Unterschrift (Betreuer)

TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

D I P L O M A R B E I T

Adaptive Isogeometrische Randelementmethode

für die
Hypersinguläre Integralgleichung

ausgeführt am
Institut für Analysis und Scientific Computing
der Technischen Universität Wien

unter der Anleitung von
Ao.Univ.Prof. Dipl.Math. Dr.techn. Dirk Praetorius
Dipl.-Ing. Gregor Gantner

durch
Stefan Schimanko
Südhangsiedlung 9
3300 Amstetten

Datum

Unterschrift (Student)

TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

DIPLOMA THESIS

Adaptive Isogeometric Boundary Element Method

for the
Hyper-Singular Integral Equation

written at the
Institute for Analysis and Scientific Computing
of the Vienna University of Technology

supervised by
Ao.Univ.Prof. Dipl.Math. Dr.techn. Dirk Praetorius
Dipl.-Ing. Gregor Gantner

by
Stefan Schimanko
Südhangsiedlung 9
3300 Amstetten

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen selbstständig ausgeführt habe und alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, genannt habe.

Datum	Unterschrift

Abstract and Outline

The present work deals with the effective numerical solution of the hyper-singular integral equation

$$Wu = (1/2 - K')\phi \quad \text{on } \Gamma := \partial\Omega$$

for given boundary data ϕ with integral mean zero and a bounded Lipschitz domain $\Omega \subset \mathbb{R}^2$. Here, W denotes the hyper-singular integral operator and K' is the adjoint double-layer integral operator, cf. Section 1. This integral equation is an equivalent formulation of the Neumann problem

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

where $\partial_n P$ denotes the normal derivative of P. Indeed, $P|_\Gamma$ conincides with u up to an additvie constant. On the other hand, P can be regained by u and ϕ via the representation formula (or: third Green formula), cf. Subsection 1.3.

In this thesis, we present and analyze an adaptive isogeometric boundary element method for the hyper-singular integral equation. The main principle of isogeometric analysis (IGA) is to use the same kind of functions, which represent the problem geometry, for the discretization of the partial differential equation at hand. Nowadays, the problem geometry Ω resp. its boundary Γ is usually represented in computer aided design (CAD) by so-called non-uniform rational B-splines (NURBS), T-splines, or hierarchical splines. The advantage of this approach is that, unlike standard approaches, no meshing of the geometry Ω resp. its boundary Γ is necessary in order to define suitable approximation spaces. Thomas Hughes and collaborators originally invented this concept for finite element methods (IGAFEM) for three dimensional geometries, cf. [CHB09] and [HCB05].
However, CAD typically describes only the boundary Γ, not the volume Ω itself. Hence, the most appealing approach is the boundary element method (BEM), if applicable, i.e., if the fundamental solution of the considered differential operator is explicitly known. In our case, the fundamental solution G of the Laplacian is given by

$$G(x, y) = -\frac{1}{2\pi} \log |x - y|.$$

We consider the variational formulation of the hypersingular integral equation, i.e.,

$$\langle Wu; v\rangle_\Gamma + \langle 1; u\rangle_\Gamma \langle 1; v\rangle_\Gamma = \langle (1/2 - K')\phi; v\rangle_\Gamma \quad \text{for all } v \in H^{1/2}(\Gamma),$$

cf. Subsection 1.3. The basic idea of BEM is to replace the trace space $H^{1/2}(\Gamma)$ in the variational formulation by a finite dimensional subspace. We end up with a discrete variational formulation, which is solvable numerically. Then, the discrete solution $U_\star \in \mathcal{X}_\star$ is an approximation of the (weak) solution u of the hypersingular integral equation.
Although there exists rich literature on IGAFEM, only little is found for isogeometric BEM (IGABEM). First, IGABEM was considered in [PGK+09] for two dimensions, and in [SSE+13] for three dimensions. Concerning a posteriori error estimation for IGABEM, the articles [FGHP16a], [FGHP16b], and [FGP15] consider simple two dimensional model problems for weakly-singular integral equations which are equivalent formulations of the Dirichlet problem.
This thesis now deals with the hyper-singular integral equation. To approximate the (weak) solution u of the hyper-singular integral equation, we employ an adaptive Galerkin IGABEM of the type

$$\boxed{\texttt{SOLVE}} \quad \Longrightarrow \quad \boxed{\texttt{ESTIMATE}} \quad \Longrightarrow \quad \boxed{\texttt{MARK}} \quad \Longrightarrow \quad \boxed{\texttt{REFINE}} \tag{1}$$

with ansatz spaces consisting of p-th order NURBS. Notice that the usually used piecewise polynomials for standard BEM are just special NURBS. Due to numerical reasons, we also incorporate an approximation of the given Neumann data ϕ into the adaptive scheme. To this end, we employ the L^2-orthogonal projection to replace ϕ by some discrete data $\Pi_\star\phi$. As a consequence, we have to deal with oscillation

terms in the analysis. However, the big advantage of this approach is that the appearing integrals can be computed with reliable quadratures. Especially the adjoint double-layer integral operator K' caused stability problems in the beginning, since possible singularities of ϕ as well as the singular kernel of the boundary integral operator K' had to be treated simultaneously. These severe stability issues are avoided by use of the approximation $\Pi_\star \phi$.

The outline of the thesis is as follows: In Section 1, we introduce the necessary Sobolev spaces on the boundary Γ. We also give the precise definitions of the hyper-singular integral operator W and the adjoint double-layer integral operator K'. In the adaptive scheme, we will compute discrete solutions. To this end, we introduce two equivalent variational formulations resp. their discrete counterparts. Additionally, we comment on the equivalence of the hyper-singular integral equation and the Neumann problem.

In Section 2, we first introduce B-splines. Based on these, NURBS are defined and some basic properties are deduced. Afterwards, the boundary discretization and the corresponding discretization spaces for the isogeometric boundary element method are discussed.

Section 3 introduces two locally equivalent weighted residual error estimators $\mu_\star$ and $\widetilde{\rho}_\star$ which also include the data oscillation terms of the approximation $\Pi_\star \phi$ of ϕ. For standard BEM, [CS95] first proposed the weighted residual error estimator which was sharpened later in [Car97]. While the local contributions of $\mu_\star$ are related to the nodes of a partition of the boundary Γ, the local contributions of the auxiliary estimator $\widetilde{\rho}_\star$ are related to elements of the partition. We note that we employ the node-based error estimator $\mu_\star$ to steer the adaptive IGABEM algorithm, while the auxiliary estimator $\widetilde{\rho}_\star$ is better suited for the numerical analysis. Hence, we first collect some technical auxiliary results which are used to analyse $\widetilde{\rho}_\star$ and the oscillation terms derived from the approximation of the Neumann data ϕ.
Further, we formally state the adaptive Algorithm (1) steered by $\mu_\star$. First, we compute the approximation $\Pi_\ell \phi$ and solve the discrete variational formulation of the hypersingular integral equation with the perturbed right-hand side $(1/2 - K')\Pi_\ell \phi$. The next step is to compute the local contributions of μ_ℓ and to mark the nodes with the largest contributions. Based on this *a posteriori* infomation, the refinement strategy steers both the local mesh-size of the underlying partition and the local smoothness of the NURBS, which leads to a refined mesh and a larger ansatz space. We repeat this loop until a given tolerance for the overall error $\mu_\star$ is reached.

Because of the local equivalence of $\mu_\star$ and $\widetilde{\rho}_\star$, the mathematical results of $\widetilde{\rho}_\star$ will transfer to $\mu_\star$ and will thus allow to analyze the adaptive strategy based on $\mu_\star$. Our main results read as follow: First, the error estimator is linearly convergent with respect to the number of adaptive steps, cf. Section 4. This means that there exist constants $0 < q_{\mathrm{lin}} < 1$ and $C_{\mathrm{lin}} > 0$ such that

$$\mu_{\ell+k} \leq C_{\mathrm{lin}}\, q_{\mathrm{lin}}^k\, \mu_\ell \quad \text{for all } k, \ell \in \mathbb{N}_0.$$

Second, cf. Section 5, the estimator μ_ℓ decays at optimal algebraic convergence rate. This means that if a convergence rate of s in the number of knots is possible for suitably chosen meshes which do not necessarily have to be nested, i.e., $\mathcal{O}(|\mathcal{K}_\star|^{-s})$, then the nested meshes generated by the adaptive algorithm lead to the same order of convergence of the estimator. Hence, there exists a constant $C_{\mathrm{opt}} > 0$ such that

$$\mu_\ell \leq C_{\mathrm{opt}} \left(|\mathcal{K}_\ell| - |\mathcal{K}_0|\right)^{-s} \quad \text{for all } \ell \in \mathbb{N}_0,$$

where $|\mathcal{K}_\ell|$ denotes the number of knots of the the partition of the ℓ-th step of the adaptive algorithm.

In Section 6, numerical aspects of the implementation of the hyper-singular integral equation are discussed. To underline our analytical findings, numerical experiments are given in Section 7.

The codes of the implementation are given in Appendix A, which also concludes the thesis.

Danksagung

Mein Dank richtet sich an erster Stelle an meine Betreuer Dirk Praetorius und Gregor Gantner, die mir immer mit Rat und Tat zur Seite gestanden sind und auf diese Weise sehr zum Gelingen dieser Arbeit beigetragen haben. Ebenso darf natürlich die ganze Arbeitsgruppe nicht unerwähnt bleiben, mit der ich viele unterhaltsame und interessante Stunden verbringen und genießen durfte.

Des Weiteren möchte ich mich bei meinen Studienkollegen bedanken, im Speziellen vor allem bei Alexander Haberl und Fabian Mußnig, denen ich nicht nur diverse Übungsbeispiele, sowie Mitschriften und Lernunterlagen zu verdanken habe, sondern die besonders auch abseits der Universität zu echten Freunden geworden sind. Auch auf das jährliche Treffen in Weyregg bei Stefanie Renner freue ich mich jeden Sommer, genauso wie auf den regelmäßigen „Stammtisch“ mit meinen Freunden.

Zu guter Letzt gilt mein Dank meiner Familie, die mich immer in meinen Bestrebungen unterstützt hat und für mich da gewesen ist. Egal was passiert, ich weiß, ich kann auf euch zählen, vielen Dank dafür.

Finanziell wurde diese Arbeit durch das FWF Projekt *Optimal Isogeometric Boundary Element Method* (P29096) unterstützt.

Contents

1 Introduction

Throughout, we assume that Ω is a bounded Lipschitz domain in $\mathbb{R}^2$ whose boundary $\Gamma := \partial\Omega$ can be parametrized by a fixed regular closed curve $\gamma\colon [a,b] \to \Gamma$. This means that γ is continuous, piecewise continuously differentiable, and $\gamma|_{[a,b)}$ is bijective. We also demand for the left and right derivative of γ, that $\gamma'^{\ell} \neq 0$ for $t \in (a,b]$ and $\gamma'^{r} \neq 0$ for $t \in [a,b)$. Additionally, we assume that

$$\gamma'^{\ell}(t) + c\gamma'^{r}(t) \neq 0 \quad \text{for all } c > 0 \text{ and all } t \in [a,b]. \tag{1.1}$$

We consider the hyper-singular integral equation

$$Wu = (1/2 - K')\phi \quad \text{on } \Gamma, \tag{1.2}$$

for given Neumann data ϕ, the hyper-singular integral operator W, and the adjoint of the double-layer integral operator K. With the fundamental solution of the Laplacian

$$G(x,y) = -\frac{1}{2\pi}\log|x-y|, \tag{1.3}$$

these operators are defined by

$$\begin{aligned} Wu(x) &:= -\partial_{n_x}\int_\Gamma \partial_{n_y} G(x,y)u(y)\,ds_y, \\ Ku(x) &:= \int_\Gamma \partial_{n_y} G(x,y)u(y)\,ds_y, \end{aligned} \tag{1.4}$$

where n_z denotes the outer unit normal vector at $z \in \Gamma$ and ∂_{n_z} is the normal derivative.

Throughout the work, $|\cdot|$ denotes the absolute value of scalars, the Euclidean norm of vectors in $\mathbb{R}^2$, the measure of a set in $\mathbb{R}$, the cardinality of a discrete set, or the arclength of a curve in $\mathbb{R}^2$, which will be clear from the context. The notation $A \lesssim B$ is an abbriviation for $A \leq cB$ with a generic constant $c > 0$. Moreover, $A \simeq B$ is an abbreviation for $A \lesssim B \lesssim A$. All mesh-related quantities have the same index, e.g., $\mathcal{K}_\star$ is the set of knots of the partition $[\mathcal{T}_\star]$, and $h_\star$ is the corresponding mesh-size etc. The analogous notation is used for partitions $[\mathcal{T}_+]$, $[\mathcal{T}_\bullet]$, $[\mathcal{T}_\ell]$, etc.

1.1 Function spaces

For any measurable subset $\Gamma_0 \subseteq \Gamma$ resp. any interval $\Gamma_0 \subseteq \mathbb{R}$, we denote the Lebesgue space of all square integrable functions by $L^2(\Gamma_0)$. The corresponding norm is

$$\|u\|^2_{L^2(\Gamma_0)} := \int_{\Gamma_0} |u(x)|^2\,dx,$$

and we define the Sobolev space $H^0(\Gamma_0) := L^2(\Gamma_0)$.
If $u \in L^2(\Gamma_0)$ is differentiable along the arc, ∇u denotes the arclength derivative. We define the Sobolev space $H^1(\Gamma_0) := \big\{u \in L^2(\Gamma_0) : \nabla u \in L^2(\Gamma_0)\big\}$ with corresponding norm

$$\begin{aligned} \|u\|^2_{H^1(\Gamma_0)} &:= \|u\|^2_{L^2(\Gamma_0)} + |u|^2_{H^1(\Gamma_0)}, \\ |u|^2_{H^1(\Gamma_0)} &:= \int_{\Gamma_0} |\nabla u(x)|^2\,dx. \end{aligned}$$

Furthermore, we will also need the Sobolev space $H^{1/2}(\Gamma_0) := \big\{u \in L^2(\Gamma_0) : \|u\|_{H^{1/2}(\Gamma_0)} < \infty\big\}$ with corresponding norm

$$\begin{aligned} \|u\|^2_{H^{1/2}(\Gamma_0)} &:= \|u\|^2_{L^2(\Gamma_0)} + |u|^2_{H^{1/2}(\Gamma_0)}, \\ |u|^2_{H^{1/2}(\Gamma_0)} &:= \int_{\Gamma_0}\int_{\Gamma_0} \frac{|u(x)-u(y)|^2}{|x-y|^2}\,dy\,dx. \end{aligned}$$

The dual space of $H^{1/2}(\Gamma)$ is denoted by $H^{-1/2}(\Gamma)$, where duality is understood with respect to the extended $L^2(\Gamma)$-scalar product, i.e., for $u \in H^{1/2}(\Gamma)$ and $\phi \in L^2(\Gamma)$,

$$\langle \phi; u \rangle_\Gamma := \int_\Gamma \phi(x) u(x)\, dx.$$

We note that $H^{1/2}(\Gamma) \subseteq L^2(\Gamma) \subseteq H^{-1/2}(\Gamma)$ form a Gelfand triple and all inclusions are dense and compact, cf. [SS11, Proposition 2.5.2].

1.2 Hyper-singular integral operator W

The hyper-singular integral operator W is a well-defined linear and continuous operator

$$W \colon H^{1/2}(\Gamma) \to H^{-1/2}(\Gamma).$$

Moreover, W is symmetric and positive semi-definite, i.e.,

$$\langle Wu; v \rangle_\Gamma = \langle u; Wv \rangle_\Gamma \quad \text{and} \quad \langle Wu; u \rangle_\Gamma \geq 0 \quad \text{for all } u, v \in H^{1/2}(\Gamma),$$

where $\langle \cdot; \cdot \rangle_\Gamma$ denotes the extended $L^2(\Gamma)$ scalar product.
The bilinear form

$$\langle\!\langle u; v \rangle\!\rangle_W := \langle Wu; v \rangle_\Gamma$$

is a scalar product on $H^{1/2}_\star(\Gamma) := \big\{ v \in H^{1/2} \,:\, \langle 1; v \rangle_\Gamma = 0 \big\}$, see, e.g., [SS11, Section 3.5.1].
Therefore,

$$\langle\!\langle u; v \rangle\!\rangle_{W+S} := \langle Wu; v \rangle_\Gamma + \langle 1; u \rangle_\Gamma \langle 1; v \rangle_\Gamma \quad \text{for all } u, v \in H^{1/2}(\Gamma), \tag{1.5}$$

defines a scalar product on $H^{1/2}(\Gamma)$. Apparently, $\langle\!\langle \cdot; \cdot \rangle\!\rangle_{W+S}$ is bilinear and symmetric. Since W is positive semi-definite and $\langle\!\langle \cdot; \cdot \rangle\!\rangle_W$ is a scalar product on $H^{1/2}_\star(\Gamma)$, it follows, that $\langle\!\langle \cdot; \cdot \rangle\!\rangle_{W+S}$ is also positive definite on $H^{1/2}(\Gamma)$.
According to the Rellich compactness theorem, the induced norm $|\!|\!| u |\!|\!|^2_{W+S} := \langle\!\langle u; u \rangle\!\rangle_{W+S}$ is an equivalent norm on $H^{1/2}(\Gamma)$.

1.3 Neumann problem

The double-layer integral operator K is a well-defined linear and continuous operator

$$K \colon H^{1/2}(\Gamma) \to H^{1/2}(\Gamma). \tag{1.6}$$

Moreover, its adjoint is a well-defined linear and continuous operator

$$K' \colon H^{-1/2}(\Gamma) \to H^{-1/2}(\Gamma). \tag{1.7}$$

The restriction of K' to $L^2(\Gamma)$ even maps to $L^2(\Gamma)$. Further details are found in, e.g., [Pra07, Theorem 4.26].

The hyper-singular integral equation (1.2) stems from the Neumann problem

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

Recall the first Green's formula

$$\langle \nabla P; \nabla Q \rangle_\Omega = \langle \phi; Q \rangle_\Gamma \quad \text{for } Q \in H^1(\Omega).$$

If we plug-in the constant function $Q \equiv 1$, we see that

$$0 = \langle \phi; 1 \rangle_\Gamma \tag{1.9}$$

has to be fulfilled to allow weak solutions of (1.8). Moreover, solutions can only be unique up to additive constants, i.e., if P is a solution of (1.8) and $\alpha \in \mathbb{R}$, then $P+\alpha$ is also a solution of (1.8). To fix additive constants, we use the Sobolev space

$$H^1_\star(\Omega) := \left\{ v \in H^1(\Omega) \,:\, \langle v; 1\rangle_\Gamma = 0 \right\}.$$

It is well known, that there exists a unique solution $P \in H^1_\star(\Omega)$ of (1.8), if $\phi \in H^{-1/2}(\Gamma)$ satisfies (1.9), see, e.g., [Ste08, Theorem 4.9].
The Neumann problem (1.8) is equivalent to the hypersingular integral equation (1.2) in the following sense. If $P \in H^1(\Omega)$ solves (1.8), the trace $u := P|_\Gamma$ solves (1.2). Conversely, if $u \in H^{1/2}(\Gamma)$ solves (1.2), the function $P(x) := \int_\Gamma G(x,y)\phi(y)\,ds_y - \int_\Gamma \partial_{n_y} G(x,y)u(y)\,ds_y$ solves the Neumann problem (1.8), where $G(x,y)$ is the fundamental solution of the Laplacian (1.3), see [Pra07, Theorem 4.13].
However, this equivalence does not state the unique solvability of the hypersingular integral equation (1.2).

We assume $\phi \in H^{-1/2}_\star(\Gamma) := \left\{ v \in H^{-1/2}(\Gamma) \,:\, \langle \psi; 1\rangle_\Gamma = 0 \right\}$ due to (1.9). It holds that $\langle Wv; 1\rangle_\Gamma = 0$ for all $v \in H^{1/2}(\Gamma)$, since W is symmetric and the constant functions are in the kernel of W, see [Pra07, Proposition 4.15]. Furthermore, from [Pra07, Proposition 4.15], it follows $\langle (1/2 - K')\phi; 1\rangle_\Gamma = 0$ for all $\phi \in H^{-1/2}_\star(\Gamma)$.
Under the constraint, that the solution u of (1.2) is in $H^{1/2}_\star(\Gamma)$, the hyper-singular integral equation is recast into the variational formulation: Find $u \in H^{1/2}_\star(\Gamma)$ such that

$$\langle\!\langle u; v\rangle\!\rangle_W = \langle (1/2 - K')\phi; v\rangle_\Gamma \quad \text{for all } v \in H^{1/2}_\star(\Gamma). \tag{1.10}$$

Thereof, we can deduce the variational formulation: Find $u \in H^{1/2}(\Gamma)$ such that

$$\langle\!\langle u; v\rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi; v\rangle_\Gamma \quad \text{for all } v \in H^{1/2}(\Gamma). \tag{1.11}$$

According to the Riesz theorem, the equation (1.10) admits a unique solution $u \in H^{1/2}_\star(\Gamma)$.
On the other hand, if we look at the formulation (1.11), we also get from the Riesz theorem a unique solution $u \in H^{1/2}(\Gamma)$. From

$$|\Gamma|\langle u; 1\rangle_\Gamma = \underbrace{\langle Wu; 1\rangle_\Gamma}_{=0} + \langle 1; 1\rangle_\Gamma \langle u; 1\rangle_\Gamma = \langle\!\langle u; 1\rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi; 1\rangle_\Gamma = 0,$$

it even follows that $u \in H^{1/2}_\star(\Gamma)$. Therefore, both variational formulations are equivalent.

To approximate u, we use a boundary element method. This means that we replace the space $H^{1/2}(\Gamma)$ in (1.11) by a finite dimensional subspace $\mathcal{X}_\star$ and look for a solution $U_\star \in \mathcal{X}_\star$. Hence, we consider the discrete variational formulation: Find $U_\star \in \mathcal{X}_\star$ such that

$$\langle\!\langle U_\star; V_\star\rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi; V_\star\rangle_\Gamma \quad \text{for all } V_\star \in \mathcal{X}_\star, \tag{1.12}$$

which can be solved numerically.

2 Discretization

In this section, we introduce B-splines as well as NURBS and deduce some basic properties. Additionally, the discretization of the boundary Γ is discussed and the ansatz spaces are defined.

2.1 B-splines

Consider *knots* $\check{\mathcal{K}} := (t_i)_{i\in\mathbb{Z}}$ on $\mathbb{R}$ such that $t_{i-1} \le t_i$ for $i \in \mathbb{Z}$ and $\lim_{i\to\pm\infty} t_i = \pm\infty$. The multiplicity of any knot t_i is denoted by $\#t_i$. We define the corresponding set of *nodes* $\check{\mathcal{N}} := \{t_i : i \in \mathbb{Z}\} = \{\check{z}_j : j \in \mathbb{Z}\}$ with $\check{z}_{j-1} < \check{z}_j$ for $j \in \mathbb{Z}$. For $i \in \mathbb{Z}$, the i-th *B-spline* of degree p is defined inductively by

$$\begin{aligned} B_{i,0} &:= \chi_{[t_{i-1},t_i)}, \\ B_{i,p} &:= \beta_{i-1,p} B_{i,p-1} + (1-\beta_{i,p}) B_{i+1,p-1} \quad \text{for } p \in \mathbb{N}, \end{aligned} \tag{2.1}$$

where, for $t \in \mathbb{R}$,

$$\beta_{i,p}(t) := \begin{cases} \frac{t-t_i}{t_{i+p}-t_i} & \text{if } t_i \neq t_{i+p} \\ 0 & \text{if } t_i = t_{i+p} \end{cases}$$

To stress the dependence on the knots $\check{\mathcal{K}}$, also the notation $B_{i,p}^{\check{\mathcal{K}}} := B_{i,p}$ and $\beta_{i,p}^{\check{\mathcal{K}}} := \beta_{i,p}$ is used.

Lemma 2.1 *Let $I = [a,b)$ be a finite interval and $p \in \mathbb{N}_0$. Then, the following assertions* (i)–(vi) *hold:*

(i) *The set $\{B_{i,p}|_I : i \in \mathbb{Z}, B_{i,p}|_I \neq 0\}$ is a basis for the space of all right-continuous $\check{\mathcal{N}}$-piecewise polynomials of degree lower or equal p on I with break points $\check{\mathcal{N}} \cap (a,b)$ and which are, at each break point t_i, $p - \#t_i$ times continuously differentiable if $p - \#t_i \ge 0$.*

(ii) *For $i \in \mathbb{Z}$, the i-th B-spline $B_{i,p}$ of degree p vanishes outside the interval $[t_{i-1}, t_{i+p})$. It is positive on the open interval (t_{i-1}, t_{i+p}).*

(iii) *For $i \in \mathbb{Z}$, the i-th B-spline $B_{i,p}$ of degree p is completely determined by the $p+2$ knots $t_{i-1}, \dots, t_{i+p}$.*

(iv) *The B-splines of degree p form a (locally finite) partition of unity, i.e.,*

$$\sum_{i\in\mathbb{Z}} B_{i,p} = 1 \quad \text{on } \mathbb{R}.$$

(v) *Let $i \in \mathbb{Z}$. For $z = t_{i-p} = \dots = t_i < t_{i+1}$, it holds $B_{i+1-p,p}(z) = 1$.*

(vi) *Let $i \in \mathbb{Z}$. For $t_{i-p-1} < z = t_{i-p} = \dots = t_i$, it holds $\lim_{t\to z-} B_{i-p,p}(t) = 1$.*

Proof. The proof of (i) is found in [de 86, Theorem 6], and (ii)–(iii) are proved in [de 86, Section 2]. (iv) is proved in [de 86, page 9– 10].
For (v) and (vi), we use induction on p.

(v) We show $B_{i+1-p,p}(z) = 1$ by induction on p. For $p = 0$ and $z = t_i < t_{i+1}$, it follows

$$B_{i+1,0}(z) = \chi_{[t_i,t_{i+1})}(z) = \chi_{[t_i,t_{i+1})}(t_i) = 1.$$

In the induction step, we suppose that the claim is true up to $p-1$, i.e.,

$$z = t_{i-(p-1)} = \dots = t_i < t_{i+1} \quad \Longrightarrow \quad B_{i+1-(p-1),p-1}(z) = 1. \tag{2.2}$$

Then, we need to prove the statement for p using (2.2). Let $z = t_{i-p} = \dots = t_i < t_{i+1}$. Then, definition (2.1) shows

$$B_{i+1-p,p}(z) = \beta_{i-p,p}(z) B_{i+1-p,p-1}(z) + (1-\beta_{i+1-p,p}(z)) B_{i+1-(p-1),p-1}(z).$$

If we take a look at the specific factors, we see

$$\beta_{i-p,p}(z) = 0, \quad \text{since } t_{i-p} = t_i,$$
$$\beta_{i+1-p,p}(z) = \frac{z - t_{i+1-p}}{t_{i+1} - t_{i+1-p}} = 0.$$

From the induction hypothesis (2.2), we derive

$$B_{i+1-(p-1),p-1}(z) = 1.$$

Hence, there holds

$$B_{i+1-p,p}(z) = 1,$$

which concludes the induction step.

(vi) We show $\lim_{t\to z-} B_{i-p,p}(t) = 1$ by induction on p. Let $p = 0$ and $t_{i-1} < z = t_i$. Then,

$$\lim_{t\to z-} B_{i,0}(t) = \lim_{t\to z-} \chi_{[t_{i-1},t_i)}(t) = 1.$$

In the induction step, we suppose that the claim is true up to $p-1$, i.e.,

$$t_{i-p} < z = t_{i-(p-1)} = \cdots = t_i \quad \Longrightarrow \quad \lim_{t\to z-} B_{i-(p-1),p-1)}(t) = 1. \tag{2.3}$$

Let $t_{i-p-1} < z = t_{i-p} = \cdots = t_i$. Via the definition of B-splines (2.1), it follows, if the limits exist,

$$\begin{aligned}\lim_{t\to z-} \beta_{i-p-1,p}(t) \lim_{t\to z-} B_{i-p,p-1}(t) + \lim_{t\to z-} (1 - \beta_{i-p,p}(t)) \lim_{t\to z-} B_{i-(p-1),p-1}(t)\\ = \lim_{t\to z-} \big(\beta_{i-p-1,p}(t) B_{i-p,p-1}(t) + (1 - \beta_{i-p,p}(t)) B_{i-(p-1),p-1}(t)\big)\\ = \lim_{t\to z-} B_{i-p,p}(t).\end{aligned}$$

As before, we consider the individual terms,

$$\lim_{t\to z-} \beta_{i-p-1,p}(t) = \lim_{t\to z-} \frac{t - t_{i-p-1}}{t_{i-1} - t_{i-p-1}} = 1,$$
$$\lim_{t\to z-} \beta_{i-p,p}(t) = 0 \quad \text{since } t_{i-p} = t_i.$$

According to the induction hypothesis (2.3), it holds that

$$\lim_{t\to z-} B_{i-(p-1),p-1}(t) = 1.$$

Therefore,

$$\lim_{t\to z-} B_{i-p,p}(t) = \lim_{t\to z-} B_{i-p,p-1}(t) + 1.$$

From (ii) and (iv), it follows directly

$$0 \le \lim_{t\to z-} B_{i-p,p}(t) \le 1.$$

Hence, $\lim_{t\to z-} B_{i-p,p-1}(t) = 0$ as well as $\lim_{t\to z-} B_{i-p,p}(t) = 1$, which concludes the proof. □

2.2 NURBS

Additionally to the knots $\check{\mathcal{K}} = (t_i)_{i\in\mathbb{Z}}$, we consider positive weights $\mathcal{W} := (w_i)_{i\in\mathbb{Z}}$ with $w_i > 0$ and define for $i \in \mathbb{Z}$ and $p \in \mathbb{N}_0$ the i-th NURBS by

$$R_{i,p} := \frac{w_i B_{i,p}}{\sum_{\ell\in\mathbb{Z}} w_\ell B_{\ell,p}}. \tag{2.4}$$

We also use the notation $R_{i,p}^{\check{\mathcal{K}},\mathcal{W}} := R_{i,p}$. Note that the denominator is locally finite and positive. In addition, we define for $p \in \mathbb{N}_0$ the B-spline space

$$\mathscr{S}^p(\check{\mathcal{K}}) := \left\{ \sum_{i\in\mathbb{Z}} a_i B_{i,p}^{\check{\mathcal{K}}} \,:\, a_i \in \mathbb{R} \right\} \tag{2.5}$$

as well as the NURBS space

$$\mathscr{N}^p(\check{\mathcal{K}},\mathcal{W}) := \left\{ \sum_{i\in\mathbb{Z}} a_i R_{i,p}^{\check{\mathcal{K}},\mathcal{W}} \,:\, a_i \in \mathbb{R} \right\} = \frac{\mathscr{S}^p(\check{\mathcal{K}})}{\sum_{\ell\in\mathbb{Z}} w_\ell B_{\ell,p}^{\check{\mathcal{K}},\mathcal{W}}}. \tag{2.6}$$

2.3 Boundary discretization

We discuss the different quantities of the discretization of Γ and introduce some further notation.

- **Nodes $z_j = \gamma(\check{z}_j) \in \check{\mathcal{N}}_\star$**
  Let $\mathcal{N}_\star := \{z_j \,:\, j = 1,\dots,n\}$ and $z_0 := z_n$ be a set of nodes. Suppose that $z_j = \gamma(\check{z}_j)$ for some $\check{z}_j \in [a,b]$ with $a = \check{z}_0 < \check{z}_1 < \check{z}_2 < \dots < \check{z}_n = b$ such that $\gamma|_{[\check{z}_{j-1},\check{z}_j]} \in C^1([\check{z}_{j-1},\check{z}_j])$.

- **Multiplicity $\#z_j$ and knots $\mathcal{K}_\star, \check{\mathcal{K}}_\star$**
  Let $p \in \mathbb{N}_0$ be some fixed polynomial order. Each node z_j has a multiplicity $\#z_j \in \{1,2,\dots,p+1\}$ with $\#z_0 = \#z_n = p+1$ and $\#z_j \le p$ for $j = 1,\dots,n-1$. This induces knots

  $$\mathcal{K}_\star = (\underbrace{z_1,\dots,z_1}_{\#z_1 \text{ times}},\dots,\underbrace{z_n,\dots,z_n}_{\#z_n \text{ times}}) \tag{2.7}$$

  with corresponding knots $\check{\mathcal{K}}_\star := \gamma|_{(a,b]}^{-1}(\mathcal{K}_\star)$ on the parameter domain $[a,b]$.

- **Elements T,$[T]$ and partitions $\mathcal{T}_\star$, $[\mathcal{T}_\star]$**
  Let $\mathcal{T}_\star = \{T_1,\dots,T_n\}$ be a partition of Γ into compact and connected segments $T_j = \gamma(\check{T}_j)$ with $\check{T}_j = [\check{z}_{j-1},\check{z}_j]$. We define

  $$[\mathcal{T}_\star] := \{[T] \,:\, T \in \mathcal{T}_\star\} \quad \text{with} \quad [T] := (T, \#z_{T,1}, \#z_{T,2}), \tag{2.8}$$

  where $z_{T,1} = z_{j-1}$ and $z_{T,2} = z_j$ are the two nodes of $T = T_j$.

- **Local mesh-sizes $h_{\star,T}, \check{h}_{\star,T}$ and $h_\star, \check{h}_\star$**
  The arclength of each element $T \in \mathcal{T}_\star$ is denoted by $h_{\star,T}$. We define the local mesh-width function $h_\star \in L^\infty(\Gamma)$ by $h_\star|_T = h_{\star,T}$. Additionally, we define for each element $T \in \mathcal{T}_\star$ its length $\check{h}_\star|_T := |\gamma^{-1}(T)|$ with respect to the parameter domain $[a,b]$. This gives rise to $\check{h}_\star \in L^\infty(\Gamma)$ with $\check{h}_\star|_T = \check{h}_{\star,T}$. Note that the lengths $h_{\star,T}$ and $\check{h}_{\star,T}$ of an element T are equivalent, and the equivalence constants depend only on γ.

