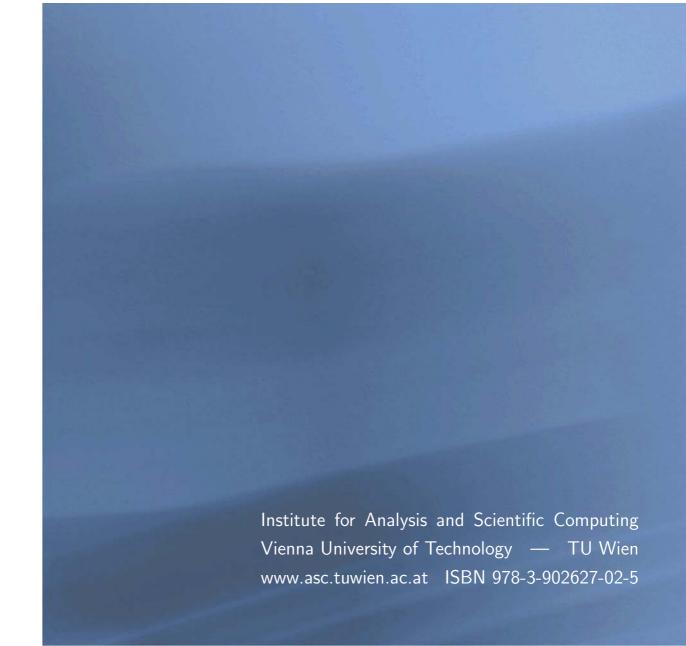# HILBERT – A MATLAB Implementation of Adaptive BEM

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

Most recent ASC Reports

ASC
TU WIEN

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`. Throughout Section 1, we give a concise introduction to BEM. Furthermore, the reader will be introduced to the data structures of `HILBERT` and mesh-refinement strategies. In the further sections, we discuss our approach of solving the Dirichlet problem (Section 2), the Neumann problem (Section 3), and the mixed boundary value problem with Dirichlet and Neumann boundary conditions (Section 4). 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 adequate 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. BOUNDARY ELEMENT METHOD

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

$$(1.1) \qquad\qquad -\Delta u = f \quad \text{in } \Omega$$

can explicitly be written in the form

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

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

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

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

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

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

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

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

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

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

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 2, 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. In Section 3, we consider the Neumann problem, where the normal derivative ϕ is known and where the unknown trace g has to be computed. Finally, in Section 4, 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 $\Gamma_N \subset \Gamma$.

1.1. Discretization of the Boundary. 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 $\Gamma_D \cap \Gamma_N = \emptyset$.

A triangulation or mesh of Γ is a finite set $\mathcal{T}_\ell = \{E_1, \ldots, E_N\}$ such that the elements $E_j \in \mathcal{T}_\ell$ are affine line segments, i.e., there holds

$$(1.8) \qquad E_j = [a_j, b_j] := \operatorname{conv}\{a_j, b_j\}.$$

If Γ is partitioned into Γ_D and Γ_N, one usually assumes that this partition is resolved by $\mathcal{T}_\ell$, i.e., $E_j \in \mathcal{T}_\ell$ satisfies either $E_j \subseteq \overline{\Gamma}_D$ or $E_j \subseteq \overline{\Gamma}_N$. With each element $E_j = [a_j, b_j]$, we associate the affine mapping

$$(1.9) \qquad \gamma_j : [-1, 1] \to E_j, \quad \gamma_j(s) = \frac{1}{2}\big(a_j + b_j + s(b_j - a_j)\big)$$

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

Finally, $\mathcal{K}_\ell = \{z_1, \ldots, z_N\}$ denotes the set of all nodes of the triangulation $\mathcal{T}_\ell$, and we note that there holds $\#\mathcal{T}_\ell = \#\mathcal{K}_\ell$ for the closed boundary Γ.

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

$$(1.10) \qquad f_\ell \circ \gamma_j \in \mathcal{P}^p[-1, 1],$$

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

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

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

$$(1.11) \qquad h_\ell|_E := \operatorname{length}(E) \quad \text{for all } E \in \mathcal{T}_\ell.$$

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

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

2

1.3. Boundary Integrals. Let $I \subset \mathbb{R}$ be a compact interval in $\mathbb{R}$. For $E_j \in \mathcal{T}_\ell$, 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

$$(1.12) \qquad \int_{E_j} f \, d\Gamma = \int_{E_j} f(x) \, d\Gamma(x) := \int_I (f \circ \pi_j)(t) \, |\pi_j'(t)| \, dt.$$

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

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

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

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

holds

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

1.4. Arclength Derivative. Let $I \subset \mathbb{R}$ be a compact interval in $\mathbb{R}$. For $E_j \in \mathcal{T}_\ell$, 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 is then defined by

$$(1.13) \qquad (f' \circ \pi_j)(t) = \frac{1}{|\pi_j'(t)|} (f \circ \pi_j)'(t) \quad \text{for all } x = \pi_j(t) \in E_j.$$

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

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

whereas the arclength parametrization β_j leads to

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

1.5. Data Structure. The set of nodes $\mathcal{K}_\ell = \{z_1, \ldots, z_N\}$ of the triangulation $\mathcal{T}_\ell = \{E_1, \ldots, E_N\}$ 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

    `coordinates(j,:) = [ `x_j` `y_j` ]`.

If Γ is not split into several parts, the triangulation $\mathcal{T}_\ell$ 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

    `elements(i,:) = [ j  k ]`,

where the nodes are given in counterclockwise order, i.e., the parametrization of the boundary element $E_i \subset \Gamma$ is mathematically positive.

If Γ is split into Dirichlet boundary Γ_D and Neumann boundary Γ_N, the triangulation $\mathcal{T}_\ell$ 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.

LISTING 1

```
1  function [coordinates,varargout] = refineBoundaryMesh(coordinates,varargin)
```

3

```matlab
 2  %*** fix the blow-up factor for the K-mesh constant,
 3  %*** where we assume C(Mesh_0) = 1, i.e., the initial mesh is uniform
 4  kappa = 2;
 5
 6  %*** count number of boundary parts from input
 7  %*** nB will hold this number
 8  nB = 0;
 9
10  for iter = 1 : (nargin - 1)
11
12      if size(varargin{iter},2) == 2
13          nB = nB + 1;
14          nE_boundary(iter) = size(varargin{iter},1);
15      else
16          break;
17      end
18
19  end
20
21  %*** check the correct number of input parameters
22  if ~( (nargin == (nB+1)) || (nargin == (2*nB+1)) )
23      error('refineBoundaryMesh: Wrong number of input arguments!');
24  end
25
26  %*** check the correct number of output parameters
27  if ~( (nargout == (nB+1)) || (nargout == (2*nB+1)) )
28      error('refineBoundaryMesh: Wrong number of output arguments!');
29  end
30
31  %*** check, if user asks for father2son fields in output
32  if nargout == (2*nB+1)
33      output_father2son = true;
34  else
35      output_father2son = false;
36  end
37
38  %*** obtain set of all elements of the boundary partition
39  elements = cat(1,varargin{1 : nB});
40
41  %*** indices of a boundary part w.r.t. entire field elements
42  ptr_boundary = cumsum([0,nE_boundary]);
43
44  %*** 1. determine whether uniform or adaptive mesh-refinement
45  %*** 2. in case of adaptive mesh-refinement compute vector marked
46  %***    of marked elements w.r.t. entire field elements
47  if (nB+1) == nargin
48      refinement = 'uniform';
49  else
50      refinement = 'adaptive';
51      marked = zeros(0,1); % marked elements w.r.t. entire field elements
52
53      for iter = 1 : nB
54          marked = [marked; varargin{iter + nB} + ptr_boundary(iter)];
55      end
56
57  end
58
```

```matlab
59  nC = size(coordinates,1); % number of coordinates
60  nE = size(elements,1);    % number of elements
61
62  if strcmp(refinement,'adaptive')
63
64      %*** if element Ej is marked and if its neighbour Ek satisfies
65      %*** hk >= kappa*hj, we (recursively) mark Ek for refinement as well
66
67      %*** marked elements Ej will be refined, i.e., flag(j) = 1
68      flag = zeros(nE,1);
69      flag(marked) = 1;
70
71      %*** determine neighbouring elements
72      node2element = zeros(nC,2);
73      node2element(elements(:,1),2) = (1:nE)';
74      node2element(elements(:,2),1) = (1:nE)';
75      element2neighbour = [ node2element(elements(:,1),1), ...
76                            node2element(elements(:,2),2) ];
77
78      %*** compute (squared) local mesh-size
79      h = sum((coordinates(elements(:,1),:)-coordinates(elements(:,2),:)).^2,2)';
80
81      %*** the formal recursion is avoided by sorting elements by mesh-size
82      [tmp,sorted_elements] = sort(h);
83      for j = sorted_elements
84          if flag(j)
85              neighbours = element2neighbour(j,:);
86              neighbours = neighbours( find(neighbours) );
87              flag( neighbours(h(neighbours) >= kappa*h(j)) ) = 1;
88          end
89      end
90
91      %*** obtain vector of marked elements
92      marked = find(flag);
93      nM = length(marked);
94
95      %*** compute and add new nodes
96      coordinates = [coordinates;zeros(nM,2)];
97      coordinates((1:nM)+nC,:) = ( coordinates(elements(marked,1),:) ...
98                                  + coordinates(elements(marked,2),:) )*0.5;
99
100     %*** refinement of mesh iterates over each boundary part
101     for iter = 1:nB
102
103         %*** determine which marked elements belong to boundary part
104         idx = find( (ptr_boundary(iter) < marked) ...
105                     & (marked <= ptr_boundary(iter+1)) );
106         nM_boundary = length(idx);
107
108         %*** allocate new elements
109         new_elements = [varargin{iter};zeros(nM_boundary,2)];
110
111         %*** generate new elements
112         new_elements((1:nM_boundary)+nE_boundary(iter),:) ...
113             = [ nC + idx, elements(marked(idx),2) ];
114         new_elements( marked(idx) - ptr_boundary(iter),2 ) = nC + idx;
115
```

```
116          %*** add new_elements and father2son to output
117          varargout{iter} = new_elements;
118
119          %*** compute father2son only if desired
120          if output_father2son == true
121
122              %*** generate father2son
123              father2son = repmat((1:nE_boundary(iter))',1,2);
124              father2son( marked(idx) − ptr_boundary(iter),2 ) ...
125                  = (1:nM_boundary)' + nE_boundary(iter);
126
127              %*** add new_elements and father2son to output
128              varargout{nB+iter} = father2son;
129          end
130
131      end
132
133  elseif strcmp(refinement,'uniform')
134
135      %*** compute and add new nodes
136      coordinates = [coordinates;zeros(nE,2)];
137      coordinates((nC+1):end,:) = ( coordinates(elements(:,1),:) ...
138                                  + coordinates(elements(:,2),:) )*0.5;
139
140      %*** uniform refinement of mesh iterates over each boundary part
141      for iter = 1:nB
142
143          %*** generate new elements
144          idx = (ptr_boundary(iter)+1):ptr_boundary(iter+1);
145          varargout{iter} = [ varargin{iter}(:,1),nC + idx' ; ...
146                              nC + idx', varargin{iter}(:,2) ];
147
148          %*** compute father2son only if desired
149          if output_father2son == true
150
151              %*** build father2son
152              varargout{nB+iter} ...
153                  = [(1:nE_boundary(iter))', ...
154                     (1:nE_boundary(iter))' + nE_boundary(iter)];
155          end
156
157      end
158  end
```

1.6. Local Refinement of Boundary Element Mesh (Listing 1). In many cases, one is not interested in computing only one approximation U with respect to a fixed given boundary element mesh $\mathcal{T}$, but in computing a sequence of more and more accurate approximations U_ℓ corresponding to a sequence $\mathcal{T}_\ell$ 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 1, where the data structure of `coordinates`, `elements`, `dirichlet`, and `neumann` is described in Section 1.5 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 $\mathcal{T}_\ell = \{E_1, \ldots, E_N\}$ is described by `coordinates` and `elements`. Then,

  `[coordinates_fine,elements_fine,father2son] ...`
  `    = refineBoundaryMesh(coordinates,elements)`

  provides the uniformly refined mesh $\widehat{\mathcal{T}}_\ell = \{e_1, \ldots, e_{2N}\}$, where each element $E_i \in \mathcal{T}_\ell$ is bisected in certain sons $e_j, e_k \in \widehat{\mathcal{T}}_\ell$. 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 $\mathcal{M}_\ell \subseteq \mathcal{T}_\ell$ 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 $\mathcal{T}_{\ell+1}$ which is only refined locally in the sense that all elements of $\mathcal{M}_\ell$ are refined. If an element $E_i \in \mathcal{T}_\ell$ is not refined, there holds $E_i = e_j \in \mathcal{T}_{\ell+1}$, 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 triangulation $\mathcal{T}_\ell = \{E_1, \ldots, E_N\}$ is described in terms of `coordinates`, `dirichlet`, and `neumann`, cf. Section 1.5. Then,

  `[coordinates_fine,dirichlet_fine,neumann_fine,dir2son,neu2son] ...`
  `    = refineBoundaryMesh(coordinates,dirichlet,neumann)`

  provides the uniformly refined mesh $\widehat{\mathcal{T}}_\ell$. 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 $\mathcal{T}_\ell|_{\Gamma_D} = \{E_1^D, \ldots, E_{N_D}^D\}$ and $\widehat{\mathcal{T}}_\ell|_{\Gamma_D} = \{e_1^D, \ldots, e_{2N_D}^D\}$. 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 $\mathcal{M}_\ell^D \subseteq \mathcal{T}_\ell|_{\Gamma_D}$ and $\mathcal{M}_\ell^N \subseteq \mathcal{T}_\ell|_{\Gamma_N}$ 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 $\mathcal{T}_{\ell+1}$ 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 $E_i = [a, b] \in \mathcal{T}_\ell$ is bisected into two sons $e_j, e_k \in \mathcal{T}_{\ell+1}$ (or $\widehat{\mathcal{T}}_\ell$). 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 $p = \texttt{elements\_fine(father2son(i,1),1)}$.

