodes are first
[coordinates,dirichlet,neumann] = ...
    buildMixedElements(coordinates,dirichlet,neumann);

%*** initialize Dirichlet data
g_ = zeros(size(coordinates,1),1);
g_(unique(dirichlet)) = g(coordinates(unique(dirichlet),:));

%*** maximal number of elements
nEmax = 100;

%*** adaptivity parameter
theta = 0.25;
rho = 0.25;

%*** adaptive mesh-refining algorithm
while (size(neumann,1)+size(dirichlet,1) < nEmax )

    %*** refine mesh uniformly
    [coordinates_fine,dirichlet_fine,neumann_fine,...
        father2dirichlet,father2neumann] ...
        = refineBoundaryMesh(coordinates,dirichlet,neumann);

    %*** rearrange indices such that Neumann nodes are first
    [coordinates,dirichlet,neumann] = ...
        buildMixedElements(coordinates,dirichlet,neumann);

    %*** prolongate Dirichlet data to uniformly refined mesh
    g_fine = buildMixedDirichlet(coordinates_fine,dirichlet_fine,neumann_fine, ...
                                 father2neumann,neumann,g_,@g);

    %*** compute integral operators for fine mesh
    elements_fine = [dirichlet_fine;neumann_fine];
    V_fine = buildV(coordinates_fine,elements_fine);
    K_fine = buildK(coordinates_fine,elements_fine);
    W_fine = buildW(coordinates_fine,elements_fine);

    %*** compute right-hand side for fine mesh
    [b1_fine,b2_fine] = buildMixedRHS(coordinates_fine,dirichlet_fine, ...
                                      neumann_fine,g_fine,V_fine,K_fine,W_fine, ...
                                      @g,@phi);

    %*** compute degrees of freedom for fine mesh
    nC_fine = size(coordinates_fine,1);
    nD_fine = size(dirichlet_fine,1);
    freeNeumann_fine = setdiff(1:nC_fine,unique(dirichlet_fine));
    freeDirichlet_fine = 1:nD_fine;
    nN_fine = length(freeNeumann_fine);

    %*** shrink integral operators and right-hand side
    W_fine = W_fine(freeNeumann_fine,freeNeumann_fine);
    K_fine = K_fine(freeDirichlet_fine,freeNeumann_fine);
    V_fine = V_fine(freeDirichlet_fine,freeDirichlet_fine);
    b1_fine = b1_fine(freeNeumann_fine);
    b2_fine = b2_fine(freeDirichlet_fine);

    %*** compute Galerkin solution for fine mesh
    x = [ W_fine K_fine' ; -K_fine V_fine ] \ [ b1_fine ; b2_fine ];

    %*** compute coefficient vectors w.r.t. S1(GammaN) and P0(GammaD)
    dim_fine = length(unique(neumann_fine));
    xN_fine = zeros(dim_fine,1);
    xN_fine(freeNeumann_fine) = x(1:nN_fine);      %** dof on Neumann boundary
    xD_fine = x((1:nD_fine) + nN_fine);            %** dof on Nirichlet boundary

    %*** compute (h-h/2)-error estimator tilde-mu on the associated boundaries
    muD_tilde = computeEstSlpMuTilde(coordinates,dirichlet,father2dirichlet, ...
                                     xD_fine);
    muN_tilde = computeEstHypMuTilde(neumann_fine,neumann,father2neumann, ...
                                     xN_fine);

    %*** compute data oscillations for fine mesh
    oscD_fine = computeOscDirichlet(coordinates_fine,dirichlet_fine,@g);
    oscD = sum(oscD_fine(father2dirichlet),2);
    oscN_fine = computeOscNeumann(coordinates_fine,neumann_fine,@phi);
    oscN = sum(oscN_fine(father2neumann),2);

    %*** mark elements for refinement
    [marked_dirichlet,marked_neumann] ...
        = markElements(theta,rho,muD_tilde + oscD,muN_tilde + oscN);

    %*** generate new mesh
    [coordinates_new,dirichlet_new,neumann_new,...
        father2dirichlet,father2neumann] ...
        = refineBoundaryMesh(coordinates,dirichlet,neumann,...
                             marked_dirichlet,marked_neumann);
```