- **Local mesh-ratios $\check{\kappa}_\star$**
  The local mesh-ratio is defined by

  $$\check{\kappa}_\star := \max\left\{\check{h}_{\star,T}/\check{h}_{\star,T'} \,:\, T,T' \in \mathcal{T}_\star \text{ with } T \cap T' \neq \emptyset\right\}. \tag{2.9}$$

- **Patches $\omega_\star(z), \omega_\star(\Gamma_0), \omega_\star(\mathcal{E}), [\omega_\star](\mathcal{E})$ and $\bigcup \mathcal{E}$**

  For each set $\Gamma_0 \subseteq \Gamma$ and $m \in \mathbb{N}_0$, we define inductively

  $$\omega_\star^m(\Gamma_0) := \begin{cases} \Gamma_0, & \text{if } m = 0, \\ \omega_\star(\Gamma_0) := \bigcup\{T \in \mathcal{T}_\star \,:\, T \cap \Gamma_0 \neq \emptyset\}, & \text{if } m = 1, \\ \omega_\star(\omega_\star^{m-1}(\Gamma_0)), & \text{if } m > 1. \end{cases} \tag{2.10}$$

For nodes $z \in \Gamma$, we abbreviate $\omega_\star(z) := \omega_\star(\{z\})$.
Analogously, for each set $\mathcal{E} \subseteq [\mathcal{T}_\star]$ and $m \in \mathbb{N}_0$, we define inductively

$$[\omega_\star^m](\mathcal{E}) := \begin{cases} \mathcal{E}, & \text{if } m = 0, \\ [\omega_\star](\mathcal{E}) := \{[T] \in [\mathcal{T}_\star] \,:\, \exists\, [T'] \in \mathcal{E},\ T \cap T' \neq \emptyset\}, & \text{if } m = 1, \\ [\omega_\star]([\omega_\star^{m-1}](\mathcal{E})), & \text{if } m > 1. \end{cases} \tag{2.11}$$

We also need

$$\bigcup \mathcal{E} := \bigcup \{T \in \mathcal{T}_\star \,:\, [T] \in \mathcal{E}\} \subseteq \Gamma \tag{2.12}$$

and

$$\omega_\star^m(\mathcal{E}) := \omega_\star^m\Big(\bigcup \mathcal{E}\Big). \tag{2.13}$$

2.4 Mesh-refinement

Let ref $(\cdot)$ be a given deterministic mesh-refinement strategy such that, for each mesh $[\mathcal{T}_\star]$ and an arbitrary set of marked nodes $\mathcal{M}_\star \subseteq \mathcal{N}_\star$ the refinement $[\mathcal{T}_+] := \text{ref}\,([\mathcal{T}_\star], \mathcal{M}_\star)$ is also a mesh in the same sense as above. Suppose that the marked nodes belong to the union of the refined elements, i.e., $\mathcal{M}_\star \subseteq \bigcup([\mathcal{T}_\star]\backslash[\mathcal{T}_+])$, and that the knots $\mathcal{K}_\star$ form a subsequence of the knots $\mathcal{K}_+$. This implies the following estimate

$$|[\mathcal{T}_\star]\backslash[\mathcal{T}_+]| \leq 2(|\mathcal{K}_+| - |\mathcal{K}_\star|), \tag{2.14}$$

since $[\mathcal{T}_\star]\backslash[\mathcal{T}_+]$ is the set of all elements in which a new knot is inserted and one new knot can be inserted in at most 2 elements of the old mesh, i.e., at the intersection of 2 elements.
We write $[\mathcal{T}_+] \in \text{ref}\,([\mathcal{T}_\star])$, if there exist finitely many meshes $[\mathcal{T}_1], \dots, [\mathcal{T}_\ell]$ and subsets $\mathcal{M}_j \subseteq \mathcal{N}_j$ of the corresponding nodes such that $[\mathcal{T}_\star] = [\mathcal{T}_1], [\mathcal{T}_+] = [\mathcal{T}_\ell]$, and $[\mathcal{T}_j] = \text{ref}\,([\mathcal{T}_{j-1}], \mathcal{M}_{j-1})$ for all $j = 2, \dots, \ell$, where we formally allow $\ell = 1$, i.e., $[\mathcal{T}_\star] \in \text{ref}\,([\mathcal{T}_\star])$.

Later on, we need some assumptions concerning the mesh-refinement strategy, which are as follows:

Assumption 2.2 *Let $[\mathcal{T}_0]$ be an arbitrary initial mesh and $[\mathbb{T}] := \text{ref}\,([\mathcal{T}_0])$ be the set of all possible refinements. Suppose that the mesh-refinement strategy satisfies the properties* (M1)-(M3)*:*

(M1) *There exists a constant $\check{\kappa}_{\max} \geq 1$ such that the local mesh-ratios* (2.9) *are uniformly bounded*

$$\check{\kappa}_\star \leq \check{\kappa}_{\max} \quad \text{for all } [\mathcal{T}_\star] \in [\mathbb{T}]. \tag{2.15}$$

(M2) *There is a common refinement $[\mathcal{T}_\star \oplus \mathcal{T}_+] \in \text{ref}\,([\mathcal{T}_\star]) \cap \text{ref}\,([\mathcal{T}_+])$ for all $[\mathcal{T}_\star], [\mathcal{T}_+] \in [\mathbb{T}]$, such that the knots $\mathcal{K}_\star \oplus \mathcal{K}_+$ of $[\mathcal{T}_\star \oplus \mathcal{T}_+]$ satisfy the overlay estimate*

$$|\mathcal{K}_\star \oplus \mathcal{K}_+| \leq |\mathcal{K}_\star| + |\mathcal{K}_+| - |\mathcal{K}_0|. \tag{2.16}$$

(M3) *There exists a constant $C_{\text{mesh}} > 0$ which depends only on $[\mathcal{T}_0]$, such that for each sequence $([\mathcal{T}_\ell])_{\ell \in \mathbb{N}_0}$ of meshes generated by successive mesh-refinement, i.e., $[\mathcal{T}_j] = \text{ref}\,([\mathcal{T}_{j-1}], \mathcal{M}_{j-1})$ for all $j \in \mathbb{N}$ and arbitrary $\mathcal{M}_j \subseteq \mathcal{N}_j$, the following closure estimate holds:*

$$|\mathcal{K}_\ell| - |\mathcal{K}_0| \leq C_{\text{mesh}} \sum_{j=0}^{\ell-1} |\mathcal{M}_j| \quad \text{for all } \ell \in \mathbb{N}. \tag{2.17}$$

In the implementation of the adaptive IGABEM Algorithm 3.5, one can use either pure h-refinement based on local biscection or a strategy, which also takes the multiplicity of the marked nodes into account and will be specified below.
The h-refinement strategy is rather simple: Let $[\mathcal{T}_\star] \in [\mathbb{T}]$ and let $\mathcal{M}_\star \subseteq \mathcal{N}_\star$ be a set of marked nodes. The refined mesh $[\mathcal{T}_+] := \text{ref}\,([\mathcal{T}_\star], \mathcal{M}_\star)$, is obtained the following way:

INPUT: mesh $[\mathcal{T}_\star]$, marked nodes $\mathcal{M}_\star$.

(i) Mark every element $T \in \mathcal{T}_\star$ such that $T \cap \mathcal{M}_\star \neq \emptyset$.

(ii) Recursively, mark further elements $T' \in \mathcal{T}_\star$, if there exists a marked element $T \in \mathcal{T}_\star$ with $T \cap T' \neq \emptyset$ and $\check{h}_{\star,T'} > 2\check{\kappa}_0 \check{h}_{\star,T}$.

(iii) Refine all marked elements $T \in \mathcal{T}_\star$ via bisection in the parameter domain by insertion of a knot of multiplicity one, and obtain $[\mathcal{T}_+]$.

OUTPUT: refined mesh $[\mathcal{T}_+]$.

The second strategy used differs from the first one and also considers the node multiplicity:

INPUT: mesh $[\mathcal{T}_\star]$, marked nodes $\mathcal{M}_\star$.

(i) Consider each element $T \in \mathcal{T}_\star$. If both nodes of T belong to $\mathcal{M}_\star$, the element T is marked.

(ii) Define $\mathcal{M}_\star^0 := \{T' \cap \mathcal{N}_\star \,:\, T' \in \mathcal{T}_\star, T' \cap \mathcal{N}_\star \subseteq \mathcal{M}_\star\}$. For all nodes $z \in \big\{z' \in \mathcal{M}_\star \setminus \mathcal{M}_\star^0 \,:\, \#z' < p\big\}$, increase the multiplicity by one. For $z \in \big\{z' \in \mathcal{M}_\star \setminus \mathcal{M}_\star^0 \,:\, \#z' = p\big\}$, mark all elements $T \in \mathcal{T}_\star$ with $z \in T$.

(iii) Recursively, further elements $T' \in \mathcal{T}_\star$ will be marked if there exists a marked element $T \in \mathcal{T}_\star$ with $T \cap T' \neq \emptyset$ and $\check{h}_{\star,T'} > 2\check{\kappa}_0 \check{h}_{\star,T}$.

(iv) Refine all marked elements $T \in \mathcal{T}_\star$ via bisection in the parameter domain and obtain $[\mathcal{T}_+]$.

OUTPUT: refined mesh $[\mathcal{T}_+]$.

According to [AFF+13], the first strategy satisfies the assumptions (M1)–(M3). [FGP, Proposition 2.2] shows that (M1)–(M3) also hold for the second strategy.

For the proof of the main result, i.e., the optimality of the proposed Algorithm 3.5, we use some auxiliary error estimator $\widetilde{\rho}$, which relies on an equivalent mesh-size function $\widetilde{h}$ from [FGHP16b, Proposition 4.2], which also takes the knot multiplicities into account.

Proposition 2.3 *Under the assumption* (M1) *there exists a modified mesh-size function* $\widetilde{h}\colon [\mathbb{T}] \to L^\infty(\Gamma)$ *such that* $\widetilde{h}([\mathcal{T}_\star])$ *is* $\mathcal{T}_\star$*-piecewise constant for* $[\mathcal{T}_\star] \in [\mathbb{T}]$ *with the following properties: There exist constants* $C_{\mathrm{wt}} > 0$ *and* $0 < q_{\mathrm{ctr}} < 1$ *such that for all* $[\mathcal{T}_\star] \in [\mathbb{T}]$ *and all refinements* $[\mathcal{T}_+] \in \operatorname{ref}([\mathcal{T}_\star])$*, the corresponding mesh-sizes* $\widetilde{h}_\star := \widetilde{h}([\mathcal{T}_\star])$ *and* $\widetilde{h}_+ := \widetilde{h}([\mathcal{T}_+])$ *satisfy equivalence*

$$C_{\mathrm{wt}}^{-1}\check{h}_\star \le \widetilde{h}_\star \le C_{\mathrm{wt}}\check{h}_\star \quad \text{resp.} \quad C_{\mathrm{wt}}^{-1}\check{h}_+ \le \widetilde{h}_+ \le C_{\mathrm{wt}}\check{h}_+, \tag{2.18}$$

as well as reduction,

$$\widetilde{h}_+ \le \widetilde{h}_\star, \tag{2.19}$$

contraction on the patch of refined elements,

$$\widetilde{h}_+|_{\omega_+([\mathcal{T}_+]\setminus[\mathcal{T}_\star])} \le q_{\mathrm{ctr}}\, \widetilde{h}_\star|_{\omega_+([\mathcal{T}_+]\setminus[\mathcal{T}_\star])}, \tag{2.20}$$

and equality on the non-contracting part,

$$\widetilde{h}_+|_{\Gamma\setminus\omega_+([\mathcal{T}_+]\setminus[\mathcal{T}_\star])} = \widetilde{h}_\star|_{\Gamma\setminus\omega_\star([\mathcal{T}_\star]\setminus[\mathcal{T}_+])}. \tag{2.21}$$

The constants $C_{\mathrm{wt}} > 0$ *and* $0 < q_{\mathrm{ctr}} < 1$ *depend only on* $\check{\kappa}_{\max}, p$*, and* γ*.*

Proof. For all $[\mathcal{T}_\star] \in [\mathbb{T}]$, the modified mesh-size $\widetilde{h}_\star \in L^\infty(\Gamma)$ is defined as

$$\widetilde{h}|_T = |\gamma^{-1}(\omega_\star(T))| \cdot q_1^{\sum_{z\in\mathcal{N}_\star\cap\omega_\star(T)} \#z} \quad \text{for all } T \in \mathcal{T}_\star,$$

with a suitable constant $0 < q_1 < 1$. The proof of the properties can be found in [FGHP16b, Proposition 4.2]. □

2.5 Discretization spaces

Let $[\mathcal{T}_0]$ be a given initial mesh with corresponding knots $\mathcal{K}_0$ such that for each $\check{z}_j \in \check{\mathcal{N}}_0$ it holds $\#\check{z}_j \leq p$ for $j = 1, \dots, n-1$ and $\#\check{z}_0 = \#\check{z}_n = p+1$. Suppose that $\mathcal{W}_0 = (w_i)_{i=1-p}^{N-p}$ are given initial weights with $N = |\mathcal{K}_0|$.
We extend the transformed knot sequence $\check{\mathcal{K}}_0 = (t_i)_{i=1}^N$ arbitrarily to $(t_i)_{i\in\mathbb{Z}}$ with $t_{-p} = \cdots = t_0 = a$, $t_i \leq t_{i+1}$, $\lim_{i\to\pm\infty} t_i = \pm\infty$ and $\mathcal{W}_0 = (w_i)_{i\in\mathbb{Z}}$ with $w_i > 0$ and assume $w_{1-p} = w_{N-p}$. Additionally, we define $w_{\min} := \min(\mathcal{W}_0)$ and $w_{\max} := \max(\mathcal{W}_0)$. For the extended sequences, we also write $\check{\mathcal{K}}_0$ and $\mathcal{W}_0$ and set

$$\begin{aligned}\mathcal{X}_0 &:= \left\{ \sum_{i\in\mathbb{Z}} a_i R_{i,p}|_{[a,b)} \circ \gamma|_{[a,b)}^{-1} \,:\, a_i \in \mathbb{R} \text{ and } a_{1-p} = a_{N-p} \right\} \\ &= \left\{ \sum_{i\in\mathbb{Z}} a_i R_{i,p}|_{[a,b)} \circ \gamma|_{[a,b)}^{-1} \,:\, a_i \in \mathbb{R} \right\} \cap C(\Gamma)\end{aligned} \tag{2.22}$$

The equality in (2.22) holds, because for $j = 1, \dots, n-1$ the multiplicity satisfies $\#\check{z}_j \leq p$ and therefore with Lemma 2.1 (i) we get at least continuity at each $\gamma(t_j)$. In combination with $a_{1-p} = a_{N-p}$ and $w_{1-p} = w_{N-p}$, Lemma 2.1 (v) shows the continuity at $\gamma(t_0) = \gamma(t_N)$.
We note that $\mathcal{X}_0 \subset H^1(\Gamma) \subset H^{1/2}(\Gamma)$ and that Lemma 2.1 implies that the set

$$\left\{ R_{i,p}|_{[a,b)} \circ \gamma|_{[a,b)}^{-1} \,:\, i = 2-p, \dots, N-p-1 \right\} \cup \left\{ \big(R_{1-p,p}|_{[a,b)} + R_{N-p,p}|_{[a,b)} \big) \circ \gamma|_{[a,b)}^{-1} \right\} \tag{2.23}$$

is a basis of $\mathcal{X}_0$. Due to Lemma 2.1, this basis does not depend on how the sequences $\check{\mathcal{K}}_0$ and $\mathcal{W}_0$ are extended.
Let $[\mathcal{T}_\star] \in [\mathbb{T}]$ be a mesh with knots $\mathcal{K}_\star$. Via *knot insertion* from $\mathcal{K}_0$ to $\mathcal{K}_\star$, one obtains unique corresponding weights $\mathcal{W}_\star$. These are chosen such that the denominators of the NURBS functions do not change. This implies nestedness

$$\mathcal{X}_\star \subseteq \mathcal{X}_+ \quad \text{for all } [\mathcal{T}_\star] \in [\mathbb{T}] \text{ and all } [\mathcal{T}_+] \in \operatorname{ref}(\mathcal{T}_\star),$$

where the spaces $\mathcal{X}_\star$ resp. $\mathcal{X}_+$ are defined analogously to (2.22). A basis of $\mathcal{X}_\star$ is given similarly as in (2.23). For further details, we refer, e.g., to [FGP15, Section 4.2].

To discretize the right-hand side ϕ of (1.2), we consider the space

$$\mathcal{P}^p(\mathcal{T}_\star) := \left\{ \check{\Psi}_\star \circ \gamma|_{[a,b)}^{-1} \,:\, \check{\Psi}_\star \in \mathcal{P}^p(\check{\mathcal{T}}_\star) \right\}, \tag{2.24}$$

where $\mathcal{P}^p(\check{\mathcal{T}}_\star)$ denotes the set of all $\check{\mathcal{N}}_\star$-piecewise polynomials in the parameter domain. We define $\check{\mathcal{K}}_{\star,p+1}$ as the knots, which are obtained from $\check{\mathcal{K}}_\star$ by increasing all multiplicities to $p+1$. With this definition, Lemma 2.1 provides a basis of $\mathcal{P}^p(\mathcal{T}_\star)$

$$\left\{ B_{i,p}^{\check{\mathcal{K}}_{\star,p+1}}|_{[a,b)} \circ \gamma|_{[a,b)}^{-1} \,:\, i = 1-p, \dots, N_{p+1}-p \right\}, \tag{2.25}$$

where N_{p+1} denotes the number of knots $\check{\mathcal{K}}_{\star,p+1}$ in $(a,b]$.

3 Error estimators and adaptive algorithm

Throughout, for given Neumann data ϕ with $\langle \phi; 1\rangle_\Gamma = 0$ and additional regularity $\phi \in L^2(\Gamma)$, let $u \in H^{1/2}(\Gamma)$ and $U_\star \in \mathcal{X}_\star$ be the unique solutions to

$$\langle\!\langle u; v\rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi; v\rangle_\Gamma \qquad \text{for all } v \in H^{1/2}(\Gamma), \tag{3.1}$$

$$\langle\!\langle U_\star; V_\star\rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi_\star; V_\star\rangle_\Gamma \qquad \text{for all } V_\star \in \mathcal{X}_\star, \tag{3.2}$$

where $\phi_\star$ is optionally defined either as $\phi_\star := \phi$ or $\phi_\star := \Pi_\star \phi$ for all $[\mathcal{T}_\star] \in [\mathbb{T}]$ with the L^2-orthogonal projection $\Pi_\star \colon L^2(\Gamma) \to \mathcal{P}^p(\mathcal{T}_\star)$ onto $\mathcal{P}^p(\mathcal{T}_\star)$. The two possibilites are considered due to numerical reasons, which will become clear in Section 6, see Remark 6.2.

3.1 Error estimators

Note that $(1/2 - K')\phi_\star$ is in $L^2(\Gamma)$ due to $\phi \in L^2(\Gamma)$, wherefore the following estimators are well-defined. We define the node-based error estimator

$$\mu_\star(z) := \|h_\star^{1/2}\big((1/2 - K')\phi_\star - WU_\star\big)\|_{L^2(\omega_\star(z))} + \|h_\star^{1/2}(\phi - \phi_\star)\|_{L^2(\omega_\star(z))} \quad \text{for all } z \in \mathcal{N}_\star, \tag{3.3}$$

and

$$\mu_\star := \mu_\star(\mathcal{N}_\star) \quad \text{with} \quad \mu_\star^2(\mathcal{E}_\star) := \sum_{z \in \mathcal{E}_\star} \mu_\star(z)^2 \quad \text{for all } \mathcal{E}_\star \subseteq \mathcal{N}_\star. \tag{3.4}$$

Additionally, we define the auxiliary elementwise error estimator with local contributions

$$\widetilde{\rho}_\star([T]) := \|\widetilde{h}_\star^{1/2}\big((1/2 - K')\phi_\star - WU_\star\big)\|_{L^2(T)} + \|\widetilde{h}_\star^{1/2}(\phi - \phi_\star)\|_{L^2(T)} \quad \text{for all } [T] \in [\mathcal{T}_\star], \tag{3.5}$$

and

$$\widetilde{\rho}_\star := \widetilde{\rho}_\star([\mathcal{T}_\star]) \quad \text{with} \quad \widetilde{\rho}_\star([\mathcal{E}_\star])^2 := \sum_{[T] \in [\mathcal{E}_\star]} \widetilde{\rho}_\star([T])^2 \quad \text{for all} \quad [\mathcal{E}_\star] \subseteq [\mathcal{T}_\star]. \tag{3.6}$$

As a direct consequence of Proposition 2.3 and the equivalence $h_\star \simeq \check{h}_\star$, there exists a constant $C_{\rm loc} > 0$ such that

$$C_{\rm loc}^{-1}\, \widetilde{\rho}_\star([T])^2 \le \mu_\star(z)^2 \le C_{\rm loc} \sum_{\substack{T' \in \mathcal{T}_\star \\ z \in T'}} \widetilde{\rho}_\star([T'])^2 \quad \text{for all } z \in \mathcal{N}_\star \text{ and } T \in \mathcal{T}_\star \text{ with } z \in T, \tag{3.7}$$

as well as,

$$C_{\rm loc}^{-1}\, \widetilde{\rho}_\star^2 \le \mu_\star^2 \le C_{\rm loc}\, \widetilde{\rho}_\star^2. \tag{3.8}$$

The constant $C_{\rm loc} > 0$ depends only on $\check{\kappa}_{\max}, p$, and γ.

3.2 Auxiliary results

Before we turn towards the properties of the auxiliary estimator $\widetilde{\rho}_\star$, we collect some auxiliary results, which will be needed later.

3.2.1 Scott-Zhang type operator $J_\star$

To begin with, we introduce a Scott-Zhang type operator $J_\star$ for $[\mathcal{T}_\star] \in [\mathbb{T}]$.

It holds $R_{i,p} = w_i B_{i,p}/w$, where $w = \sum_{\ell \in \mathbb{Z}} w_\ell B_{\ell,p}$ is the fixed denominator from Section 2.5. In [BdVBSV14, Section 2.1.5], it is shown that, for $i \in \{1-p, \dots, N-p\}$, there exist dual basis functions $B_{i,p}^* \in L^2([a,b])$ with $\operatorname{supp} B_{i,p}^* = \operatorname{supp} B_{i,p}$ and

$$\int_{[a,b]} B_{i,p}^*(t) B_{j,p}(t)\, dt = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

and

$$\|B^*_{i,p}\|_{L^2(\mathrm{supp} B_{i,p})} \le (2p+3)9^p |\mathrm{supp} B_{i,p}|^{-1/2}. \tag{3.9}$$

Define $R^*_{i,p} := B^*_{i,p} w/w_i$ with the denominator w from before, and $\widehat{R}_{i,p} := R_{i,p}|_{[a,b)} \circ \gamma|^{-1}_{[a,b)}$. For $[\mathcal{T}_\star] \in [\mathbb{T}]$, we define the following Scott-Zhang type operator

$$\begin{aligned} &J_\star \colon L^2(\Gamma) \to \mathcal{X}_\star, \\ &J_\star(v) := \sum_{i \in \{1-p,\dots,N-p-1\}} \left(\int_{[a,b]} R^*_{i,p}(t) v(\gamma(t))\, dt \right) \widehat{R}_{i,p} + \left(\int_{[a,b]} R^*_{1-p,p}(t) v(\gamma(t))\, dt \right) \widehat{R}_{N-p,p}. \end{aligned} \tag{3.10}$$

Lemma 3.1 *Let $[\mathcal{T}_\star] \in [\mathbb{T}]$. The Scott-Zhang type operator* (3.10) *satisfies the following properties:*

(i) *Local projection property: For $T \in \mathcal{T}_\star$ and $v \in L^2(\Gamma)$, the property $v|_{\omega^p_\star(T)} \in \mathcal{X}_\star|_{\omega^p_\star(T)} := \{\xi|_{\omega^p_\star(T)} : \xi \in \mathcal{X}_\star\}$ implies $v|_T = (J_\star v)|_T$.*

(ii) *Local L^2-stability: There exists a constant $C_{\mathrm{sz}} > 0$ such that for $v \in L^2(\Gamma)$ and $T \in \mathcal{T}_\star$, there holds*

$$\|J_\star v\|_{L^2(T)} \le C_{\mathrm{sz}} \|v\|_{L^2(\omega^p_\star(T))}. \tag{3.11}$$

The constant $C_{\mathrm{sz}} > 0$ depends only on $\check{\kappa}_{\max}, p, w_{\max}$, and γ.

(iii) *Local approximation properties: There exists a constant $C_{\mathrm{sz}} > 0$ such that for all $v \in H^{1/2}(\Gamma)$*

$$\|h_\star^{-1/2}(1 - J_\star)v\|_{L^2(\Gamma)} \le C_{\mathrm{sz}} |v|_{H^{1/2}(\Gamma)}, \tag{3.12}$$

and if v is even in $H^1(\Gamma)$

$$\|h_\star^{-1+\sigma}(1 - J_\star)v\|_{L^2(\Gamma)} \le C_{\mathrm{sz}} \|h_\star^{\sigma} \nabla v\|_{L^2(\Gamma)} \quad \text{for all } \sigma \in \mathbb{R}, \tag{3.13}$$

as well as,

$$|(1 - J_\star)v|_{H^{1/2}(\Gamma)} \le C_{\mathrm{sz}} \|h_\star^{1/2} \nabla v\|_{L^2(\Gamma)}. \tag{3.14}$$

The constant $\widetilde{C}_{\mathrm{sz}} > 0$ depends only on $\check{\kappa}_{\max}, p, w_{\max}$, and γ.

Proof. Let $I := \{1-p, \dots, N-p-1\}$ throughout the proof.

(i) As $v|_{\omega^p_\star(T)} \in \mathcal{X}_\star|_{\omega^p_\star(T)}$, it can be rewritten in terms of the NURBS-basis with coefficients $a_j \in \mathbb{R}$

$$v|_{\omega^p_\star(T)} = \sum_{j \in \{2-p,\dots,N-p-1\}} a_j \widehat{R}_{j,p}|_{\omega^p_\star(T)} + a_{1-p}(\widehat{R}_{1-p,p} + \widehat{R}_{N-p,p})|_{\omega^p_\star(T)}.$$

Therefore, it holds for $i \in I$ with $\mathrm{supp}\, \widehat{R}_{i,p} \subseteq \omega^p_\star(T)$

$$\begin{aligned} \int_{[a,b]} R^*_{i,p}(t) v(\gamma(t))\, dt =& \sum_{j \in \{2-p,\dots,N-p-1\}} a_j \int_{[a,b]} R^*_{i,p}(t) R_{j,p}(t)\, dt + \\ & a_{1-p} \int_{[a,b]} R^*_{i,p}(t)(R_{1-p,p} + R_{N-p,p})(t)\, dt \\ =& \sum_{j \in \{2-p,\dots,N-p-1\}} a_j \int_{[a,b]} \frac{B^*_{i,p}(t) w(t)}{w_i} \frac{B_{j,p}(t) w_j}{w(t)}\, dt + \\ & a_{1-p} \int_{[a,b]} \frac{B^*_{i,p}(t) w(t)}{w_i} \left(\frac{B_{1-p,p}(t) w_{1-p}}{w(t)} + \frac{B_{N-p,p}(t) w_{N-p}}{w(t)} \right) dt \\ =& a_i. \end{aligned}$$

If $\widehat{R}_{i,p}|_T \neq 0$, it follows $\operatorname{supp} \widehat{R}_{i,p} \subseteq \omega_\star^p(T)$. Together with the preceding identity, this implies

$$\begin{aligned}(J_\star v)|_T =& \sum_{i\in I} \left(\int_{[a,b]} R_{i,p}^*(t) v(\gamma(t))\, dt \right) \widehat{R}_{i,p}|_T + \left(\int_{[a,b]} R_{1-p,p}^*(t) v(\gamma(t))\, dt \right) \widehat{R}_{N-p,p}|_T \\ =& \sum_{i\in I} a_i \widehat{R}_{i,p}|_T + a_{1-p} \widehat{R}_{N-p,p}|_T \\ =& v|_T\end{aligned}$$

and concludes the proof of (i).

(ii) Recall $\operatorname{supp} R_{i,p}^\star = \operatorname{supp} R_{i,p}$ and $\widehat{R}_{i,p} = R_{i,p}|_{[a,b)} \circ \gamma|_{[a,b)}^{-1}$. We use $0 \le \widehat{R}_{i,p} \le 1$ and (3.9) to see

$$\begin{aligned}\|J_\star v\|_{L^2(T)} =& \Bigg\| \sum_{i\in I} \left(\int_{[a,b]} R_{i,p}^*(t) v(\gamma(t))\, dt \right) \widehat{R}_{i,p} \\ &+ \left(\int_{[a,b]} R_{1-p,p}^*(t) v(\gamma(t))\, dt \right) \widehat{R}_{N-p,p} \Bigg\|_{L^2(T)} \\ \le& \sum_{\substack{i\in I \\ |\operatorname{supp}\widehat{R}_{i,p}\cap T|>0}} \left| \int_{[a,b]} R_{i,p}^*(t) v(\gamma(t))\, dt \right| \|\widehat{R}_{i,p}\|_{L^2(T)} \\ &+ \chi_{(0,\infty)}(|T\cap \operatorname{supp}\widehat{R}_{N-p,p}|) \left| \int_{[a,b]} R_{1-p,p}^*(t) v(\gamma(t))\, dt \right| \|\widehat{R}_{N-p,p}\|_{L^2(T)} \\ \lesssim& \sum_{\substack{i\in I \\ |\operatorname{supp}\widehat{R}_{i,p}\cap T|>0}} \|R_{i,p}^*\|_{L^2(\operatorname{supp} R_{i,p})} \|v\|_{L^2(\operatorname{supp}\widehat{R}_{i,p})} h_{\star,T}^{1/2} \\ \overset{(3.9)}{\lesssim}& \sum_{\substack{i\in I \\ |\operatorname{supp}\widehat{R}_{i,p}\cap T|>0}} \|v\|_{L^2(\operatorname{supp}\widehat{R}_{i,p})} \lesssim \|v\|_{L^2(\omega_\star^p(T))}.\end{aligned}$$

The hidden constants depend only on $\check{\kappa}_{\max}, p, w_{\max}$, and γ.

(iii) Let $T \in \mathcal{T}_\star$. We define $v_{\omega_\star^p(T)} := \frac{1}{|\omega_\star^p(T)|} \int_{\omega_\star^p(T)} v(x)\, ds_x$. It holds

$$\begin{aligned}\|(1-J_\star)v\|_{L^2(T)} &= \|v - J_\star v\|_{L^2(T)} \\ &\overset{\text{(i)}}{=} \|v - v_{\omega_\star^p(T)} - J_\star(v - v_{\omega_\star^p(T)})\|_{L^2(T)} \\ &\le \|v - v_{\omega_\star^p(T)}\|_{L^2(T)} + \underbrace{\|J_\star(v - v_{\omega_\star^p(T)})\|_{L^2(T)}}_{\overset{(3.11)}{\lesssim} \|v - v_{\omega_\star^p(T)}\|_{L^2(\omega_\star^p(T))}} \\ &\lesssim \|v - v_{\omega_\star^p(T)}\|_{L^2(\omega_\star^p(T))}\end{aligned} \tag{3.15}$$

Let $\sigma \in (0,1]$ and $w \in H^\sigma(\Gamma_0)$ for $\Gamma_0 \subseteq \Gamma$. Then, [Fae00, Lemma 2.5] states that the following inequality holds

$$\|w\|_{L^2(\Gamma_0)}^2 \le \frac{1}{2} |\Gamma_0|^{2\sigma} |w|_{H^\sigma(\Gamma_0)}^2 + \frac{1}{|\Gamma_0|} \left| \int_{\Gamma_0} w\, ds \right|^2. \tag{3.16}$$

Hence, for $\sigma = 1/2$, it follows from (3.15) and (3.16)

$$\begin{aligned}\|(1-J_\star)v\|_{L^2(T)}^2 &\lesssim \frac{1}{2} \underbrace{|\omega_\star^p(T)|}_{\simeq h_{\star,T}} \underbrace{|v - v_{\omega_\star^p(T)}|_{H^{1/2}(\omega_\star^p(T))}^2}_{\overset{\text{def.}}{=} |v|_{H^{1/2}(\omega_\star^p(T))}^2} + \frac{1}{|\omega_\star^p(T)|} \underbrace{\left| \int_{\omega_\star^p(T)} v - v_{\omega_\star^p(T)}\, ds \right|^2}_{=0} \\ &\lesssim h_{\star,T} |v|_{H^{1/2}(\omega_\star^p(T))}^2,\end{aligned}$$

which immediately shows (3.12).
Now, we assume that v is even in $H^1(\Gamma)$. We proceed as before and use (3.16) for $v \in H^1(\Gamma)$. This proves (3.13).
With the help of [Fae00, Lemma 2.3] and [FGHP16a, Lemma 4.5], we see

$$\begin{aligned}|(1-J_\star)v|^2_{H^{1/2}(\Gamma)} &\overset{[\text{Fae00}]}{\lesssim} \sum_{z\in\mathcal{N}_\star} \big(|(1-J_\star)v|^2_{H^{1/2}(\omega_\star(z))} + \|h_\star^{-1/2}(1-J_\star)v\|^2_{L^2(\omega_\star(z))}\big)\\ &\overset{[\text{FGHP16a}]}{\lesssim} \sum_{z\in\mathcal{N}_\star} \big(\|h_\star^{1/2}\nabla((1-J_\star)v)\|^2_{L^2(\omega_\star(z))} + \|h_\star^{-1/2}(1-J_\star)v\|^2_{L^2(\omega_\star(z))}\big).\end{aligned}$$

Together with (3.12) and (3.13), we finally see

$$|(1-J_\star)v|_{H^{1/2}(\Gamma)} \lesssim \|h_\star^{1/2}\nabla v\|_{L^2(\Gamma)}.$$

This proves the last point. □

3.2.2 Convergence of uniform refinement

The next lemma states, that independently of the starting mesh $[\mathcal{T}_\star] \in [\mathbb{T}]$, uniform refinement always leads to convergence.