- **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

$$(1.14) \qquad \kappa(\mathcal{T}_\ell) := \sup \left\{ \text{length}(E_j)/\text{length}(E_k) \, : \, E_j, E_k \in \mathcal{T}_\ell \text{ with } E_j \cap E_k \neq \emptyset \right\} \geq 1$$

which is the maximal ratio of the element widths of neighbouring elements. Let $\mathcal{T}_0$ be a given initial mesh. Let $\mathcal{T}_\ell$ be inductively obtained by refinement of arbitrary sets of marked elements $\mathcal{M}_j \subseteq \mathcal{T}_j$ 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 $\sup_{\ell \in \mathbb{N}} \kappa(\mathcal{T}_\ell) < \infty$. Our refinement rule guarantees

$$(1.15) \qquad \sup_{\ell \in \mathbb{N}} \kappa(\mathcal{T}_\ell) \leq 2 \, \kappa(\mathcal{T}_0)$$

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

$$(1.16) \qquad \#\mathcal{T}_\ell - \#\mathcal{T}_0 \lesssim \sum_{j=0}^{\ell-1} \#\mathcal{M}_j,$$

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

Finally, we give a rough overview on the code:

- The treatment of the variable input-/output parameters is 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 $E_i \in \mathcal{T}_\ell$ is marked for refinement, any neighbour E_j with

$$h_\ell|_{E_j}/h_\ell|_{E_i} \geq 2$$

  is recursively marked for refinement as well (Line 68–92). This guarantees $\kappa(\mathcal{T}_\ell) \leq 2\kappa(\mathcal{T}_0)$ for all generated meshes $\mathcal{T}_\ell$.
- 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.

2. Symm's Integral Equation

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

$$(2.1) \qquad V\phi = (K + 1/2)g \quad \text{on } \Gamma$$

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

$$(2.2) \qquad -\Delta u = 0 \text{ in } \Omega \quad \text{with} \quad u = g \text{ on } \Gamma.$$

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 (2.1) is the normal derivative $\phi = \partial_n u$ of the solution $u \in H^1(\Omega)$ of (2.2).

Note that (2.1) can equivalently be written in variational form

$$(2.3) \qquad \langle V\phi, \psi \rangle_\Gamma = \langle (K + 1/2)g, \psi \rangle_\Gamma \quad \text{for all } \psi \in H^{-1/2}(\Gamma),$$

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 $\text{diam}(\Omega) < 1$, one can show that the left-hand side

$$(2.4) \qquad \langle\!\langle \phi, \psi \rangle\!\rangle_V := \langle V\phi, \psi \rangle_\Gamma \quad \text{for } \phi, \psi \in H^{-1/2}(\Gamma)$$

of (2.3) 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 (2.3) 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 (2.3), we first replace the continuous Dirichlet data $g \in H^1(\Gamma)$ by its nodal interpolant

$$(2.5) \qquad G_\ell := \sum_{j=1}^N g(z_j)\zeta_j \in \mathcal{S}^1(\mathcal{T}_\ell) \subset H^1(\Gamma)$$

where $\{\zeta_1, \dots, \zeta_N\}$ denotes the set of canonical basis functions of $\mathcal{S}^1(\mathcal{T}_\ell)$. Second, we replace the entire function space $H^{-1/2}(\Gamma)$ in (2.3) by the finite-dimensional space $\mathcal{P}^0(\mathcal{T}_\ell)$. Since the discrete space $\mathcal{P}^0(\mathcal{T}_\ell)$ is a subspace of $H^{-1/2}(\Gamma)$, $\langle\!\langle \cdot, \cdot \rangle\!\rangle_V$ from (2.4) is also a scalar product on $\mathcal{P}^0(\mathcal{T}_\ell)$. Consequently, there is a unique Galerkin solution $\Phi_\ell \in \mathcal{P}^0(\mathcal{T}_\ell)$ of

$$(2.6) \qquad \langle V\Phi_\ell, \Psi_\ell \rangle_\Gamma = \langle (K + 1/2)G_\ell, \Psi_\ell \rangle_\Gamma \quad \text{for all } \Psi_\ell \in \mathcal{P}^0(\mathcal{T}_\ell).$$

According to Linear Algebra, (2.6) holds for all $\Psi_\ell \in \mathcal{P}^0(\mathcal{T}_\ell)$ if and only if it holds for all (canonical) basis functions $\chi_k \in \mathcal{B}_\ell = \{\chi_1, \dots, \chi_N\}$ of $\mathcal{P}^0(\mathcal{T}_\ell)$. With the coefficient vector $\mathbf{x} \in \mathbb{R}^N$ of the ansatz

$$(2.7) \qquad \Phi_\ell = \sum_{j=1}^N \mathbf{x}_j \chi_j$$

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 (2.6) 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

$$(2.8) \quad \mathbf{V}_{kj} = \langle V\chi_j, \chi_k \rangle_\Gamma, \quad \mathbf{K}_{kj} = \langle K\zeta_j, \chi_k \rangle_\Gamma, \quad \mathbf{M}_{kj} = \langle \zeta_j, \chi_k \rangle_\Gamma \quad \text{for all } j, k = 1, \dots, N,$$

the last equation becomes

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

Altogether, the Galerkin formulation (2.6) is thus equivalent to the linear system

$$(2.9) \qquad \mathbf{V}\mathbf{x} = \mathbf{K}\mathbf{g} + \frac{1}{2}\,\mathbf{M}\mathbf{g}.$$

We stress that $\mathbf{V}$ is symmetric and positive definite since it stems from a scalar product. In particular, the linear system (2.9) has a unique solution $\mathbf{x} \in \mathbb{R}^N$.

2.1. Matrices for Integral Operators. The matrices $\mathbf{V}, \mathbf{K} \in \mathbb{R}^{N \times N}$ defined in (2.8) are implemented in the programming language C via the MATLAB-MEX-Interface.

LISTING 2

```
1  function b = buildSymmRHS(coordinates,elements,uD)
2  %*** nodal evaluation of Dirichlet data
3  uDh = uD(coordinates);
4
5  %*** compute DLP-matrix for P0 x S1
6  K = buildK(coordinates,elements);
7
8  %*** compute mass-type matrix for P0 x S1
9  nE = size(elements,1);
10 h = sqrt(sum((coordinates(elements(:,1),:)-coordinates(elements(:,2),:)).^2,2));
11 I = reshape(repmat(1:nE,2,1),2*nE,1);
12 J = reshape(elements',2*nE,1);
13 A = reshape(repmat(0.5*h,1,2)',2*nE,1);
14 M = sparse(I,J,A);
15
16 %*** build right-hand side vector
17 b = K*uDh + M*uDh*0.5;
```

2.2. Building of Right-Hand Side Vector (Listing 2). To compute the vector

$$(2.10) \qquad \mathbf{b} := \mathbf{K}\mathbf{g} + \frac{1}{2}\mathbf{M}\mathbf{g} \in \mathbb{R}^N$$

from (2.9), 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 $E_k \in \mathcal{T}_\ell$, 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

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

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 $k = 1, \ldots, N$, one consequently expects a computational complexity of order $\mathcal{O}(N^2 \log N)$ which can even be observed experimentally.

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

Altogether, the documentation of Listing 2 reads as follows:

- The function takes the mesh $\mathcal{T}_\ell$ 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 $\mathbf{h} \in \mathbb{R}^N$ contains $\mathbf{h}_j = \text{length}(E_j)$ (Line 10), where we stress that the Euclidean length `h = norm(b−a)` can also be computed via `h = sqrt(sum((b−a).^2,2))` if $\mathbf{a}, \mathbf{b} \in \mathbb{R}^2$ are row-vectors. Then, the vectors $\mathbf{I}, \mathbf{J}, \mathbf{A} \in \mathbb{R}^{2N}$ 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).

<div align="center">LISTING 3</div>

```
1  function osc = computeOscDirichlet(coordinates,elements,uD)
2  %*** compute midpoints of all elements
3  midpoints = 0.5*( coordinates(elements(:,1),:) + coordinates(elements(:,2),:) );
4
5  %*** evaluate Dirichlet data at element midpoints
6  uD_midpoints = uD(midpoints);
7
8  %*** evaluate Dirichlet data at all nodes
9  uD_nodes = uD(coordinates);
10
11 %*** compute oscillations of Dirichlet data via adapted Newton–Cotes formula
12 osc = 4/3*( uD_nodes(elements(:,1))+uD_nodes(elements(:,2))−2*uD_midpoints ).^2;
```

2.3. Computation of Data Oscillations for Dirichlet Data (Listing 3). Instead of solving the correct variational form (2.3), we solve

$$(2.11) \qquad \langle V\phi_\ell\,,\,\psi\rangle_\Gamma = \langle (K+1/2)G_\ell\,,\,\psi\rangle_\Gamma \quad \text{for all } \psi \in H^{-1/2}(\Gamma)$$

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 (2.3) and the exact solution $\phi_\ell \in H^{-1/2}(\Gamma)$ of the perturbed formulation (2.11) is controlled by

$$(2.12) \qquad \|\phi - \phi_\ell\|_V \lesssim \|h_\ell^{1/2}(g - G_\ell)'\|_{L^2(\Gamma)} =: \text{osc}_{D,\ell}$$

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

We now aim for a numerical approximation of the local contributions

$$\text{osc}_{D,\ell}(E_j) = \|h_\ell^{1/2}(g - G_\ell)'\|_{L^2(E_j)} = \text{length}(E_j)^{1/2}\|(g - G_\ell)'\|_{L^2(E_j)} \quad \text{for all } E_j \in \mathcal{T}_\ell.$$

For $E_j = [a_j, b_j] \in \mathcal{T}_\ell$ and $h := \text{length}(E_j) = |b_j - a_j|$, let $\gamma_j : [-1, 1] \to E_j$ denote the reference parametrization from (1.9). Recall that $|\gamma_j'| = h/2$. With the definition of a boundary integral from Section 1.3 and the definition of the arclength derivative from Section 1.4, we obtain

$$(2.13) \qquad \|v'\|_{L^2(E_j)}^2 = \int_{E_j} (v')^2 \, d\Gamma \overset{\text{Def}}{=} \frac{h}{2} \int_{-1}^1 \big((v' \circ \gamma_j)(s)\big)^2 \, ds \overset{\text{Def}}{=} \frac{2}{h} \int_{-1}^1 \big((v \circ \gamma_j)'(s)\big)^2 \, ds.$$

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

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

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

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

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

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

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,

(2.14)
$$\mathrm{osc}_{D,\ell}(E_j)^2 \approx \widetilde{\mathrm{osc}}_{D,\ell}(E_j)^2 := 2 \, \mathrm{quad}_2 \big((p_j')^2 \big) = 2 \sum_{k=1}^3 \omega_k \big(p_j'(s_k) \big)^2$$
$$= \frac{4}{3} \left(g(a_j) + g(b_j) - 2\,g(m_j) \right)^2.$$

Altogether, the documentation of Listing 3 now reads as follows:

- The function takes the mesh $\mathcal{T}_\ell$ 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 (2.14) is realized (Line 12), simultaneously for all elements $E_j \in \mathcal{T}_\ell$.

- The function returns the column vector of elementwise Dirichlet data oscillations

$$\mathbf{v} := \big(\widetilde{\mathrm{osc}}_{D,\ell}(E_1)^2, \ldots, \widetilde{\mathrm{osc}}_{D,\ell}(E_N)^2\big) \in \mathbb{R}^N$$

so that $\mathrm{osc}_{D,\ell} \approx \big(\sum_{j=1}^N \mathbf{v}_j\big)^{1/2}$.