```
91
92      [coordinates_new,dirichlet_new,neumann_new] = ...
93          buildMixedElements(coordinates_new,dirichlet_new,neumann_new);
94
95      %*** prolongate Dirichlet data to adaptively refined mesh
96      g_ = buildMixedDirichlet(coordinates_new,dirichlet_new,neumann_new, ...
97                               father2neumann,neumann,g_,@g);
98
99      coordinates = coordinates_new;
100     dirichlet = dirichlet_new;
101     neumann = neumann_new;
102 end
```

7.7.1. Implementation of Adaptive Algorithm (Listing 23). The MATLAB script of Listing 23 realizes the adaptive algorithm from the beginning of this section.

- We first order the nodes such that nodes on $\overline{\Gamma}_N$ are first (Line 6–7) and compute the nodal vector of $U_{D,0}$ (Line 10-11)
- We use the adaptivity parameter $\theta = 1/4$ in (7.28) and mark at least 25% of elements with the largest indicators (Line 17–18).

The remainder of the code consists of the adaptive loop, where $\mathcal{E}_\ell$ is a given mesh with associated discrete Dirichlet data $U_{D,\ell}$.

- We generate the mesh $\widehat{\mathcal{E}}_\ell$ (Line 24–26), order the nodes such that nodes on $\overline{\Gamma}_N$ are first (Line 29–30), and compute the nodal vector of $\widehat{U}_{D,\ell}$ (Line 33-34).
- We build the discrete integral operators related to $\widehat{\mathcal{E}}_\ell$ (Line 37–40) and the corresponding right-hand side (Line 43–45). Note that the latter is built with respect to the improved data $(\widehat{U}_{D,\ell}, \widehat{\Phi}_{N,\ell})$ instead of $(U_{D,\ell}, \Phi_{N,\ell})$.
- By definition, the degrees of freedom are the elements on the Dirichlet boundary, which are the first N_D elements (Line 51), as well as the nodes $\widehat{\mathcal{K}}_\ell \backslash \overline{\Gamma}_D$, which lie inside of Γ_N (Line 50).
- To lower the storage, we restrict the discrete operators and the right-hand side to the degrees of freedom (Line 55–59). For instance, $\mathbf{V}$ is only needed for elements on Γ_D, and $\mathbf{W}$ is only needed for nodes $z_\ell \in \mathcal{K}_\ell \backslash \overline{\Gamma}_D$.
- Finally (Line 62), we compute the coefficient vector $\widehat{\mathbf{x}}$ of $\widehat{\mathbf{U}}_\ell$ by solving (7.22).
- Next, we aim to obtain the basis vectors $\widehat{x}^N$ and $\widehat{x}^D$ of $U_{N,\ell}$ and $\Phi_{D,\ell}$, respectively. To use the functions from Section 6, we have to represent $U_{N,\ell}$ in the nodal basis of $\mathcal{S}^1(\mathcal{E}_\ell|_{\overline{\Gamma}_N})$. This is done in Line 65–67. The coefficients of $\Phi_{D,\ell}$ with respect to $\mathcal{P}^0(\mathcal{E}_\ell|_{\overline{\Gamma}_D})$ are obtained in Line 68.
- We compute the local contributions of the error estimator $\widetilde{\mu}_\ell^2 = \widetilde{\mu}_{D,\ell}^2 + \widetilde{\mu}_{N,\ell}^2$ (Line 71—74).
- Next, we compute the data oscillations (Line 77–80). By definition, we have used the data for $\widehat{\mathcal{E}}_\ell$ and have to sum the son contributions to obtain the oscillations on the coarse mesh $\mathcal{E}_\ell$.
- In Line 83–84, the Dörfler marking (7.28) is realized.
- Finally, the new mesh $\mathcal{E}_{\ell+1}$ is created (Line 87–90) and ordered (Line 92–93), and we compute the nodal vector of $U_{D,\ell+1}$ (Line 96–97).

Acknowledgement. The authors PG, MK, MM, and DP are partially funded through the research project *Adaptive Boundary Element Method*, funded by the Austria Science Fund (FWF) under grant P21732. The author SFL acknowledges a grant of the graduate school *Differential Equations – Models in Science and Engineering*, funded by the Austrian Science Fund (FWF) under grant W800-N05.

References

[AEFMGKP] M. Aurada, M. Ebner, S. Ferraz-Leite, M. Mayr, P. Goldenits, M. Karkulik, D. Praetorius: *HILBERT — A Matlab implementation of adaptive BEM*, software download at `http://www.asc.tuwien.ac.at/abem/hilbert/`

[AFP] M. Aurada, S. Ferraz-Leite, D. Praetorius: *Estimator Reduction and Convergence of Adaptive FEM and BEM*, ASC Report **27/2009**, Institute for Analysis and Scientific Computing, Vienna University of Technology, 2009, submitted for publication.