Lemma 3.2 *Let $u \in H^{1/2}(\Gamma)$ be the solution to (1.11), let $\epsilon > 0$ and $[\mathcal{T}_\star] \in [\mathbb{T}]$. Then, there exists a refinement $[\mathcal{T}_\bullet] \in \operatorname{ref}([\mathcal{T}_\star])$ such that the corresponding Galerkin solution $U_\bullet \in \mathcal{X}_\bullet$ satisfies*

$$\|u - U_\bullet\|_{H^{1/2}(\Gamma)} \le \epsilon. \tag{3.17}$$

Proof. We split the proof into three steps.
Step 1: We show a Céa-type quasi-optimality: Let $[\mathcal{T}_\bullet] \in [\mathbb{T}]$ be arbitrary. Since the energy norm $|\!|\!| \cdot |\!|\!|_{W+S}$ is an equivalent norm to $\|\cdot\|_{H^{1/2}(\Gamma)}$, the following inequality is satisfied

$$\|u - U_\bullet\|^2_{H^{1/2}(\Gamma)} \lesssim |\!|\!| u - U_\bullet |\!|\!|^2_{W+S} = \langle\!\langle u - U_\bullet; u - U_\bullet\rangle\!\rangle_{W+S}.$$

Due to the Galerkin orthogonality, it holds for all $V_\bullet \in \mathcal{X}_\bullet$

$$\langle\!\langle u - U_\bullet; u - U_\bullet\rangle\!\rangle_{W+S} = \langle\!\langle u - U_\bullet; u - V_\bullet\rangle\!\rangle_{W+S} \lesssim |\!|\!| u - U_\bullet |\!|\!|_{W+S} |\!|\!| u - V_\bullet |\!|\!|_{W+S}$$

Taking the infimum over all $V_\bullet \in \mathcal{X}_\bullet$ and using again the norm equivalence, we get the Céa-type quasi-optimality

$$\|u - U_\bullet\|_{H^{1/2}(\Gamma)} \lesssim \min_{V_\bullet\in\mathcal{X}_\bullet} \|u - V_\bullet\|_{H^{1/2}(\Gamma)}.$$

Step 2: We use the Scott-Zhang type operator $J_\bullet$: Since $H^1(\Gamma)$ is a dense subset of $H^{1/2}(\Gamma)$, there exists a function $v \in H^1(\Gamma)$ with

$$\|u - v\|_{H^{1/2}(\Gamma)} \le \frac{\epsilon}{2},$$

and we can estimate

$$\min_{V_\bullet\in\mathcal{X}_\bullet} \|u - V_\bullet\|_{H^{1/2}(\Gamma)} \le \|u - v\|_{H^{1/2}(\Gamma)} + \|v - J_\bullet v\|_{H^{1/2}(\Gamma)} \le \frac{\epsilon}{2} + \|v - J_\bullet v\|_{H^{1/2}(\Gamma)}.$$

From (3.13) and (3.14), we get

$$\begin{aligned}\|v - J_\bullet v\|^2_{H^{1/2}(\Gamma)} &= \|v - J_\bullet v\|^2_{L^2(\Gamma)} + |v - J_\bullet v|^2_{H^{1/2}(\Gamma)}\\ &\lesssim \|h_\bullet^{1/2}\nabla v\|^2_{L^2(\Gamma)} \lesssim \|\widetilde{h}_\bullet\|_{L^\infty(\Gamma)} |\nabla v|^2_{H^1(\Gamma)}.\end{aligned}$$

Step 3: We combine (i) and (ii): There exists $[T_\bullet] \in [\mathbb{T}]$ with $\|\widetilde{h}_\bullet\|_{L^\infty(\Gamma)} \ll 1$ sufficiently small such that

$$\|u - U_\bullet\|_{H^{1/2}(\Gamma)} \le \frac{\epsilon}{2} + \|v - J_\bullet v\|_{H^{1/2}(\Gamma)} \le \frac{\epsilon}{2} + \frac{\epsilon}{2} \le \epsilon.$$

This concludes the proof. □

3.2.3 Inverse estimates

Next, we present some inverse estimates which stem from [AFF+15, Theorem 3.1], [FGHP16b, Proposition 4.1] and [FGP].

Theorem 3.3 *Let $[\mathcal{T}_\star] \in [\mathbb{T}]$. Then, there exists a constant $C_{\rm inv} > 0$ such that*

$$\|h_\star^{1/2}(1/2 - K')\psi\|_{L^2(\Gamma)} \le C_{\rm inv}\big(\|\psi\|_{H^{-1/2}(\Gamma)} + \|h_\star^{1/2}\psi\|_{L^2(\Gamma)}\big) \quad \text{for all } \psi \in L^2(\Gamma) \tag{3.18}$$

and

$$\|h_\star^{1/2} W v\|_{L^2(\Gamma)} \le C_{\rm inv}\big(\|v\|_{H^{1/2}(\Gamma)} + \|h_\star^{1/2}\nabla v\|_{L^2(\Gamma)}\big) \quad \text{for all } v \in H^1(\Gamma). \tag{3.19}$$

The constant $C_{\rm inv} > 0$ depends only on $\check{\kappa}_{\max}$ and γ.
Moreover, there exists a constant $\widetilde{C}_{\rm inv} > 0$ such that

$$\|h_\star^{1/2}\Psi_\star\|_{L^2(\Gamma)} + \|h_\star^{1/2}(1/2 - K')\Psi_\star\|_{L^2(\Gamma)} \le \widetilde{C}_{\rm inv}\|\Psi_\star\|_{H^{-1/2}(\Gamma)} \quad \text{for all } \Psi_\star \in \mathcal{P}^p(\mathcal{T}_\star) \tag{3.20}$$

as well as,

$$\|h_\star^{1/2}\nabla V_\star\|_{L^2(\Gamma)} + \|h_\star^{1/2} W V_\star\|_{L^2(\Gamma)} \le \widetilde{C}_{\rm inv}\|V_\star\|_{H^{1/2}(\Gamma)} \quad \text{for all } V_\star \in \mathcal{X}_\star. \tag{3.21}$$

The constant $\widetilde{C}_{\rm inv} > 0$ depends only on $\check{\kappa}_{\max}, p, w_{\min}, w_{\max}$, and γ.

Proof. The proof of (3.18) and (3.19) is found in [AFF+15, Theorem 3.1] with $w_h := h_\star^{1/2}$. To see (3.20), one can directly use (3.18) in combination with the inverse estimate from [FGHP16b, Proposition 4.1]. Estimate (3.21) is considered in [FGP] and follows by standard interpolation techniques together with the fact that $J_\star$ is L^2- and H^1-stable. □

3.2.4 Approximation property of $\Pi_\star$

Due to numerical reasons, we have to approximate the Neumann data ϕ, which leads to an additional consistency error. The following lemma helps to control this error. We follow the ideas of [CP06, Theorem 4.1].

Lemma 3.4 *Let $\Pi_\star\colon L^2(\Gamma) \to \mathcal{P}^p(\mathcal{T}_\star)$ be the L^2-orthogonal projection onto $\mathcal{P}^p(\mathcal{T}_\star)$. Then, there exists a constant $C_{\rm apx} > 0$ such that*

$$\|\psi - \Pi_\star\psi\|_{H^{-1/2}(\Gamma)} \le C_{\rm apx}\|h_\star^{1/2}(\psi - \Pi_\star\psi)\|_{L^2(\Gamma)} \quad \text{for all } \psi \in L^2(\Gamma). \tag{3.22}$$

The constant $C_{\rm apx} > 0$ depends only on the boundary Γ.

Proof. For $\Pi_0\colon L^2(\Gamma) \to \mathcal{P}^0(\mathcal{T}_\star)$ being the L^2-orthogonal projection onto $\mathcal{P}^0(\mathcal{T}_\star)$ and each $T \in \mathcal{T}_\star$, it holds for all $\phi \in L^2(T)$ that

$$\begin{aligned}\|\psi - \Pi_\star\phi\|_{L^2(T)} &= \inf_{\phi_h \in \mathcal{P}^p(\mathcal{T}_\star)} \|\phi - \phi_h\|_{L^2(T)} \\ &\le \inf_{\phi_h \in \mathcal{P}^0(\mathcal{T}_\star)} \|\phi - \phi_h\|_{L^2(T)} = \|\phi - \Pi_0\phi\|_{L^2(T)}.\end{aligned} \tag{3.23}$$

Next, we use [Fae00, Lemma 2.5], which leads to

$$\|\phi - \Pi_0\phi\|_{L^2(T)} \lesssim \underbrace{|T|^{1/2}}_{=h_{\star,T}^{1/2}} \underbrace{|\phi - \Pi_0\phi|_{H^{1/2}(T)}}_{\overset{\text{def.}}{=} |\phi|_{H^{1/2}(T)}} \le h_{\star,T}^{1/2}\|\phi\|_{H^{1/2}(T)} \quad \text{for all } \phi \in H^{1/2}(T). \tag{3.24}$$

Now, let $\psi \in L^2(\Gamma)$ and $\phi \in H^{1/2}(\Gamma)$. Since the orthogonal projection is self-adjoint, it holds that

$$\langle (1 - \Pi_\star)\psi; \phi\rangle_\Gamma = \langle \psi; (1 - \Pi_\star)\phi\rangle_\Gamma \le \sum_{T \in \mathcal{T}_\star} \|\psi\|_{L^2(T)}\|\phi - \Pi_\star\phi\|_{L^2(T)}. \tag{3.25}$$

Combining (3.23)–(3.25), we get

$$\langle (1-\Pi_\star)\psi;\phi\rangle_\Gamma \le \sum_{T\in\mathcal{T}_\star} \|\psi\|_{L^2(T)}\|\phi-\Pi_\star\phi\|_{L^2(T)} \overset{(3.24)}{\lesssim} \sum_{T\in\mathcal{T}_\star} \|h_\star^{1/2}\psi\|_{L^2(T)}\|\phi\|_{H^{1/2}(T)}$$
$$\le \|h_\star^{1/2}\psi\|_{L^2(\Gamma)}\|\phi\|_{H^{1/2}(\Gamma)}.$$

Replacing ψ by $\psi-\Pi_\star\psi$ and using the idempotency of the orthogonal projection, it follows that

$$\|\psi-\Pi_\star\psi\|_{H^{-1/2}(\Gamma)} = \|(1-\Pi_\star)(\psi-\Pi_\star\psi)\|_{H^{-1/2}(\Gamma)}$$
$$= \sup_{\substack{\phi\in H^{1/2}(\Gamma)\\ \phi\neq 0}} \frac{\langle (1-\Pi_\star)(\psi-\Pi_\star\psi);\phi\rangle_\Gamma}{\|\phi\|_{H^{1/2}(\Gamma)}} \le C_{\mathrm{apx}}\|h_\star^{1/2}(\psi-\Pi_\star\psi)\|_{L^2(\Gamma)}.$$

This concludes the proof. □

3.3 Data approximation error $\widetilde{\mathrm{osc}}_\star$

We first look at the oscillation term of the error estimator $\rho_\star$. For $[\mathcal{T}_\star]\in[\mathbb{T}]$ and $\phi\in L^2(\Gamma)$, we define

$$\widetilde{\mathrm{osc}}_\star([T]) := \|\widetilde{h}_\star^{1/2}(\phi-\phi_\star)\|_{L^2(T)}, \tag{3.26}$$

as well as

$$\widetilde{\mathrm{osc}}_\star := \widetilde{\mathrm{osc}}_\star([\mathcal{T}_\star]) \quad \text{with } \widetilde{\mathrm{osc}}_\star([\mathcal{E}_\star])^2 := \sum_{[T]\in[\mathcal{E}_\star]} \widetilde{\mathrm{osc}}_\star([T])^2 \quad \text{for all } [\mathcal{E}_\star]\subseteq[\mathcal{T}_\star]. \tag{3.27}$$

Proposition 3.5 *Let $[\mathcal{T}_+]\in\mathrm{ref}([\mathcal{T}_\star])$ with $[\mathcal{T}_\star]\in[\mathbb{T}]$. For the data oscillation terms $\widetilde{\mathrm{osc}}_+$ and $\widetilde{\mathrm{osc}}_\star$, the following properties* (i)–(iii) *hold:*

(i) *Stability on non-contracting part: There holds that*

$$\widetilde{\mathrm{osc}}_+([\mathcal{T}_+]\setminus[\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star])) = \widetilde{\mathrm{osc}}_\star([\mathcal{T}_\star]\setminus[\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+])). \tag{3.28}$$

(ii) *Reduction on contracting part: With the constant q_{ctr} from* (2.20)*, there holds that*

$$\widetilde{\mathrm{osc}}_+([\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star]))^2 \le q_{\mathrm{ctr}}\,\widetilde{\mathrm{osc}}_\star([\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))^2. \tag{3.29}$$

(iii) *Discrete reliability: There exists a constant $C_{\widetilde{\mathrm{osc}}}>0$ such that*

$$\|\phi_+-\phi_\star\|_{H^{-1/2}(\Gamma)} \le C_{\widetilde{\mathrm{osc}}}\,\widetilde{\mathrm{osc}}_\star([\mathcal{T}_\star]\setminus[\mathcal{T}_+]). \tag{3.30}$$

The constant $C_{\widetilde{\mathrm{osc}}}>0$ depends only on $\check{\kappa}_{\max}, p$ and γ.

Proof. We only consider $\phi_\star := \Pi_\star\phi$, since the other case is trivial. The proof works analogously as in [FFK+15, Proposition 3.7].
Since $\Pi_\star$ is the L^2-orthogonal projection onto $\mathcal{P}^p(\mathcal{T}_\star)$, we know that it is the $\mathcal{T}_\star$-elementwise best approximation. Hence, it holds that $\phi_\star=\phi_+$ on the non-contracting part $\Gamma\setminus\omega_\star([\mathcal{T}_\star]\setminus[\mathcal{T}_+])\subseteq\bigcup([\mathcal{T}_\star]\cap[\mathcal{T}_+])$, and (2.21) proves (3.28).
Note that $\|\phi-\phi_+\|_{L^2(T)}\le\|\phi-\phi_\star\|_{L^2(T)}$ for all $T\in\mathcal{T}_+$. This and $\widetilde{h}_+\le q_{\mathrm{ctr}}\widetilde{h}_\star$ on the contracting part $\omega_\star([\mathcal{T}_\star]\setminus[\mathcal{T}_+])=\omega_+([\mathcal{T}_+]\setminus[\mathcal{T}_\star])$, which follows directly from the definition of $\widetilde{h}$ in Proposition 2.3, imply

$$\widetilde{\mathrm{osc}}_+([\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star])) = \|\widetilde{h}_+^{1/2}(\phi-\phi_+)\|_{L^2(\omega_+([\mathcal{T}_+]\setminus[\mathcal{T}_\star]))} \le q_{\mathrm{ctr}}^{1/2}\,\|\widetilde{h}_\star^{1/2}(\phi-\phi_\star)\|_{L^2(\omega_+([\mathcal{T}_+]\setminus[\mathcal{T}_\star]))}$$
$$= q_{\mathrm{ctr}}^{1/2}\widetilde{\mathrm{osc}}_\star([\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+])),$$

which concludes the proof of (3.29).
For (3.30), we need the following elementwise identities, which we get by the fact that Π_+ as well as $\Pi_\star$ are orthogonal projections and that $\mathcal{X}_\star\subseteq\mathcal{X}_+$,

$$\Pi_+(1-\Pi_\star) = \Pi_+-\Pi_\star = (1-\Pi_\star)\Pi_+.$$

For $\phi \in L^2(\Gamma)$, this reads as follows

$$\Pi_+(\phi - \phi_\star) = \phi_+ - \phi_\star = \phi_+ - \Pi_\star \phi_+.$$

Now, we use (3.22), $\phi_\star = \phi_+$ in $\bigcup([\mathcal{T}_\star] \cap [\mathcal{T}_+])$, $\widetilde{h}_\star \simeq h_\star$ and $\|\Pi_+\psi\|_{L^2(T)} \le \|\psi\|_{L^2(T)}$ for $\psi \in L^2(\Gamma)$ and $T \in \mathcal{T}_\star$, to see

$$\begin{aligned}
\|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)} &= \|\phi_+ - \Pi_\star\phi_+\|_{H^{-1/2}(\Gamma)} \overset{(3.22)}{\lesssim} \|\widetilde{h}_\star^{1/2}(\phi_+ - \Pi_\star\phi_+)\|_{L^2(\Gamma)} \\
&= \|\widetilde{h}_\star^{1/2}(\phi_+ - \phi_\star)\|_{L^2(\bigcup([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))} = \|\widetilde{h}_\star^{1/2}\Pi_+(\phi - \phi_\star)\|_{L^2(\bigcup([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))} \\
&\le \|\widetilde{h}_\star^{1/2}(\phi - \phi_\star)\|_{L^2(\bigcup([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))} \\
&= \widetilde{\operatorname{osc}}_\star([\mathcal{T}_\star] \setminus [\mathcal{T}_+]).
\end{aligned}$$

We see that the hidden constant $C_{\widetilde{\operatorname{osc}}} > 0$ depends only on C_{apx} from (3.22) and the equivalence constant C_{wt} from (2.18). □

3.4 Error estimator $\widetilde{\rho}_\star$

The following theorem states some basic properties of $\widetilde{\rho}_\star$. A similar version was already proved in [FFK+15, Theorem 3.8].

Theorem 3.6 *Let $[\mathcal{T}_+] \in \operatorname{ref}([\mathcal{T}_\star])$ and $U_\star$ resp. U_+ be the corresponding Galerkin solutions to* (3.2)*. The error estimators $\widetilde{\rho}_\star$ resp. $\widetilde{\rho}_+$ satisfy the following properties* (i)–(iii)*:*

(i) *Stability on non-contracting part: There exists a constant $C_{\mathrm{stab}} > 0$ such that*

$$\begin{aligned}
&\big|\widetilde{\rho}_+\big([\mathcal{T}_+] \setminus [\omega_+]([\mathcal{T}_+] \setminus [\mathcal{T}_\star])\big) - \widetilde{\rho}_\star\big([\mathcal{T}_\star] \setminus [\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big)\big| \\
&\qquad\qquad \le C_{\mathrm{stab}}\big(\|U_+ - U_\star\|_{H^{1/2}(\Gamma)} + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)}\big).
\end{aligned} \tag{3.31}$$

(ii) *Reduction on contracting part: There exist constants $0 < q_{\mathrm{red}} < 1$ and $C_{\mathrm{red}} > 0$ such that*

$$\begin{aligned}
\widetilde{\rho}_+\big([\omega_+]([\mathcal{T}_+] \setminus [\mathcal{T}_\star])\big)^2 &\le q_{\mathrm{red}}\, \widetilde{\rho}_\star\big([\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big)^2 \\
&\quad + C_{\mathrm{red}}\,\big(\|U_+ - U_\star\|^2_{H^{1/2}(\Gamma)} + \|\phi_+ - \phi_\star\|^2_{H^{-1/2}(\Gamma)}\big).
\end{aligned} \tag{3.32}$$

(iii) *Discrete reliability: There exist constants $C_{\mathrm{drl}}, C_{\mathrm{ref}} > 0$ such that*

$$\|U_+ - U_\star\|_{H^{1/2}(\Gamma)} + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)} \le C_{\mathrm{drl}}\, \widetilde{\rho}_\star\big([\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big). \tag{3.33}$$

Obviously,

$$\big|[\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big| \le C_{\mathrm{ref}}\big|[\mathcal{T}_\star] \setminus [\mathcal{T}_+]\big|. \tag{3.34}$$

The constants $C_{\mathrm{stab}}, C_{\mathrm{red}}, C_{\mathrm{drl}}, C_{\mathrm{ref}} > 0$ and $0 < q_{\mathrm{red}} < 1$ depend only on $\check{\kappa}_{\max}, p, w_{\min}, w_{\max}$ and γ.

Proof. We prove the three points for $\phi_\star := \phi$ as well as $\phi_\star := \Pi_\star\phi$. Note that in the case $\phi_\star = \phi$ some argumentations become trivial.

(i) Stability on non-contracting part (3.31): We recall that $\widetilde{h}_+ = \widetilde{h}_\star$ on $\Gamma \setminus \omega_\star([\mathcal{T}_\star] \setminus [\mathcal{T}_+]) = \Gamma \setminus \omega_+([\mathcal{T}_+] \setminus [\mathcal{T}_\star]) \subseteq \bigcup([\mathcal{T}_\star] \cap [\mathcal{T}_+])$. We use the stability (3.28) of the data oscillations, the mesh-size

equivalence (2.21), and the reverse triangle inequality to see

$$
\begin{aligned}
&\big|\widetilde{\rho}_+\big([\mathcal{T}_+]\setminus[\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star])\big)-\widetilde{\rho}_\star\big([\mathcal{T}_\star]\setminus[\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+])\big)\big|\\
&\quad=\Big|\Big(\sum_{[T]\in[\mathcal{T}_+]\setminus[\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star])}\tilde{\rho}_+(T)^2\Big)^{1/2}-\Big(\sum_{[T]\in[\mathcal{T}_\star]\setminus[\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+])}\tilde{\rho}_\star(T)^2\Big)^{1/2}\Big|\\
&\quad\le\Big(\sum_{[T]\in[\mathcal{T}_+]\setminus[\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star])}\big(\|\widetilde{h}_+^{1/2}\big((1/2-K')\phi_+-WU_+\big)\|_{L^2(T)}\\
&\qquad\qquad-\|\widetilde{h}_+^{1/2}\big((1/2-K')\phi_\star-WU_\star\big)\|_{L^2(T)}\big)^2\Big)^{1/2}\\
&\quad\le\Big(\sum_{[T]\in[\mathcal{T}_+]\setminus[\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star])}\|\widetilde{h}_+^{1/2}\big((1/2-K')\phi_+-WU_+\big)-\widetilde{h}_+^{1/2}\big((1/2-K')\phi_\star-WU_\star\big)\|_{L^2(T)}^2\Big)^{1/2}\\
&\quad=\|\widetilde{h}_+^{1/2}\big((1/2-K')\phi_+-WU_+\big)-\widetilde{h}_+^{1/2}\big((1/2-K')\phi_\star-WU_\star\big)\|_{L^2(\Gamma\setminus\omega_+([\mathcal{T}_+]\setminus[\mathcal{T}_\star]))}\\
&\quad\le\|\widetilde{h}_+^{1/2}W(U_+-U_\star)\|_{L^2(\Gamma)}+\|\widetilde{h}_+^{1/2}(1/2-K')(\phi_+-\phi_\star)\|_{L^2(\Gamma)}.
\end{aligned}
$$

The inverse estimates (3.20) and (3.21) lead to

$$
\lesssim\|U_+-U_\star\|_{H^{1/2}(\Gamma)}+\|\phi_+-\phi_\star\|_{H^{-1/2}(\Gamma)},
$$

which concludes the proof of (i).

(ii) Reduction on contracting part (3.32): By definition, the error estimator satisfies

$$
\begin{aligned}
&\widetilde{\rho}_+([\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star]))^2\\
&\quad=\sum_{[T]\in[\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star])}\Big(\|\widetilde{h}_+^{1/2}\big((1/2-K')\phi_+-WU_+\big)\|_{L^2(T)}+\|\widetilde{h}_+^{1/2}(\phi-\phi_+)\|_{L^2(T)}\Big)^2\\
&\quad=\sum_{[T]\in[\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star])}\Big(\|\widetilde{h}_+^{1/2}\big((1/2-K')(\phi_+-\phi_\star+\phi_\star)-W(U_+-U_\star+U_\star)\big)\|_{L^2(T)}\\
&\qquad\qquad+\|\widetilde{h}_+^{1/2}(\phi-\phi_+)\|_{L^2(T)}\Big)^2.
\end{aligned}
$$

It holds $\widetilde{h}_+\le q_{\mathrm{ctr}}\widetilde{h}_\star$ on $\omega_+([\mathcal{T}_+]\setminus[\mathcal{T}_\star])=\omega_\star([\mathcal{T}_\star]\setminus[\mathcal{T}_+])$. Hence, with Young's inequality for $\delta>0$ and, for the case $\phi_+:=\Pi_+\phi$, the fact that Π_+ is the $\mathcal{T}_+$-elementwise L^2-best approximation, we have with a constant $C_\delta>0$ which depends only on δ and the inverse estimates (3.20)–(3.21), that

$$
\begin{aligned}
&\widetilde{\rho}_+([\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star]))^2\\
&\quad\le\sum_{[T]\in[\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star])}\Big((1+\delta)q_{\mathrm{ctr}}\big(\|\widetilde{h}_\star^{1/2}\big((1/2-K')\phi_\star-WU_\star\big)\|_{L^2(T)}+\|\widetilde{h}_\star^{1/2}(\phi-\phi_\star)\|_{L^2(T)}\big)^2\\
&\qquad\qquad+2(1+\delta^{-1})\|\widetilde{h}_+^{1/2}(1/2-K')(\phi_+-\phi_\star)\|_{L^2(T)}^2\\
&\qquad\qquad+2(1+\delta^{-1})\|\widetilde{h}_+^{1/2}W(U_+-U_\star)\|_{L^2(T)}^2\Big)\\
&\quad\le(1+\delta)q_{\mathrm{ctr}}\,\widetilde{\rho}_\star([\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))^2+C_\delta\big(\|U_+-U_\star\|_{H^{1/2}(\Gamma)}^2+\|\phi_+-\phi_\star\|_{H^{-1/2}(\Gamma)}^2\big).
\end{aligned}
$$

We choose $\delta>0$ sufficiently small and set $q_{\mathrm{red}}:=(1+\delta)q_{\mathrm{ctr}}$.

(iii) Discrete reliability (3.33): We split the proof into seven steps.
Step 1: Let $U_{\star,+}\in\mathcal{X}_+$ denote the unique Galerkin solution to

$$
\langle\!\langle U_{\star,+};V_+\rangle\!\rangle_{W+S}=\langle(1/2-K')\phi_\star;V_+\rangle_\Gamma\quad\text{for all } V_+\in\mathcal{X}_+.
$$

Both $U_{\star,+}\in\mathcal{X}_+$ as well as $U_\star\in\mathcal{X}_\star$ are Galerkin solutions to the same right-hand side $f=(1/2-K')\phi_\star$.

Step 2: In this step, we show that $\|U_{\star,+} - U_\star\|^2_{H^{1/2}(\Gamma)} \simeq \langle f - WU_\star; (1-J_\star)(U_{\star,+} - U_\star)\rangle_\Gamma$: From Section 1.2 we know, that $\|\cdot\|_{H^{1/2}(\Gamma)}$ and $|||\cdot|||_{W+S}$ are equivalent norms, whence we get with the Galerkin orthogonality

$$\begin{aligned}\|U_{\star,+} - U_\star\|^2_{H^{1/2}(\Gamma)} &\simeq |||\, U_{\star,+} - U_\star \,|||^2_{W+S} = \langle\!\langle U_{\star,+} - U_\star; U_{\star,+} - U_\star\rangle\!\rangle_{W+S} \\ &= \langle\!\langle U_{\star,+} - U_\star; (1-J_\star)(U_{\star,+} - U_\star)\rangle\!\rangle_{W+S} \\ &= \langle W(U_{\star,+} - U_\star); (1-J_\star)(U_{\star,+} - U_\star)\rangle_\Gamma \\ &\qquad + \langle 1; U_{\star,+} - U_\star\rangle_\Gamma \langle 1; (1-J_\star)(U_{\star,+} - U_\star)\rangle_\Gamma\end{aligned}$$

The second summand is zero, due to

$$0 = \langle\!\langle U_{\star,+} - U_\star; 1\rangle\!\rangle_{W+S} = \underbrace{\langle W(U_{\star,+} - U_\star); 1\rangle_\Gamma}_{=\langle U_{\star,+} - U_\star; W1\rangle_\Gamma = 0} + \langle 1; U_{\star,+} - U_\star\rangle_\Gamma \underbrace{\langle 1; 1\rangle_\Gamma}_{=|\Gamma|}$$

This implies

$$\|U_{\star,+} - U_\star\|^2_{H^{1/2}(\Gamma)} \simeq \langle f - WU_\star; (1-J_\star)(U_{\star,+} - U_\star)\rangle_\Gamma. \tag{3.35}$$

Step 3: We apply Lemma 3.1 (i) to $U_{\star,+} - U_\star$: Let $T \in \mathcal{T}_\star$ with $T \subseteq \Gamma \setminus \omega_\star^{2p}([\mathcal{T}_\star] \setminus [\mathcal{T}_+])$ up to finitely many points. It holds $\omega_\star^{2p}(T) \subseteq \bigcup([\mathcal{T}_\star] \cap [\mathcal{T}_+])$, wherefore $(U_{\star,+} - U_\star)|_{\omega_\star^p(T)} \in \mathcal{X}_\star|_{\omega_\star^p(T)}$. Hence, we can use Lemma 3.1 (i), to see

$$J_\star(U_{\star,+} - U_\star)|_T = (U_{\star,+} - U_\star)|_T$$

Step 4: We use Step 1, Step 2, and the Cauchy-Schwarz inequality to get

$$\begin{aligned}&\|U_{\star,+} - U_\star\|^2_{H^{1/2}(\Gamma)} \\ &\quad\lesssim \langle f - WU_\star; (1-J_\star)(U_{\star,+} - U_\star)\rangle_\Gamma \\ &\quad= \langle f - WU_\star; (1-J_\star)(U_{\star,+} - U_\star)\rangle_{\omega_\star^{2p}([\mathcal{T}_\star]\setminus[\mathcal{T}_+])} \\ &\quad\overset{\text{C.S.}}{\leq} \|\widetilde{h}_\star^{1/2}(f - WU_\star)\|_{L^2(\omega_\star^{2p}([\mathcal{T}_\star]\setminus[\mathcal{T}_+])} \|\widetilde{h}_\star^{-1/2}(1-J_\star)(U_{\star,+} - U_\star)\|_{L^2(\omega_\star^{2p}([\mathcal{T}_\star]\setminus[\mathcal{T}_+])}\end{aligned}$$

With the approximation property (3.12) of $J_\star$, it follows directly

$$\|U_{\star,+} - U_\star\|^2_{H^{1/2}(\Gamma)} \overset{(3.12)}{\lesssim} \widetilde{\rho}_\star([\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+]))\|U_{\star,+} - U_\star\|_{H^{1/2}(\Gamma)}.$$

Hence, we end up with

$$\|U_{\star,+} - U_\star\|_{H^{1/2}(\Gamma)} \lesssim \widetilde{\rho}_\star([\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])). \tag{3.36}$$

We note that, in the case $\phi_\star := \phi$, this step already concludes the proof of discrete reliability, since $U_{\star,+} = U_+$.

Step 5: Next, the equivalence of $\|\cdot\|_{H^{1/2}(\Gamma)}$ and $|||\cdot|||_{W+S}$ on $H^{1/2}(\Gamma)$, the definition of U_+ and $U_{\star,+}$, as well as the continuity of K' lead to

$$\begin{aligned}\|U_+ - U_{\star,+}\|^2_{H^{1/2}(\Gamma)} &\lesssim |||\, U_+ - U_{\star,+} \,|||^2_{W+S} = \langle\!\langle U_+ - U_{\star,+}; U_+ - U_{\star,+}\rangle\!\rangle_{W+S} \\ &= \langle (1/2 - K')\phi_+ - (1/2 - K')\phi_\star; U_+ - U_{\star,+}\rangle_\Gamma \\ &\lesssim \|(1/2 - K')\phi_+ - (1/2 - K')\phi_\star\|_{H^{-1/2}(\Gamma)} \|U_+ - U_{\star,+}\|_{H^{1/2}(\Gamma)} \\ &\lesssim \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)} \|U_+ - U_{\star,+}\|_{H^{1/2}(\Gamma)},\end{aligned}$$

which is equivalent to

$$\|U_+ - U_{\star,+}\|_{H^{1/2}(\Gamma)} \lesssim \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)}. \tag{3.37}$$

Step 6: We know from the discrete reliability (3.30) of the data oscillations that

$$\|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)} \lesssim \widetilde{\operatorname{osc}}_\star([\mathcal{T}_\star] \setminus [\mathcal{T}_+]) \leq \widetilde{\rho}_\star([\mathcal{T}_\star] \setminus [\mathcal{T}_+]). \tag{3.38}$$

Step 7: Finally, we combine (3.36)–(3.38) to get

$$\begin{aligned}\|U_+ - U_\star\|_{H^{1/2}(\Gamma)} + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)} \\ &\le \|U_+ - U_{\star,+}\|_{H^{1/2}(\Gamma)} + \|U_{\star,+} - U_\star\|_{H^{1/2}(\Gamma)} + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)} \\ &\lesssim \widetilde{\rho}_\star([\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])),\end{aligned}$$

which concludes the proof. □

From the aforegoing basic properties, we can deduce the following corollary.