Remark 2.1. *For smooth Dirichlet data g and uniform meshes with mesh-size h, there holds*

$$\mathrm{osc}_{D,\ell} = \mathcal{O}(h^{3/2}) \quad and \quad \left| \mathrm{osc}_{D,\ell} - \Big(\sum_{j=1}^N \mathbf{v}_j\Big)^{1/2} \right| = \mathcal{O}(h^{5/2}).$$

Therefore, the quadrature error is of higher order when compared to the discretization order.

LISTING 4

```
1  function err = computeErrNeumann(coordinates,elements,p,phi)
2  %*** arbitrary quadrature on [-1,1] with exactness n >= 2, e.g., gauss(2)
3  quad_nodes = [-1 1]/sqrt(3);
4  quad_weights = [1;1];
5
6  %*** the remaining code is independent of the chosen quadrature rule
7  nE = size(elements,1);
8  nQ = length(quad_nodes);
9
10 %*** build vector of evaluations points as (nQ*nE x 2)-matrix
11 a = coordinates(elements(:,1),:);
12 b = coordinates(elements(:,2),:);
13 sx = reshape(a,2*nE,1)*(1-quad_nodes) + reshape(b,2*nE,1)*(1+quad_nodes);
14 sx = 0.5*reshape(sx',nQ*nE,2);
15
16 %*** phi(sx) usually depends on the normal vector, whence phi takes sx and the
17 %*** nodes of the respective element to compute the normal
18 a_sx = reshape(repmat(reshape(a,2*nE,1),1,nQ)',nE*nQ,2);
19 b_sx = reshape(repmat(reshape(b,2*nE,1),1,nQ)',nE*nQ,2);
20
21 %*** perform all necessary evaluations of phi as (nE x nQ)-matrix
22 phi_sx = reshape(phi(sx,a_sx,b_sx),nQ,nE)';
23
24 %*** compute vector of (squared) element-widths
25 h = sum((a-b).^2,2);
26
27 %*** compute Neumann error simultaneously for all elements
28 err_sx = (phi_sx - repmat(reshape(p,nE,1),1,nQ)).^2;
29 err = 0.5*h.*(err_sx*quad_weights);
```

2.4. Approximate Computation of Error $\|\phi - \Phi_\ell\|_V$ (Listing 4). We assume that the exact Neumann data satisfy $\phi \in L^2(\Gamma)$. Let $\Phi_\ell^* \in \mathcal{P}^0(\mathcal{T}_\ell)$ 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 $\mathcal{P}^0(\mathcal{T}_\ell)$. With the technique from [FP, AGP], we obtain

$$\|\phi - \Phi_\ell^*\|_V \leq \|\phi - \Pi_\ell \phi\|_V \lesssim \|h_\ell^{1/2}(\phi - \Pi_\ell \phi)\|_{L^2(\Gamma)}$$

as well as

$$\|\Phi_\ell^* - \Phi_\ell\|_V \lesssim \mathrm{osc}_{D,\ell},$$

13

where $\mathrm{osc}_{D,\ell}$ denote the Dirichlet data oscillations from Section 2.3. Note that Π_ℓ is even the $\mathcal{T}_\ell$-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

$$\mathrm{err}_{N,\ell}(E_j)^2 = \mathrm{length}(E_j) \int_{E_j} |\phi - \mathbf{x}_j|^2 \, d\Gamma = \frac{\mathrm{length}(E_j)^2}{2} \int_{-1}^1 |\phi \circ \gamma_j(s) - \mathbf{x}_j|^2 \, ds$$

(2.15)
$$\approx \frac{\mathrm{length}(E_j)^2}{2} \mathrm{quad}_n\big((\phi \circ \gamma_j - \mathbf{x}_j)^2\big) =: \widetilde{\mathrm{err}}_{N,\ell}(E_j)^2,$$

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 4 now simply reads as follows:

- The function takes the given mesh $\mathcal{T}_\ell$ in form of the arrays `coordinates` and `elements`, the coefficient vector $\mathbf{p} = \mathbf{x}$ 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 $E_j = [a_j, b_j] \in \mathcal{T}_\ell = \{E_1, \ldots, E_N\}$ 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 (2.15). 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 $\texttt{err}$ such that $\texttt{err}(j) \approx \text{length}(E_j)\|\phi - \Phi_\ell\|^2_{L^2(E_j)}$. More precisely, there holds $\text{err}^2_{N,\ell} \approx \sum_{j=1}^N \texttt{err}(j) = \widetilde{\text{err}}^2_{N,\ell}$.

Remark 2.2. *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. $\qquad\square$

2.5. 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 $\widehat{\mathcal{T}}_\ell = \{e_1, \dots, e_{2N}\}$ be the uniform refinement of the triangulation $\mathcal{T}_\ell = \{E_1, \dots, E_N\}$. Let $\Phi_\ell \in \mathcal{P}^0(\mathcal{T}_\ell)$ and $\widehat{\Phi}_\ell \in \mathcal{P}^0(\widehat{\mathcal{T}}_\ell)$ be the Galerkin solutions (2.6) with respect to $\mathcal{T}_\ell$ and $\widehat{\mathcal{T}}_\ell$ and the same approximate Dirichlet data G_ℓ, i.e., there holds

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

as well as

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

With $\phi_\ell \in H^{-1/2}(\Gamma)$ the exact solution of (2.11), one can expect

$$(2.16) \qquad \|\phi_\ell - \Phi_\ell\|_V \approx \|\widehat{\Phi}_\ell - \Phi_\ell\|_V =: \eta_\ell,$$

which results in

$$(2.17) \qquad \|\phi - \Phi_\ell\|_V \leq \|\phi - \phi_\ell\|_V + \|\phi_\ell - \Phi_\ell\|_V \lesssim \text{osc}_{D,\ell} + \eta_\ell$$

according to the triangle inequality and (2.12).

Clearly, the Galerkin solution $\widehat{\Phi}_\ell$ with respect to the uniformly refined mesh $\widehat{\mathcal{T}}_\ell$ 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

$$(2.18) \qquad \eta_\ell \sim \|\widehat{\Phi}_\ell - \Pi_\ell\widehat{\Phi}_\ell\|_V =: \widetilde{\eta}_\ell$$

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 $\mathcal{P}^0(\mathcal{T}_\ell)$, which simply reads

$$(2.19) \qquad (\Pi_\ell\widehat{\Phi}_\ell)|_{E_i} = \frac{1}{\text{length}(E_i)} \int_{E_i} \widehat{\Phi}_\ell \, d\Gamma \quad \text{for all } E_i \in \mathcal{T}_\ell$$

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 $E_i \in \mathcal{T}_\ell$. This is different for the error estimators μ_ℓ and $\widetilde{\mu}_\ell$ discussed in the following. For instance, one can prove that

$$(2.20) \qquad \eta_\ell \sim \mu_\ell := \|h_\ell^{1/2}(\widehat{\Phi}_\ell - \Phi_\ell)\|_{L^2(\Gamma)} = \Big(\sum_{i=1}^N \text{length}(E_i)\|\widehat{\Phi}_\ell - \Phi_\ell\|^2_{L^2(E_i)} \Big)^{1/2}.$$

Then, the local contributions

$$(2.21) \qquad \mu_\ell(E_i) := \text{length}(E_i)^{1/2}\|\widehat{\Phi}_\ell - \Phi_\ell\|_{L^2(E_i)} \quad \text{for all } E_i \in \mathcal{T}_\ell$$

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

$$(2.22) \qquad \eta_\ell \sim \widetilde{\mu}_\ell := \| h_\ell^{1/2} (\widehat{\Phi}_\ell - \Pi_\ell \widehat{\Phi}_\ell) \|_{L^2(\Gamma)}.$$

In the following subsections, we first discuss the computation of the *global* error estimators η_ℓ and $\widetilde{\eta}_\ell$ from (2.16) and (2.18). Then, we give an implementation of the *local* error estimators μ_ℓ and $\widetilde{\mu}_\ell$ from (2.20) and (2.22), where our functions return the local contributions, see e.g. (2.21), to steer an adaptive mesh-refinement.

Remark 2.3. *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*

$$(2.23) \qquad \eta_\ell \leq \widetilde{\eta}_\ell \lesssim \widetilde{\mu}_\ell \leq \mu_\ell \lesssim \eta_\ell,$$

cf. [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 2.4. *Mathematically, the error estimate (2.17) involves the so-called saturation assumption: Assume that we could compute the Galerkin solutions Φ_ℓ^* and $\widehat{\Phi}_\ell^*$ with respect to $\mathcal{T}_\ell$ and $\widehat{\mathcal{T}}_\ell$ for the non-perturbed variational formulation (2.3), 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*

$$(2.24) \qquad \| \phi - \widehat{\Phi}_\ell^* \|_V \leq q \, \| \phi - \Phi_\ell^* \|_V$$

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 (2.24), one can prove that

$$(2.25) \qquad \eta_\ell \leq \| \phi_\ell - \Phi_\ell \|_V \leq (1 - q^2)^{-1/2} \, \eta_\ell$$

which is the mathematical basis of (2.16), cf. [FP].

We stress that this assumption is somewhat natural and can, for instance, be proven for the finite element method. For the boundary element method, however, (2.24) still remains open.

Finally, one can prove that (2.24) is sufficient and in some sense even necessary to guarantee (2.17). ☐

Remark 2.5. *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 2.4, 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 5

```
1  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);
```

2.5.1. Computation of Error Estimator η_ℓ (Listing 5). In this section, we aim to compute the error estimator $\eta_\ell = \|\widehat{\Phi}_\ell - \Phi_\ell\|_V$ from (2.16). Let $\widehat{\chi}_j$ denote the characteristic function associated with some fine-mesh element $e_j \in \widehat{\mathcal{T}}_\ell$. 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 $\mathcal{P}^0(\mathcal{T}_\ell)$ and $\mathcal{P}^0(\widehat{\mathcal{T}}_\ell)$, 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 $\mathcal{P}^0(\mathcal{T}_\ell) \subset \mathcal{P}^0(\widehat{\mathcal{T}}_\ell)$, 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

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

where $\widehat{\mathbf{V}}$ is the matrix for the simple-layer potential (2.8) with respect to the fine mesh $\widehat{\mathcal{T}}_\ell$. With these observations, the documentation of Listing 5 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 $\widehat{\mathcal{T}}_\ell$. Besides this, the $(N \times 2)$-array `father2son` links the indices of elements $E_i \in \mathcal{T}_\ell$ with the indices of the sons $e_j, e_k \in \widehat{\mathcal{T}}_\ell$ 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).

<div align="center">LISTING 6</div>

```
1  function est = computeEstSlpEtaTilde(father2son,V_fine,x_fine)
2  %*** compute L2-projection Pi_coarse*phi_fine onto coarse mesh
3  pi_x_fine = 0.5*( x_fine(father2son(:,1)) + x_fine(father2son(:,2)) );
4
5  %*** compute coefficient vector of (1-Pi_coarse)*phi_fine
6  x_fine(father2son(:,1)) = x_fine(father2son(:,1)) - pi_x_fine;
7  x_fine(father2son(:,2)) = x_fine(father2son(:,2)) - pi_x_fine;
8
9  %*** compute energy ||| (1-Pi_coarse)*phi_fine |||^2
10 est = x_fine'*(V_fine*x_fine);
```

2.5.2. Computation of Error Estimator $\widetilde{\boldsymbol{\eta}}_\ell$ (Listing 6). We adopt the notation of Section 2.5.1 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 $e_j, e_k \in \widehat{\mathcal{T}}_\ell$ be the sons of $E_i \in \mathcal{T}_\ell$, 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

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

for the L^2-projection Π_ℓ defined in (2.19). Representing $\Pi_\ell \widehat{\Phi}_\ell \in \mathcal{P}^0(\mathcal{T}_\ell)$ with respect to the fine-mesh $\widehat{\mathcal{T}}_\ell$, 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 $e_j, e_k \in \widehat{\mathcal{T}}_\ell$ are the sons of some element $E_i \in \mathcal{T}_\ell$. As in Section 2.5.1, 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 6 reads as follows:

- The function takes the simple-layer potential matrix $\widehat{\mathbf{V}}$ for the fine mesh $\widehat{\mathcal{T}}_\ell$ and the coefficient vector $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ of $\widehat{\Phi}_\ell$. Moreover, the link between $\mathcal{T}_\ell$ and $\widehat{\mathcal{T}}_\ell$ is provided by means of `father2son`.
- We first compute the coefficient vector of $\Pi_\ell \widehat{\Phi}_\ell$ with respect to the coarse mesh $\mathcal{T}_\ell$ (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).