[AGP] M. Aurada, P. Goldenits, D. Praetorius: *Convergence of Data Perturbed Adaptive Boundary Element Methods*, ASC Report **40/2009**, Institute for Analysis and Scientific Computing, Vienna University of Technology, 2009, submitted for publication.

[AGKP] M. Aurada, P. Goldenits, M. Karkulik, D. Praetorius: *Adaptive BEM for some Mixed Boundary Value Problem*, work in progress 2010.

[BLM] S. Börm, M. Löhndorf, J. Melenk: *Approximation of Integral Operators by Variable-Order Interpolation*, Numer. Math. **99** (2005), 605-643.

[CKNS] J. Cascon, C. Kreuzer, R. Nochetto, K. Siebert: *Quasi-Optimal Convergence Rate for an Adaptive Finite Element Method*, SIAM J. Numer. Anal. **46** (2008), 2524–2550.

[CP1] C. Carstensen, D. Praetorius: *Averaging Techniques for the Effective Numerical Solution of Symm's Integral Equation of the First Kind*, SIAM J. Sci. Comput., **27** (2006), 1226–1260.

[CP2] C. Carstensen, D. Praetorius: *Averaging Techniques for the A Posteriori BEM Error Control for a Hypersingular Integral Equation in Two Dimensions*, SIAM J. Sci. Comput., **29** (2007), 782–810.

[D] W. Doerfler: *A Convergent Adaptive Algorithm for Possion's Equation*, SIAM J. Numer. Anal. **33** (1996), 1106–1124.

[DN] W. Doerfler, R. Nochetto: *Small Data Oscillation Implies the Saturation Assumption*, Numer. Math. 91 (2002), 1–12.

[EFFP] C. Erath, S. Ferraz-Leite, S. Funken, D. Praetorius: *Energy Norm Based A Posteriori Error Estimation for Boundary Element Methods in Two Dimensions*, Appl. Numer. Math., in press (2009).

[EFGP] C. Erath, S. Funken, P. Goldenits, D. Praetorius: *Simple Error Estimators for the Galerkin BEM for some Hypersingular Integral Equation in 2D*, submitted for publication (2009).

[FOP] S. Ferraz-Leite, C. Ortner, D. Praetorius: *Convergence of Simple Adaptive Galerkin Schemes Based on $h - h/2$ Error Estimators*, ASC Report **06/2009**, Institute for Analysis and Scientific Computing, Vienna University of Technology, 2009, submitted for publication.

[FP] S. Ferraz-Leite, D. Praetorius: *Simple A Posteriori Error Estimators for the h-Version of the Boundary Element Method*, Computing **83** (2008), 135–162.

[FPW] S. Funken, D. Praetorius, P. Wissgott: *Efficient Implementation of Adaptive P1-FEM in MATLAB* ASC Report **19/2008**, Institute for Analysis and Scientific Computing, Vienna University of Technology, 2008, submitted for publication.

[H] W. Hackbusch: *Hierarchische Matrizen — Algorithmen und Analysis*, Springer, Berlin 2009.

[M] M. Maischak: *The Analytical Computation of the Galerkin Elements for the Laplace, Lamé and Helmholtz Equation in 2D-BEM*, Preprint, Institut für Angewandte Mathematik, Universität Hannover, Hannover, 1999.

[Ma] M. Mayr: *Stabile Implementierung der Randelementmethode auf stark adaptierten Netzen*, Bachelor thesis (in German), Institute for Analysis and Scientific Computing, Vienna University of Technology, 2010.

[Mc] W. McLean: *Strongly Elliptic Systems and Boundary Integral Equations*, Cambridge University Press, Cambridge, 2000.

[MSW] P. Mund, E. Stephan, J. Weisse: *Two-Level Methods for the Single Layer Potential in $\mathbb{R}^3$*, Computing **60** (1998), 243–266.

[RS] S. Rjasanov, O. Steinbach: *The Fast Solution of Boundary Integral Equations*, Springer, New York, 2007.

[SS] S. Sauter, C. Schwab: *Randelementmethoden: Analysis, Numerik und Implementierung schneller Algorithmen*, Teubner, Wiesbaden, 2004.

[S] O. Steinbach: *Numerical Approximation Methods for Elliptic Boundary Value Problems: Finite and Boundary Elements*, Springer, New York, 2008.

[Stev] R. Stevenson: *The Completion of Locally Refined Simplicial Partitions Created by Bisection*, Math. Comp. **77** (2008), 227–241.

Institute for Analysis and Scientific Computing, Vienna University of Technology, Wiedner Hauptstrasse 8-10, A-1040 Wien, Austria

E-mail address: Dirk.Praetorius@tuwien.ac.at (corresponding author)