Corollary 3.7 *Let $[\mathcal{T}_+] \in \mathrm{ref}([\mathcal{T}_\star])$ for some $[\mathcal{T}_\star] \in [\mathbb{T}]$ and $U_\star$ resp. U_+ be the corresponding Galerkin solutions to* (3.2) *and u the solution to* (3.1)*. The error estimators $\widetilde{\rho}_\star$ resp. $\widetilde{\rho}_+$ satisfy the following properties:*

(i) *Quasi-monotonicity: There exists a constant $C_{\mathrm{mon}} > 0$ such that*

$$\widetilde{\rho}_+ \le C_{\mathrm{mon}}\, \widetilde{\rho}_\star. \tag{3.39}$$

(ii) *Reliability: With the constant C_{drl} from* (3.33)*, it holds*

$$\|u - U_\star\|_{H^{1/2}(\Gamma)} \le C_{\mathrm{drl}}\, \widetilde{\rho}_\star. \tag{3.40}$$

(iii) *Estimator reduction: For $0 < \theta \le 1$, there exist constants $C_{\mathrm{est}} > 0$ and $0 < q_{\mathrm{est}} \le 1$ such that the following implication holds*

$$\theta \widetilde{\rho}_\star^2 \le \widetilde{\rho}_\star([\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+]))^2 \quad \Longrightarrow \quad \widetilde{\rho}_+^2 \le q_{\mathrm{est}}\, \widetilde{\rho}_\star^2 + C_{\mathrm{est}}\big(\|U_+ - U_\star\|_{H^{1/2}(\Gamma)}^2 + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)}^2\big) \tag{3.41}$$

The constant $C_{\mathrm{mon}} > 0$ depends only on $\check{\kappa}_{\max}, p, w_{\min}, w_{\max}$, and γ. The constants $C_{\mathrm{est}} > 0$ and $0 < q_{\mathrm{est}} \le 1$ depend additionally on θ.

Proof. As before, the proof holds for $\phi_\star := \phi$ as well as $\phi_\star := \Pi_\star \phi$.

(i) Quasi-monotonicity (3.39): It holds that

$$\widetilde{\rho}_+^2 = \widetilde{\rho}_+([\omega_+]([\mathcal{T}_+] \setminus [\mathcal{T}_\star]))^2 + \widetilde{\rho}_+([\mathcal{T}_+] \setminus [\omega_+]([\mathcal{T}_+] \cap [\mathcal{T}_\star]))^2.$$

The reduction property (3.32) yields

$$\begin{aligned}&\widetilde{\rho}_+([\omega_+]([\mathcal{T}_+] \setminus [\mathcal{T}_\star]))^2 \\ &\qquad \lesssim \widetilde{\rho}_\star([\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+]))^2 + \|U_+ - U_\star\|_{H^{1/2}(\Gamma)}^2 + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)}^2.\end{aligned}$$

The stability (3.31) implies

$$\begin{aligned}&\widetilde{\rho}_+([\mathcal{T}_+] \setminus [\omega_+]([\mathcal{T}_+] \setminus [\mathcal{T}_\star]))^2 \\ &\qquad \lesssim \widetilde{\rho}_\star([\mathcal{T}_\star] \setminus [\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+]))^2 + \|U_+ - U_\star\|_{H^{1/2}(\Gamma)}^2 + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)}^2.\end{aligned}$$

Together with the discrete reliability (3.33), these two estimates yield

$$\widetilde{\rho}_+^2 \lesssim \widetilde{\rho}_\star^2.$$

This concludes the proof.

(ii) Reliability (3.40): Due to Lemma 3.2, there exists for all $\epsilon > 0$ a mesh $[\mathcal{T}_\bullet] \in \mathrm{ref}\,([\mathcal{T}_\star])$ such that $\|u - U_{\phi,\bullet}\|_{H^{1/2}(\Gamma)} \le \epsilon$, where $U_{\phi,\bullet}$ is the unique solution to

$$\langle\!\langle U_{\phi,\bullet}; V_\bullet \rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi; V_\bullet \rangle_\Gamma \quad \text{for all } V_\bullet \in \mathcal{X}_\bullet.$$

We define $U_{\phi,\star} \in \mathcal{X}_\star$ analogously, and apply the discrete reliability (3.33) to see

$$\begin{aligned}\|u - U_\star\|_{H^{1/2}(\Gamma)} &\le \|u - U_{\phi,\bullet}\|_{H^{1/2}(\Gamma)} + \|U_{\phi,\bullet} - U_{\phi,\star}\|_{H^{1/2}(\Gamma)} + \|U_{\phi,\star} - U_\star\|_{H^{1/2}(\Gamma)} \\ &\le \epsilon + C_{\mathrm{drl}}^{1/2}\, \|\widetilde{h}_\star^{1/2}\big((1/2-K')\phi - WU_{\phi,\star}\big)\|_{L^2(\omega_\star^{2p}([\mathcal{T}_\star]\setminus[\mathcal{T}_\bullet]))} \\ &\qquad + \|U_{\phi,\star} - U_\star\|_{H^{1/2}(\Gamma)}. \end{aligned} \tag{3.42}$$

In the case $\phi_\star := \phi$, this already concludes the proof of reliability, since $U_{\phi,\star} = U_\star$ and $\epsilon > 0$ is arbitrary.
For the square of the last summand, we obtain via norm equivalence, the continuity of the adjoint double-layer operator K', (3.22), and (2.21) that

$$\begin{aligned}\|U_{\phi,\star} - U_\star\|_{H^{1/2}(\Gamma)}^2 &\simeq |\!|\!| U_{\phi,\star} - U_\star |\!|\!|_{W+S}^2 = \langle\!\langle U_{\phi,\star} - U_\star; U_{\phi,\star} - U_\star\rangle\!\rangle_{W+S} \\ &= \langle (1/2-K')\phi - (1/2-K')\phi_\star; U_{\phi,\star} - U_\star\rangle_\Gamma \\ &\le \|(1/2-K')(\phi-\phi_\star)\|_{H^{-1/2}(\Gamma)} \|U_{\phi,\star} - U_\star\|_{H^{1/2}(\Gamma)} \\ &\lesssim \|\phi-\phi_\star\|_{H^{-1/2}(\Gamma)} \|U_{\phi,\star} - U_\star\|_{H^{1/2}(\Gamma)} \\ &\lesssim \|\widetilde{h}_\star^{1/2}(\phi-\phi_\star)\|_{L^2(\Gamma)} \|U_{\phi,\star} - U_\star\|_{H^{1/2}(\Gamma)}. \end{aligned} \tag{3.43}$$

To get an estimate for the second summand of (3.42) we use the auxiliary results (3.18), (3.21), (3.22) as well as (3.43) to see

$$\begin{aligned}&\|\widetilde{h}_\star^{1/2}\big((1/2-K')(\phi-\phi_\star+\phi_\star) - W(U_{\phi,\star} - U_\star + U_\star)\big)\|_{L^2(\omega_\star^{2p}([\mathcal{T}_\star]\setminus[\mathcal{T}_\bullet]))} \\ &\qquad \le \|\widetilde{h}_\star^{1/2}\big((1/2-K')\phi_\star - WU_\star\big)\|_{L^2(\omega_\star^{2p}([\mathcal{T}_\star]\setminus[\mathcal{T}_\bullet]))} \\ &\qquad\qquad + \|\widetilde{h}_\star^{1/2}\big((1/2-K')(\phi-\phi_\star)\|_{L^2(\omega_\star^{2p}([\mathcal{T}_\star]\setminus[\mathcal{T}_\bullet]))} \\ &\qquad\qquad + \|\widetilde{h}_\star^{1/2} W(U_{\phi,\star} - U_\star)\|_{L^2(\omega_\star^{2p}([\mathcal{T}_\star]\setminus[\mathcal{T}_\bullet]))} \\ &\qquad \overset{(3.18)+(3.21)}{\lesssim} \widetilde{\rho}_\star + \|\phi-\phi_\star\|_{H^{-1/2}(\Gamma)} + \|\widetilde{h}_\star^{1/2}(\phi-\phi_\star)\|_{L^2(\Gamma)} + \|U_{\phi,\star} - U_\star\|_{H^{1/2}(\Gamma)} \\ &\qquad \overset{(3.22)+(3.43)}{\lesssim} \widetilde{\rho}_\star. \end{aligned}$$

Altogether, we have

$$\|u - U_\star\|_{H^{1/2}(\Gamma)} \lesssim \epsilon + \widetilde{\rho}_\star.$$

Since $\epsilon > 0$ was arbitrary, this concludes the proof.

(iii) Estimator reduction (3.41): Obviously,

$$\widetilde{\rho}_+^2 = \widetilde{\rho}_+([\mathcal{T}_+]\setminus[\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star]))^2 + \widetilde{\rho}_+([\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star]))^2.$$

We use the stability (3.31) to see

$$\begin{aligned}\widetilde{\rho}_+([\mathcal{T}_+]\setminus[\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star]))^2 &\le (1+\delta)\,\widetilde{\rho}_\star([\mathcal{T}_\star]\setminus[\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+])^2 \\ &\qquad + 2(1+\delta^{-1})\, C_{\mathrm{stab}}^2 \big(\|U_+ - U_\star\|_{H^{1/2}(\Gamma)}^2 + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)}^2\big) \end{aligned}$$

For the remaining part, the reduction property (3.32) yields

$$\begin{aligned}&\widetilde{\rho}_+([\omega_+]([\mathcal{T}_+]\setminus[\mathcal{T}_\star]))^2 \\ &\qquad \le q_{\mathrm{red}}\, \widetilde{\rho}_\star([\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))^2 + C_{\mathrm{red}}\big(\|U_+ - U_\star\|_{H^{1/2}(\Gamma)}^2 + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)}^2\big) \end{aligned}$$

We define $C_{\mathrm{est}} := 2(1+\delta^{-1})\, C_{\mathrm{stab}}^2 + C_{\mathrm{red}}$ and end up with

$$\begin{aligned}\widetilde{\rho}_+^2 &\le (1+\delta)\widetilde{\rho}_\star([\mathcal{T}_\star]\setminus[\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))^2 + q_{\mathrm{red}}\, \widetilde{\rho}_\star([\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))^2 \\ &\qquad + C_{\mathrm{est}}\big(\|U_+ - U_\star\|_{H^{1/2}(\Gamma)}^2 + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)}^2\big). \end{aligned}$$

The assumption $\theta\widetilde{\rho}_\star^2 \le \widetilde{\rho}_\star([\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))^2$ implies

$$\begin{aligned}(1+\delta)\widetilde{\rho}_\star([\mathcal{T}_\star]\setminus[\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))^2 + q_{\mathrm{red}}\widetilde{\rho}_\star([\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))^2 \\ &= (1+\delta)\widetilde{\rho}_\star^2 - (1+\delta-q_{\mathrm{red}})\widetilde{\rho}_\star([\omega_\star]([\mathcal{T}_\star]\setminus[\mathcal{T}_+]))^2 \\ &\le \big(1+\delta-\theta(1+\delta-q_{\mathrm{red}})\big)\widetilde{\rho}_\star^2.\end{aligned}$$

For sufficiently small $\delta > 0$, this proves the estimator reduction with $q_{\mathrm{est}} := 1+\delta-\theta(1+\delta-q_{\mathrm{red}})$. □

3.5 Adaptive algorithm

We consider the following adaptive algorithm:

INPUT: adaptivity parameter $0 < \theta \le 1$, polynomial order $p \in \mathbb{N}$, initial mesh $[\mathcal{T}_0]$, inital weights $\mathcal{W}_0$, $\ell := 0$.

(i) Compute discrete approximation ϕ_ℓ.

(ii) Compute the Galerkin solution $U_\ell \in \mathcal{X}_\ell$ of (3.2).

(iii) Compute refinement indicators $\mu_\ell(z)$ for all $z \in \mathcal{N}_\ell$.

(iv) Determine an up to the multiplicative constant $C_{\mathrm{mark}} > 0$ minimal set of nodes $\mathcal{M}_\ell \subseteq \mathcal{N}_\ell$ such that

$$\theta\mu_\ell^2 \le \sum_{z\in\mathcal{M}_\ell} \mu_\ell(z)^2. \tag{3.44}$$

(v) Generate the refined mesh $[\mathcal{T}_{\ell+1}] := \mathrm{ref}\,([\mathcal{T}_\ell], \mathcal{M}_\ell)$.

(vi) Increment $\ell \leftarrow \ell+1$ and go to (i).

OUTPUT: Galerkin solutions U_ℓ and error estimators μ_ℓ for all $\ell \in \mathbb{N}_0$.

4 Linear convergence of adaptive IGABEM

The following theorem and corollary are the analogons to [FFK+15, Theorem 4.2] resp. [FFK+15, Corollary 4.3].

Theorem 4.1 *Let $[\mathcal{T}_+] \in \operatorname{ref}([\mathcal{T}_\star])$ for some $[\mathcal{T}_\star] \in [\mathbb{T}]$ and $U_\star \in \mathcal{X}_\star$ resp. $U_+ \in \mathcal{X}_+$ the corresponding Galerkin solutions to* (3.2)*. By $u_\star, u_+ \in H^{1/2}(\Gamma)$, we denote the unique solutions to*

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

Suppose that the set $[\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])$ satisfies the Dörfler marking

$$\theta\, \widetilde{\rho}_\star^2 \le \widetilde{\rho}_\star([\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+]))^2 \tag{4.1}$$

for some $0 < \theta \le 1$. Then, there are constants $\alpha, \beta > 0$ and $0 < q_{\mathrm{lin}} < 1$ such that the quasi-errors

$$\Delta_\star := |\!|\!| u_\star - U_\star |\!|\!|_{W+S}^2 + \alpha \widetilde{\rho}_\star^2 + \beta \widetilde{\operatorname{osc}}_\star^2,$$
$$\Delta_+ := |\!|\!| u_+ - U_+ |\!|\!|_{W+S}^2 + \alpha \widetilde{\rho}_+^2 + \beta \widetilde{\operatorname{osc}}_+^2$$

satisfy the contraction property

$$\Delta_+ \le q_{\mathrm{lin}}\, \Delta_\star. \tag{4.2}$$

The constants $\alpha, \beta > 0$ and $0 < q_{\mathrm{lin}} < 1$ depend only on C_{drl} from (3.33)*, C_{est} and q_{est} from* (3.41)*, as well as on Γ.*

Proof. We split the proof into six steps:
Step 1: First, we recall the norm equivalence $|\!|\!| \cdot |\!|\!|_{W+S} \simeq \|\cdot\|_{H^{1/2}(\Gamma)}$, i.e.,

$$C_1^{-1} \|v\|_{H^{1/2}(\Gamma)} \le |\!|\!| v |\!|\!|_{W+S} \le C_1 \|v\|_{H^{1/2}(\Gamma)} \quad \text{for all } v \in H^{1/2}(\Gamma), \tag{4.3}$$

where $C_1 > 0$ depends only on Γ.
Step 2: We show that

$$|\!|\!| u_+ - u_\star |\!|\!|_{W+S}^2 \lesssim \widetilde{\operatorname{osc}}_\star([\mathcal{T}_\star] \setminus [\mathcal{T}_+])^2.$$

It holds that

$$\begin{aligned}
\|u_+ - u_\star\|_{H^{1/2}(\Gamma)}^2 &\simeq |\!|\!| u_+ - u_\star |\!|\!|_{W+S}^2 = \langle\!\langle u_+ - u_\star; u_+ - u_\star\rangle\!\rangle_{W+S} \\
&= \langle (1/2 - K')\phi_+ - (1/2 - K')\phi_\star; u_+ - u_\star\rangle_\Gamma \\
&\lesssim \|(1/2 - K')\phi_+ - (1/2 - K')\phi_\star\|_{H^{-1/2}(\Gamma)} \|u_+ - u_\star\|_{H^{1/2}(\Gamma)} \\
&\lesssim \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)} \|u_+ - u_\star\|_{H^{1/2}(\Gamma)},
\end{aligned}$$

which is equivalent to

$$\|u_+ - u_\star\|_{H^{1/2}(\Gamma)}^2 \lesssim \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)}^2.$$

Together with the norm equivalence and the discrete reliability (3.30) of the data oscillation, we have with a constant $C_2 > 0$

$$|\!|\!| u_+ - u_\star |\!|\!|_{W+S}^2 \le C_2\, \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)}^2 \le C_2\, C_{\widetilde{\operatorname{osc}}}^2\, \widetilde{\operatorname{osc}}_\star([\mathcal{T}_\star] \setminus [\mathcal{T}_+])^2, \tag{4.4}$$

where $C_2 > 0$ depends only on Γ.
Step 3: The next step is to derive an estimation for $|\!|\!| u_+ - U_+ |\!|\!|_{W+S}^2$. We recall the Galerkin orthogonality,

$$\langle\!\langle u_+ - U_+; V_+\rangle\!\rangle_{W+S} = 0 \quad \text{for all } V_+ \in \mathcal{X}_+,$$

which yields the Pythagorean identity

$$\| u_+ - U_+ \|_{W+S}^2 = \| u_+ - U_\star \|_{W+S}^2 - \| U_+ - U_\star \|_{W+S}^2. \tag{4.5}$$

For $\epsilon > 0$, the triangle inequality and Young's inequality yield

$$\begin{aligned} \| u_+ - U_\star \|_{W+S}^2 &\le \big(\| u_+ - u_\star \|_{W+S} + \| u_\star - U_\star \|_{W+S} \big)^2 \\ &\le (1+\epsilon) \| u_\star - U_\star \|_{W+S}^2 + (1+\epsilon^{-1}) \| u_+ - u_\star \|_{W+S}^2. \end{aligned} \tag{4.6}$$

Next, we combine (4.4), (4.5) and (4.6), which leads to

$$\begin{aligned} \| u_+ - U_+ \|_{W+S}^2 \le (1+\epsilon) \| u_\star - U_\star \|_{W+S}^2 - \| U_+ - U_\star \|_{W+S}^2 \\ + (1+\epsilon^{-1}) C_2 C_{\widetilde{\mathrm{osc}}}^2 \, \widetilde{\mathrm{osc}}_\star([\mathcal{T}_\star] \setminus [\mathcal{T}_+])^2. \end{aligned} \tag{4.7}$$

Step 4: Now, we use Step 3 to get a first estimation for Δ_+. To this end, we define $C_3 := (1 + \epsilon^{-1}) \, C_2 \, C_{\widetilde{\mathrm{osc}}}^2 + \alpha \, C_{\mathrm{est}} \, C_{\widetilde{\mathrm{osc}}}^2$ and use the estimator reduction (3.41), (4.7), the norm equivalence (4.3), and (4.4) to see

$$\begin{aligned} \Delta_+ &\le (1+\epsilon) \| u_\star - U_\star \|_{W+S}^2 - \| U_+ - U_\star \|_{W+S}^2 + (1+\epsilon^{-1}) C_2 C_{\widetilde{\mathrm{osc}}}^2 \, \widetilde{\mathrm{osc}}_\star([\mathcal{T}_\star] \setminus [\mathcal{T}_+])^2 \\ &\quad + \alpha \, q_{\mathrm{est}} \, \widetilde{\rho}_\star^2 + \alpha C_{\mathrm{est}} \big(\|U_+ - U_\star\|_{H^{1/2}(\Gamma)}^2 + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)}^2 \big) + \beta \, \widetilde{\mathrm{osc}}_+^2 \\ &\le (1+\epsilon) \| u_\star - U_\star \|_{W+S}^2 + \alpha \, q_{\mathrm{est}} \, \widetilde{\rho}_\star^2 + C_3 \, \widetilde{\mathrm{osc}}_\star([\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+]))^2 + \beta \, \widetilde{\mathrm{osc}}_+^2 \\ &\quad + (\alpha C_{\mathrm{est}} C_1^2 - 1) \| U_+ - U_\star \|_{W+S}^2. \end{aligned} \tag{4.8}$$

Step 5: In this step, we fix the free parameters $\alpha, \beta > 0$. Let $\alpha := C_{\mathrm{est}}^{-1} C_1^{-2}$, wherefore the last term in (4.8) vanishes.
We recall that $\widetilde{h}_+ \le q_{\mathrm{ctr}} \widetilde{h}_\star$ on $\omega_+([\mathcal{T}_+] \setminus [\mathcal{T}_\star]) = \omega_\star([\mathcal{T}_\star] \setminus [\mathcal{T}_+])$, while $\widetilde{h}_+ = \widetilde{h}_\star$ on $\Gamma \setminus \omega_+([\mathcal{T}_+] \setminus [\mathcal{T}_\star])$, see Proposition 2.3. Let $\chi_{\omega_+([\mathcal{T}_+] \setminus [\mathcal{T}_\star])}$ be the characteristic function on $\omega_+([\mathcal{T}_+] \setminus [\mathcal{T}_\star])$. Hence, it holds

$$(1 - q_{\mathrm{ctr}}) \widetilde{h}_\star \chi_{\omega_+([\mathcal{T}_+] \setminus [\mathcal{T}_\star])} \le \widetilde{h}_\star - \widetilde{h}_+ \quad \text{pointwise a.e. on } \Gamma.$$

Together with the best approximation property of Π_+, this yields

$$\begin{aligned} (1 - q_{\mathrm{ctr}}) \widetilde{\mathrm{osc}}_\star([\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+]))^2 &= \sum_{[T] \in [\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])} \|(1 - q_{\mathrm{ctr}})^{1/2} \, \widetilde{h}_\star^{1/2} (\phi - \phi_\star)\|_{L^2(T)}^2 \\ &\le \sum_{[T] \in [\mathcal{T}_\star]} \|(\widetilde{h}_\star - \widetilde{h}_+)^{1/2} (\phi - \phi_\star)\|_{L^2(T)}^2 \\ &\le \|\widetilde{h}_\star^{1/2} (\phi - \phi_\star)\|_{L^2(\Gamma)}^2 - \|\widetilde{h}_+^{1/2} (\phi - \phi_+)\|_{L^2(\Gamma)}^2 \\ &= \widetilde{\mathrm{osc}}_\star^2 - \widetilde{\mathrm{osc}}_+^2 \end{aligned}$$

Step 4 shows

$$\begin{aligned} \Delta_+ &\le (1+\epsilon) \| u_\star - U_\star \|_{W+S}^2 + \alpha \, q_{\mathrm{est}} \, \widetilde{\rho}_\star^2 + C_3 \, \widetilde{\mathrm{osc}}_\star([\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+]))^2 + \beta \, \widetilde{\mathrm{osc}}_+^2 \\ &\le (1+\epsilon) \| u_\star - U_\star \|_{W+S}^2 + \alpha \, q_{\mathrm{est}} \, \widetilde{\rho}_\star^2 + C_3 (1 - q_{\mathrm{ctr}})^{-1} (\widetilde{\mathrm{osc}}_\star^2 - \widetilde{\mathrm{osc}}_+^2) + \beta \, \widetilde{\mathrm{osc}}_+^2 \end{aligned}$$

We set $\beta := C_3 (1 - q_{\mathrm{ctr}})^{-1}$ and end up with

$$\Delta_+ \le (1+\epsilon) \| u_\star - U_\star \|_{W+S}^2 + \alpha \, q_{\mathrm{est}} \, \widetilde{\rho}_\star^2 + \beta \, \widetilde{\mathrm{osc}}_\star^2. \tag{4.9}$$

Step 6: Since $u_\star$ and $U_\star$ are determined by the same right-hand side $f = (1/2 - K') \Pi_\star \phi$, norm equivalence (4.3) together with reliability (3.40) yields

$$C_4 \| u_\star - U_\star \|_{W+S}^2 \le \widetilde{\rho}_\star^2,$$

where $C_4 := C_1^{-2} \, C_{\mathrm{drl}}^{-2}$.
Obviously, $\widetilde{\mathrm{osc}}_\star^2 \le \widetilde{\rho}_\star^2$. Hence, we get from (4.9) for $\delta > 0$ that

$$\Delta_+ \le (1 + \epsilon - \alpha \, \delta \, C_4) \| u_\star - U_\star \|_{W+S}^2 + \alpha \, (q_{\mathrm{est}} + 2\delta) \widetilde{\rho}_\star^2 + (\beta - \alpha \, \delta) \widetilde{\mathrm{osc}}_\star^2 \le q_{\mathrm{lin}} \Delta_\star,$$

where

$$q_{\rm lin} := \max\{1+\epsilon-\alpha\,\delta\,C_4, q_{\rm est}+2\delta, (\beta-\alpha\,\delta)/\beta\}.$$

The choice $\delta < (1-q_{\rm est})/2$ and $\epsilon < \alpha\,\delta\,C_4$ finally concludes the proof with $0 < q_{\rm lin} < 1$. □

As a direct consequence, we get the following corollary, which states the linear convergence of the node-based error estimator $\mu_\star$.

Corollary 4.2 *Suppose that* (M1) *from Assumption 2.2 is satisfied for the mesh-refinement. Then for each* $0 < \theta \le 1$*, Algorithm 3.5 guarantees linear convergence of the error estimator sequence, i.e., there exist constants* $0 < q_{\rm lin} < 1$ *and* $C_{\rm lin} > 0$ *such that*

$$\mu_{\ell+k} \le C_{\rm lin}\, q_{\rm lin}^k\, \mu_\ell \quad \text{for all } k,\ell \in \mathbb{N}_0. \tag{4.10}$$

The constants $q_{\rm lin}$ *and* $C_{\rm lin}$ *depend only on* $p, w_{\min}, w_{\max}, \gamma, \theta$*, and* $\check{\kappa}_{\max}$*.*

Proof. Local equivalence of $\widetilde{\rho}_\ell$ and μ_ℓ, the Dörfler marking (3.44) and $\mathcal{M}_\ell \subseteq \omega_\ell([\mathcal{T}_\ell]\setminus[\mathcal{T}_{\ell+1}])$ imply

$$\theta\widetilde{\rho}_\ell^2 \simeq \theta\mu_\ell^2 \le \sum_{z\in\mathcal{M}_\ell} \mu_\ell(z)^2 \simeq \sum_{\substack{[T]\in[\mathcal{T}_\ell]\\ T\subseteq\omega_\ell(\mathcal{M}_\ell)}} \widetilde{\rho}_\ell([T])^2 \le \sum_{[T]\in[\omega_\ell]([\mathcal{T}_\ell]\setminus[\mathcal{T}_{\ell+1}])} \widetilde{\rho}_\ell([T])^2 = \widetilde{\rho}_\ell([\omega_\ell]([\mathcal{T}_\ell]\setminus[\mathcal{T}_{\ell+1}]))^2$$

where the hidden constants depend only on $\check{\kappa}_{\max}, p$, and γ. Hence, $\widetilde{\rho}_\ell$ satisfies the Dörfler marking with some parameter $0 < \widetilde{\theta} < 1$, and Theorem 4.1 applies. Inductively, the contraction property (4.2) leads to

$$\Delta_{\ell+k} \le q_{\rm lin}^k\, \Delta_\ell \quad \text{for all } \ell,k \in \mathbb{N}_0,$$

and the norm equivalence $|\!|\!|\cdot|\!|\!|_{W+S} \simeq \|\cdot\|_{H^{1/2}(\Gamma)}$ combined with the reliability (3.40) provides

$$\widetilde{\rho}_\ell^2 \lesssim \Delta_\ell \lesssim \widetilde{\rho}_\ell^2.$$

From this, we infer

$$\mu_{\ell+k} \simeq \widetilde{\rho}_{\ell+k}^2 \simeq \Delta_{\ell+k} \le q_{\rm lin}^k\, \Delta_\ell \simeq q_{\rm lin}^k\, \widetilde{\rho}_\ell^2 \simeq q_{\rm lin}^k\, \mu_\ell^2 \quad \text{for all } \ell,k \in \mathbb{N}_0$$

and hence conclude the proof. □

5 Optimal convergence of adaptive IGABEM

In this section, we want to show that the proposed Algorithm 3.5 converges even with the optimal algebraic rate in the following sense: Following [FGHP16b, Chapter 3] and [FFK+15, Chapter 5], let

$$[\mathbb{T}_N] := \{[\mathcal{T}_\star] \in [\mathbb{T}] \,:\, |\mathcal{K}_\star| - |\mathcal{K}_0| \le N\} \tag{5.1}$$

be the finite set of all refinements having at most N knots more than $[\mathcal{T}_0]$. For $s > 0$, we define

$$\mathbb{A}_s := \Big\{ u \in H^{1/2}(\Gamma) \,:\, \|u\|_{\mathbb{A}_s} < \infty \Big\} \tag{5.2}$$

with

$$\|u\|_{\mathbb{A}_s} := \sup_{N \in \mathbb{N}_0} \big((N+1)^s \min_{[\mathcal{T}_\star] \in [\mathbb{T}_N]} \mu_\star \big). \tag{5.3}$$

This means, that the error estimator μ can provide an algebraic convergence rate of $\mathcal{O}(N^{-s})$, if the optimal meshes are chosen, which are not necessarily nested. The main result states that the sequence of estimators μ_ℓ generated by Algorithm 3.5 also decays asymptotically with any rate $s > 0$, if $\|u\|_{\mathbb{A}_s} < \infty$.

Up to now, the assumption (M1) of Assumption 2.2 was enough. For the optimality, we also need the assumptions (M2)-(M3) on the mesh-refinement.

The proof of the next lemma follows the the ideas of [FFK+15, Lemma 5.2].

Lemma 5.1 (Optimality of Dörfler marking) *With the constants $C_{\rm stab}$ from*(3.31) *and $C_{\rm drl}$ from* (3.33)*, we set*

$$\widetilde{\theta}_{\rm opt} := (1 + C_{\rm stab}^2\, C_{\rm drl}^2)^{-1}. \tag{5.4}$$

For all $0 < \widetilde{\theta} < \widetilde{\theta}_{\rm opt}$, there exists $0 < q_{\rm opt} < 1$ such that for any $[\mathcal{T}_\star] \in [\mathbb{T}]$ and $[\mathcal{T}_+] \in {\rm ref}\,([\mathcal{T}_\star])$ the following implication is satisfied:

$$\widetilde{\rho}_+^2 \le q_{\rm opt}\, \widetilde{\rho}_\star^2 \quad \Longrightarrow \quad \widetilde{\theta}\, \widetilde{\rho}_\star^2 \le \widetilde{\rho}_\star\big([\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big)^2. \tag{5.5}$$

The constant $0 < q_{\rm opt} < 1$ depends only on $\widetilde{\theta}$, $C_{\rm stab}$, and $C_{\rm drl}$.

Proof. From the stability property (3.31) of the error estimator and Young's inequality, we get for $\delta > 0$

$$\begin{aligned}
\widetilde{\rho}_\star^2 &= \widetilde{\rho}_\star\big([\mathcal{T}_\star] \setminus [\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big)^2 + \widetilde{\rho}_\star\big([\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big)^2 \\
&\le (1+\delta)\widetilde{\rho}_+\big([\mathcal{T}_+] \setminus [\omega_+]([\mathcal{T}_+] \setminus [\mathcal{T}_\star])\big)^2 + \widetilde{\rho}_\star\big([\omega_\star]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big)^2 \\
&\qquad + (1+\delta^{-1}) C_{\rm stab}^2 \left(\|U_+ - U_\star\|_{H^{1/2}(\Gamma)} + \|\phi_+ - \phi_\star\|_{H^{-1/2}(\Gamma)} \right)^2.
\end{aligned}$$

Now, we take the assumption $\widetilde{\rho}_+^2 \le q_{\rm opt}\, \widetilde{\rho}_\star^2$ into account, where $q_{\rm opt}$ is fixed later on. Together with the discrete reliability (3.33) and the last inequality, we end up with

$$\widetilde{\rho}_\star^2 \le (1+\delta) q_{\rm opt}\, \widetilde{\rho}_\star^2 + (1 + (1+\delta^{-1})\, C_{\rm stab}^2\, C_{\rm drl}^2) \widetilde{\rho}_\star\big([\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big)^2.$$

Hence, this yields

$$\widetilde{\theta}\, \widetilde{\rho}_\star^2 \le \widetilde{\rho}_\star\big([\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big)^2 \quad \text{for all } 0 < \widetilde{\theta} \le \frac{1 - (1+\delta) q_{\rm opt}}{1 + (1+\delta^{-1}) C_{\rm stab}^2\, C_{\rm drl}^2}.$$

For each $0 < \widetilde{\theta} < \widetilde{\theta}_{\rm opt}$, there exist $\delta > 0$ and $0 < q_{\rm opt} < 1$ such that $\widetilde{\theta} < \frac{1-(1+\delta)q_{\rm opt}}{1+(1+\delta^{-1})C_{\rm stab}^2\, C_{\rm drl}^2}$, which concludes the proof. □

The crucial part of the proof of the main result is the following lemma. It is the analogous version of [FGHP16b, Lemma 5.6] and its proof is more or less verbatim the same.

Lemma 5.2 *Let $s > 0$, $u \in \mathbb{A}_s$ and $0 < \widetilde{\theta} < \widetilde{\theta}_{\rm opt}$. Then, there exist constants $C_5, C_6 > 0$ such that for all $[\mathcal{T}_\star] \in [\mathbb{T}]$ there is a refinement $[\mathcal{T}_+] \in \operatorname{ref}([\mathcal{T}_\star])$ with*

$$\big|[\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big| \le C_5\, C_6^{1/s}\, \|u\|_{\mathbb{A}_s}^{1/s}\, \widetilde{\rho_\star}^{-1/s}, \tag{5.6}$$

and

$$\widetilde{\theta}\, \widetilde{\rho}_\star^2 \le \widetilde{\rho}_\star\big([\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+])\big)^2. \tag{5.7}$$

The constants C_5 and C_6 depend only on $C_{\rm drl}$ from (3.33), $C_{\rm mon}$ from (3.39), $q_{\rm opt}$ from (5.5), and p.