<center>LISTING 7</center>

```
1  function ind = computeEstSlpMu(coordinates,elements,father2son,x_fine,x_coarse)
2  %*** compute (squared) local mesh-size
3  h = sum((coordinates(elements(:,1),:) - coordinates(elements(:,2),:)).^2,2);
4
5  %*** compute coefficient vector of (phi_fine - phi_coarse) w.r.t. to fine mesh
6  x_fine(father2son(:,1)) = x_fine(father2son(:,1)) - x_coarse;
7  x_fine(father2son(:,2)) = x_fine(father2son(:,2)) - x_coarse;
8
9  %*** compute ind(j) = diam(Ej)*|| phi_fine - phi_coarse ||_{L2(Ej)}^2
10 ind = 0.5*h.*( x_fine(father2son(:,1)).^2 + x_fine(father2son(:,2)).^2 );
```

2.5.3. Computation of Error Estimator $\boldsymbol{\mu}_\ell$ (Listing 7). In this section, we discuss the implementation of

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

<center>18</center>

We adopt the notation of Section 2.5.1, 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 $E_i \in \mathcal{T}_\ell$ and sons $e_j, e_k \in \widehat{\mathcal{T}}_\ell$ with $E_i = e_j \cup e_k$, we obtain

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

This implies

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

Altogether, the documentation of Listing 7 reads as follows:

- As input arguments, the function takes the triangulation $\mathcal{T}_\ell$, the link between $\mathcal{T}_\ell$ and $\widehat{\mathcal{T}}_\ell$, 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 (2.26) and returns the vector

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

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

<div align="center">LISTING 8</div>

```
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 );
```

2.5.4. Computation of Error Estimator $\widetilde{\mu}_\ell$ (Listing 8).

In this section, we finally aim to compute

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

We adopt the notation of the preceding Sections 2.5.1–2.5.3, 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 8 reads as follows:

- The function takes the triangulation $\mathcal{T}_\ell$, the link between $\mathcal{T}_\ell$ and $\widehat{\mathcal{T}}_\ell$, 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
$$\mathbf{v} := (\widetilde{\mu}_\ell(E_1)^2, \ldots, \widetilde{\mu}_\ell(E_N)^2) \in \mathbb{R}^N.$$

In particular, there holds $\widetilde{\mu}_\ell = \left(\sum_{i=1}^N \mathbf{v}_i\right)^{1/2}$.

2.6. Adaptive Mesh-Refinement. Usually computing time and memory requirements are limiting quantities for numerical simulations. Therefore, one aims to choose the triangulation 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, where the error appears to be large, i.e., where the local contributions $\widetilde{\mu}_\ell(E_j)$ are large. For the error estimator $\widetilde{\mu}_\ell$ from Section 2.5.4, a possible adaptive algorithm reads as follows:

Input: Initial mesh $\mathcal{T}_0$, 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 $\widehat{\mathcal{T}}_\ell$.
    (ii) Compute Galerkin solution $\widehat{\Phi}_\ell \in \mathcal{P}^0(\widehat{\mathcal{T}}_\ell)$.
    (iii) Compute refinement indicators $\widetilde{\mu}_\ell(E)^2$ and oscillation terms $\mathrm{osc}_{D,\ell}(E)^2$ for all $E \in \mathcal{T}_\ell$.
    (iv) Find minimal set $\mathcal{M}_\ell \subseteq \mathcal{T}_\ell$ such that

$$(2.27) \qquad \theta\left(\widetilde{\mu}_\ell^2 + \mathrm{osc}_{D,\ell}^2\right) = \theta \sum_{E \in \mathcal{T}_\ell} \widetilde{\mu}_\ell(E)^2 + \mathrm{osc}_{D,\ell}(E)^2 \leq \sum_{E \in \mathcal{M}_\ell} \widetilde{\mu}_\ell(E)^2 + \mathrm{osc}_{D,\ell}(E)^2.$$

    (v) Refine at least marked elements $E \in \mathcal{M}_\ell$ and obtain mesh $\mathcal{T}_{\ell+1}$ with $\kappa(\mathcal{T}_{\ell+1}) \leq 2\kappa(\mathcal{T}_0)$.
    (vi) Stop provided that $\#\mathcal{T}_{\ell+1} \geq N_{\max}$; otherwise, increase counter $\ell \mapsto \ell + 1$ and go to (i).
Output: Adaptively generated mesh $\widehat{\mathcal{T}}_\ell$ and corresponding discrete solution $\widehat{\Phi}_\ell \in \mathcal{P}^0(\widehat{\mathcal{T}}_\ell)$.

The marking criterion (2.27) 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], we prove that the saturation assumption (2.24) 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 (2.24) holds (at least every finitely many steps), we then 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.

<div align="center">LISTING 9</div>

```
1  function varargout = markElements(theta,varargin)
2  %*** check whether optional parameter rho is given or not
3  if nargin == nargout +1
4      rho = 0;
5  else
6      rho = varargin{1};
7      varargin = varargin(2:end);
8  end
9
10 %*** enforce input parameters to be column vectors and count their length
11 nE = zeros(1,nargout+1);
```

```
12  for j = 1:nargout
13      nE(j+1) = length(varargin{j});
14      varargin{j} = reshape(varargin{j},nE(j+1),1);
15  end
16
17  %*** generate set of all indicators
18  indicators = cat(1,varargin{:});
19  nE = cumsum(nE);
20
21  %*** realization of Doerfler marking
22  [indicators,idx] = sort(indicators,'descend');
23  sum_indicators = cumsum(indicators);
24  ell = max(ceil(rho*nE(end)),find(sum_indicators>=sum_indicators(end)*theta,1));
25  marked = idx(1:ell);
26
27  %*** split subset marked into subsets with respect to input vectors
28  for j = 1:nargout
29      varargout{j} = marked( marked>nE(j) & marked<=nE(j+1) ) - nE(j);
30  end
```

2.6.1. Dörfler Marking (2.27) for Local Mesh-Refinement (Listing 9). We realize the marking criterion (2.27) in a generalized form which is suitable even for mixed boundary value problems or the FEM-BEM coupling. Suppose that $\mathcal{T}_\ell^{(1)}, \ldots, \mathcal{T}_\ell^{(n)}$ are pairwise disjoint meshes which provide indicators $\varrho_\ell^{(k)}(E)$ for all $E \in \mathcal{T}_\ell^{(k)}$. We formally define $\mathcal{T}_\ell := \bigcup_{j=1}^n \mathcal{T}_\ell^{(j)}$ and $\varrho_\ell(E) := \varrho_\ell^{(k)}(E)$ for all $E \in \mathcal{T}_\ell^{(k)}$ and $k = 1, \ldots, n$. For given $0 < \theta < 1$, we then aim to find the minimal set $\mathcal{M}_\ell \subseteq \mathcal{T}_\ell$ such that

$$(2.28) \qquad \theta \sum_{E \in \mathcal{T}_\ell} \varrho_\ell(E) \leq \sum_{E \in \mathcal{M}_\ell} \varrho_\ell(E).$$

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

A second generalization is concerned with the minimal cardinality of $\mathcal{M}_\ell$. For analytical convergence results, the minimal set $\mathcal{M}_\ell$ with (2.28) 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 $\mathcal{M}_\ell \subseteq \overline{\mathcal{M}}_\ell \subseteq \mathcal{T}_\ell$ with

$$(2.29) \qquad \frac{\#\overline{\mathcal{M}}_\ell}{\#\mathcal{T}_\ell} \geq \rho \quad \text{and} \quad \varrho_\ell(E) \geq \varrho_\ell(E') \quad \text{for all } E \in \overline{\mathcal{M}}_\ell \text{ and } E' \in \mathcal{T}_\ell \backslash \overline{\mathcal{M}}_\ell.$$

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 (2.27) includes (optionally) these generalizations (2.28)–(2.29):

- For Symm's integral equation and the stated adaptive algorithm, the function `markElements` is called by

      marked = markElements(theta,mutilde+osc)

  where `mutilde` and `osc` are the output of `computeEstSlpMuTilde` from Section 2.5.4 and `computeOscDirichlet` from Section 2.3. Note that these functions return vectors of quadratic terms $\widetilde{\mu}_\ell(E)^2$ and $\mathrm{osc}_\ell(E)^2$, respectively. The function `markElements` then returns the indices corresponding to the minimal set $\mathcal{M}_\ell$ for $\varrho_\ell(E) = \widetilde{\mu}_\ell(E)^2 + \mathrm{osc}_\ell(E)^2$.
- Alternatively, the function can be called by

      marked = markElements(theta,rho,mutilde+osc)

and returns the indices corresponding to the minimal set $\overline{\mathcal{M}}_\ell \supseteq \mathcal{M}_\ell$ with (2.29).

- 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 $E_j^{(1)} \in \mathcal{T}_\ell^{(1)} = \{E_1^{(1)},\dots,E_{N^{(1)}}^{(1)}\}$.
  The function returns the indices corresponding to the sets $\mathcal{M}_\ell^{(k)} \subseteq \mathcal{T}_\ell^{(k)}$ (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 $\mathcal{T}_\ell = \{E_1,\dots,E_N\}$. Moreover, `nE` becomes a vector of pointers so that `nE(j)+1` and `nE(j + 1)` give the start and the end of $\mathcal{T}_\ell^{(j)}$ 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 $\mathcal{T}_\ell^{(k)} = \{E_1^{(k)},\dots,E_{N^{(k)}}^{(k)}\}$ (Line 28–30).

LISTING 10

```
1   % adaptiveSymm provides the implementation of an adaptive mesh-refining
2   % algorithm for Symm's 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      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
```

2.6.2. 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 (2.27) and mark at least the 25% of elements
  with the largest indicators (Line 8–9).

3. Hypersingular Integral Equation

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

$$(3.1) \qquad Wu = (1/2 - K')\phi \quad \text{on } \Gamma$$

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

$$(3.2) \qquad -\Delta u = 0 \text{ in } \Omega \quad \text{with} \quad \partial_n u = \phi \text{ on } \Gamma.$$

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 $\phi \in C(\mathcal{T}_\ell) \subset L^2(\Gamma) \subset H^{-1/2}(\Gamma)$. The exact solution $u \in H^{1/2}(\Gamma)$ of the integral formulation (3.1) is just the Dirichlet data $u|_\Gamma$ of the solution $u \in H^1(\Omega)$ of (3.2).

Due to the fact, that there holds $Wc = 0$ for all constant functions $c \in \mathbb{R}$, the solutions of (3.1) and (3.2) 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, (3.1) can equivalently be formulated in variational form: Find $u_* \in H_*^{1/2}(\Gamma) := \{v \in H^{1/2}(\Gamma) : \int_\Gamma v \, d\Gamma = 0\}$ such that

$$(3.3) \qquad \langle Wu_*, v_* \rangle_\Gamma = \langle (1/2 - K')\phi, v_* \rangle_\Gamma \quad \text{for all } v_* \in H_*^{1/2}(\Gamma).$$

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

$$(3.4) \qquad \langle\!\langle u, v \rangle\!\rangle_{W+S} := \langle Wu, v \rangle_\Gamma + \left(\int_\Gamma u \, d\Gamma \right)\left(\int_\Gamma v \, d\Gamma \right) \quad \text{for all } u, v \in H^{1/2}(\Gamma),$$

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

$$(3.5) \qquad \langle\!\langle u, v \rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi, v \rangle_\Gamma \quad \text{for all } v \in H^{1/2}(\Gamma).$$

One can prove that $\langle\!\langle \cdot, \cdot \rangle\!\rangle_{W+S}$ from (3.4) 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, (3.5) 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 (3.5), we thus obtain

$$\left(\int_\Gamma u \, d\Gamma \right) \text{length}(\Gamma) = \langle Wu, 1 \rangle_\Gamma + \left(\int_\Gamma u \, d\Gamma \right)\left(\int_\Gamma 1 \, d\Gamma \right)$$
$$= \langle\!\langle u, 1 \rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi, 1 \rangle_\Gamma = 0$$

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 (3.5) thus becomes

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

i.e. (3.5) reduces to (3.3). Altogether we obtain that the unique solution u of (3.5) is also the unique solution of (3.3), i.e., (3.5) is an equivalent formulation of (3.3).