Proof. Define $\zeta := C_{\rm mon}^{-2}\, q_{\rm opt}$ and $\delta^2 := \zeta\, \widetilde{\rho}_\star^2$. We split the proof into three steps:
Step 1: There exists a mesh $[\mathcal{T}_\delta] \in [\mathbb{T}]$ such that

$$\widetilde{\rho}_\delta \le \delta \quad \text{and} \quad |\mathcal{K}_\delta| - |\mathcal{K}_0| \le \|u\|_{\mathbb{A}_s}^{1/s}\, \delta^{-1/s}.$$

Let $N \in \mathbb{N}_0$ be minimal with $\|u\|_{\mathbb{A}_s} \le (N+1)^s\, \delta$. If $N = 0$, we can set $[\mathcal{T}_\delta] = [\mathcal{T}_0]$ due to $\widetilde{\rho}_0 \le \|u\|_{\mathbb{A}_s} \le \delta$. If $N > 0$, minimality of N yields $\|u\|_{\mathbb{A}_s}^{1/s}\, \delta^{-1/s} > N$. Now, we choose $[\mathcal{T}_\delta] \in [\mathbb{T}]$ such that

$$\widetilde{\rho}_\delta = \min_{[\mathcal{T}_\bullet] \in [\mathbb{T}_N]} \widetilde{\rho}_\bullet.$$

By definition of $\|\cdot\|_{\mathbb{A}_s}$, it follows that

$$\widetilde{\rho}_\delta \le (N+1)^{-s} \|u\|_{\mathbb{A}_s} \le \delta.$$

By definition of $[\mathbb{T}_N]$, we get

$$|\mathcal{K}_\delta| - |\mathcal{K}_0| \le N < \|u\|_{\mathbb{A}_s}^{1/s}\, \delta^{-1/s}.$$

Step 2: We take the common refinement $[\mathcal{T}_+] := [\mathcal{T}_\star \oplus \mathcal{T}_\delta]$ of (M2). Now, the assumptions on the refinement come into play. The overlay estimate (M2) and Step 1 imply

$$|\mathcal{K}_+| - |\mathcal{K}_\star| \le \big(|K_\delta| + |\mathcal{K}_\star| - |\mathcal{K}_0|\big) - |\mathcal{K}_\star| = |\mathcal{K}_\delta| - |\mathcal{K}_0| < \|u\|_{\mathbb{A}_s}^{1/s}\, \delta^{-1/s}.$$

With estimate (2.14), we see

$$\big|[\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+]))\big| \lesssim \big|[\mathcal{T}_\star] \setminus [\mathcal{T}_+]\big| \le 2\big(|\mathcal{K}_+| - |\mathcal{K}_\star|\big).$$

Combining the last two estimates, we get

$$\big|[\omega_\star^{2p}]([\mathcal{T}_\star] \setminus [\mathcal{T}_+]))\big| \lesssim \|u\|_{\mathbb{A}_s}^{1/s}\, \zeta^{-1/(2s)}\, \widetilde{\rho_\star}^{-1/s},$$

which proves (5.6).
Step 3: The quasi-monotonicity (3.39) yields

$$\widetilde{\rho}_+^2 \le C_{\rm mon}^2\, \widetilde{\rho}_\delta^2 \le C_{\rm mon}^2\, \delta^2 = q_{\rm opt}\, \widetilde{\rho}_\star^2.$$

Finally, we apply Lemma 5.1 to see (5.7). □

Finally, we state our main result, cf. [FGHP16b, Theorem 3.2] resp. [FFK+15, Theorem 5.1].

Theorem 5.3 *Under Assumption 2.2 on the mesh-refinement strategy, there exists a constant $0 < \theta_{\rm opt} < 1$ such that for all $0 < \theta < \theta_{\rm opt}$, there exists a constant $C_{\rm opt} > 0$ such that, for all $s > 0$ the following implication holds:*

$$u \in \mathbb{A}_s \quad \Longrightarrow \quad \mu_\ell \le \frac{C_{\rm opt}^{1+s}}{(1 - q_{\rm lin}^{1/s})^s} \|u\|_{\mathbb{A}_s} \big(|\mathcal{K}_\ell| - |\mathcal{K}_0|\big)^{-s} \quad \text{for all } \ell \in \mathbb{N}_0 \tag{5.8}$$

with the constant $q_{\rm lin}$ from (4.10). The constant $\theta_{\rm opt}$ depends only on $\widetilde{\theta}_{\rm opt}$ from (5.4) and $C_{\rm loc}$ from (3.8). Moreover, $C_{\rm opt}$ depends only on $C_{\rm mesh}$ from (2.17), $C_{\rm loc}$ from (3.8), $C_{\rm mark}$ from (3.44), $C_{\rm lin}$ from (4.10), and C_5, C_6 from (5.6).

Proof. Let $C_{\rm loc} > 0$ be the constant from (3.8). Define $\theta_{\rm opt} := \widetilde{\theta}_{\rm opt}/C_{\rm loc}^2$ and $\widetilde{\theta} := C_{\rm loc}^2\,\theta$. In addition, assume that θ is sufficiently small, i.e., $\theta < \theta_{\rm opt}$. Hence, $\widetilde{\theta} < \widetilde{\theta}_{\rm opt}$. We let $\ell \in \mathbb{N}_0$, $j \le \ell$ and choose a refinement $[\mathcal{T}_+] \in {\rm ref}\,([\mathcal{T}_j])$ according to Lemma 5.2. With (3.8), we get

$$\begin{aligned}\theta\,\mu_j^2 \le C_{\rm loc}^{-1}\,\widetilde{\theta}\,\widetilde{\rho}_j^2 &\le C_{\rm loc}^{-1}\,\widetilde{\rho}_j([\omega_j^{2p}]([\mathcal{T}_j]\setminus[\mathcal{T}_+])))^2\\ &= C_{\rm loc}^{-1}\sum_{[T]\in[\omega_j^{2p}]([\mathcal{T}_j]\setminus[\mathcal{T}_+])}\widetilde{\rho}_j([T])^2 \le \sum_{z\in\mathcal{N}_j\cap\omega_j^{2p}([\mathcal{T}_j]\setminus[\mathcal{T}_+])}\mu_j(z)^2.\end{aligned}$$

We see, that $\mathcal{N}_j\cap\omega_j^{2p}([T_j]\setminus[\mathcal{T}_+])$ satisfies the Dörfler marking of Algorithm 3.5. Moreover, the chosen set $\mathcal{M}_j$ of Algorithm 3.5 has minimal cardinality. Hence, we derive with (5.6) and the constant $C_{\rm mark} > 0$ of Algorithm 3.5 that

$$|\mathcal{M}_j| \le C_{\rm mark}\,\big|\mathcal{N}_j\cap\omega_j^{2p}([T_j]\setminus[\mathcal{T}_+])\big| \le 2\,C_{\rm mark}\,\big|[\omega_j^{2p}]([T_j]\setminus[\mathcal{T}_+])\big| \le 2\,C_{\rm mark}\,C_5\,C_6^{1/s}\,\|u\|_{\mathbb{A}_s}^{1/s}\,\widetilde{\rho}_j^{-1/s}.$$

Next, we use the mesh-closure estimate (M3) and end up with

$$\begin{aligned}|\mathcal{K}_\ell| - |\mathcal{K}_0| \le C_{\rm mesh}\sum_{j=0}^{\ell-1}|\mathcal{M}_j| &\le 2\,C_{\rm mark}\,C_{\rm mesh}\,C_5\,C_6^{1/s}\,\|u\|_{\mathbb{A}_s}^{1/s}\sum_{j=0}^{\ell-1}\widetilde{\rho}_j^{-1/s}\\ &\le 2\,C_{\rm mark}\,C_{\rm mesh}\,C_5\,C_6^{1/s}\,C_{\rm loc}^{1/s}\,\|u\|_{\mathbb{A}_s}^{1/s}\sum_{j=0}^{\ell-1}\mu_j^{-1/s}.\end{aligned}$$

With the linear convergence (4.10), we see

$$\mu_\ell \le C_{\rm lin}\,q_{\rm lin}^{\ell-j}\,\mu_j \quad \text{for all } j = 0,\dots,\ell.$$

Hence, with the help of the geometric series, this finally yields

$$\begin{aligned}|\mathcal{K}_\ell| - |\mathcal{K}_0| &\le 2\,C_{\rm mark}\,C_{\rm mesh}\,C_5\,C_6^{1/s}\,C_{\rm loc}^{1/s}\,\|u\|_{\mathbb{A}_s}^{1/s}\sum_{j=0}^{\ell-1}\mu_j^{-1/s}\\ &\le 2\,C_{\rm mark}\,C_{\rm mesh}\,C_5\,(C_6\,C_{\rm lin}\,C_{\rm loc})^{1/s}\|u\|_{\mathbb{A}_s}^{1/s}\,\mu_\ell^{-1/s}\sum_{j=0}^{\ell-1}(q_{\rm lin}^{1/s})^{\ell-j}\\ &\le (C_6\,C_{\rm lin}\,C_{\rm loc})^{1/s}\,\frac{2\,C_{\rm mark}\,C_{\rm mesh}\,C_5}{1-q_{\rm lin}^{1/s}}\|u\|_{\mathbb{A}_s}^{1/s}\,\mu_\ell^{-1/s}.\end{aligned}$$

The latter is equivalent to

$$\mu_\ell \le \frac{C_6\,C_{\rm lin}\,C_{\rm loc}\,(2\,C_{\rm mark}\,C_{\rm mesh}\,C_5)^s}{(1-q_{\rm lin}^{1/s})^s}\|u\|_{\mathbb{A}_s}\big(|\mathcal{K}_\ell| - |\mathcal{K}_0|\big)^{-s},$$

which concludes the proof. □

6 Numerical approximation

This section is closely related to [Gan14, Section 5], where the numerical solution of Symm's integral equation is discussed. However, most of the appearing integrals are of the same type and therefore, they are treated in a similar way.

Let γ be the parametrization of the boundary of the open set $\Omega \subset \mathbb{R}^2$ of Section 1. For $a = \check{z}_0^\gamma < \cdots < \check{z}_{n_\gamma}^\gamma = b$, we assume, that $\gamma|_{[\check{z}_{j-1}^\gamma, \check{z}_j^\gamma]}$ is even two times continuously differentiable for $j = 1, \ldots, n_\gamma$, and, additionally, that γ is positively orientated. The latter means, that the outer normal vector ν at any point $x = \gamma(t) = \begin{pmatrix} \gamma_1(t) \\ \gamma_2(t) \end{pmatrix} \in \Gamma \setminus \{\gamma(\check{z}_1^\gamma), \ldots, \gamma(\check{z}_{n_\gamma}^\gamma)\}$ satisfies

$$\nu(x) = \frac{1}{|\gamma'(t)|} \begin{pmatrix} \gamma_2'(t) \\ -\gamma_1'(t) \end{pmatrix}.$$

We consider the equivalent variational formulation (1.11) of the hyper-singular integral equation

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

and its boundary element discretization via the finite dimensional subspace $\mathcal{X}_h \subset H^{1/2}(\Gamma)$ with $\phi_h := \phi$ or $\phi_h = \Pi_h \phi$, where $\Pi_h \colon L^2(\Gamma) \to \mathcal{P}^p(\mathcal{T}_h)$ is the L^2-orthogonal projection onto $\mathcal{P}^p(\mathcal{T}_h)$. The discrete solution $U_h \in \mathcal{X}_h$ then solves

$$\langle\!\langle U_h; V_h \rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi_h; V_h \rangle_\Gamma \quad \text{for all } V_h \in \mathcal{X}_h. \tag{6.1}$$

To calculate $\Pi_h \phi$, we solve the linear system

$$\langle \Pi_h \phi; \Psi_h \rangle_\Gamma = \langle \phi; \Psi_h \rangle_\Gamma \quad \text{for all } \Psi_h \in \mathcal{P}^p(\mathcal{T}_h).$$

We use the notation of Section 2.5 and abbreviate

$$\widehat{B}_{i,p} := B_{i,p}^{\check{\mathcal{K}}_{h,p+1}}|_{[a,b)} \circ \gamma|_{[a,b)}^{-1} \quad \text{for } i = 1-p, \ldots, N_{p+1} - p.$$

Now, we use the ansatz $\Pi_h \phi = \sum_{i=1-p}^{N_{p+1}-p} x_{h,i} \widehat{B}_{i,p}$. Hence, we have for $j = 1-p, \ldots, N_{p+1} - p$

$$\sum_{i=1-p}^{N_{p+1}-p} x_{h,i} \langle \widehat{B}_{i,p}; \widehat{B}_{j,p} \rangle_\Gamma = \langle \phi; \widehat{B}_{j,p} \rangle_\Gamma,$$

which is a uniquely solvable system of linear equations.
Now, we recall the basis of $\mathcal{X}_h$, which we mentioned in Section 2.5

$$\left\{ R_{i,p}|_{[a,b)} \circ \gamma|_{[a,b)}^{-1} \,:\, i = 2-p, \ldots, N-p-1 \right\} \cup \left\{ \left(R_{1-p,p}|_{[a,b)} + R_{N-p,p}|_{[a,b)} \right) \circ \gamma|_{[a,b)}^{-1} \right\}.$$

We introduce the following notation:

$$\begin{aligned} \overline{R}_{1-p} &:= (R_{1-p,p}|_{[a,b)} + R_{N-p,p}|_{[a,b)}) \circ \gamma|_{[a,b)}^{-1} \\ \overline{R}_i &:= R_{i,p}|_{[a,b)} \circ \gamma_{[a,b)}^{-1} \quad \text{for } i = 2-p, \ldots, N-p-1. \end{aligned}$$

Let $U_h \in \mathcal{X}_h$ be the solution to (6.1). Then, there exists a unique coefficient vector $c_h := (c_{h,1-p}, \ldots, c_{h,N-p-1})^T$ with $U_h = \sum_{i=1-p}^{N-p-1} c_{h,i} \overline{R}_i$.
Therefore, it holds for $j = 1-p, \ldots, N-p-1$ that

$$\sum_{i=1-p}^{N-p-1} c_{h,i} \langle\!\langle \overline{R}_i; \overline{R}_j \rangle\!\rangle_{W+S} = \langle (1/2 - K')\phi; \overline{R}_j \rangle_\Gamma$$

or explicitly,

$$\sum_{i=1-p}^{N-p-1} c_{h,i} \Big(\langle W\overline{R}_i; \overline{R}_j \rangle_\Gamma + \langle \overline{R}_i; 1 \rangle_\Gamma \langle \overline{R}_j; 1 \rangle_\Gamma \Big) = \langle (1/2 - K')\phi; \overline{R}_j \rangle_\Gamma.$$

With the real, symmetric, and positive definite matrix

$$W_h + S_h := \Big(\underbrace{\langle W\overline{R}_i; \overline{R}_j\rangle_\Gamma}_{=:W_h} + \underbrace{\langle \overline{R}_i; 1\rangle_\Gamma \langle \overline{R}_j; 1\rangle_\Gamma}_{=:S_h} \Big)_{i,j=1-p}^{N-p-1}$$

and the right-hand side vector

$$F_h := \Big(\langle (1/2 - K')\phi_h; \overline{R}_j\rangle_\Gamma \Big)_{j=1-p}^{N-p-1},$$

it holds that

$$(W_h + S_h)c_h = F_h.$$

Therefore, we have to solve this system of linear equations, to compute the solution U_h. Only for very special geometries, the arising integrals and/or double integrals can be computed analytically. To approximate these integrals numerically, we want to use a tensorial Gauss quadrature. For a continuous function $f \in C([0,1]^2)$ and weight functions $\theta_1, \theta_2 \in L^1([0,1])$, we set

$$I(f) := \int_{[0,1]} \int_{[0,1]} f(s,t)\theta_1(s)\theta_2(t)\, dt\, ds.$$

We define the tensorial Gauss quadrature of order $n := (n_1, n_2)$ for the nodes ξ_{1,q_1}, ξ_{2,q_2} and weights $\omega_{1,q_1}, \omega_{2,q_2}$ of the Gaussian quadrature on $[0,1]$ by

$$\begin{aligned} Q^n \colon C([0,1]^2) &\to \mathbb{R} \\ f &\mapsto \sum_{q_1=1}^{n_1} \sum_{q_2=1}^{n_2} f(\xi_{1,q_1}, \xi_{2,q_2})\omega_{1,q_1}\omega_{2,q_2}, \end{aligned}$$

and the quadrature error by

$$E^n := I - Q^n.$$

We denote for $g \in C([0,1])$

$$\begin{aligned} I_\ell g &:= \int_{[0,1]} g(r)\theta_\ell(r)\, dr, \\ Q_\ell^{n_\ell} g &:= \sum_{q_\ell=1}^{n_\ell} g(\xi_{\ell,q_\ell})\omega_{\ell,q_\ell}, \\ E_\ell^{n_\ell} &:= I_\ell - Q_\ell^{n_\ell} \quad \text{for } \ell = 1,2. \end{aligned}$$

The following theorem is important for the convergence of the quadrature error.

Theorem 6.1 *For arbitrary $f \in C([0,1]^2)$, there holds the error estimate*

$$|E^n f| \le \|\theta_1\|_{L^1([0,1])} \max_{t\in[0,1]} |E_2^{n_2} f(\cdot,t)| + \|\theta_2\|_{L^1([0,1])} \max_{s\in[0,1]} |E_1^{n_1} f(s,\cdot)|,$$

where the right-hand side converges to zero for $n_1 \to \infty$ and $n_2 \to \infty$.

Proof. As the proof of the inequality is shown in [SS11, Theorem 5.3.15], we will only prove the convergence of the right-hand side. We only consider the first summand, since the second works analogously. The aim is to show, that each subsequence of a given sequence has a subsequence itself, which converges to zero. As a consequence, the full sequence itself converges to zero. Let $(t_{n_2})_{n_2\in\mathbb{N}}$ be a sequence in $[0,1]$ such that

$$|E_2^{n_2} f(\cdot, t_{n_2})| = \max_{t\in[0,1]} |E_2^{n_2} f(\cdot,t)|,$$

and $|E_2^{n_2(k)} f(\cdot, t_{n_2(k)})|$ the according subsequence. The sequence $t_{n_2(k)}$ takes values in the compact intervall $[0,1]$. Therefore, there exists some $t_\infty \in [0,1]$ and a subsequence $t_{n_2(k(j))}$ such that $t_{n_2(k(j))}$ converges to t_∞ as $j \to \infty$. We get the following estimate

$$|E_2^{n_2(k(j))} f(\cdot, t_{n_2(k(j))})| \leq |E_2^{n_2(k(j))} f(\cdot, t_{n_2(k(j))}) - E_2^{n_2(k(j))} f(\cdot, t_\infty)| + |E_2^{n_2(k(j))} f(\cdot, t_\infty)|$$

The last summand of the right-hand side is just the quadrature error of a standard Gauss quadrature on $[0,1]$ with the weight function θ_2, which converges to 0, compare e.g. [Pra06, Chapter 6.3]. So the only remaining factor is the first term,

$$\begin{aligned}
&|E_2^{n_2(k(j))} f(\cdot, t_{n_2(k(j))}) - E_2^{n_2(k(j))} f(\cdot, t_\infty)| \\
&\quad \leq |I_2 f(\cdot, t_{n_2(k(j))}) - I_2 f(\cdot, t_\infty)| + |Q_2^{n_2(k(j))} f(\cdot, t_{n_2(k(j))}) - Q_2^{n_2(k(j))} f(\cdot, t_\infty)| \\
&\quad \leq 2\|\theta_2\|_{L^1([0,1])} \|f(\cdot, t_{n_2(k(j))}) - f(\cdot, t_\infty)\|_{L^\infty([0,1])}
\end{aligned}$$

The mapping $t \to f(\cdot, t)$ from $[0,1]$ to $C([0,1])$ is continuous. This yields the convergence to zero of the last term. This concludes our prove. □

Remark 6.2 *The previous lemma hints, why we consider the two different choices of ϕ_h, namely, $\phi_h := \phi$ or $\phi_h := \Pi_h \phi$. To get convergence of the quadrature error, we need certain smoothness of ϕ, see also Lemma 6.5–6.7. In practice, for the interesting cases, where adaptive refinement leads to a higher convergence order than uniform refinement, this smoothness is not given. This leads to numerical problems for the computation of the right-hand side F_h as well as the pointwise evaluation of $K'\phi(\cdot)$. Hence, we use the approximation $\Pi_h \phi$ instead. However, as a consequence, we have to take care of the additional data oscillations. As we have seen, the convergence theory therefore gets more complicated.*

6.1 Approximation of W_h

In this section, we follow the ideas of [Gan14, Section 5.1].

For $t \in [a,b]$ and $x = \gamma(t) \in \Gamma$, we use the following abbreviations

$$\begin{aligned}
\Gamma_\ell &:= \gamma([t_{\ell-1} t_\ell]), \\
\begin{pmatrix} \nu_1(x) \\ \nu_2(x) \end{pmatrix} &:= \nu(x) = \frac{1}{|\gamma'(t)|} \begin{pmatrix} \gamma_2'(t) \\ -\gamma_1'(t) \end{pmatrix}, \\
H_\ell &:= t_\ell - t_{\ell-1}, \\
\gamma_\ell(t) &:= \gamma(t_{\ell-1} + (t_\ell - t_{\ell-1})t), \\
\operatorname{curl}_\Gamma \overline{R}_\ell(x) &:= \nu_1(x) \frac{\partial}{\partial x_2} \overline{R}_\ell(x) - \nu_2(x) \frac{\partial}{\partial x_1} \overline{R}_\ell(x),
\end{aligned}$$

where $\overline{R}_\ell$ on the right hand side of $\operatorname{curl}_\Gamma \overline{R}_\ell$ is an appropriate extension to a neighbourhood, compare [Ste08, Section 6.5] for more details.

For $i, j = 2-p, \ldots, N-p-1$, it holds due to [Ste08, Theorem 6.15]

$$\begin{aligned}
\langle W\overline{R}_i; \overline{R}_j \rangle_\Gamma &= -\frac{1}{2\pi} \int_\Gamma \operatorname{curl}_\Gamma \overline{R}_j(x) \int_\Gamma \log|x-y| \operatorname{curl}_\Gamma \overline{R}_i(y)\, ds_y\, ds_x \\
&= -\frac{1}{2\pi} \sum_{k=1}^N \sum_{m=1}^N \int_{\Gamma_k} \operatorname{curl}_\Gamma \overline{R}_j(x) \int_{\Gamma_m} \log|x-y| \operatorname{curl}_\Gamma \overline{R}_i(y)\, ds_y\, ds_x.
\end{aligned}$$

If we insert the definition of the curl operator and the parametrization, we get

$$
\begin{aligned}
&= -\frac{1}{2\pi}\sum_{k=1}^{N}\sum_{m=1}^{N}\int_{\Gamma_k}[\nu_1(x)\frac{\partial}{\partial x_2}\overline{R}_j(x)-\nu_2(x)\frac{\partial}{\partial x_1}\overline{R}_j(x)]\\
&\quad\int_{\Gamma_m}\log|x-y|[\nu_1(y)\frac{\partial}{\partial y_2}\overline{R}_i(y)-\nu_2(y)\frac{\partial}{\partial y_1}\overline{R}_i(y)]\,ds_y\,ds_x\\
&= -\frac{1}{2\pi}\sum_{k=1}^{N}\sum_{m=1}^{N}\int_{[t_{k-1},t_k]}[\nu_1(\gamma(t))\frac{\partial}{\partial x_2}\overline{R}_j(\gamma(t))-\nu_2(\gamma(t))\frac{\partial}{\partial x_1}\overline{R}_j(\gamma(t))]\cdot|\gamma'(t)|\\
&\quad\int_{[t_{m-1},t_m]}\log|\gamma(t)-\gamma(s)|[\nu_1(\gamma(s))\frac{\partial}{\partial y_2}\overline{R}_i(\gamma(s))-\nu_2(\gamma(s))\frac{\partial}{\partial y_1}\overline{R}_i(\gamma(s))]\cdot|\gamma'(s)|\,ds\,dt.
\end{aligned}
$$

The characterization of ν and some further simplifications provide

$$
\begin{aligned}
&= -\frac{1}{2\pi}\sum_{k=1}^{N}\sum_{m=1}^{N}\int_{[t_{k-1},t_k]}[\gamma_2'(t)\frac{\partial}{\partial x_2}\overline{R}_j(\gamma(t))+\gamma_1'(t)\frac{\partial}{\partial x_1}\overline{R}_j(\gamma(t))]\\
&\quad\int_{[t_{m-1},t_m]}\log|\gamma(t)-\gamma(s)|[\gamma_2'(s)\frac{\partial}{\partial y_2}\overline{R}_i(\gamma(s))+\gamma_1'(s)\frac{\partial}{\partial y_1}\overline{R}_i(\gamma(s))]\,ds\,dt\\
&= -\frac{1}{2\pi}\sum_{k=1}^{N}\sum_{m=1}^{N}\int_{[t_{k-1},t_k]}\frac{\partial}{\partial t}\overline{R}_j(\gamma(t))\int_{[t_{m-1},t_m]}\log|\gamma(t)-\gamma(s)|\frac{\partial}{\partial s}\overline{R}_i(\gamma(s))\,ds\,dt\\
&= -\frac{1}{2\pi}\sum_{k=1}^{N}\sum_{m=1}^{N}\int_{[t_{k-1},t_k]}R_j'(t)\int_{[t_{m-1},t_m]}\log|\gamma(t)-\gamma(s)|R_i'(s)\,ds\,dt\\
&= -\frac{1}{2\pi}\sum_{k=\max(j,1)}^{\min(j+p,N)}\sum_{m=\max(i,1)}^{\min(i+p,N)}\int_{[t_{k-1},t_k]}\int_{[t_{m-1},t_m]}\log|\gamma(t)-\gamma(s)|R_j'(t)R_i'(s)\,ds\,dt.
\end{aligned}
$$

We use the transformations $t\mapsto t_{k-1}+(t_k-t_{k-1})t$ and $s\mapsto t_{m-1}+(t_m-t_{m-1})s$

$$
= -\frac{1}{2\pi}\sum_{k=\max(j,1)}^{\min(j+p,N)}\sum_{m=\max(i,1)}^{\min(i+p,N)}H_kH_m\int_{[0,1]}\int_{[0,1]}\log|\gamma_k(t)-\gamma_m(s)|R_{j,k}'(t)R_{i,m}'(s)\,ds\,dt.
$$

We now have to distinguish between different cases, where the summands are unequal to zero, i.e., $H_k, H_m\neq 0$. These cases are treated in a similar way to [Gan14, Section 5]

1. Case: $\boldsymbol{\gamma([t_{k-1},t_k])\cap\gamma([t_{m-1},t_m])=\emptyset}$: In this case, the integrand has no singularities. Therefore, we can use a tensorial Gauss quadrature with weight function 1 to approximate the integral and Theorem 6.1 is applicable.

2. Case: $\boldsymbol{\gamma([t_{k-1},t_k])=\gamma([t_{m-1},t_m])}$: This implies, that $k=m$. As a result we get

$$
\begin{aligned}
&\int_{[0,1]}\int_{[0,1]}\log|\gamma_k(t)-\gamma_m(s)|R_{j,k}'(t)R_{i,m}'(s)\,ds\,dt\\
&\quad=\int_{[0,1]}\int_{[0,1]}\log|\gamma_k(t)-\gamma_k(s)|R_{j,k}'(t)R_{i,k}'(s)\,ds\,dt.
\end{aligned}
$$

The transformation $s \mapsto t - s$ and splitting the integral yield

$$
\begin{aligned}
&= \int_{[0,1]} \int_{[t-1,t]} \log|\gamma_k(t) - \gamma_k(t-s)| R'_{j,k}(t) R'_{i,k}(t-s)\, ds\, dt \\
&= \int_{[0,1]} \int_{[t-1,0]} \log|\gamma_k(t) - \gamma_k(t-s)| R'_{j,k}(t) R'_{i,k}(t-s)\, ds\, dt \\
&\quad + \int_{[0,1]} \int_{[0,t]} \log|\gamma_k(t) - \gamma_k(t-s)| R'_{j,k}(t) R'_{i,k}(t-s)\, ds\, dt
\end{aligned} \tag{6.2}
$$

Take a look at the first integral of (6.2). We use the transformations $t \mapsto 1-t$ and $s \mapsto -s$ to get

$$
\begin{aligned}
&\int_{[0,1]} \int_{[t-1,0]} \log|\gamma_k(t) - \gamma_k(t-s)| R'_{j,k}(t) R'_{i,k}(t-s)\, ds\, dt \\
&= \int_{[0,1]} \int_{[-t,0]} \log|\gamma_k(1-t) - \gamma_k(1-t-s)| R'_{j,k}(1-t) R'_{i,k}(1-t-s)\, ds\, dt \\
&= \int_{[0,1]} \int_{[0,t]} \log|\gamma_k(1-t) - \gamma_k(1-t+s)| R'_{j,k}(1-t) R'_{i,k}(1-t+s)\, ds\, dt.
\end{aligned}
$$

Now we use the transformation $(s,t) \mapsto (st,t)$ such that the singularities are either at $s=0$ or $t=0$. Afterwards, we apply basic logarithmic identities

$$
\begin{aligned}
&= \int_{[0,1]} \int_{[0,1]} \log|\gamma_k(1-t) - \gamma_k(1-t+st)| R'_{j,i}(1-t) R'_{i,k}(1-t+st)\, t\, ds\, dt \\
&= \int_{[0,1]} \int_{[0,1]} \log\Big|\frac{\gamma_k(1-t) - \gamma_k(1-t+st)}{st}\Big| R'_{j,k}(1-t) R'_{i,k}(1-t+st)\, t\, ds \\
&\quad + \int_{[0,1]} \int_{[0,1]} \log(s) R'_{j,k}(1-t) R'_{i,k}(1-t+st)\, t\, ds\, dt \\
&\quad + \int_{[0,1]} \int_{[0,1]} \log(t) R'_{j,k}(1-t) R'_{i,k}(1-t+st)\, t\, ds\, dt.
\end{aligned} \tag{6.3}
$$

Consider the second integral in (6.2). We use again the transformation $(s,t) \mapsto (st,t)$, followed by basic logarithmic identities, to see

$$
\begin{aligned}
&\int_{[0,1]} \int_{[0,t]} \log|\gamma_k(t) - \gamma_k(t-s)| R'_{j,k}(t) R'_{i,k}(t-s)\, ds\, dt \\
&= \int_{[0,1]} \int_{[0,1]} \log|\gamma_k(t) - \gamma_k(t-st)| R'_{j,k}(t) R'_{i,k}(t-st)\, t\, ds\, dt \\
&= \int_{[0,1]} \int_{[0,1]} \log\Big|\frac{\gamma_k(t) - \gamma_k(t-st)}{st}\Big| R'_{j,k}(t) R'_{i,k}(t-st)\, t\, ds\, dt \\
&\quad + \int_{[0,1]} \int_{[0,1]} \log(s) R'_{j,k}(t) R'_{i,k}(t-st)\, t\, ds\, dt \\
&\quad + \int_{[0,1]} \int_{[0,1]} \log(t) R'_{j,k}(t) R'_{i,k}(t-st)\, t\, ds\, dt
\end{aligned} \tag{6.4}
$$

To approximate the integrals in (6.3) respecitvely (6.4), we use a tensorial Gauss quadrature with weight functions 1 and $\log(\cdot)$ accordingly. This is justifiable due to the following lemma, which allows us to apply Theorem 6.1.

Lemma 6.3 *Let the parametrization γ be $q \geq 2$ times continuously differentiable on all $[\check{z}^\gamma_{\ell-1}, \check{z}^\gamma_\ell]$ with $\ell \in \{1, \dots, n_\gamma\}$, where $\big\{\check{z}^\gamma_j : j = 1, \dots, n_\gamma\big\}$ is the set of all nodes. Then the integrands in* (6.3) *and* (6.4) *are, up to* $\log(s)$ *resp.* $\log(t)$*, $q-1$ times continuously partially differentiable on* $[0,1] \times [0,1]$.