Galerkin Discretization. To discretize (3.5), we first replace the Neumann data $\phi \in H_*^{-1/2}(\Gamma) \cap L^2(\Gamma)$ by its L^2-projection $\Phi_\ell \in \mathcal{P}^0(\mathcal{T}_\ell)$,

$$(3.6) \qquad \Phi_\ell|_{E_j} = \frac{1}{\text{length}(E_j)} \int_{E_j} \phi \, d\Gamma =: \mathbf{p}_j \quad \text{for all } E_j \in \mathcal{T}_\ell.$$

According to this definition, there holds

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

i.e. there holds $\Phi_\ell \in H_*^{-1/2}(\Gamma)$, too. Second, we replace the function space $H^{1/2}(\Gamma)$ in (3.5) by the finite-dimensional space $\mathcal{S}^1(\mathcal{T}_\ell)$. Since $\mathcal{S}^1(\mathcal{T}_\ell)$ is a subspace of $H^{1/2}(\Gamma)$, $\langle\!\langle \cdot, \cdot \rangle\!\rangle_{W+S}$ from (3.4) is also a scalar product on $\mathcal{S}^1(\mathcal{T}_\ell)$. Consequently, there exists a unique Galerkin solution $U_\ell \in \mathcal{S}^1(\mathcal{T}_\ell)$ of the discretized problem

$$(3.7) \qquad \langle\!\langle U_\ell, V_\ell \rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell, V_\ell \rangle_\Gamma \quad \text{for all } V_\ell \in \mathcal{S}^1(\mathcal{T}_\ell).$$

As in the continuous case, the discrete solution U_ℓ automatically satisfies $\int_\Gamma U_\ell \, d\Gamma = 0$ which follows from $1 = \sum_{j=1}^N \zeta_j \in \mathcal{S}^1(\mathcal{T}_\ell)$, which allows us to plug $V_\ell = 1$ in (3.7). Indeed

$$\left(\int_\Gamma U_\ell \, d\Gamma \right) \text{length}(\Gamma) = \langle WU_\ell, 1 \rangle_\Gamma + \left(\int_\Gamma U_\ell \, d\Gamma \right)\left(\int_\Gamma 1 \, d\Gamma \right)$$

$$= \langle\!\langle U_\ell, 1 \rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell, 1 \rangle_\Gamma = 0.$$

According to Linear Algebra, (3.7) holds for all $V_\ell \in \mathcal{S}^1(\mathcal{T}_\ell)$ if and only if it holds for all basis functions $\zeta_k \in \{\zeta_1, \ldots, \zeta_N\}$ of $\mathcal{S}^1(\mathcal{T}_\ell)$. With $\mathbf{p} \in \mathbb{R}^N$ from (3.6) and the coefficient vector $\mathbf{x} \in \mathbb{R}^N$ of the ansatz

$$(3.8) \qquad U_\ell = \sum_{j=1}^N \mathbf{x}_j \zeta_j,$$

the Galerkin formulation (3.7) is thus equivalent to

$$(3.9) \quad \sum_{j=1}^N \mathbf{x}_j \langle\!\langle \zeta_j, \zeta_k \rangle\!\rangle_{W+S} = \langle\!\langle U_\ell, \zeta_k \rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell, \zeta_k \rangle_\Gamma = \sum_{j=1}^N \mathbf{p}_j \langle (1/2 - K')\chi_j, \zeta_k \rangle_\Gamma$$

for all $k = 1, \ldots, N$. In the context of Symm's integral equation of Section 2, 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. (2.8). 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 (3.9), we define matrices $\mathbf{W}, \mathbf{S} \in \mathbb{R}^{N \times N}$ by

$$(3.10) \qquad \mathbf{W}_{kj} = \langle W\zeta_j, \zeta_k \rangle_\Gamma, \quad \mathbf{S}_{kj} = \left(\int_\Gamma \zeta_j \, d\Gamma \right)\left(\int_\Gamma \zeta_k \, d\Gamma \right) \quad \text{for all } j, k = 1, \ldots, N.$$

Then there holds

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

Altogether, the Galerkin system (3.7) is equivalently stated by

$$(3.11) \qquad (\mathbf{W} + \mathbf{S})\mathbf{x} = \frac{1}{2}\mathbf{M}^T \mathbf{p} - \mathbf{K}^T \mathbf{p}.$$

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

(3.12) $\qquad \mathbf{S} = \mathbf{c}\mathbf{c}^T \quad$ with the column vector $\quad \mathbf{c} \in \mathbb{R}^N$ with $\mathbf{c}_j := \int_\Gamma \zeta_j \, d\Gamma.$

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

(3.13) $\qquad \langle Wu \,,\, v \rangle_\Gamma = \langle Vu' \,,\, v' \rangle_\Gamma \quad$ for all $u, v \in H^1(\Gamma).$

Since $\zeta_j' \in \mathcal{P}^0(\mathcal{T}_\ell)$, this gives a direct link between the matrices $\mathbf{W}$ and $\mathbf{V}$. Finally, we stress that the matrix $\mathbf{W} + \mathbf{S}$ from (3.11) is symmetric and positive definite since it stems from a scalar product. Consequently, the linear system (3.11) has a unique solution $\mathbf{x} \in \mathbb{R}^N$.

LISTING 11

```
1  function S = buildHypsingStabilization(coordinates,elements)
2  nE = size(elements,1);
3
4  %*** compute local mesh—size
5  h = sqrt(sum((coordinates(elements(:,1),:)−coordinates(elements(:,2),:)).^2,2));
6
7  %*** build vector with entries c(j) = int_Gamma hatfunction(j) ds
8  c = 0.5*accumarray(reshape(elements,2*nE,1),[h;h]);
9
10 %*** build stabilization matrix
11 S = c*c';
```

3.1. Compute Stabilization for Hypersingular Integral Equation (Listing 11). The kernel of the hypersingular integral operator W is the space of constant functions. Since $1 = \sum_{j=1}^N \zeta_j \in \mathcal{S}^1(\mathcal{T}_\ell)$, the corresponding matrix $\mathbf{W}$ defined by $\mathbf{W}_{kj} = \langle W\zeta_j \,,\, \zeta_k \rangle_\Gamma$ for all $j, k \in \{1, \ldots, N\}$ 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 (3.4). It thus remains to assemble the rank-1-matrix $\mathbf{S} = \mathbf{c}\mathbf{c}^T \in \mathbb{R}^{N \times N}$ from (3.12). 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 $E_i \in \mathcal{T}_\ell$ which include $z_k \in \mathcal{K}_\ell$ as a node. The vector $\mathbf{c}$ can be assembled $\mathcal{T}_\ell$-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 $\mathrm{h}(j)$ contains the element-width $\text{length}(E_j)$.

```
1  nE = size(elements,1);
2  h = sqrt(sum((coordinates(elements(:,1),:)−coordinates(elements(:,2),:)).^2,2));
3  c = zeros(nE,1);
4  for j = 1:nE
5      nodes = elements(j,:);
6      c(nodes) = c(nodes) + 0.5*h(j);
7  end
```

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

- The function takes the mesh $\mathcal{T}_\ell$ described by the arrays `coordinates` and `elements`.

26

- 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 **S** (Line 11).

<div align="center">LISTING 12</div>

```
1  function b = buildHypsingRHS(coordinates,elements,phi)
2  %*** arbitrary quadrature on [−1,1] with exactness n >= 2, e.g., gauss(2)
3  quad_nodes = [−1 1]/sqrt(3);
4  quad_weights = [1;1];
5
6  %*** the remaining code is independent of the chosen quadrature rule
7  nE = size(elements,1);
8  nQ = length(quad_nodes);
9
10 %*** build vector of evaluations points as (nQ*nE x 2)−matrix
11 a = coordinates(elements(:,1),:);
12 b = coordinates(elements(:,2),:);
13 sx = reshape(a,2*nE,1)*(1−quad_nodes) + reshape(b,2*nE,1)*(1+quad_nodes);
14 sx = 0.5*reshape(sx',nQ*nE,2);
15
16 %*** phi(sx) usually depends on the normal vector, whence phi takes sx and the
17 %*** nodes of the respective element to compute the normal
18 a_sx = reshape(repmat(reshape(a,2*nE,1),1,nQ)',nE*nQ,2);
19 b_sx = reshape(repmat(reshape(b,2*nE,1),1,nQ)',nE*nQ,2);
20
21 %*** perform all necessary evaluations of phi as (nE x nQ)−matrix
22 phi_sx = reshape(phi(sx,a_sx,b_sx),nQ,nE)';
23
24 %*** compute elementwise integral mean of phi
25 phi_mean = (phi_sx*quad_weights*0.5)';
26
27 %*** compute DLP−matrix for P0 x S1
28 K = buildK(coordinates,elements);
29
30 %*** compute mass−type matrix for P0 x S1
31 nE = size(elements,1);
32 h = sqrt(sum((coordinates(elements(:,1),:)−coordinates(elements(:,2),:)).^2,2));
33 I = reshape(repmat(1:nE,2,1),2*nE,1);
34 J = reshape(elements',2*nE,1);
35 A = reshape(repmat(0.5*h,1,2)',2*nE,1);
36 M = sparse(I,J,A);
37
38 %*** build right−hand side vector
39 b =  (phi_mean*M*0.5 − phi_mean*K)';
```

3.2. 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 transpose matrices of **K** and **M**, the right-hand side vector for (3.11) reads

(3.14)
$$\mathbf{b} := \frac{1}{2}\mathbf{M}^T\mathbf{p} - \mathbf{K}^T\mathbf{p} = \left(\frac{1}{2}\mathbf{p}^T\mathbf{M} - \mathbf{p}^T\mathbf{K}\right)^T,$$

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:

$$(3.15)\quad \mathbf{p}_j := \frac{1}{\text{length}(E_j)} \int_{E_j} \phi \, d\Gamma = \frac{1}{2} \int_{-1}^{1} \phi \circ \gamma_j \, ds \approx \frac{1}{2} \text{quad}_n(\phi \circ \gamma_j) =: \widetilde{\mathbf{p}}_j \quad \text{for all } E_j \in \mathcal{T}_\ell.$$

Here, we use the parametrization $\gamma_j : [-1, 1] \to E_j$ from (1.9). Moreover, $\text{quad}_n(\cdot)$ denotes a quadrature rule which is exact of order $n \in \mathbb{N}$, i.e., $\text{quad}_n(p) = \int_{-1}^{1} p \, ds$ for all $p \in \mathcal{P}^n[-1, 1]$. In our concrete 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 3.3.

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

- The function takes as input the given mesh $\mathcal{T}_\ell$ 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 $\mathbf{x}(j,:)$, $\mathbf{a}(j,:)$, and $\mathbf{b}(j,:)$ correspond to a point $x_j \in [a_j, b_j] \subset \mathbb{R}^2$. The entry $\mathbf{y}(j)$ 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 $E_j = [a_j, b_j] \in \mathcal{T}_\ell = \{E_1, \ldots, E_N\}$ 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 an $(N \times 2)$-array `phi_sx` such that $\mathtt{phi\_sx}(j,:)$ 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 (3.15), namely $\mathtt{phi\_mean}(j) = \text{quad}_n(\phi \circ \gamma_j)/2$ (Line 25).
- Next we build $\mathbf{K}$ (Line 28) and $\mathbf{M}$ (Line 31–36), cf. Section 2.2 above.
- Finally (Line 39), the function computes and returns the vector $\mathbf{b}$ as described in (3.14).

LISTING 13

```
1  function osc = computeOscNeumann(coordinates,elements,phi)
2  %*** arbitrary quadrature on [−1,1] with exactness n >= 2, e.g., gauss(2)
3  quad_nodes = [−1 1]/sqrt(3);
4  quad_weights = [1;1];
5
6  %*** the remaining code is independent of the chosen quadrature rule
7  nE = size(elements,1);
8  nQ = length(quad_nodes);
9
10 %*** build vector of evaluations points as (nQ*nE x 2)−matrix
11 a = coordinates(elements(:,1),:);
12 b = coordinates(elements(:,2),:);
13 sx = reshape(a,2*nE,1)*(1−quad_nodes) + reshape(b,2*nE,1)*(1+quad_nodes);
14 sx = 0.5*reshape(sx',nQ*nE,2);
15
16 %*** phi(sx) usually depends on the normal vector, whence phi takes sx and the
17 %*** nodes of the respective element to compute the normal
18 a_sx = reshape(repmat(reshape(a,2*nE,1),1,nQ)',nE*nQ,2);
```

```
19  b_sx = reshape(repmat(reshape(b,2*nE,1),1,nQ)',nE*nQ,2);
20
21  %*** perform all necessary evaluations of phi as (nE x nQ)-matrix
22  phi_sx = reshape(phi(sx,a_sx,b_sx),nQ,nE)';
23
24  %*** compute elementwise integral mean of phi
25  phi_mean = phi_sx*quad_weights*0.5;
26
27  %*** compute vector of (squared) element-widths
28  h = sum((a-b).^2,2);
29
30  %*** compute oscillation terms
31  osc_sx = (phi_sx - repmat(phi_mean,1,nQ)).^2;
32  osc = 0.5*h.*(osc_sx*quad_weights);
```

3.3. Computation of Data Oscillations for Neumann Data (Listing 13). Instead of solving the correct variational form (3.5), we solve

$$(3.16) \qquad \langle\!\langle u_\ell\,,v\rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell\,,v\rangle_\Gamma \quad \text{for all } v \in H^{1/2}(\Gamma)$$

with perturbed right-hand side, where we use the approximation $\Phi_\ell \approx \phi$. Analytically [AGP], the error between the exact solution $u \in H^{1/2}(\Gamma)$ of (3.5) and the exact solution $u_\ell \in H^{1/2}(\Gamma)$ of the perturbed formulation (3.16) is controlled by

$$(3.17) \qquad \|u - u_\ell\|_{W+S} \lesssim \|h_\ell^{1/2}(\phi - \Phi_\ell)\|_{L^2(\Gamma)} =: \mathrm{osc}_{N,\ell}.$$

We now aim for a numerical approximation of the local contributions

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

where —as for the computation of the right-hand side vector $\mathbf{b}$ in Section 3.2— $\mathbf{p}_j$ abbreviates the integral mean

$$(3.18) \qquad \mathbf{p}_j := \frac{1}{\mathrm{length}(E_j)}\int_{E_j}\phi\,d\Gamma = \frac{1}{2}\int_{-1}^{1}\phi\circ\gamma_j\,ds \approx \frac{1}{2}\,\mathrm{quad}_n(\phi\circ\gamma_j) =: \widetilde{\mathbf{p}}_j.$$

Here, we use the parametrization $\gamma_j : [-1,1] \to E_j$ from (1.9). 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 the same quadrature rule, the local Neumann oscillations are approximated which results in

$$\mathrm{osc}_{N,\ell}(E_j)^2 = \mathrm{length}(E_j)\int_{E_j}|\phi - \mathbf{p}_j|^2\,d\Gamma = \frac{\mathrm{length}(E_j)^2}{2}\int_{-1}^{1}|\phi\circ\gamma_j(s) - \mathbf{p}_j|^2\,ds$$

$$(3.19) \qquad\qquad\qquad \approx \frac{\mathrm{length}(E_j)^2}{2}\,\mathrm{quad}_n\big((\phi\circ\gamma_j - \widetilde{\mathbf{p}}_j)^2\big) =: \widetilde{\mathrm{osc}}_{N,\ell}(E_j)^2.$$

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 $\mathcal{T}_\ell$ 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 3.2.

- We realize Equation (3.19). 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 $\text{length}(E_j)^2/2$ provides the $(N \times 1)$-array `osc` such that $\texttt{osc}(j) \approx \text{length}(E_j)\|\phi - \Phi_\ell\|_{L^2(E_j)}^2$. More precisely, there holds $\text{osc}_{N,\ell}^2 \approx \widetilde{\text{osc}}_{N,\ell}^2 = \sum_{j=1}^{N} \texttt{osc}(j)$.

<div align="center">LISTING 14</div>

```
1  function err = computeErrDirichlet(coordinates,elements,g,uD)
2  %*** compute midpoints of all elements
3  midpoints = 0.5*( coordinates(elements(:,1),:) + coordinates(elements(:,2),:) );
4
5  %*** compute p = (uD − uDh) at element midpoints
6  p_midpoints = uD(midpoints) − 0.5*sum(g(elements),2);
7
8  %*** compute p = (uD − uDh) at all nodes
9  p_nodes = uD(coordinates) − g;
10
11 %*** evaluate derivative p' at all elements (left,midpoint,right)
12 p_prime = [p_nodes(elements) p_midpoints] * [−3 −1 1 ; −1 1 3 ; 4 0 −4]*0.5;
13
14 %*** compute Dirichlet error simultaneously for all elements
15 err = 2*p_prime.^2*[1;4;1]/3;
```

3.4. Approximate Computation of Error $\|u - U_\ell\|_{W+S}$ (Listing 14). We assume that the exact Dirichlet data satisfy additional regularity $u \in H^1(\Gamma)$. Let $U_\ell^* \in \mathcal{S}^1(\mathcal{T}_\ell)$ 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 $\mathcal{S}^1(\mathcal{T}_\ell)$. 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

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

where $\text{osc}_{N,\ell}$ denotes the Neumann data oscillations from Section 3.3. We therefore obtain

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

For the numerical realization of

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

we use the same ideas as for the Dirichlet data oscillations in Section 2.3, where

$$(3.20) \qquad \text{err}_{D,\ell}(E_j)^2 = 2 \int_{-1}^{1} \big((u - U_\ell) \circ \gamma_j\big)'(s)^2 \, ds \approx \text{quad}_2\big((p_j')^2\big) =: \widetilde{\text{err}}_{D,\ell}(E_j).$$

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 rowwise as

$$(3.21) \qquad \big(p'_j(-1), p'_j(0), p'_j(+1)\big) = \big(v(a_j), v(b_j), v(m_j)\big) \begin{pmatrix} -3/2 & -1/2 & +1/2 \\ -1/2 & +1/2 & +3/2 \\ +2 & 0 & -2 \end{pmatrix}.$$

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 $\mathcal{T}_\ell$ 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 (3.21), 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 (3.20), and the function returns the column vector `err`, where $\mathtt{err}(j) = \widetilde{\mathrm{err}}_{D,\ell}(E_j)^2$. In particular, there holds $\mathrm{err}_{D,\ell} \approx \widetilde{\mathrm{err}}_{D,\ell} := \big(\sum_{j=1}^N \mathtt{err}(j) \big)^{1/2}$.

Remark 3.1. *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. □

3.5. 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 $\widehat{\mathcal{T}}_\ell = \{e_1, \ldots, e_{2N}\}$ be the uniform refinement of the triangulation $\mathcal{T}_\ell$. Let $U_\ell \in \mathcal{S}^1(\mathcal{T}_\ell)$ and $\widehat{U}_\ell \in \mathcal{S}^1(\widehat{\mathcal{T}}_\ell)$ be the Galerkin solutions of (3.7) with respect to $\mathcal{T}_\ell$ and $\widehat{\mathcal{T}}_\ell$ and the same approximate Neumann data Φ_ℓ, i.e. there holds

$$\langle\!\langle U_\ell, V_\ell \rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell, V_\ell \rangle_\Gamma \quad \text{for all } V_\ell \in \mathcal{S}^1(\mathcal{T}_\ell)$$

and

$$\langle\!\langle \widehat{U}_\ell, \widehat{V}_\ell \rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell, \widehat{V}_\ell \rangle_\Gamma \quad \text{for all } \widehat{V}_\ell \in \mathcal{S}^1(\widehat{\mathcal{T}}_\ell).$$

As for Symm's integral equation, one can expect

$$(3.22) \qquad \|u_\ell - U_\ell\|_{W+S} \approx \|\widehat{U}_\ell - U_\ell\|_{W+S} = \|\widehat{U}_\ell - U_\ell\|_W =: \eta_\ell,$$

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

$$(3.23) \qquad \langle\!\langle u_\ell, v \rangle\!\rangle_{W+S} = \langle (1/2 - K')\Phi_\ell, v \rangle_\Gamma \quad \text{for all } v \in H^{1/2}(\Gamma).$$

According to (3.17), (3.22) and the triangle inquality, there holds

$$(3.24) \qquad \|u - U_\ell\|_{W+S} \leq \|u - u_\ell\|_{W+S} + \|u_\ell - U_\ell\|_{W+S} \lesssim \mathrm{osc}_{N,\ell} + \eta_\ell.$$

Clearly, the Galerkin solution $\widehat{U}_\ell$ with respect to $\mathcal{S}^1(\widehat{\mathcal{T}}_\ell)$ 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 2, 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

$$(3.25) \qquad \eta_\ell \sim \|\widehat{U}_\ell - I_\ell \widehat{U}_\ell\|_{W+S} =: \widetilde{\eta}_\ell$$

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 $\mathcal{S}^1(\mathcal{T}_\ell)$, 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 $\mathcal{T}_\ell$ 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 $E_i \in \mathcal{T}_\ell$. This is different for the error estimators μ_ℓ and $\widetilde{\mu}_\ell$ discussed in the following. For instance, one can prove that

$$(3.26) \qquad \eta_\ell \sim \mu_\ell := \|h_\ell^{1/2}(\widehat{U}_\ell - U_\ell)'\|_{L^2(\Gamma)} = \Big(\sum_{i=1}^{N} \mathrm{length}(E_i) \|(\widehat{U}_\ell - U_\ell)'\|_{L^2(E_i)}^2 \Big)^{1/2}.$$

The local contributions

$$(3.27) \qquad \mu_\ell(E_i) := \mathrm{length}(E_i)^{1/2} \|(\widehat{U}_\ell - U_\ell)'\|_{L^2(E_i)} \quad \text{for all } E_i \in \mathcal{T}_\ell$$

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

$$(3.28) \qquad \eta_\ell \sim \widetilde{\mu}_\ell := \|h_\ell^{1/2}(\widehat{U}_\ell - I_\ell \widehat{U}_\ell)'\|_{L^2(\Gamma)}.$$

In the following subsections, we first discuss the computation of the *global* error estimators η_ℓ and $\widetilde{\eta}_\ell$ from (3.22) and (3.25). Then, we give an implementation of the *local* error estimators μ_ℓ and $\widetilde{\mu}_\ell$ from (3.26) and (3.28), where our functions return the local contributions, see e.g. (3.27), to steer an adaptive mesh-refinement.

Remark 3.2. *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*

$$(3.29) \qquad \eta_\ell \leq \widetilde{\eta}_\ell \lesssim \widetilde{\mu}_\ell \leq \mu_\ell \lesssim \eta_\ell,$$

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 3.3. *Mathematically, the error estimate (3.22) respectively (3.24) involves the so-called saturation assumption: Assume that we could compute the Galerkin solutions U_ℓ^* and $\widehat{U}_\ell^*$ with respect to $\mathcal{T}_\ell$ and $\widehat{\mathcal{T}}_\ell$ for the non-perturbed variational formulation (3.5), 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*

$$(3.30) \qquad \|u - \widehat{U}_\ell^*\|_{W+S} \leq q \, \|u - U_\ell^*\|_{W+S}$$

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 (3.30), one can prove that

$$(3.31) \qquad \eta_\ell \leq \|u_\ell - U_\ell\|_{W+S} \leq (1 - q^2)^{-1/2} \, \eta_\ell.$$

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, (3.30) still remains open.

Finally, one can prove that (3.30) is sufficient and in some sense even necessary to guarantee (3.24). □

Remark 3.4. *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 3.4, 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. □

<div align="center">LISTING 15</div>

```
1  function est = computeEstHypEta(elements_fine,elements_coarse,father2son,...
2                                              W_fine,x_fine,x_coarse)
3  nC = length(x_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 %*** compute coefficient vector of (u_fine - u_coarse) w.r.t. fine mesh
16 x_fine(idx) = x_fine(idx) - x_coarse;
17 x_fine(mid) = x_fine(mid) - 0.5*sum(x_coarse(elements_coarse),2);
18
19 %*** compute energy ||| u_fine - u_coarse |||^2
20 est = x_fine'*(W_fine*x_fine);
```

3.5.1. 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 (3.22). 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 $\mathcal{S}^1(\mathcal{T}_\ell)$ and $\mathcal{S}^1(\widehat{\mathcal{T}}_\ell)$, 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 2.5.1, there holds $\mathcal{S}^1(\mathcal{T}_\ell) \subset \mathcal{S}^1(\widehat{\mathcal{T}}_\ell)$ which gives us 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

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

$$= (\widehat{\mathbf{x}} - \widehat{\mathbf{y}}) \cdot (\widehat{\mathbf{W}} + \widehat{\mathbf{S}})(\widehat{\mathbf{x}} - \widehat{\mathbf{y}}),$$

<div align="center">33</div>

where $\widehat{\mathbf{W}}$ denotes the matrix of the hypersingular integral operator and $\widehat{\mathbf{S}}$ the matrix of the stabilization term contributions (3.10) with respect to the fine mesh, cf. Section 3, *Continuous Model Problem* and *Galerkin Discretization*. 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 $\widehat{\mathcal{T}}_\ell$ stored in W_fine. Besides this, the function takes the coarse triangulation described by the $(N \times 2)$-array elements_coarse and the fine triangulation described by the $(2N \times 2)$-array elements_fine. Finally, the $(N \times 2)$-array father2son links the indices of elements $E_i \in \mathcal{T}_\ell$ with the indices of its sons $e_j, e_k \in \widehat{\mathcal{T}}_\ell$ 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);
```

3.5.2. Computation of Error Estimator $\widetilde{\boldsymbol{\eta}}_\ell$ (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 3.5.1 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 $e_j, e_k \in \widehat{\mathcal{T}}_\ell$ being the sons of $E_i \in \mathcal{T}_\ell$, 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

(3.32) $$I_\ell \widehat{U}_\ell(z_i) = \frac{1}{2} \left(I_\ell \widehat{U}_\ell(z_j) + I_\ell \widehat{U}_\ell(z_k) \right) = \frac{1}{2} \left(\widehat{U}_\ell(z_j) + \widehat{U}_\ell(z_k) \right),$$

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 $I_\ell \widehat{U}_\ell \in \mathcal{S}^1(\mathcal{T}_\ell)$ with respect to the fine-mesh $\widehat{\mathcal{T}}_\ell$, we obtain

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

where $\widehat{\mathbf{z}} \in \mathbb{R}^{2N}$ just denotes the coefficient vector. As in Section 3.5.1, 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 $\widehat{\mathcal{T}}_\ell$ stored in W_fine. Besides this, the function takes the coarse triangulation described by the $(N \times 2)$-array elements_coarse and the fine triangulation described by the $(2N \times 2)$-array elements_fine. Moreover, the link between $\mathcal{T}_\ell$ and $\widehat{\mathcal{T}}_\ell$ 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).

<div align="center">LISTING 17</div>