Proof. We just consider the integrals in (6.3), since the ones in (6.4) can be treated analogously. Obviously, the statement of the lemma holds for the second and third summand. Consider the singular part of the integrand of the first summand and let $s, t \in (0,1]$,

$$\begin{aligned}
\frac{\gamma(t_{k-1}+H_k(1-t))-\gamma(t_{k-1}+H_k(1-t+st))}{st} &= \frac{\int_{[t_{k-1}+H_k(1-t+st),t_{k-1}+H_k(1-t)]}\gamma'(\tau)\,d\tau}{st} \\
&= \frac{(t_{k-1}+H_k(1-t)-t_{k-1}-H_k(1-t+st))\int_{[0,1]}\gamma'(t_{k-1}+H_k(1-t+st)-H_k st\tau)\,d\tau}{st} \\
&= -H_k\int_{[0,1]}\gamma'(t_{k-1}+H_k(1-t+st)-H_k st\tau)\,d\tau
\end{aligned}$$

This term can be continuously extended for $s=0$ or $t=0$ by $-H_k\gamma|'_{[t_{k-1},t_k]}(t_{k-1}+H_k(1-t))$. Due to the smoothness of the integrand, the term is $q-1$ times continuously partially differentiable. Since γ is injective and γ' vanishes nowhere, the modulus is strictly positive for all $s,t\in[0,1]$. Therefore, we can apply $\log(|\cdot|)$, which has no influence on the differentiability. This concludes the proof, since the remaining parts of the integrand fullfill the needed differentiability. □

3. Case: $\gamma([t_{k-1},t_k])\cap\gamma([t_{m-1},t_m])=\{z\}$ for some $z\in\Gamma$: In this case, there exists a singularity of the integrand at either $t=0$ and $s=1$, or at $t=1$ and $s=0$. Without loss of generality, we consider the case $t=0$ and $s=1$. The other case can be treated analogously.
Obviously, $t_{k-1}=t_m$, or $t_m=b$ and $t_{k-1}=a$. Therefore, after the transformation $s\mapsto 1-s$ and splitting the inner integral, we get

$$\begin{aligned}
&\int_{[0,1]}\int_{[0,1]}\log|\gamma_k(t)-\gamma_m(s)|R'_{j,k}(t)R'_{i,m}(s)\,ds\,dt \\
&\quad=\int_{[0,1]}\int_{[0,t]}\log|\gamma_k(t)-\gamma_m(1-s)|R'_{j,k}(t)R'_{i,m}(1-s)\,ds\,dt \\
&\qquad+\int_{[0,1]}\int_{[t,1]}\log|\gamma_k(t)-\gamma_m(1-s)|R'_{j,k}(t)R'_{i,m}(1-s)\,ds\,dt
\end{aligned}\tag{6.5}$$

Consider the first integral in (6.5). We use the Duffy transformation $(s,t)\mapsto(st,t)$, which transforms the singularity to $t=0$. With basic logarithmic identities, it follows

$$\begin{aligned}
&\int_{[0,1]}\int_{[0,t]}\log|\gamma_k(t)-\gamma_m(1-s)|R'_{j,k}(t)R'_{i,m}(1-s)\,ds\,dt \\
&\quad=\int_{[0,1]}\int_{[0,1]}\log|\gamma_k(t)-\gamma_m(1-st)|R'_{j,k}(t)R'_{i,m}(1-st)\,t\,ds\,dt \\
&\quad=\int_{[0,1]}\int_{[0,1]}\log\Big|\frac{\gamma_k(t)-\gamma_m(1-st)}{t}\Big|R'_{j,k}(t)R'_{i,m}(1-st)\,t\,ds\,dt \\
&\qquad-\int_{[0,1]}\int_{[0,1]}\log(t)R'_{j,k}(t)R'_{i,m}(1-st)\,t\,ds\,dt
\end{aligned}\tag{6.6}$$

For the second integral in (6.5), we use Fubini's Theorem, followed by the transformation $(s,t)\mapsto(s,st)$ and basic logarithmic identities. This gives

$$\begin{aligned}
&\int_{[0,1]}\int_{[t,1]}\log|\gamma_k(t)-\gamma_m(1-s)|R'_{j,k}(t)R'_{i,m}(1-s)\,ds\,dt \\
&\quad=\int_{[0,1]}\int_{[0,s]}\log|\gamma_k(t)-\gamma_m(1-s)|R'_{j,k}(t)R'_{i,m}(1-s)\,dt\,ds \\
&\quad=\int_{[0,1]}\int_{[0,1]}\log|\gamma_k(st)-\gamma_m(1-s)|R'_{j,k}(st)R'_{i,m}(1-s)\,s\,dt\,ds \\
&\quad=\int_{[0,1]}\int_{[0,1]}\log\Big|\frac{\gamma_k(st)-\gamma_m(1-s)}{s}\Big|R'_{j,k}(st)R'_{i,m}(1-s)\,s\,dt\,ds \\
&\qquad-\int_{[0,1]}\int_{[0,1]}\log(s)R'_{j,k}(st)R'_{i,m}(1-s)\,s\,dt\,ds
\end{aligned}\tag{6.7}$$

For the final integrals in (6.6) and (6.7), we can use a tensorial Gauss quadrature with weight functions 1, $\log(t)$, and $\log(s)$ accordingly. Again, Theorem 6.1 can be applied, because of the following lemma.

Lemma 6.4 *Let the parametrization γ be $q \geq 2$ times continuously differentiable on all $[\check{z}^\gamma_{\ell-1}, \check{z}^\gamma_\ell]$ with $\ell \in \{1, \dots, n_\gamma\}$. Then the integrands in* (6.6) *and* (6.7) *are, up to* $\log(s)$ *resp.* $\log(t)$*, $q-1$ times continuously partially differentiable on* $[0,1] \times [0,1]$.

Proof. We just consider the integrals in (6.6), since the ones in (6.7) can be treated analogously. Since $R'_{j,k}(t)R'_{i,m}(1-st)\,t$ is smooth, the statement of the lemma holds for the second summand. Consider the singular part of the integrand of the first summand $\log\left|\frac{\gamma_k(t)-\gamma_m(1-st)}{t}\right|$. Without loss of generality, assume $t_{k-1} = t_m$. Let $t \in (0,1]$, $s \in [0,1]$. Then,

$$\begin{aligned}
&\frac{\gamma(t_{k-1} + (t_k - t_{k-1})t) - \gamma(t_{m-1} + (t_m - t_{m-1})(1-st))}{t} \\
&\quad = \frac{\int_{[t_m - H_m st, t_m + H_k t]} \gamma'(\tau)\, d\tau}{t} = \int_{[-H_m s, H_k]} \gamma'(t_m + t\tau)\, d\tau.
\end{aligned}$$

We can extend this term for $t = 0$ continuously by $H_m s \gamma'^\ell(t_m) + H_k \gamma'^r(t_m)$. The integrand is smooth on $[-H_m s, 0]$ resp. $[0, H_k]$. Hence, the term is $q-1$ times continuously partially differentiable. Due to (1.1) the modulus of it is strictly positive, and we can apply $\log(|\cdot|)$ without affecting the differentiability. The smoothness of the remaining factors in (6.6) is obvious.
The case $t_m = b$ and $t_{k-1} = a$ can be treated analogously. □.

Up to this point, we assumed $i, j = 2-p, \dots, N-p-1$ for $\langle W\overline{R}_i; \overline{R}_j\rangle_\Gamma$. However, the remaining cases, i.e., $i = j = 1-p$, $i \neq j = 1-p$ and $1-p = i \neq j$, can be treated similarly by splitting $\overline{R}_{1-p}$ into its two summands.

6.2 Approximation of S_h

We use the following abbreviations for $t \in [a,b]$:

$$\begin{aligned}
\Gamma_\ell &:= \gamma([t_{\ell-1}, t_\ell]), \\
H_\ell &:= t_\ell - t_{\ell-1}, \\
R_{k,\ell} &:= R_k(t_{\ell-1} + (t_\ell - t_{\ell-1})t), \\
\gamma_\ell(t) &:= \gamma(t_{\ell-1} + (t_\ell - t_{\ell-1})t).
\end{aligned}$$

Let $i, j \neq 1-p$. Then,

$$\begin{aligned}
\langle \overline{R}_i; 1\rangle_\Gamma \langle \overline{R}_j; 1\rangle_\Gamma &= \int_\Gamma \int_\Gamma \overline{R}_i(x)\overline{R}_j(y)\, ds_y\, ds_x \\
&= \sum_{k=1}^N \sum_{m=1}^N \int_{\Gamma_k} \int_{\Gamma_m} \overline{R}_i(x)\overline{R}_j(y)\, ds_y\, ds_x \\
&= \sum_{k=1}^N \sum_{m=1}^N \int_{[t_{k-1},t_k]} \int_{[t_{m-1},t_m]} \overline{R}_i(\gamma(t))\overline{R}_j(\gamma(s))|\gamma'(t)||\gamma'(s)|\, ds\, dt \\
&= \sum_{k=1}^N \sum_{m=1}^N \int_{[t_{k-1},t_k]} \int_{[t_{m-1},t_m]} R_i(t)R_j(s)|\gamma'(t)||\gamma'(s)|\, ds\, dt \\
&= \sum_{k=\max(i,1)}^{\min(i+p,N)} \sum_{m=\max(j,1)}^{\min(j+p,N)} H_k H_m \int_{[0,1]} \int_{[0,1]} R_{i,k}(t)R_{j,m}(s)|\gamma'_k(t)||\gamma'_m(s)|\, ds\, dt
\end{aligned}$$

As the integrand is smooth, we can use Gauss quadrature with weight function 1 and apply Theorem 6.1. The other cases for i, j can be treated as above by splitting $\overline{R}_{1-p}$ into its two summands.

6.3 Approximation of F_h

This section is the analogon of [Gan14, Section 5.2].

We use the following abbreviations:

$$\begin{aligned}
\Gamma_\ell &:= \gamma([t_{\ell-1}, t_\ell]),\\
H_\ell &:= t_\ell - t_{\ell-1},\\
R_{k,\ell} &:= R_k(t_{\ell-1} + (t_\ell - t_{\ell-1})t),\\
\gamma_\ell(t) &:= \gamma(t_{\ell-1} + (t_\ell - t_{\ell-1})t).
\end{aligned}$$

For $j \neq 1-p$, it holds that

$$\langle (1/2 - K')\phi_h; \overline{R}_j\rangle_\Gamma = \langle \phi_h; (1/2 - K)\overline{R}_j\rangle_\Gamma = \langle \phi_h; 1/2\overline{R}_j\rangle_\Gamma - \langle \phi_h; K\overline{R}_j\rangle_\Gamma.$$

We consider the first summand,

$$\begin{aligned}
\langle \phi_h; 1/2\overline{R}_j\rangle_\Gamma = \frac{1}{2}\int_\Gamma \phi_h(x)\overline{R}_j(x)\,ds_x &= \frac{1}{2}\sum_{k=1}^{N}\int_{\Gamma_k} \phi_h(x)\overline{R}_j(x)\,ds_x\\
&= \frac{1}{2}\sum_{k=\max(j,1)}^{\min(j+p,N)}\int_{[t_{k-1},t_k]} \phi_h(\gamma(t))R_j(t)|\gamma'(t)|\,dt\\
&= \frac{1}{2}\sum_{k=\max(j,1)}^{\min(j+p,N)} H_k\int_{[0,1]} \phi_h(\gamma_k(t))R_{j,k}(t)|\gamma_k'(t)|\,dt.
\end{aligned}$$

We use Gauss quadrature with weight function 1. Theorem 6.1 is applicable if ϕ_h is at least piecewise continuous.

The second summand is more difficult,

$$\begin{aligned}
\langle \phi_h; K\overline{R}_j\rangle_\Gamma &= \int_\Gamma\int_\Gamma \phi_h(x)\partial_{n_y}G(x,y)\overline{R}_j(y)\,ds_y\,ds_x\\
&= \sum_{k=1}^{N}\sum_{m=1}^{N}\int_{\Gamma_k}\int_{\Gamma_m} \phi_h(x)\partial_{n_y}G(x,y)\overline{R}_j(y)\,ds_y\,ds_x\\
&= \sum_{k=1}^{N}\sum_{m=1}^{N}\int_{[t_{k-1},t_k]}\int_{[t_{m-1},t_m]} \phi_h(\gamma(t))\partial_{n_{\gamma(s)}}G(\gamma(t),\gamma(s))\overline{R}_j(\gamma(s))|\gamma'(t)||\gamma'(s)|\,ds\,dt\\
&= \sum_{k=1}^{N}\sum_{m=\max(j,1)}^{\min(j+p,N)} H_kH_m\int_{[0,1]}\int_{[0,1]} \phi_h(\gamma_k(t))\partial_{n_{\gamma(s)}}G(\gamma_k(t),\gamma_m(s))R_{j,m}(s)|\gamma_k'(t)||\gamma_m'(s)|\,ds\,dt.
\end{aligned}$$

Again, we have to consider 3 cases.

1. Case: $\gamma([t_{k-1}, t_k]) \cap \gamma([t_{m-1}, t_m]) = \emptyset$: As before, the integrand does not have any singularity in this case. Therefore, we can use Gauss quadrature with weight function 1.

2. Case: $\gamma([t_{k-1}, t_k]) = \gamma([t_{m-1}, t_m])$: It holds $m = k$, and we can simplify the integral as follows:

$$\begin{aligned}
&\int_{[0,1]}\int_{[0,1]} \phi_h(\gamma_k(t))\partial_{n_y}G(\gamma_k(t),\gamma_m(s))R_{j,m}(s)|\gamma_k'(t)||\gamma_m'(s)|\,ds\,dt\\
&\quad= \int_{[0,1]}\int_{[0,1]} \phi_h(\gamma_k(t))\partial_{n_y}G(\gamma_k(t),\gamma_k(s))R_{j,k}(s)|\gamma_k'(t)||\gamma_k'(s)|\,ds\,dt.
\end{aligned}$$

We use the transformation $s \mapsto t - s$ and split the integral

$$
\begin{aligned}
&= \int_{[0,1]} \int_{[t-1,t]} \phi_h(\gamma_k(t)) \partial_{n_y} G(\gamma_k(t), \gamma_k(t-s)) R_{j,k}(t-s) |\gamma_k'(t)| |\gamma_k'(t-s)| \, ds \, dt \\
&= \int_{[0,1]} \int_{[0,t]} \phi_h(\gamma_k(t)) \partial_{n_y} G(\gamma_k(t), \gamma_k(t-s)) R_{j,k}(t-s) |\gamma_k'(t)| |\gamma_k'(t-s)| \, ds \, dt \\
&\quad + \int_{[0,1]} \int_{[t-1,0]} \phi_h(\gamma_k(t)) \partial_{n_y} G(\gamma_k(t), \gamma_k(t-s)) R_{j,k}(t-s) |\gamma_k'(t)| |\gamma_k'(t-s)| \, ds \, dt
\end{aligned} \tag{6.8}
$$

Consider the first summand in (6.8). Using the transformation $(s,t) \mapsto (st, t)$, we get

$$
\begin{aligned}
&\int_{[0,1]} \int_{[0,t]} \phi_h(\gamma_k(t)) \partial_{n_y} G(\gamma_k(t), \gamma_k(t-s)) R_{j,k}(t-s) |\gamma_k'(t)| |\gamma_k'(t-s)| \, ds \, dt \\
&= \int_{[0,1]} \int_{[0,1]} \phi_h(\gamma_k(t)) \partial_{n_y} G(\gamma_k(t), \gamma_k(t-st)) R_{j,k}(t-st) |\gamma_k'(t)| |\gamma_k'(t-st)| \, t \, ds \, dt
\end{aligned} \tag{6.9}
$$

For the second summand in (6.8), we use the transformations $t \mapsto 1-t$, $s \mapsto -s$ and $(s,t) \mapsto (st, t)$,

$$
\begin{aligned}
&\int_{[0,1]} \int_{[t-1,0]} \phi_h(\gamma_k(t)) \partial_{n_y} G(\gamma_k(t), \gamma_k(t-s)) R_{j,k}(t-s) |\gamma_k'(t)| |\gamma_k'(t-s)| \, ds \, dt \\
&= \int_{[0,1]} \int_{[-t,0]} \phi_h(\gamma_k(1-t)) \partial_{n_y} G(\gamma_k(1-t), \gamma_k(1-t-s)) \\
&\qquad\qquad R_{j,k}(1-t-s) |\gamma_k'(1-t)| |\gamma_k'(1-t-s)| \, ds \, dt \\
&= \int_{[0,1]} \int_{[0,t]} \phi_h(\gamma_k(1-t)) \partial_{n_y} G(\gamma_k(1-t), \gamma_k(1-t+s)) \\
&\qquad\qquad R_{j,k}(1-t+s) |\gamma_k'(1-t)| |\gamma_k'(1-t+s)| \, ds \, dt \\
&= \int_{[0,1]} \int_{[0,1]} \phi_h(\gamma_k(1-t)) \partial_{n_y} G(\gamma_k(1-t), \gamma_k(1-t+st)) \\
&\qquad\qquad R_{j,k}(1-t+st) |\gamma_k'(1-t)| |\gamma_k'(1-t+st)| \, t \, ds \, dt
\end{aligned} \tag{6.10}
$$

Now we can again use a tensorial Gauss quadrature with weight function 1 for the resulting two integrals, due to the following lemma and Theorem 6.1.

Lemma 6.5 *Let the parametrization γ be $q \geq 2$ times and let $\phi_h \circ \gamma$ be $q-2$ times continuously differentiable on all $[\check{z}_{\ell-1}, \check{z}_\ell]$ with $\ell \in \{1, \dots, n\}$. Then the integrands in* (6.9) *and* (6.10) *are $q-2$ times continuously partially differentiable on $[0,1] \times [0,1]$.*

Proof. We will only consider the integral (6.9), as (6.10) can be treated analogously. To simplify the inegrand $\partial_{n_y} G(\gamma_k(t), \gamma_k(t-st))$, we firstly notice that

$$
\partial_{n_y} G(\gamma_k(t), \gamma_k(t-st)) = \frac{1}{2\pi} \frac{\langle \gamma_k(t) - \gamma_k(t-st); \nu(\gamma_k(t-st)) \rangle}{|\gamma_k(t) - \gamma_k(t-st)|^2}.
$$

Additionaly, we expand $\gamma(\cdot)$ into its Taylor series,

$$
\begin{aligned}
\gamma_k(t) &= \gamma_k(t-st) + \gamma_k'(t-st) H_k t (1-(1-s)) \\
&\quad + \int_{[t_{k-1}+H_k(t-st), t_{k-1}+H_k t]} \gamma''(\tau)(t_{k-1} + H_k t - \tau) \, d\tau
\end{aligned}
$$

Combining these two equations, we get for $s,t \in (0,1]$

$$\begin{aligned}
&\partial_{n_y} G(\gamma_k(t),\gamma_k(t-st)) \\
&\quad = \frac{1}{2\pi}\frac{\langle \gamma_k'(t-st)H_k t(1-(1-s)); \nu(\gamma_k(t-st))\rangle}{|\gamma_k(t)-\gamma_k(t-st)|^2} \\
&\qquad + \frac{1}{2\pi}\frac{\langle \int_{[t_{k-1}+H_k(t-st),t_{k-1}+H_k t]} \gamma''(\tau)(t_{k-1}+H_k t-\tau)\,d\tau; \nu(\gamma_k(t-st))\rangle}{|\gamma_k(t)-\gamma_k(t-st)|^2} \\
&\quad = \Big\langle \int_{[0,1]} \gamma''(t_{k-1}+H_k(t-st)+H_k t(1-(1-s))\tau)(1-\tau)\,d\tau; \nu(\gamma_k(t-st))\Big\rangle \\
&\qquad \frac{1}{2\pi}\frac{H_k^2|t(1-(1-s))|^2}{|\gamma_k(t)-\gamma_k(t-st)|^2}.
\end{aligned}$$

For $t=0$ or $s=0$, we can extend the first factor continuously by

$$\frac{1}{2}\langle \gamma|''_{[t_{k-1},t_k]}(t_{k-1}+H_k t; \nu(\gamma_k|_{[t_{k-1},t_k]}(t-st))\rangle_\Gamma.$$

The integrand is obviously $q-2$ times continuously partially differentiable. So we have to deal with the second factor,

$$\begin{aligned}
\frac{1}{2\pi}\frac{H_k^2|t(1-(1-s))|^2}{|\gamma_k(t)-\gamma_k(t-st)|^2} &= \frac{1}{2\pi}\frac{H_k^2|t(1-(1-s))|^2}{|\gamma(t_{k-1}+H_k t)-\gamma(t_{k-1}+H_k(t-st)|^2} \\
&= \frac{1}{2\pi}\frac{H_k^2|ts|^2}{|\int_{[t_{k-1}+H_k(t-st),t_{k-1}+H_k t]}\gamma'(\tau)\,d\tau|^2} \\
&= \frac{1}{2\pi}\frac{H_k^2|ts|^2}{H_k^2|st|^2|\int_{[0,1]}\gamma'(t_{k-1}+H_k(t-st)+H_k st\tau)\,d\tau|^2}.
\end{aligned}$$

This final term can be continuously extended at $t=0$ or $s=0$ by $1/(2\pi|\gamma|'_{[t_{k-1},t_k]}(t_{k-1}+H_k t)|^2)$. The integrand is $q-1$ times continuously partially differentiable, and $(s,t)\mapsto \partial_{n_y}G(\gamma_k(t),\gamma_k(t-st))$ is $q-2$ times continuously partially differentiable on $[0,1]\times[0,1]$. This proves the lemma, as the remaining factors are obviously sufficiently smooth. □

3. Case: $\gamma([t_{k-1},t_k])\cap\gamma([t_{m-1},t_m]) = \{z\}$ for some $z\in\Gamma$: As before, there is either a singularity at $t=0$ and $s=1$, or at $t=1$ and $s=0$. Without loss of generality, let the singularity occur at $t=0$ and $s=1$. The other case can be treated analogously. It follows that either $t_{k-1}=t_m$, or $t_m=b$ and $t_{k-1}=a$. First, we split the integral as follows

$$\begin{aligned}
&\int_{[0,1]}\int_{[0,1]} \phi_h(\gamma_k(t))\partial_{n_y}G(\gamma_k(t),\gamma_m(s))R_{j,m}(s)|\gamma_k'(t)||\gamma_m'(s)|\,ds\,dt \\
&\quad = \int_{[0,1]}\int_{[0,t]} \phi_h(\gamma_k(t))\partial_{n_y}G(\gamma_k(t),\gamma_m(s))R_{j,m}(s)|\gamma_k'(t)||\gamma_m'(s)|\,ds\,dt \\
&\qquad + \int_{[0,1]}\int_{[t,1]} \phi_h(\gamma_k(t))\partial_{n_y}G(\gamma_k(t),\gamma_m(s))R_{j,m}(s)|\gamma_k'(t)||\gamma_m'(s)|\,ds\,dt.
\end{aligned} \tag{6.11}$$

Considering the first summand in (6.11), we use the transformation $(s,t)\mapsto(1-st,t)$ to transform the singularity to $t=0$,

$$\begin{aligned}
&\int_{[0,1]}\int_{[0,t]} \phi_h(\gamma_k(t))\partial_{n_y}G(\gamma_k(t),\gamma_m(s))R_{j,m}(s)|\gamma_k'(t)||\gamma_m'(s)|\,ds\,dt \\
&\quad = \int_{[0,1]}\int_{[0,1]} \phi_h(\gamma_k(t))\partial_{n_y}G(\gamma_k(t),\gamma_m(1-st))R_{j,m}(1-st)|\gamma_k'(t)||\gamma_m'(1-st)|\,t\,ds\,dt.
\end{aligned} \tag{6.12}$$

For the second summand in (6.11), we use Fubini's Theorem and the transformation $(s,t)\mapsto(1-s,st)$

to transform the singularity to $s = 0$,

$$\begin{aligned}
&\int_{[0,1]}\int_{[t,1]} \phi_h(\gamma_k(t))\partial_{n_y} G(\gamma_k(t),\gamma_m(s))R_{j,m}(s)|\gamma_k'(t)||\gamma_m'(s)|\,ds\,dt \\
&\quad = \int_{[0,1]}\int_{[0,s]} \phi_h(\gamma_k(t))\partial_{n_y} G(\gamma_k(t),\gamma_m(s))R_{j,m}(s)|\gamma_k'(t)||\gamma_m'(s)|\,dt\,ds \qquad (6.13)\\
&\quad = \int_{[0,1]}\int_{[0,1]} \phi_h(\gamma_k(st)\partial_{n_y} G(\gamma_k(st),\gamma_m(1-s))R_{j,m}(1-s)|\gamma_k'(st)||\gamma_m'(1-s)|\,s\,ds\,dt.
\end{aligned}$$

To approximate the integrals (6.12) and (6.13), we can use a tensorial Gauss quadrature with weight function 1. Theorem 6.1 can be applied due to the following lemma.

Lemma 6.6 *Let the parametrization γ be $q \geq 2$ times and let $\phi_h \circ \gamma$ be $q-1$ times continuously differentiable on all $[\check{z}_{\ell-1}, \check{z}_\ell]$ with $\ell \in \{1,\dots,n\}$. Then the integrands in* (6.12) *and* (6.13) *are $q-1$ times continuously partially differentiable on $[0,1]\times[0,1]$.*

Proof. As (6.13) can be treated analogously, we will only consider equation (6.12). To this end, let first $t_{k-1} = t_m$. For $t \in (0,1]$ and $s \in [0,1]$, it holds

$$\partial_{n_y} G(\gamma_k(t),\gamma_m(1-st))\cdot t = \frac{1}{2\pi}\frac{\langle \gamma_k(t)-\gamma_m(1-st);\nu(\gamma_m(1-st))\rangle}{|\gamma_k(t)-\gamma_m(1-st)|^2}\cdot t.$$

Take a deeper look at

$$\begin{aligned}
&\frac{\gamma_k(t)-\gamma_m(1-st)}{|\gamma_k(t)-\gamma_m(1-st)|^2}\cdot t \\
&\quad = \frac{t^2}{|\gamma(t_{k-1}+H_k t)-\gamma(t_{m-1}+H_m(1-st))|^2}\cdot\frac{\gamma(t_{k-1}+H_k t)-\gamma(t_{m-1}+H_m(1-st))}{t} \\
&\quad = \frac{t^2}{|\int_{[t_{k-1}-H_m st, t_{k-1}+H_k t]}\gamma'(\tau)\,d\tau|^2}\cdot\frac{\int_{[t_{k-1}-H_m st, t_{k-1}+H_k t]}\gamma'(\tau)\,d\tau}{t} \\
&\quad = \frac{\int_{[-H_m s, H_k]}\gamma'(t_{k-1}+t\tau)\,d\tau}{|\int_{[-H_m s, H_k]}\gamma'(t_{k-1}+t\tau)\,d\tau|^2}.
\end{aligned}$$

Since $\gamma'^{\ell}(t_{k-1})$ is no negative multiple of $\gamma'^{r}(t_{k-1})$, we can extend this term at $t=0$ continuously by

$$\frac{H_m s\gamma'^{\ell}(t_{k-1}) + H_k\gamma'^{r}(t_{k-1})}{|H_m s\gamma'^{\ell}(t_{k-1}) + H_k\gamma'^{r}(t_{k-1})|^2}.$$

Additionally, it is even $q-1$ continuously partially differentiable, which proves the demanded differentiability of $(s,t)\mapsto \partial_{n_y} G(\gamma_k(t),\gamma_m(1-st))\cdot t$ on $[0,1]\times[0,1]$. As the remaining factors in (6.12) have the same smoothness, and the case $t_m = b$ and $t_{k-1} = a$ works analogously, the lemma is proved. □

The case $\langle (1/2-K')\phi_h; \overline{R}_{1-p}\rangle_\Gamma$ can be treated analogously by splitting $\overline{R}_{1-p}$ into its two summands.

6.4 Numerical evaluation of $K'\phi_h$

Let $x = \gamma(s)$ with $s \in [a,b]\setminus\{t_0,\dots,t_n\}$ and $k_s \in \{1,\dots,n\}$ with $s \in [t_{k_s-1}, t_{k_s}]$. We use the following abbreviations:

$$\begin{aligned}
\Gamma_\ell &:= \gamma([t_{\ell-1}, t_\ell]),\\
H_\ell &:= t_\ell - t_{\ell-1},\\
\gamma_\ell(t) &:= \gamma(t_{\ell-1} + (t_\ell - t_{\ell-1})t),
\end{aligned}$$

and $B(x,\epsilon) := \{y \in \mathbb{R}^2 : |x-y| < \epsilon\}$ for the open ball centered at x with radius ϵ.

It holds that

$$
\begin{aligned}
K'\phi_h(x) &= \lim_{\epsilon\to 0}\int_{\Gamma\setminus B(x,\epsilon)} \partial_{n_x} G(x,y)\phi_h(y)\,ds_y \\
&= \sum_{\substack{k=1\\k\neq k_s}}^{N} \int_{\Gamma_k} \partial_{n_x} G(x,y)\phi_h(y)\,ds_y + \lim_{\epsilon\to 0}\int_{\Gamma_{k_s}\setminus B(x,\epsilon)} \partial_{n_x} G(x,y)\phi_h(y)\,ds_y. \qquad (6.14)
\end{aligned}
$$

With the help of the mean value theorem, it follows that $|\partial_{n_x} G(x,y)| \le C$ for all $x, y \in \Gamma_{k_s}$ with $C > 0$. Hence, [Pra07, Lemma 4.23] yields the existence of the improper integral (6.14) and it follows

$$
\begin{aligned}
K'\phi_h(x) = &\sum_{\substack{k=1\\k\neq k_s}}^{N} \int_{[t_{k-1},t_k]} \partial_{n_x} G(\gamma(s),\gamma(t))\phi_h(\gamma(t))|\gamma'(t)|\,dt \\
&+ \int_{[t_{k_s-1},s]} \partial_{n_x} G(\gamma(s),\gamma(t))\phi_h(\gamma(t))|\gamma'(t)|\,dt \\
&+ \int_{[s,t_{k_s}]} \partial_{n_x} G(\gamma(s),\gamma(t))\phi_h(\gamma(t))|\gamma'(t)|\,dt \\
= &\sum_{\substack{k=1\\k\neq k_s}}^{N} H_k \int_{[0,1]} \partial_{n_x} G(\gamma(s),\gamma_k(t))\phi_h(\gamma_k(t))|\gamma_k'(t)|\,dt \\
&+ (s-t_{k_s-1})\int_{[0,1]} \partial_{n_x} G(\gamma(s),\gamma(s-(s-t_{k_s-1})t)) \\
&\qquad\qquad \cdot \phi_h(\gamma(s-(s-t_{k_s-1})t))|\gamma'(s-(s-t_{k_s-1})t)|\,dt \\
&+ (t_{k_s}-s)\int_{[0,1]} \partial_{n_x} G(\gamma(s),\gamma(s+(t_{k_s}-s)t)) \\
&\qquad\qquad \cdot \phi_h(\gamma(s+(t_{k_s}-s)t))|\gamma'(s+(t_{k_s}-s)t)|\,dt. \qquad (6.15)
\end{aligned}
$$

We approximate these final integrals in (6.15) by Gauss quadrature with weight function 1, which is justified by the following lemma.

Lemma 6.7 *Let the parametrization γ be $q \ge 2$ times and let $\phi_h \circ \gamma$ be $q-2$ times continuously differentiable on $[\check{z}_{\ell-1}, \check{z}_\ell]$ for $\ell \in \{1,\dots,n\}$. Then, the integrands in* (6.15) *are $q-2$ times continuously differentiable on* $[0,1]$.

Proof. The integrand $\partial_{n_x} G(\gamma(s),\gamma_k(t))\phi_h(\gamma_k(t))|\gamma_k'(t)|$ of the first integral in (6.15), is smooth since $\gamma(s) \neq \gamma_k(t)$ for all $k \neq k_s$. Hence, it is sufficient to consider only the second and third summand in (6.15). For $t \in (0,1]$, it holds that

$$
\partial_{n_x} G(\gamma(s),\gamma(s-(s-t_{k_s-1})t)) = -\frac{1}{2\pi}\frac{\langle \gamma(s)-\gamma(s-(s-t_{k_s-1})t);\nu(\gamma(s))\rangle}{|\gamma(s)-\gamma(s-(s-t_{k_s-1})t)|^2}.
$$

The Taylor expansion yields

$$
\begin{aligned}
\gamma(s-(s-t_{k_s-1})t) &= \gamma(s) - \gamma'(s)(s-t_{k_s-1})t + \int_{[s,s-(s-t_{k_s-1})t]} \gamma''(\tau)(s-(s-(t_{k_s-1})t-\tau)\,d\tau \\
&= \gamma(s) - \gamma'(s)(s-t_{k_s-1})t + ((s-t_{k_s-1})t)^2 \int_{[0,1]} \gamma''(s-(s-t_{k_s-1})t\tau)(1-\tau)\,d\tau
\end{aligned}
$$

Altogether, we end up with

$$
\begin{aligned}
&\partial_{n_x} G(\gamma(s), \gamma(s-(s-t_{k_s-1})t)) \\
&\quad = -\frac{1}{2\pi} \frac{\langle \gamma'(s)(s-t_{k_s-1})t; \nu(\gamma(s))\rangle_\Gamma}{|\gamma(s)-\gamma(s-(s-t_{k_s-1})t)|^2} \\
&\qquad + \frac{1}{2\pi} \frac{((s-t_{k_s-1})t)^2}{|\gamma(s)-\gamma(s-(s-t_{k_s-1})t)|^2} \Big\langle \int_{[0,1]} \gamma''(s-(s-t_{k_s-1})t\tau)(1-\tau)\, d\tau; \nu(\gamma(s)) \Big\rangle \\
&\quad = \frac{1}{2\pi} \frac{((s-t_{k_s-1})t)^2}{|\gamma(s)-\gamma(s-(s-t_{k_s-1})t)|^2} \Big\langle \int_{[0,1]} \gamma''(s-(s-t_{k_s-1})t\tau)(1-\tau)\, d\tau; \nu(\gamma(s)) \Big\rangle \\
&\quad = \frac{1}{2\pi} \frac{((s-t_{k_s-1})t)^2}{|\int_{[\gamma(s-(s-t_{k_s-1})t),\gamma(s)]} \gamma'(\tau)\, d\tau|^2} \Big\langle \int_{[0,1]} \gamma''(s-(s-t_{k_s-1})t\tau)(1-\tau)\, d\tau; \nu(\gamma(s)) \Big\rangle \\
&\quad = \frac{1}{2\pi} \frac{((s-t_{k_s-1})t)^2}{((s-t_{k_s-1})t)^2 |\int_{[0,1]} \gamma'(s-(s-t_{k_s-1})t+(s-t_{k_s-1})t\tau)\, d\tau|^2} \\
&\qquad \cdot \Big\langle \int_{[0,1]} \gamma''(s-(s-t_{k_s-1})t\tau)(1-\tau)\, d\tau; \nu(\gamma(s)) \Big\rangle
\end{aligned}
$$

which can be continuously extended at $t = 0$ by

$$
\frac{1}{2\pi} \frac{\langle \gamma''(s); \nu(\gamma(s))\rangle_\Gamma}{2|\gamma'(s)|^2}.
$$

The continuous extension is even $q-2$ times countinuously differentiable for $s \in [0,1]$. Hence, the integrand in the second summand of (6.15) is $q-2$ times continuously differentiable as well. The third summand is treated analogously. □

6.5 Numerical evaluation of WU_h

For the implementation, we used the so called NURBS-BEM package from A. Bantle, M. Bantle, and S. Funken, developed at the Institute for Numerical Mathematics at the University of Ulm. This package provides a stable black box implementation of the numerical evaluation of WU_h.