```
1  function ind = computeEstHypMu(elements_fine,elements_coarse,father2son,...
2                                                          x_fine,x_coarse)
3  nC = length(x_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 %*** compute coefficient vector of (u_fine — u_coarse) w.r.t. fine mesh
16 x_fine(idx) = x_fine(idx) — x_coarse;
```

<div align="center">35</div>

```
17  x_fine(mid) = x_fine(mid) − 0.5*sum(x_coarse(elements_coarse),2);
18
19  %*** compute h^2*|(u_fine − u_coarse)'|^2 for all fine−mesh elements
20  %*** where h denotes the diameters of the fine−mesh elements
21  grad = (x_fine(elements_fine)*[−1;1]).^2;
22
23  %*** compute (squared) indicators w.r.t. coarse mesh as described above
24  ind = 2*( grad(father2son(:,1)) + grad(father2son(:,2)) );
```

3.5.3. 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 $E_i \in \mathcal{T}_\ell$.

We adopt the notation of Section 3.5.1, 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 $E_i \in \mathcal{T}_\ell$ and sons $e_j, e_k \in \widehat{\mathcal{T}}_\ell$ 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 $(\widehat{U}_\ell - U_\ell) \in \mathcal{S}^1(\widehat{\mathcal{T}}_\ell)$ is piecewise affine, its arc-length derivative $(\widehat{U}_\ell - U_\ell)' \in \mathcal{P}^0(\widehat{\mathcal{T}}_\ell)$ is piecewise constant. Consequently, the above formula reduces to

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

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

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

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

Altogether, the documentation of Listing 17 reads as follows:

- As input arguments, the function takes the triangulation $\mathcal{T}_\ell$ represented by the $(N \times 2)$-array elements_coarse, the triangulation $\widehat{\mathcal{T}}_\ell$ represented by the $(2N \times 2)$-array elements_fine, the link between $\mathcal{T}_\ell$ and $\widehat{\mathcal{T}}_\ell$, 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 3.5.1 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 (3.34) and returns the vector

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

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

LISTING 18

```
1  function ind = computeEstHypMuTilde(elements_fine,elements_coarse,...
2                                              father2son,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 h^2*| ((1 − I_coarse)*u_fine)' |^2 for all fine−mesh elements
24 %*** where h denotes the diameters of the fine−mesh elements
25 grad = (x_fine(elements_fine)*[−1;1]).^2;
26
27 %*** compute (squared) indicators w.r.t. coarse mesh as described above
28 ind = 2*( grad(father2son(:,1)) + grad(father2son(:,2)) );
```

3.5.4. Computation of Error Estimator $\widetilde{\mu}_\ell$ (Listing 18).

In this section, we finally aim to compute

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

We adopt the notation of Section 3.5.1, 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 3.5.2 and 3.5.3, a concise documentation of Listing 18 reads as follows:

- The function takes the triangulations $\mathcal{T}_\ell$ and $\widehat{\mathcal{T}}_\ell$, the link between $\mathcal{T}_\ell$ and $\widehat{\mathcal{T}}_\ell$, and the coefficient vector $\widehat{\mathbf{x}} \in \mathbb{R}^{2N}$ of $\widehat{U}_\ell$ (Line 1–2).
- Adopting the ideas of Section 3.5.2, we compute the coefficient vector of $\widehat{U}_\ell - I_\ell\widehat{U}_\ell$ (Line 7–21).
- According to Section 3.5.3, we compute the local contributions $\text{length}(e_j)^2|(\widehat{U}_\ell - I_\ell\widehat{U}_\ell)'|^2$ for all elements $e_j \in \widehat{\mathcal{T}}_\ell$ (Line 25).
- Finally (Line 28), the function returns the vector

$$\mathbf{v} := (\widetilde{\mu}_\ell(E_1)^2, \ldots, \widetilde{\mu}_\ell(E_N)^2) \in \mathbb{R}^N.$$

In particular, there holds $\widetilde{\mu}_\ell = \left(\sum_{i=1}^{N} \mathbf{v}_i\right)^{1/2}$.

LISTING 19

```
1  % adaptiveHypsing provides the implementation of an adaptive mesh—refining
2  % 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
```

3.6. Adaptive Mesh-Refinement for Hypersingular Integral Equation (Listing 19).
Usually computing time and memory requirements are limiting quantities for numerical simulations. Therefore, one aims to choose the triangulation 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, where the error appears to be large, i.e., where the local contributions $\widetilde{\mu}_\ell(E_j)$ are large. For the error estimator $\widetilde{\mu}_\ell$ from Section 3.5.4, a possible adaptive algorithm reads as follows:

Input: Initial mesh $\mathcal{T}_0$, 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 $\widehat{\mathcal{T}}_\ell$.
    (ii) Compute Galerkin solution $\widehat{U}_\ell \in \mathcal{S}^1(\widehat{\mathcal{T}}_\ell)$.
    (iii) Compute refinement indicators $\widetilde{\mu}_\ell(E)^2$ and oscillation terms $\mathrm{osc}_{N,\ell}(E)^2$ for all $E \in \mathcal{T}_\ell$.

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

$$(3.35) \qquad \theta \left(\widetilde{\mu}_\ell^2 + \mathrm{osc}_{N,\ell}^2 \right) = \theta \sum_{E \in \mathcal{T}_\ell} \left(\widetilde{\mu}_\ell(E)^2 + \mathrm{osc}_{N,\ell}(E)^2 \right) \leq \sum_{E \in \mathcal{M}_\ell} \left(\widetilde{\mu}_\ell(E)^2 + \mathrm{osc}_{N,\ell}(E)^2 \right).$$

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

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

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

The marking criterion (3.35) 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 (3.30) 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 (3.30) holds (at least every finitely many steps), we then obtain convergence of U_ℓ to u due to $\|u - U_\ell\|_{W+S}^2 \lesssim \widetilde{\mu}_\ell^2 + \mathrm{osc}_{N,\ell}^2$.

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 (3.35) and mark at least the 25% of elements with the largest indicators (Line 8–9). The marking criterion is explained in Section 2.6.1.

4. 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 3. 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

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

For the equivalent integral formulation of (4.1), 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

$$
(4.2) \qquad u_N := u - \overline{u}_D \quad\text{and}\quad \phi_D := \partial_n u - \overline{\phi}_N.
$$

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

$$
(4.3) \qquad A = \begin{pmatrix} -K & V \\ W & K' \end{pmatrix},
$$

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

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

One can prove that (4.4) is, in fact, an equivalent formulation of the mixed boundary value problem (4.1). With the spaces

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

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

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

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

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

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

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

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 (4.4)

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

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 $\mathcal{T}_\ell$ be a triangulation of Γ. By definition, $\mathcal{T}_\ell$ then resolves Γ_D and Γ_N, cf. Section 1.1. Consequently,

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

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

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

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

Extension of the Given Dirichlet and Neumann Data. By Definition (4.2), the solution $\mathbf{u} = (u_N, \phi_D)$ of (4.9) depends on the chosen extensions $\overline{u}_D$ of u_D and $\overline{\phi}_N$ of ϕ_N. We assume additional regularity

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

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

(4.11) $$\overline{\phi}_N|_{\Gamma_N} = \phi_N \quad \text{and} \quad \overline{\phi}_N|_{\Gamma_D} = 0$$

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

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

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

Galerkin Discretization. Let $\mathcal{T}_\ell = \{E_1, \ldots, E_N\}$ be a triangulation of Γ obtained by certain refinements of the initial mesh $\mathcal{T}_0$. To discretize (4.9), we replace the continuous Dirichlet data $\overline{u}_D \in H^1(\Gamma) \subset C(\Gamma)$ by the nodal interpolant

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

and the Neumann data by its L^2-projection

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

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

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

Here, the representation for $\Phi_{N,\ell}$ shrinks to a sum over all elements on the Neumann boundary by definition (4.11) 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

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

Altogether, this leads to the following discrete version of the integral equation (4.4): Find $\mathbf{U}_\ell \in X_\ell$ with

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

where the approximated right-hand side is given by

$$(4.18) \qquad F_\ell := (1/2 - A) \begin{pmatrix} U_{D,\ell} \\ \Phi_{N,\ell} \end{pmatrix}.$$

We use (4.18) here because the right hand side of (4.4) can hardly be evaluated numerically. In order to write (4.17) as a linear system of equations

$$(4.19) \qquad \mathbf{Ax} = \mathbf{b},$$

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

- Let $\mathcal{T}_\ell = \{E_1, \ldots, E_N\}$ and assume that $\overline{\Gamma}_D = \bigcup_{j=1}^d E_j$. Then, $\{\chi_1, \ldots, \chi_N\}$ is a basis of $\mathcal{P}^0(\mathcal{T}_\ell)$ and $\{\chi_1, \ldots, \chi_d\}$ is a basis of $\mathcal{P}^0(\mathcal{T}_\ell|_{\Gamma_D})$.
- Let $\mathcal{K}_\ell = \{z_1, \ldots, z_N\}$ and assume that $\{z_1, \ldots, z_n\} = \mathcal{K}_\ell \cap \Gamma_N$. Then, $\{\zeta_1, \ldots, \zeta_N\}$ is a basis of $\mathcal{S}^1(\mathcal{T}_\ell)$ and $\{\zeta_1, \ldots, \zeta_n\}$ is a basis of $\mathcal{S}_0^1(\mathcal{T}_\ell|_{\Gamma_N})$.
- In particular, $\{(\zeta_1, 0), \ldots, (\zeta_n, 0), (0, \chi_1), \ldots, (0, \chi_d)\}$ 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 (4.19) reads as follows: According to Linear Algebra, the Galerkin system (4.17) 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

$$(4.20) \qquad \mathbf{b} \in \mathbb{R}^{n+d}, \quad \text{where} \quad \mathbf{b}_j := \langle F_\ell, (\zeta_j, 0) \rangle_{\mathcal{H}^* \times \mathcal{H}}, \quad \mathbf{b}_{n+k} := \langle F_\ell, (0, \chi_k) \rangle_{\mathcal{H}^* \times \mathcal{H}},$$

for all $j = 1, \ldots, n$ and $k = 1, \ldots, d$. Recall the matrices $\mathbf{M}, \mathbf{K} \in \mathbb{R}^{N \times N}$ defined in (2.8) and the matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ from (3.10). With the data representation (4.15), there holds

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

where we have finally used the symmetry of $\mathbf{W}$. Now, also recall the matrix $\mathbf{V} \in \mathbb{R}^{N \times N}$ from (2.8). The same type of arguments leads to

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

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

$$(4.21) \qquad \mathbf{b} = \begin{pmatrix} \left(\frac{1}{2} \mathbf{p}^T \mathbf{M} - \mathbf{p}^T \mathbf{K} - \mathbf{g}^T \mathbf{W} \right)^T |_{\Gamma_N} \\ \left(\frac{1}{2} \mathbf{M} \mathbf{g} + \mathbf{K} \mathbf{g} - \mathbf{V} \mathbf{p} \right) |_{\Gamma_D} \end{pmatrix}.$$

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

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

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

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

Altogether, we obtain the following short-hand notation for the linear system (4.19)

$$(4.22) \qquad \begin{pmatrix} \mathbf{W}|_{\Gamma_N \times \Gamma_N} & \mathbf{K}^T|_{\Gamma_N \times \Gamma_D} \\ -\mathbf{K}|_{\Gamma_D \times \Gamma_N} & \mathbf{V}|_{\Gamma_D \times \Gamma_D} \end{pmatrix} \mathbf{x} = \begin{pmatrix} \left(\frac{1}{2}\mathbf{p}^T\mathbf{M} - \mathbf{p}^T\mathbf{K} - \mathbf{g}^T\mathbf{W} \right)^T |_{\Gamma_N} \\ \left(\frac{1}{2}\mathbf{Mg} + \mathbf{Kg} - \mathbf{Vp} \right) |_{\Gamma_D} \end{pmatrix}.$$

Listing 20