7 Numerical experiment

In this section, we consider different geometries and compare uniform refinement to the adaptive Algorithm 3.5 for various problems.

The parametrization γ of the boundary Γ is of the form

$$\gamma(t) = \sum_{i\in\mathbb{Z}} C_i R_{i,p}(t) \quad \text{for all } t \in [a,b],$$

where $p \in \mathbb{N}$ is the polynomial degree, $\{R_{i,p} : i \in \mathbb{Z}\}$ are the NURBS from (2.4), and $\big(C_i\big)_{i\in\mathbb{Z}}$ are control points in $\mathbb{R}^2$. Curves represented in this way are called NURBS curves.

The main idea of isogeometric analysis is to use NURBS for both the geometry and the ansatz spaces to avoid possible data approximation errors of the boundary. Therefore, we use the polynomial degree as well as knots and weights of the geometry for the initial ansatz space $\mathcal{X}_0$.

7.1 Slit problem

Although we assumed $\Gamma := \partial\Omega$ for the whole theory in the previous sections, it is also possible to prove everything for a connected $\Gamma_0 \subsetneqq \partial\Omega$ with just some minor changes in the proofs. However, we have to adjust the ansatz and test space of the variational formulation. Instead of $H^{1/2}(\Gamma)$, we take the space $\widetilde{H}^{1/2}(\Gamma_0) := \big\{\widehat{v}|_{\Gamma_0} : \widehat{v} \in H^{1/2}(\Gamma), \operatorname{supp}\widehat{v} \subseteq \overline{\Gamma}_0\big\}$, where $\overline{\Gamma}_0$ denotes the closure of Γ_0.

Let $\Gamma_0 := (-1,1) \times \{0\} \subseteq \mathbb{R}^2$. We consider the hyper-singular integral equation

$$\langle\!\langle u; v\rangle\!\rangle_W = \langle \phi; v\rangle_\Gamma \quad \text{for all } v \in \widetilde{H}^{1/2}(\Gamma_0), \tag{7.1}$$

with the right-hand side $\phi = 1$ and the exact solution $u(x,0) = 2\sqrt{1-x^2}$. In this case, no stabilization term is necessary. The discrete variational formulation reads

$$\langle\!\langle U_\ell; V_\ell\rangle\!\rangle_W = \langle \Pi_\ell\phi; V_\ell\rangle_\Gamma \quad \text{for all } V_\ell \in \mathcal{X}_{\ell,0}, \tag{7.2}$$

where $\mathcal{X}_{\ell,0} := \mathcal{X}_\ell \cap \widetilde{H}^{1/2}(\Gamma_0)$.

Since Π_ℓ is the piecewise L^2-best approximation, it holds $\Pi_\ell\phi = \phi$. Hence, the oscillation term of the node-based error estimator μ_ℓ, which steers Algorithm 3.5, vanishes and μ_ℓ simplifies to

$$\mu_\ell(z) = \|h_\ell^{1/2}(\phi - WU_\ell)\|_{L^2(\omega_\star(z))} \quad \text{for all } z \in \mathcal{N}_\ell.$$

As a consequence, we can use the Galerkin orthogonality to compute the error in the energy norm by

$$|\!|\!| u - U_\ell |\!|\!|_W^2 = |\!|\!| u |\!|\!|_W^2 - |\!|\!| U_\ell |\!|\!|_W^2 = \pi - |\!|\!| U_\ell |\!|\!|_W^2 =: \operatorname{err}_\ell^2.$$

The energy norm $|\!|\!| u |\!|\!|_W^2 = \pi$ is computed analytically, whereas $|\!|\!| U_\ell |\!|\!|_W^2$ can be easily computed via $|\!|\!| U_\ell |\!|\!|_W^2 = c_{h_\ell} \cdot W_{h_\ell} c_{h_\ell}$. Here, W_{h_ℓ} is the stiffness matrix and c_{h_ℓ} is the coefficient vector corresponding to U_ℓ as in Section 6.

The slit is paramatrized on $[0,1]$ for a fixed $p \in \mathbb{N}$ by the NURBS curve induced by

$$\begin{aligned}
p_\gamma &= p,\\
\check{\mathcal{K}}^\gamma &= \Big(\underbrace{0,\dots,0}_{p+1 \text{ times}}, \tfrac{1}{5}, \tfrac{2}{5}, \tfrac{3}{5}, \tfrac{4}{5}, \underbrace{1,\dots,1}_{p+1 \text{ times}}\Big),\\
\mathcal{W}^\gamma &= (\underbrace{1,\dots,1}_{2p+6 \text{ times}}),\\
(C_i)_{i=1}^{N_\gamma} &= \left(\begin{pmatrix}-1\\0\end{pmatrix}, \begin{pmatrix}-1+2/(p+4)\\0\end{pmatrix}, \begin{pmatrix}-1+4/(p+4)\\0\end{pmatrix}, \dots, \begin{pmatrix}1-2/(p+4)\\0\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}\right).
\end{aligned}$$

The first and the last knot have the demanded multiplicity $p+1$ for $p \in \mathbb{N}$. Hence, we can directly use the knot resp. weight vector for the corresponding initial ansatz space $\mathcal{X}_0$.

Figure 7.1 shows the slit for $p = 1$. The images $\{\gamma(t_\ell) : \ell \in \{0, \dots, N_\gamma\}\}$ of the initial nodes are also plotted.

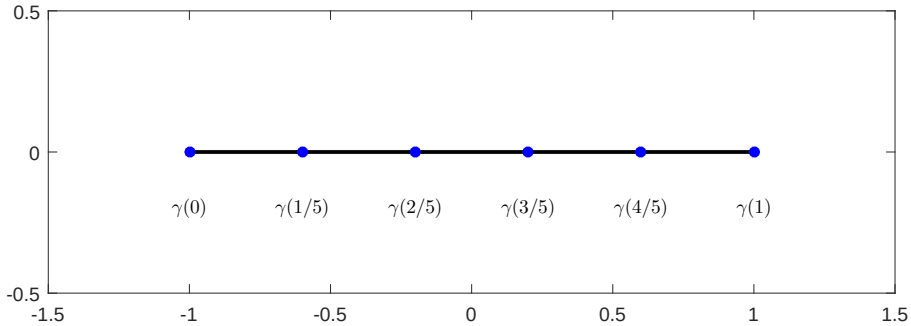

Figure 7.1: Slit geometry with initial nodes for $p = 1$.

In Figure 7.2, we compare the exact solution u of (7.1) and the discrete solution U_ℓ of (7.2) for $p = 1$. Here, the different meshes are generated by the adaptive Algorithm 3.5 with $\theta = 0.5$.

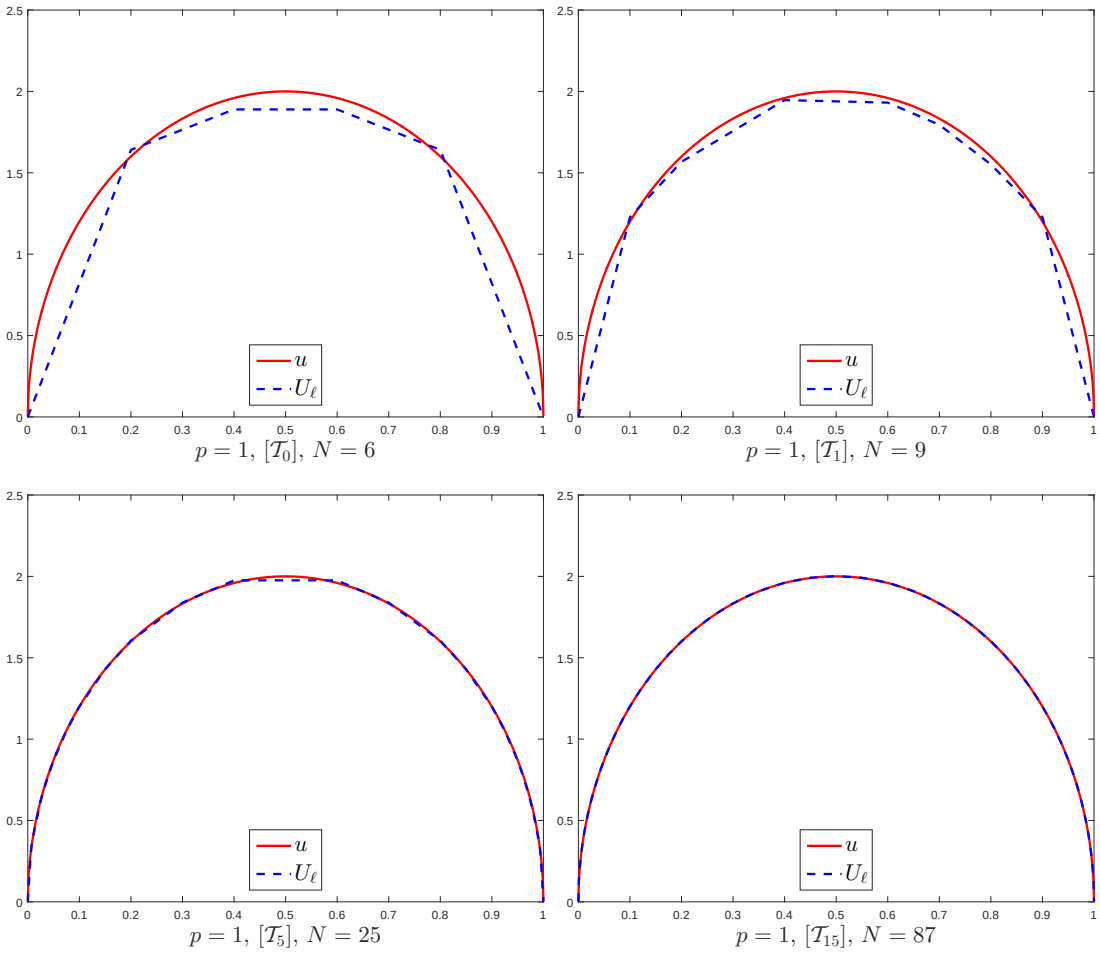

Figure 7.2: The solution u of (7.1) resp. the discrete solution U_ℓ of (7.2) are plotted for the starting mesh $[\mathcal{T}_0]$ and the refinements $[\mathcal{T}_1]$, $[\mathcal{T}_5]$ and $[\mathcal{T}_{15}]$ over the parameter domain $[0, 1]$.

First, we look at Algorithm 3.5 for $\theta = 1$, i.e., uniform refinement and choose $p \in \{1, \dots, 5\}$. The optimal order of convergence is given by $\mathcal{O}(h^{1/2+p}) = \mathcal{O}(N^{-1/2-p})$, cf. [SS11, Theorem 4.1.51]. Figure 7.3 shows in a double logarithmic plot that uniform refinement only leads to the suboptimal rate $\mathcal{O}(N^{-1/2})$, independently of the chosen p. This is due to limited regularity $u \in H^{1-\epsilon}(\Gamma) \setminus H^1(\Gamma)$ of the exact solution for all $\epsilon > 0$.

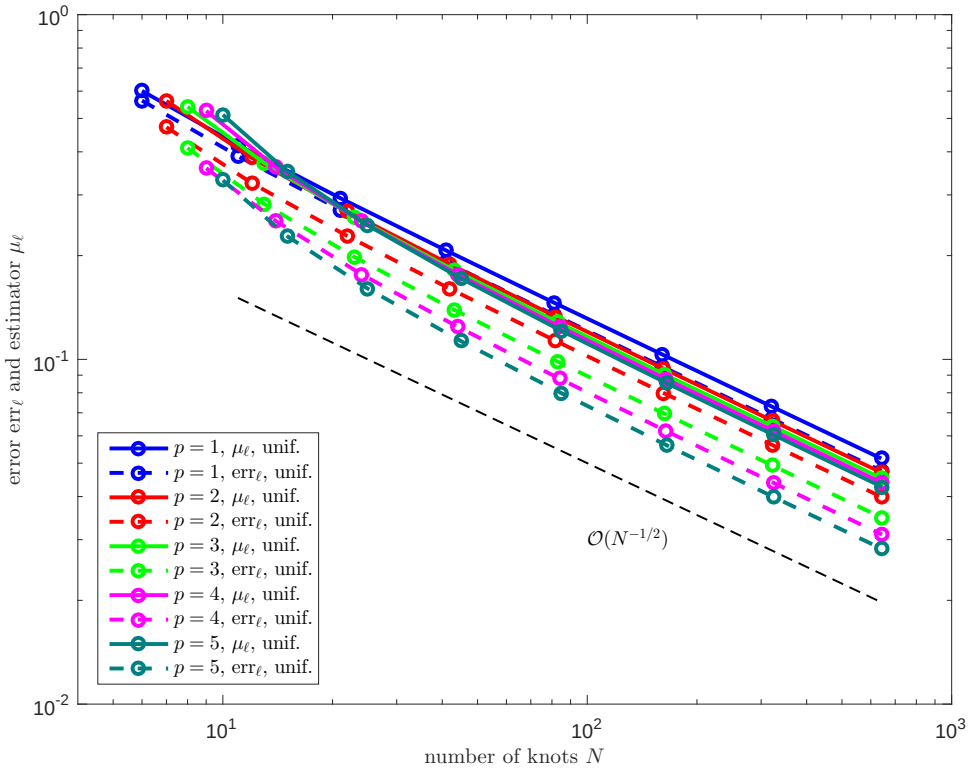

Figure 7.3: Uniform refinement leads to suboptimal order of convergence $\mathcal{O}(N^{-1/2})$ of the energy error err_ℓ as well as the error estimator μ_ℓ.

Contrary to uniform refinement, we see in Figure 7.4 that the adaptive Algorithm 3.5 with $\theta = 0.5$ regains the optimal order of convergence $\mathcal{O}(N^{-1/2-p})$ for both err_ℓ and μ_ℓ.

The histogram in Figure 7.5 shows the knot distribution of the last refinement step of the adaptive Algorithm 3.5 with $\theta = 0.5$ and $p = 1$. Clearly, the algorithm refines mainly at $t = 0$ and $t = 1$, as one would expect.

The question arises, which marking parameter θ one should choose for the adaptive Algorithm 3.5. Figure 7.6 shows that the adaptive Algorithm 3.5 converges with the optimal order of convergence $\mathcal{O}(N^{-3/2})$ for all $\theta \in \{0.1, \dots, 0.9\}$. In Figure 7.7, we consider the cumulative number of knots

$$N_{\mathrm{cum}} := \sum_{j=0}^{\ell} |\mathcal{K}_j|, \tag{7.3}$$

which are necessary to reach the prescribed accuracy $\mu_\ell < 10^{-3}$. Note that an adaptively generated mesh and thus the computational effort depend on the full history of preceeding meshes. Hence, the cumulative sum N_{cum} provides a good measure of comparison. We see that independently of the chosen p a marking parameter $\theta \approx 0.75$ is a good choice for the adaptive Algorithm 3.5 and that $\theta < 0.5$ results in higher computational effort compared to $0.5 \leq \theta$.

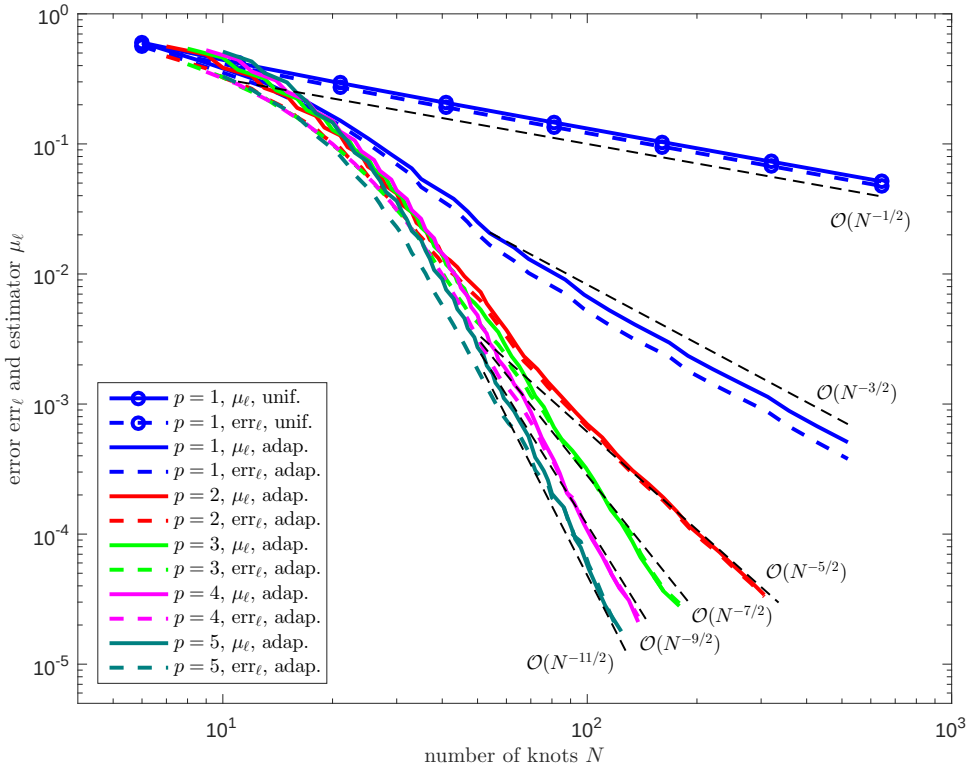

Figure 7.4: Adaptive refinement with $\theta = 0.5$ regains the optimal order of convergence $\mathcal{O}(N^{-1/2-p})$ of the energy error err_ℓ as well as the error estimator μ_ℓ.

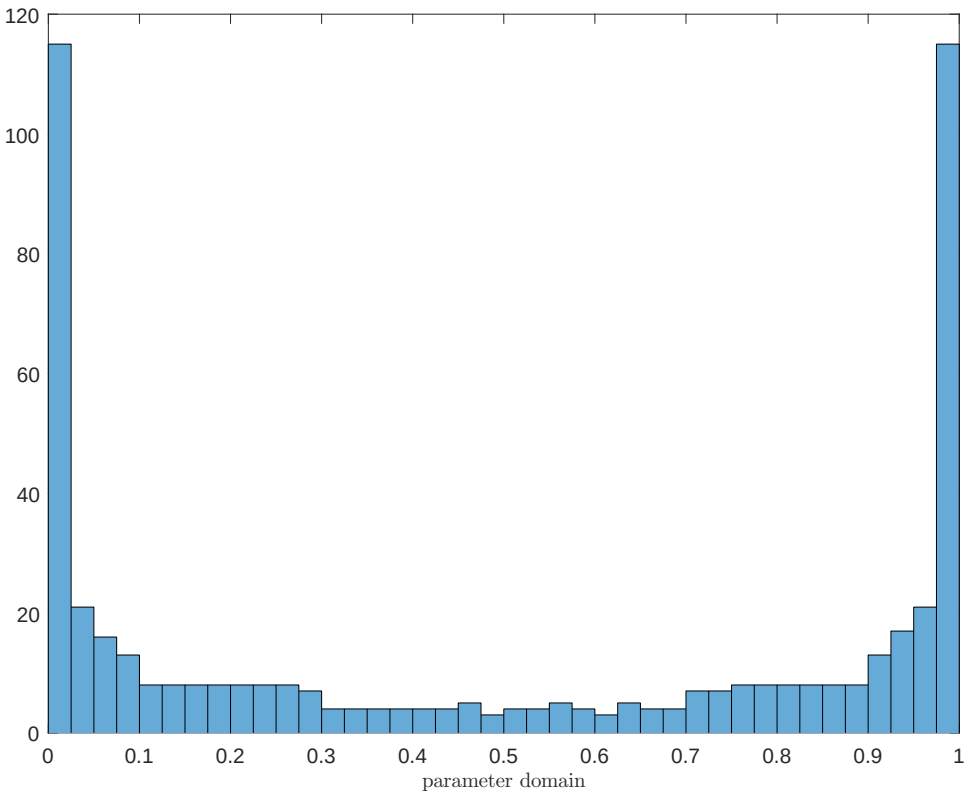

Figure 7.5: Histogram of the knot distribution of the last refinement step $[\mathcal{T}_{26}]$ with $|\mathcal{K}_{26}| = 521$ for the adaptive Algorithm 3.5 with $\theta = 0.5$ and $p = 1$ for the slit problem (7.1).

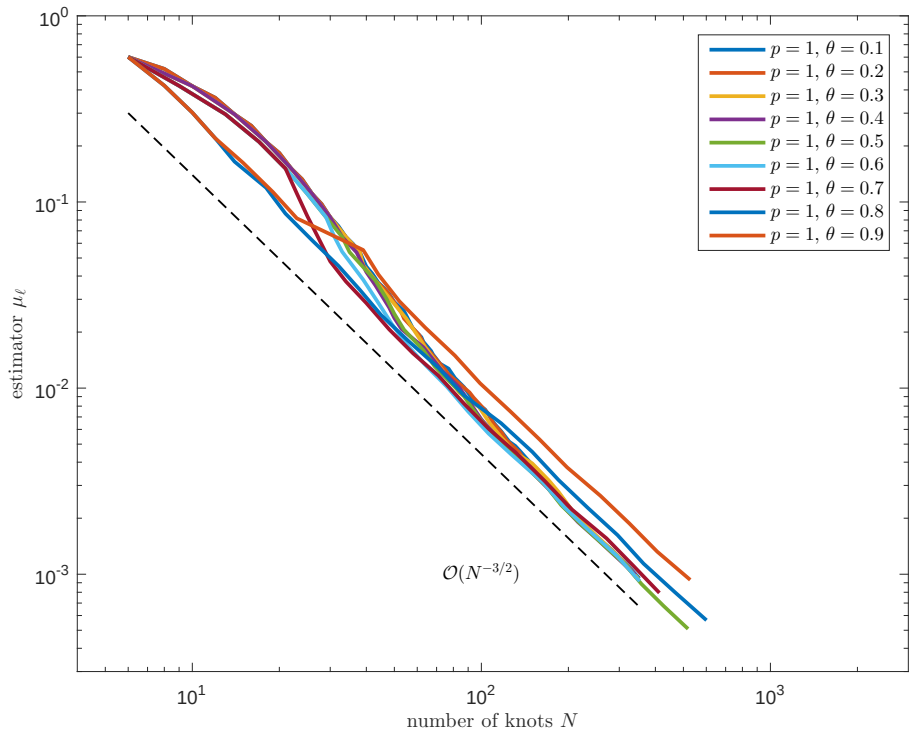

Figure 7.6: Comparison of different marking parameters θ for the adaptive Algorithm 3.5 for the slit problem with $p = 1$.

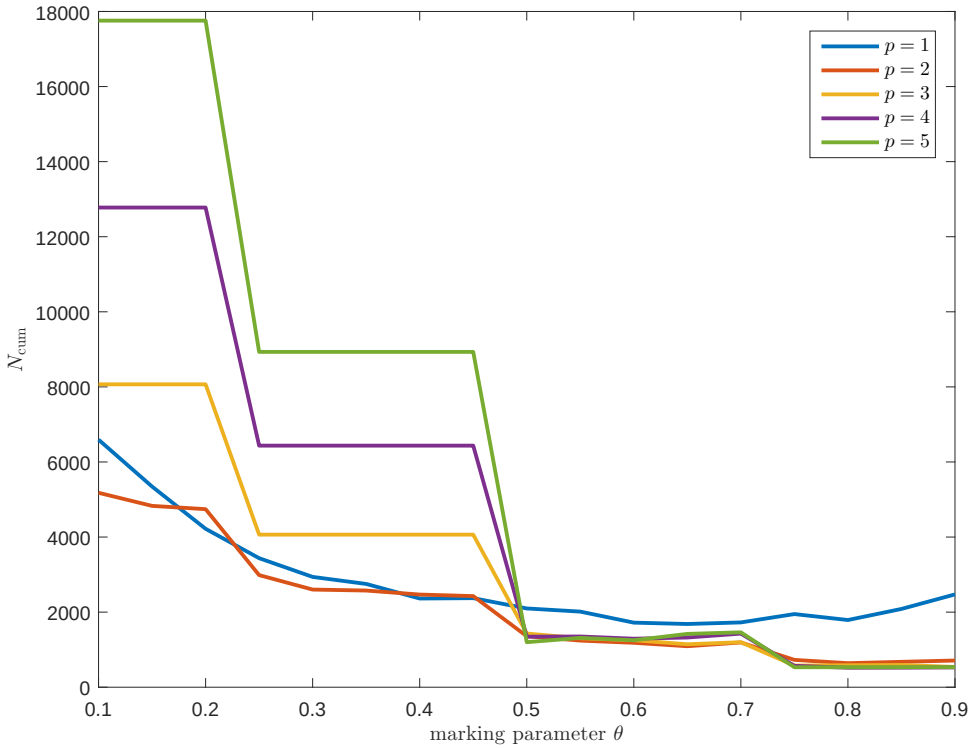

Figure 7.7: Comparison of the slit problem with $p = 1, \ldots, 5$ in terms of the cumulative number of knots $N_{\text{cum}} := \sum_{j=0}^{\ell} |\mathcal{K}_j|$ necessary to reach the prescribed accuracy $\mu_\ell < 10^{-3}$.

7.2 Indirect BEM

Let Γ be the boundary of the L-shaped domain $\Omega = (-1,1)^2 \setminus [0,1]^2$. We consider the hyper-singular integral equation

$$Wu = f \quad \text{on } \Gamma, \tag{7.4}$$

where $f(x,y) = x - c$. The constant $c = -1/8$ is chosen such that the integral mean $\int_\Gamma f\,ds = 0$ vanishes. In this case, we do not have any data oscillation terms.

The L-shape shown in Figure 7.8 is parametrized on $[0,6]$ by the NURBS curve induced by

$$\begin{aligned} p_\gamma &= 1, \\ \check{\mathcal{K}}^\gamma &= (0,0,1,2,3,4,5,6,6), \\ \mathcal{W}^\gamma &= (1,1,1,1,1,1,1,1,1), \\ (C_i)_{i=1}^{N_\gamma} &= \left(\begin{pmatrix} -1 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \end{pmatrix} \right). \end{aligned}$$

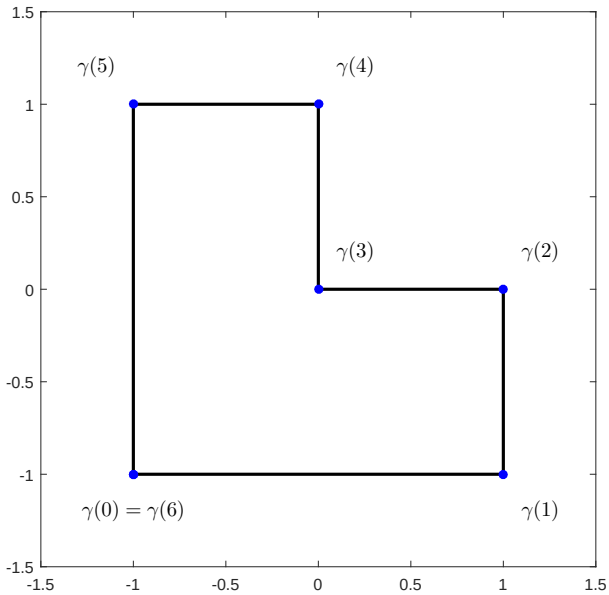

Figure 7.8: L-shape $(-1,1)^2 \setminus [0,1]^2$ with initial nodes.

Again, we compare uniform refinement of the mesh to adaptive refinement for the adaptive Algorithm 3.5 steered by the node-based error estimator μ_ℓ with $\theta = 0.5$. Since the exact solution u of (7.4) is unknown, we use the extrapolated value $\| u \|^2_{W+S} = 12.952418804$ to compute the absolute error $\| u - U_\ell \|_{W+S}$. In Figure 7.9, we see that uniform refinement leads to the order of convergence $\mathcal{O}(N^{-2/3})$, whereas the adaptive strategy regains the optimal order $\mathcal{O}(N^{-3/2})$.

The histogram in Figure 7.10 indicates that the adaptive Algorithm 3.5 refines especially at the corners $\gamma(0), \dots, \gamma(6)$.

In Figure 7.11, the error estimator μ_ℓ is plotted for different choices of the marking parameter θ of the adaptive Algorithm 3.5. For all $\theta \in \{0.1, \dots, 0.9\}$, the estimator decays at optimal order of convergence. Figure 7.12 shows the cumulative sum N_{cum}, cf. Definition (7.3), which is necessary to reach an accuracy of $\mu_\ell < 10^{-3}$, plotted over θ. We observe that the optimal marking parameter seems to be at $\theta \approx 0.8$.

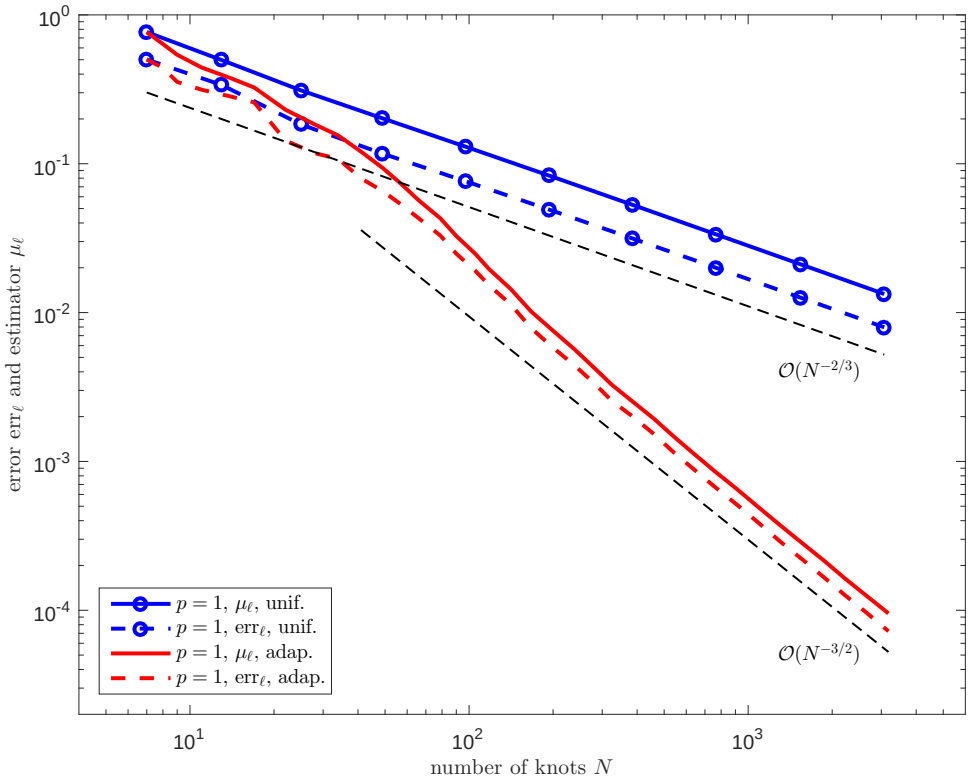

Figure 7.9: Comparison of uniform and adaptive refinement on the L-shaped domain $(-1,1)^2 \setminus [0,1]^2$.

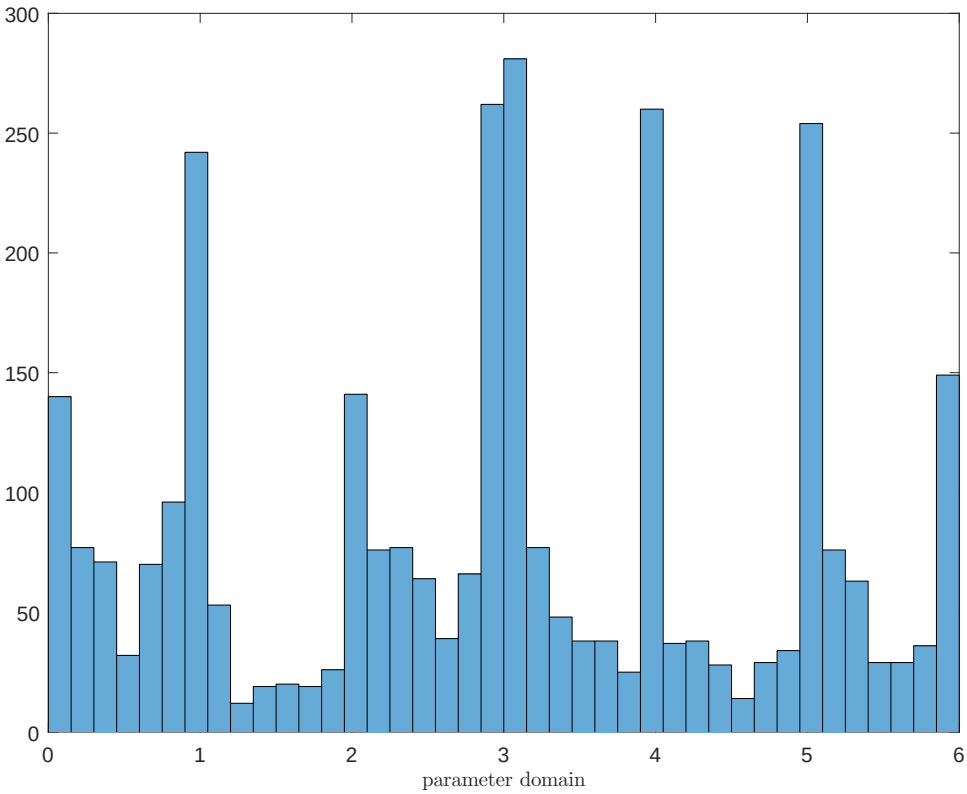

Figure 7.10: Histogram of the knot distribution of the last refinement step $[\mathcal{T}_{35}]$ with $|\mathcal{K}_{35}| = 3185$ for the adaptive Algorithm 3.5 with $\theta = 0.5$ for the problem (7.4).