```
1  function [b1,b2] = buildMixedRHS(coordinates,dirichlet,neumann,g,V,K,W,uD,phiN)
2  nD = size(dirichlet,1);
3  nN = size(neumann,1);
4  nE = nD + nN;
5  elements = [dirichlet;neumann];
6
7  %*** arbitrary quadrature on [−1,1] with exactness n >= 2, e.g., gauss(2)
8  quad_nodes = [−1 1]/sqrt(3);
9  quad_weights = [1;1];
10
11 %*** the remaining code is independent of the chosen quadrature rule
12 nQ = length(quad_nodes);
13
14 %*** build vector of evaluation points as (nQ*nN x 2)−matrix
15 a = coordinates(neumann(:,1),:);
16 b = coordinates(neumann(:,2),:);
17 sx = reshape(a,2*nN,1)*(1−quad_nodes) + reshape(b,2*nN,1)*(1+quad_nodes);
18 sx = 0.5*reshape(sx',nQ*nN,2);
19
20 %*** phiN(sx) usually depends on the normal vector, whence phi takes sx and the
21 %*** nodes of the respective element to compute the normal
22 a_sx = reshape(repmat(reshape(a,2*nN,1),1,nQ)',nN*nQ,2);
23 b_sx = reshape(repmat(reshape(b,2*nN,1),1,nQ)',nN*nQ,2);
24
25 %*** perform all necessary evaluations of phi as (nE x nQ)−matrix
26 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;
```

4.1. Build Right-Hand Side Vector (Listing 20). To compute the vector **b** from (4.19), we first recall the representation of **b** in (4.22). In the routine for the computation of **b**, we choose to compute

$$(4.23) \qquad \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix} := \begin{pmatrix} \left(\frac{1}{2}\mathbf{p}^T\mathbf{M} - \mathbf{g}^T\mathbf{W} - \mathbf{p}^T\mathbf{K} \right)^T \big|_{\Gamma_N} \\ \left(\frac{1}{2}\mathbf{Mg} + \mathbf{Kg} - \mathbf{Vp} \right) \big|_{\Gamma_D} \end{pmatrix}.$$

The documentation of Listing 20 reads as follows:

- As input, the function takes the mesh $\mathcal{T}_\ell$ described in terms of `coordinates`, `dirichlet` and `neumann`, where the last two arrays represent the Dirchlet 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 $\mathcal{T}_\ell$. 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 3.3.
- 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 2.2 for details.
- Finally (Line 43–44), the right-hand side vectors `b1` and `b2` are computed.

<div align="center">LISTING 21</div>

```
1  function g = buildMixedDirichlet(coordinates,dirichlet,neumann, ...
2                                    father2neumann,neumann_old,g_old,uD)
3  nC = size(coordinates,1);
4
5  %*** prolongate Dirichlet data on Neumann boundary
6  g = zeros(nC,1);
7  g(neumann(father2neumann(:,1),1)) = g_old(neumann_old(:,1));
8  g(neumann(father2neumann(:,2),2)) = g_old(neumann_old(:,2));
9  g(neumann(father2neumann(:,1),2)) = 0.5*sum(g_old(neumann_old),2);
10
11 %*** evaluate Dirichlet data on Dirichlet boundary
12 g(unique(dirichlet)) = uD(coordinates(unique(dirichlet),:));
```

4.2. 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 $\mathcal{T}_0$ and choose the extended Dirichlet data $\overline{u}_D$ to satisfy $\overline{u}_D|_{\Gamma_N} \in \mathcal{S}^1(\mathcal{T}_0|_{\Gamma_N})$ 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 $\mathcal{T}_\ell$.

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 $\mathcal{T}_\ell$ is described by the arrays `coordinates`, `dirichlet`, and `neumann`. The array `neumann_old` is the Neumann part of the mesh $\mathcal{T}_{\ell-1}$ and the link between $\mathcal{T}_{\ell-1}|_{\Gamma_N}$ and $\mathcal{T}_\ell|_{\Gamma_N}$ 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 $\mathcal{T}_\ell$ was also a node of $\mathcal{T}_{\ell-1}$, 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 $\mathcal{T}_\ell$ 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 $\mathcal{T}_{\ell-1}$. 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

```
1  function [coordinates,dirichlet,neumann] = buildMixedElements(coordinates, ...
2                                              dirichlet,neumann)
3  %*** determine nodes on Dirichlet and Neumann boundary
4  nC = size(coordinates,1);
5  nodes_neumann = unique(neumann);
6  nodes_dirichlet = setdiff((1:nC)',nodes_neumann);
7
8  %*** build permutation such that Neumann nodes are first
9  nodes = [nodes_neumann;nodes_dirichlet];
10 [foo,permutation] = sort(nodes);
11
12 %*** permute indices of nodes
13 coordinates(permutation,:) = coordinates;
14 neumann = permutation(neumann);
15 dirichlet = permutation(dirichlet);
```

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

For $\mathcal{S}_0^1(\mathcal{T}_\ell|_{\Gamma_N}) \subset \widetilde{H}^{1/2}(\Gamma_N)$ we aim to benefit from the functions already written for the hypersingular integral equation. To do so, we have to embed $\mathcal{S}_0^1(\mathcal{T}_\ell|_{\Gamma_N})$ into $\mathcal{S}^1(\mathcal{T}_\ell|_{\Gamma_N})$. 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 $\mathcal{S}^1(\mathcal{T}_\ell|_{\Gamma_N})$ 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`.

4.4. Computation of Data Oscillations for Mixed Boundary Data. Instead of the correct variational form (4.9)

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

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

$$(4.24) \qquad\qquad \langle\!\langle \mathbf{u}_\ell\,, \mathbf{v}\rangle\!\rangle_A = \langle F_\ell\,, \mathbf{v}\rangle_{\mathcal{H}^* \times \mathcal{H}} \quad \text{for all } \mathbf{v} \in \mathcal{H}.$$

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 2.3 and 3.3.

4.5. 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 (4.8), 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 $\mathcal{T}_\ell$ and its uniform refinement $\widehat{\mathcal{T}}_\ell$. As in Section 2.5, we define the following four error estimators for the part of the simple-layer potential:

$$\eta_{D,\ell} := \|\widehat{\Phi}_{D,\ell} - \Phi_{D,\ell}\|_{V(\Gamma_D)} \qquad\qquad \widetilde{\eta}_{D,\ell} := \|\widehat{\Phi}_{D,\ell} - \Pi_\ell \widehat{\Phi}_{D,\ell}\|_{V(\Gamma_D)}$$

$$\mu_{D,\ell} := \|h_\ell^{1/2}(\widehat{\Phi}_{D,\ell} - \Phi_{D,\ell})\|_{L^2(\Gamma_D)} \qquad \widetilde{\mu}_{D,\ell} := \|h_\ell^{1/2}(\widehat{\Phi}_{D,\ell} - \Pi_\ell \widehat{\Phi}_{D,\ell})\|_{L^2(\Gamma_D)}$$

In analogy to Section 3.5, we define the following four error estimators for the contribution of the hypersingular integral operator:

$$\eta_{N,\ell} := \|\widehat{U}_{N,\ell} - U_{N,\ell}\|_{W(\Gamma_N)} \qquad\qquad \widetilde{\eta}_{N,\ell} := \|\widehat{U}_{N,\ell} - I_\ell \widehat{U}_{N,\ell}\|_{W(\Gamma_N)}$$

$$\mu_{N,\ell} := \|h_\ell^{1/2}(\widehat{U}_{N,\ell} - U_{N,\ell})'\|_{L^2(\Gamma_N)} \qquad \widetilde{\mu}_{N,\ell} := \|h_\ell^{1/2}(\widehat{U}_{N,\ell} - I_\ell \widehat{U}_{N,\ell})'\|_{L^2(\Gamma_N)}$$

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

$$\eta_\ell^2 := \eta_{D,\ell}^2 + \eta_{N,\ell}^2 \qquad\qquad \widetilde{\eta}_\ell^2 := \widetilde{\eta}_{D,\ell}^2 + \widetilde{\eta}_{N,\ell}^2$$

$$\mu_\ell^2 := \mu_{D,\ell}^2 + \mu_{N,\ell}^2 \qquad\qquad \widetilde{\mu}_\ell^{\,2} := \widetilde{\mu}_{D,\ell}^2 + \widetilde{\mu}_{N,\ell}^2$$

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

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

Moreover, there holds efficiency in the form

$$\eta_\ell \leq \|\mathbf{u} - \mathbf{U}_\ell\|_A + C \operatorname{osc}_\ell$$

with a constant $C > 0$ which depends only on Γ and $\kappa(\mathcal{T}_\ell)$. Under a saturation assumption for the non-perturbed problem, there holds also reliability

$$(4.25) \qquad \|\mathbf{u} - \mathbf{U}_\ell\|_A \lesssim \eta_\ell + \operatorname{osc}_\ell.$$

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

$$\varrho_\ell^2 := \mu_\ell^2 + \operatorname{osc}_\ell^2 = (\mu_{D,\ell}^2 + \operatorname{osc}_{D,\ell}^2) + (\mu_{N,\ell}^2 + \operatorname{osc}_{N,\ell}^2),$$
$$\widetilde{\varrho}_\ell^2 := \widetilde{\mu}_\ell^2 + \operatorname{osc}_\ell^2 = (\widetilde{\mu}_{D,\ell}^2 + \operatorname{osc}_{D,\ell}^2) + (\widetilde{\mu}_{N,\ell}^2 + \operatorname{osc}_{N,\ell}^2).$$

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 $\widehat{\mathbf{U}}_\ell$.

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

$$(4.26) \qquad \widetilde{\varrho}_\ell(E_j)^2 := \begin{cases} \widetilde{\mu}_{D,\ell}(E_j)^2 + \operatorname{osc}_{D,\ell}(E_j)^2 & \text{if } E_j \subseteq \overline{\Gamma}_D, \\ \widetilde{\mu}_{N,\ell}(E_j)^2 + \operatorname{osc}_{N,\ell}(E_j)^2 & \text{if } E_j \subseteq \overline{\Gamma}_N. \end{cases}$$

Note that there holds

$$(4.27) \qquad \widetilde{\varrho}_\ell^2 = \sum_{j=1}^N \widetilde{\varrho}_\ell(E_j)^2.$$

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

Input: Initial mesh $\mathcal{T}_0$, 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 $\widehat{\mathcal{T}}_\ell$.
   (ii) Compute Galerkin solution $\widehat{\mathbf{U}}_\ell \in \widehat{X}_\ell$.
   (iii) Compute refinement indicators $\widetilde{\varrho}_\ell(E)^2$ for all $E \in \mathcal{T}_\ell$.
   (iv) Find minimal set $\mathcal{M}_\ell \subseteq \mathcal{T}_\ell$ such that

$$(4.28) \qquad \theta\, \widetilde{\varrho}_\ell^2 = \theta \sum_{E \in \mathcal{T}_\ell} \widetilde{\varrho}_\ell(E)^2 \leq \sum_{E \in \mathcal{M}_\ell} \widetilde{\varrho}_\ell(E)^2.$$

   (v) Refine (at least) marked elements $E \in \mathcal{M}_\ell$ and obtain mesh $\mathcal{T}_{\ell+1}$ with $\kappa(\mathcal{T}_{\ell+1}) \leq 2\kappa(\mathcal{T}_0)$.
   (vi) Stop provided that $\#\mathcal{T}_{\ell+1} \geq N_{\max}$; otherwise, increase counter $\ell \mapsto \ell + 1$ and go to (i).
Output: Adaptively generated mesh $\widehat{\mathcal{T}}_\ell$ and corresponding discrete solution $\widehat{\mathbf{U}}_\ell \in \widehat{X}_\ell$.

The marking criterion (4.28) 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 (4.25), 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 every finitely many steps), we then obtain convergence of $\mathbf{U}_\ell$ to $\mathbf{u}$ due to $\|\mathbf{u} - \mathbf{U}_\ell\|_A^2 \lesssim \varrho_\ell^2$.

References

[AFP]   M. Aurada, S. Ferraz-Leite, D. Praetorius: *Estimator Reduction and Convergence of Adaptive FEM and BEM*, submitted for publication (2009).

[AGP]   M. Aurada, P. Goldenits, D. Praetorius: *Convergence of Data Perturbed Adaptive Boundary Element Methods*, submitted for publication (2009).

[AGKP] M. Aurada, P. Goldenits, M. Karkulik, D. Praetorius: *Adaptive BEM for some Mixed Boundary Value Problem*, work in progress 2009.

[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*, submitted for publication (2009).

[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.

[MSW]   P. Mund, E. Stephan, J. Weisse: *Two-Level Methods for the Single Layer Potential in $\mathbb{R}^3$*, Computing **60** (1998), 243–266.

[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.

[WX]   S. Wandzura, H. Xiao: *Symmetric Quadrature Rules on a Triangle*, Comput. Math. Appl. **45** (2003), 1829–1840.

Institute for Analysis and Scientific Computing, Vienna University of Technology, Wiedner Hauptstrasse 8-10, A-1040 Wien, Austria

E-mail address: `markus.aurada@tuwien.ac.at` (corresponding author)