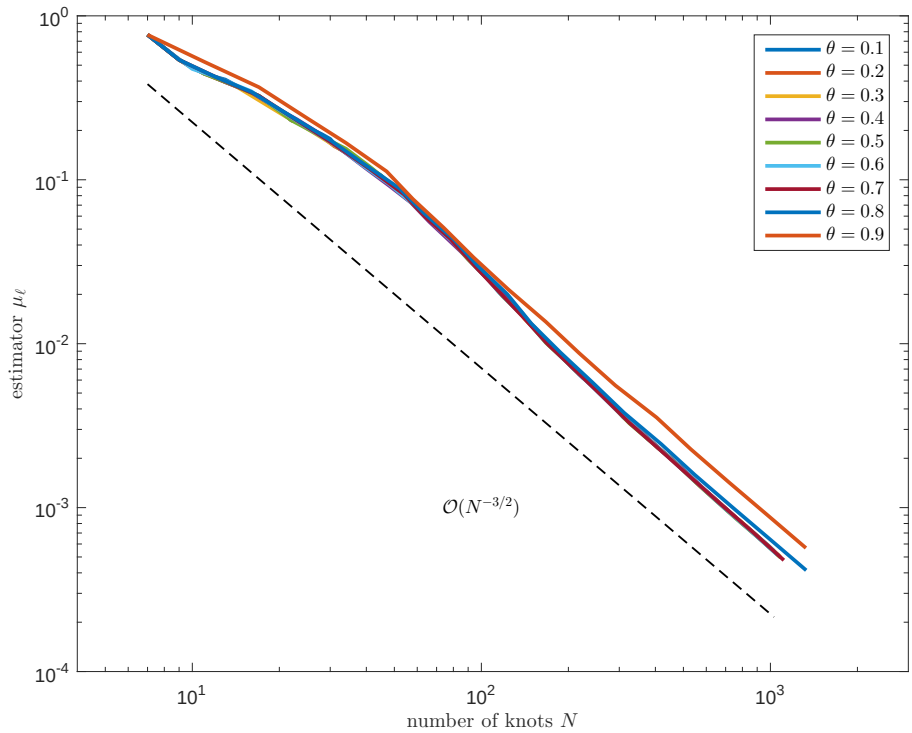

Figure 7.11: Comparison of different marking parameters θ for the adaptive Algorithm 3.5 on the L-shaped domain $(-1,1)^2 \setminus [0,1]^2$.

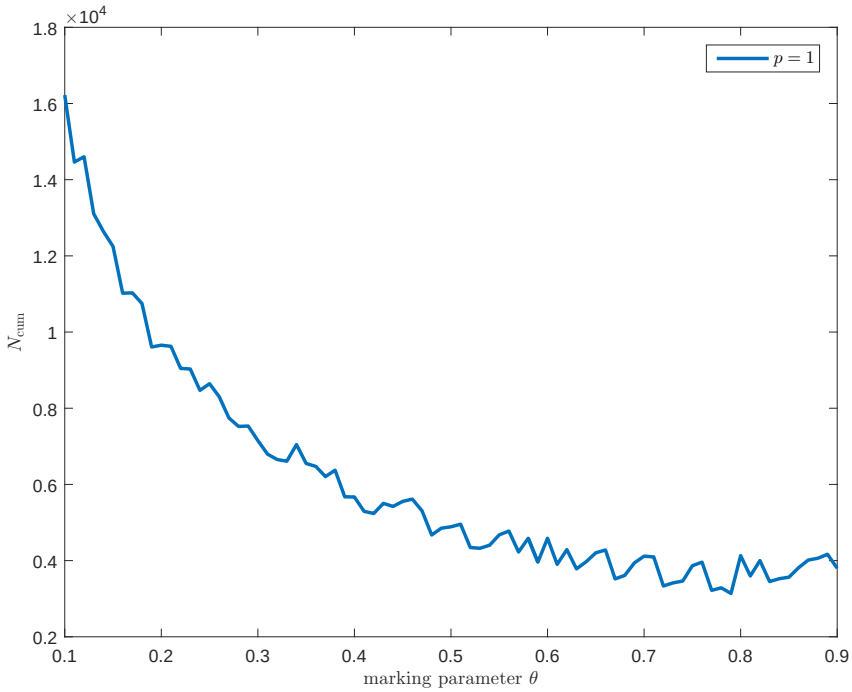

Figure 7.12: Comparison of the marking parameter θ in terms of the cumulative number of knots $N_{\text{cum}} := \sum_{j=0}^{\ell} |\mathcal{K}_j|$ necessary to reach the prescribed accuracy $\mu_\ell < 10^{-3}$.

7.3 Neumann problem with singular right-hand side ϕ

We consider the Neumann problem

$$Wu = (1/2 - K')\phi \quad \text{on } \Gamma, \tag{7.5}$$

where Γ is the boundary of the L-shaped domain Ω shown in Figure 7.13 and ϕ is given by the normal derivative of the function

$$P(x, y) = r^\tau \cos(\tau\alpha).$$

Here, (r, α) denotes the polar coordinates of $(x, y) \in \mathbb{R}^2$ and $\tau = 2/3$. The normal derivative of P reads

$$\phi(x, y) = \begin{pmatrix} \cos(\alpha)\cos(\tau\alpha) + \sin(\alpha)\sin(\tau\alpha) \\ \sin(\alpha)\cos(\tau\alpha) - \cos(\alpha)\sin(\tau\alpha) \end{pmatrix} \cdot \nu(x, y) \cdot \tau \cdot r^{\tau-1}$$

and has a generic singularity at the origin.

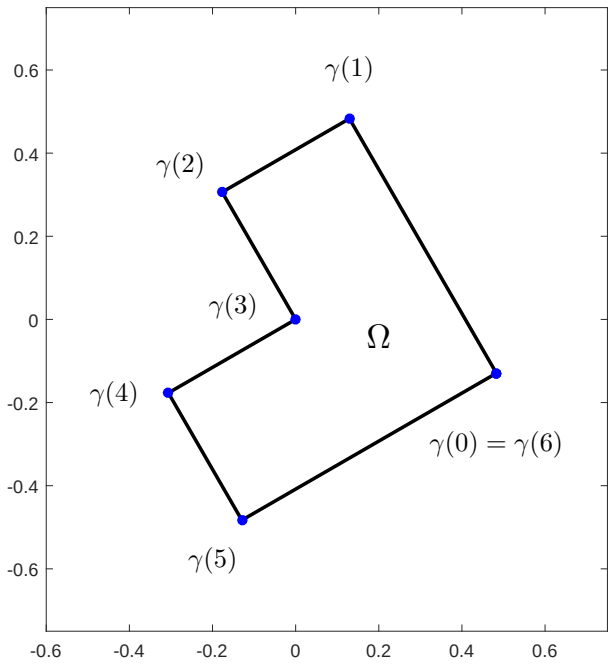

Figure 7.13: Rotated L-shape with initial nodes.

Let $\beta = \pi/12$. The rotated L-shape is parametrized on $[0, 6]$ by the NURBS curve induced by

$$\begin{aligned} p_\gamma &= 1, \\ \check{\mathcal{K}}^\gamma &= (0, 0, 1, 2, 3, 4, 5, 6, 6), \\ \mathcal{W}^\gamma &= (1, 1, 1, 1, 1, 1, 1, 1, 1), \\ (C_i)_{i=1}^{N_\gamma} &= \frac{1}{4} \cdot \begin{pmatrix} \cos(\beta) & \sin(\beta) \\ -\sin(\beta) & \cos(\beta) \end{pmatrix} \cdot \left(\begin{pmatrix} -1 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \end{pmatrix} \right). \end{aligned}$$

In Figure 7.14, the solution u of (7.5) as well as the corresponding discrete solution are plotted for different refinements. The meshes are obtained by the adaptive Algorithm 3.5 with $\theta = 0.5$. One can see, that even after 20 adaptive steps, the number of knots is rather small. We know that the right-hand side ϕ is singular at the origin. Hence, we expect that the adaptive Algorithm 3.5 refines mainly at the origin. This is underpinned by the histogram in Figure 7.15.

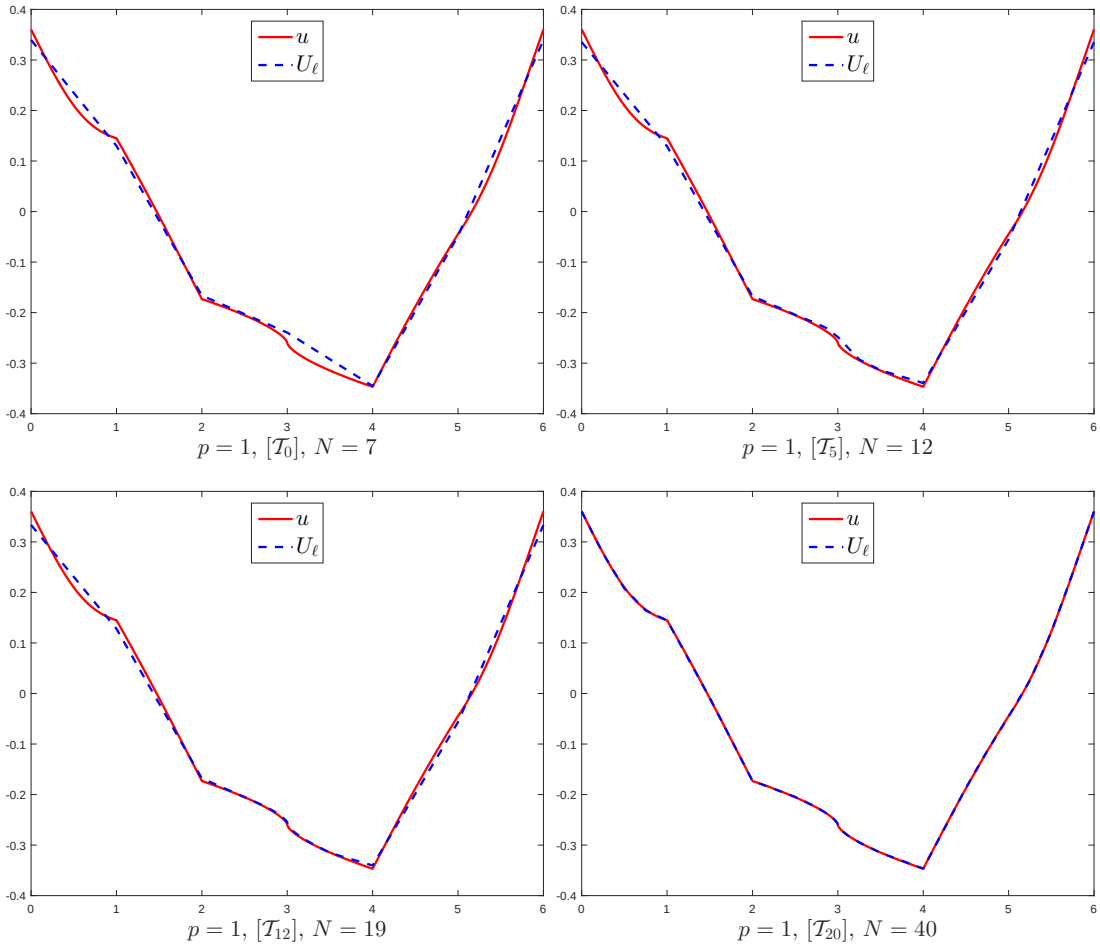

Figure 7.14: The solution u of (7.5) resp. the discrete solution U_ℓ of the according variational formulation are plotted for the initial mesh $[\mathcal{T}_0]$ and the refinements $[\mathcal{T}_5]$, $[\mathcal{T}_{12}]$, and $[\mathcal{T}_{20}]$ over the parameter domain $[0, 6]$.

Since the solution $u := P|_\Gamma$ is smooth, one would expect that the estimator decays at a rate $\mathcal{O}(N^{-3/2})$. First, we implemented the adaptive Algorithm 3.5 without any approximation of the right-hand side ϕ. We observed that neither uniform nor adaptive refinement regained this order of convergence. The problem was, that the right-hand side of the discrete variational formulation was not integrated numerically well enough. In order to solve this problem, we replaced ϕ by the L^2-projection $\Pi_\star \phi$ and incorporated the arising data oscillation terms into the overall adaptive scheme and its analysis.

We look at a uniform refinement strategy in comparison to the adaptive Algorithm 3.5 with $\theta = 0.5$. In Figure 7.16, the node-based error estimator μ_ℓ is plotted over the number of knots for both uniform and adaptive refinement. Since the function ϕ misses regularity at the origin, uniform refinement only leads to an order of convergence $\mathcal{O}(N^{-2/3})$. The adaptive strategy, on the other hand, regains the optimal order of convergence $\mathcal{O}(N^{-3/2})$. Figure 7.17 shows that this optimal order of convergence is independent of the chosen marking parameter $\theta \in \{0.1, \dots, 0.9\}$.

We prescribe an accuracy of 10^{-4} for the error estimator μ_ℓ. Figure 7.18 shows the cumuluative sum N_{cum}, cf. Definition (7.3), plotted over the marking parameter θ. We see that $\theta \approx 0.8$ is a good choice for θ.

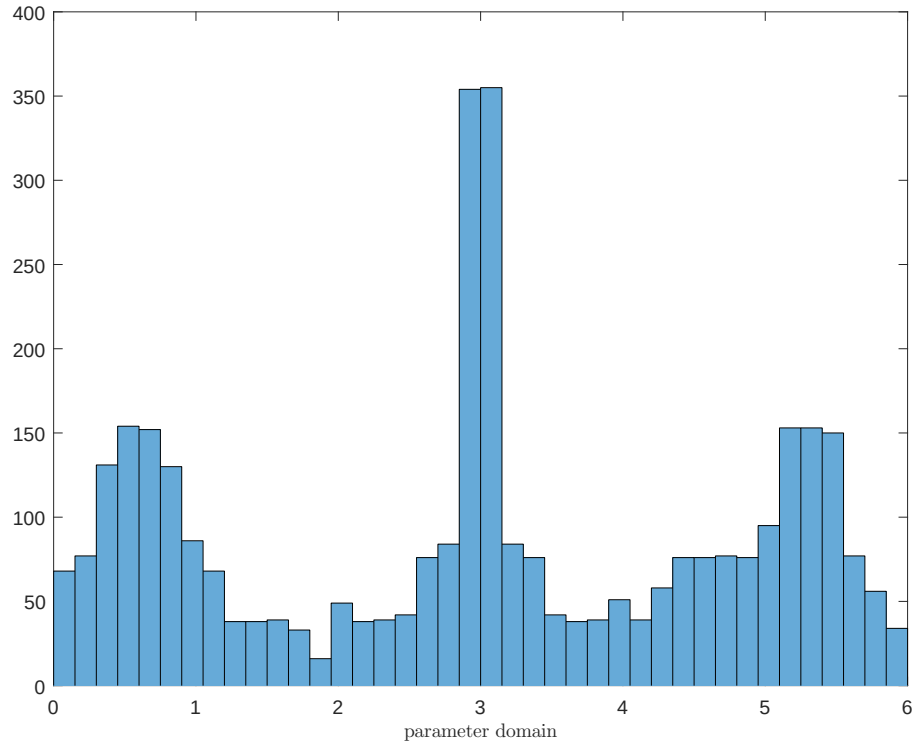

Figure 7.15: Histogram of the knot distribution of the last refinement step $[\mathcal{T}_{48}]$ with $|\mathcal{K}_{48}| = 3517$ for the adaptive Algorithm 3.5 with $\theta = 0.5$ for the Neumann problem (7.5) with singular right-hand side ϕ.

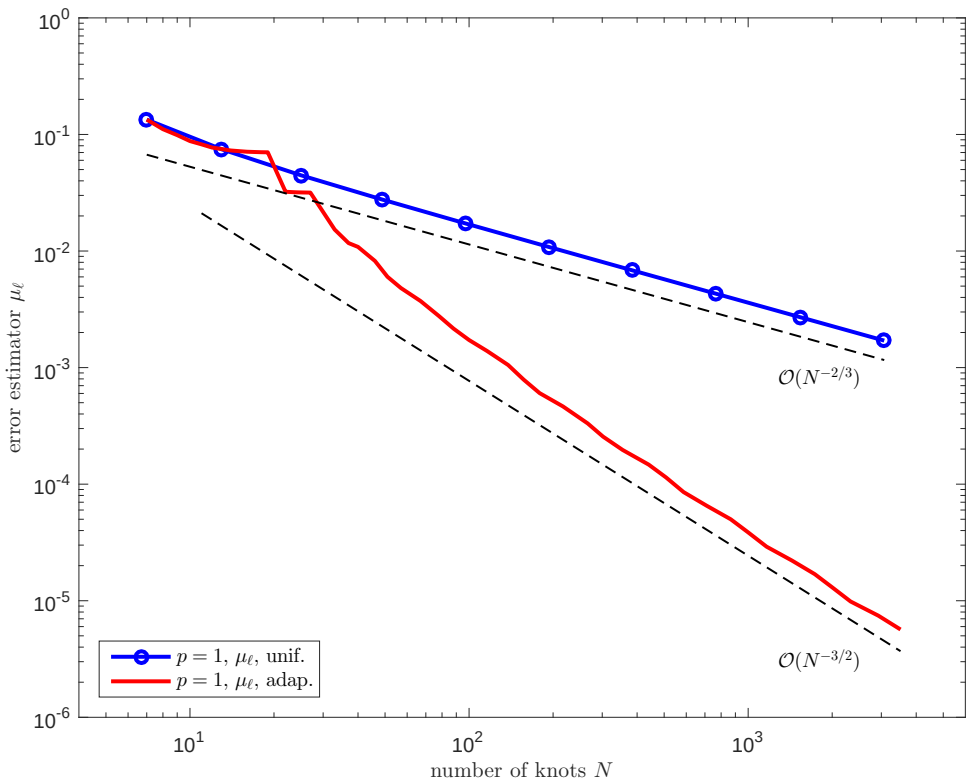

Figure 7.16: Comparison of uniform and adaptive refinement on the rotated L-shape for the Neumann problem (7.5).

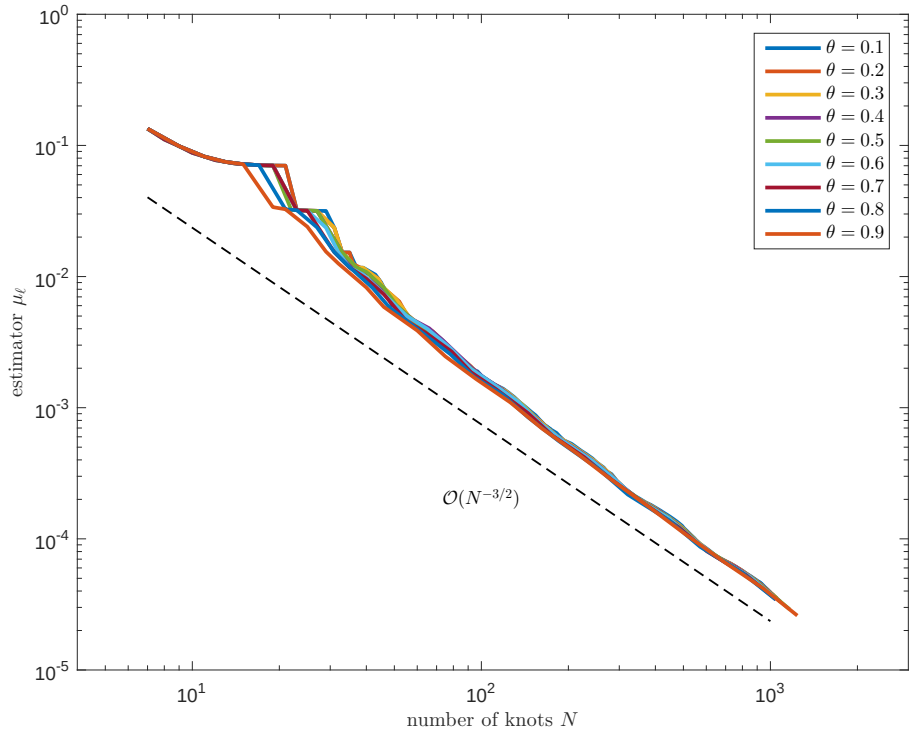

Figure 7.17: Comparison of different marking parameters θ for the adaptive Algorithm 3.5 on rotated L-shape for the Neumann problem (7.5).

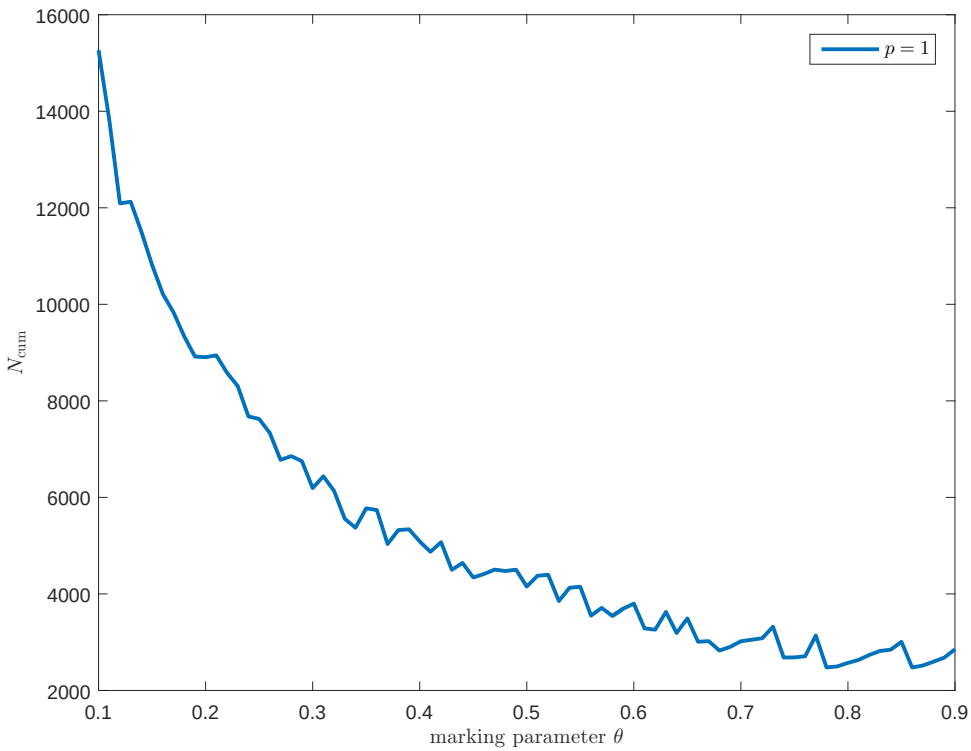

Figure 7.18: Comparison of the marking parameter θ in terms of the cumulative number of knots $N_{\text{cum}} := \sum_{j=0}^{\ell} |\mathcal{K}_j|$ necessary to reach the prescribed accuracy $\mu_\ell < 10^{-4}$.

A Implementation

This section is dedicated to the implementation of the adaptive Algorithm 3.5. All the main files needed are given below.
For better performance, most parts are coded in C. However, they can all be used in MATLAB via MEX files. One example of such a MEX file is also given.
The algorithms used are mostly based on the approaches discussed in Section 6 and [de 86].

A.1 Functions.h

In this file, some basic functions and examples for the Neumann boundary conditions ϕ are defined, cf. [Gan14, Appendix B].

```
#ifndef _Functions_h
#define _Functions_h

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define EPS 1e-14

inline int mod(int x, int y){
    // returns x mod y for integer x and positive integer y
    int z=x%y;
    if (z<0) {z=z+y;}
    return z;
}


inline int floordiv(int x, int y) {
    // returns floor(((double)x)/((double)y)) for int x and pos. int y
    if (x >= 0) {return x/y;}else{return (x-y+1)/y;}
}


inline double min(double x,double y){
    if (x<=y){return x;}else{return y;}
}


inline double max(double x, double y){
    if (x<=y){return y;}else{return x;};
}


inline int nearly_equal(double x,double y){
    if (x==0){
        if(fabs(y)<=EPS)
            return 1;
        else
            return 0;
    }
    if (y==0){
        if(fabs(x)<=EPS)
            return 1;
        else
            return 0;
    }
    if (fabs(x-y)<=EPS*max(fabs(x),fabs(y))){
        return 1;
    }else{
        return 0;
    }
}


inline int nearly_leq(double x,double y){
```

```
    if (nearly_equal(x,y) || (x<y)){
        return 1;
    }else{
        return 0;
    }
}


inline double norm(double* vector){
    return sqrt(vector[0]*vector[0]+vector[1]*vector[1]);
}


inline void mvm(double* output,double* A,int n,double* x) {
    int i;
    int j;

    for (i=0; i<n; i=i+1) {
        output[i] = 0;
        for (j=0; j<n; j=j+1) {
            output[i] = output[i] + A[i+j*n]*x[j];
        }
    }
}


inline double phi(double* x){
    // returns phi(x) for x in Gamma for various geometries

    // L-shape
    // singular right-hand side:
     double tau=2.0/3.0;
     double r=norm(x);
     double alpha=atan2(x[1],x[0]);
     if (nearly_leq(0,x[0]) && nearly_leq(x[1],0))
         return (tau*pow(r,tau-1)*(cos(alpha-tau*alpha)
                        -sin(alpha-tau*alpha)))/sqrt(2);
     else if (nearly_leq(0,x[0]))
         return (tau*pow(r,tau-1)*(cos(alpha-tau*alpha)
                        +sin(alpha-tau*alpha)))/sqrt(2);
     else if (nearly_leq(0.25,x[1]))
         return (tau*pow(r,tau-1)*(-cos(alpha-tau*alpha)
                        +sin(alpha-tau*alpha)))/sqrt(2);
     else if (nearly_leq(0,x[1]))
         return (tau*pow(r,tau-1)*(-cos(alpha-tau*alpha)
                        -sin(alpha-tau*alpha)))/sqrt(2);
     else if (nearly_leq(-0.25,x[1]))
         return (tau*pow(r,tau-1)*(-cos(alpha-tau*alpha)
                        +sin(alpha-tau*alpha)))/sqrt(2);
     else if (nearly_leq(x[1],-0.25))
         return (tau*pow(r,tau-1)*(-cos(alpha-tau*alpha)
                        -sin(alpha-tau*alpha)))/sqrt(2);

}

#endif
```

A.2 Structures.h

In this file, the main structures `NURBSData` and `QuadData` are defined. They are used to store NURBS and quadrature data, cf. [Gan14, Appendix B].

```
#ifndef _Structures_h_
#define _Structures_h_
#include <stdio.h>
#include <stdlib.h>

```

```
6
7  // NURBSData
8  typedef struct _NURBSData_{
9      double a; // left point of real interval [a,b]
10     double* knots; // non-decreasing points in (a,b],
11     //t_1=knots[0],...,t_N=knots[N-1]=b with #t_i<=p+1
12     int N; // number of t_i in (a,b]
13     double* weights; // positive weights correspoding to knots
14     int p; // polynomial degree
15     double* nodes; // unique(knots)
16     int n; // number of nodes
17     int is_rational; // 0 if all weights are one, 1 else
18 }NURBSData;
19
20
21 NURBSData* new_NURBSData(double a,double* knots,int N,double* weights,int p,
22                         double* nodes,int n,int is_rational) {
23     NURBSData* Data = malloc(sizeof(NURBSData));
24
25     Data->a = a;
26     Data->knots = knots;
27     Data->N = N;
28     Data->weights = weights;
29     Data->p = p;
30     Data->nodes=nodes;
31     Data->n=n;
32     Data->is_rational=is_rational;
33
34     return Data;
35 }
36
37
38 NURBSData* del_NURBSData(NURBSData* Data) {
39     if (Data != NULL) {
40         free(Data);
41     }
42     return NULL;
43 }
44
45
46 // get NURBSData
47 inline double get_NURBSData_a(NURBSData* Data){return Data->a;}
48 inline double* get_NURBSData_knots(NURBSData* Data){return Data->knots;}
49 inline int get_NURBSData_N(NURBSData* Data){return Data->N;}
50 inline double* get_NURBSData_weights(NURBSData* Data){return Data->weights;}
51 inline int get_NURBSData_p(NURBSData* Data){return Data->p;}
52 inline double* get_NURBSData_nodes(NURBSData* Data){return Data->nodes;}
53 inline int get_NURBSData_n(NURBSData* Data){return Data->n;}
54 inline int get_NURBSData_is_rational(NURBSData* Data)
55 {return Data->is_rational;}
56
57
58
59
60 // QuadData
61 typedef struct _QuadData_{
62     double* nodes; // nodes of Gauss quadrature in [0,1]
63     double* weights; // weights of Gauss quadrature in [0,1]
64     int n; // number of nodes resp. weights
65 }QuadData;
66
67
68 QuadData* new_QuadData(double* nodes,double* weights,int n) {
69     QuadData* Data = malloc(sizeof(QuadData));
70
71     Data->nodes = nodes;
72     Data->weights = weights;
73     Data->n = n;
74
75     return Data;
```

```
}


QuadData* del_QuadData(QuadData* Data) {
    if (Data != NULL) {
        free(Data);
    }
    return NULL;
}


// get QuadData
inline double* get_QuadData_nodes(QuadData* Data){return Data->nodes;}
inline double* get_QuadData_weights(QuadData* Data){return Data->weights;}
inline int get_QuadData_n(QuadData* Data){return Data->n;}

#endif
```

A.3 Spline.h and Spline.c

These two files contain the functions needed to evaluate and derive NURBS, cf. [Gan14, Appendix B].

```
#ifndef _Spline_h
#define _Spline_h

#include "Structures.h"
#include "Functions.h"

/* parameters:
 Data...NURBSData*, see Structures.h
 i...arbitrary unfixed integer
 t...arbitrary unfixed evaluation point in [a,b)
 wcpoints1...first component of weighted control points w_l*C_l
 for geometry corresponding to knots
 wcpoints2...second component of weighted control points w_l*C_l
 for geometry corresponding to knots
 */


inline double knotseq(NURBSData* Data,int i);
// returns t_i, (knots (b-a)-periodic extended)

int find_Span(NURBSData* Data, double t);
// returns integer index between 0 and N-1 with t in [t_index,t_{index+1})
// using binary search

int find_Span_nodes(NURBSData* Data, double t);
// returns integer index between 0 and n-1 with t in [t_index,t_{index+1})
// using binary search

double deBoor_help(NURBSData* Data,int i, double* coeffdata,int coefftype,
                   int l,double t);
// help function for function deBoor which returns
// a_i^[l](t)=(1-\beta_{i,p-l+1}(t))*a_{i-1}^[l-1](t)
// +\beta_{i,p-l+1}(t)*a_i^[l-1](t), where a_i^[0]=c_{i+1}
// (see B-Spline Basics: Algorithm 9 with k=p+1 and a_i=c_{i+1}),
// for parameter description see function deBoor


double deBoorDeriv_help(NURBSData* Data, int i, double* coeffdata,
                        int coefftype,int l,double t);
// help function for function deBoorDeriv which returns
// (not derivative of function a_i^[l])
// a_i'^[l](t)=(1-\beta_{i,p-l})*a_{i-1}'^[l-1]+\beta_{i,p-l}*a_i'^[l-1] with
// a_i'^[0]=a_i'=p*(a_i-a_{i-1})/(t_{i+p}-t_i)
// (see B-Spline Basics: Algorithm 9 and Algorithm 10 with k=p+1 and
// a_i=c_{i+1}), for parameter description see function deBoorDeriv
```

```
double deBoor(NURBSData* Data, double* coeffdata, int coefftype,double t);
// returns sum_{i in Z}c_i*B_{i,p}(t) using deBoor algorithm, where:
// if coefftype=0: coeffdata[i+p-1]=c_i for i=(1-p)...(N-#b+1)
// if coefftype=1: coeffdata[i-1]=c_i for i=1...N, c_i N-periodic extended
// if coefftype=2: coeffdata[0] is unique integer i where c_i!=0, it is c_i=1

double deBoorDeriv(NURBSData* Data,double* coeffdata,int coefftype,double t);
// returns sum_{i in Z}c_i*D^r(B_{i,p})(t) using deBoor algorithm and formula
// for derivative coefficients c_i', where:
// if coefftype=0: coeffdata[i+p-1]=c_i for i=(1-p)...(N-#b+1)
// if coefftype=1: coeffdata[i-1]=c_i for i=1...N, c_i N-periodic extended
// if coefftype=2: coeffdata[0] is unique integer i where c_i!=0, it is c_i=1

double eval_NURBS(NURBSData* Data, int i, double t);
// returns R_{i,p}(t)


double eval_NURBSDeriv(NURBSData* Data,int i, double t);
// returns D^r(R_{i,p})(t) by using quotient rule


void eval_NURBSCurve(double* output, NURBSData* Data, double* wcpoints1,
                     double* wcpoints2, double t);
// turns output[0] resp. output[1] into first resp. second coordinate of
// sum_{l in Z} C_l*R_{l,p}(t)
// =(sum_{l in Z} w_l*C_l*B_{l,p}(t))/(sum_{l in Z} w_l*B_{l,p}(t))

void eval_NURBSCurveDeriv(double* output, NURBSData* Data, double* wcpoints1,
                          double* wcpoints2, double t);
// turns output[0] resp. output[1] into first resp. second coordinate of
// right derivative of
// sum_{l in Z} C_l*R_{l,p}(t)
// =(sum_{l in Z} w_l*C_l*B_{l,p}(t))/(sum_{l in Z} w_l*B_{l,p}(t))
// using quotient rule

double eval_NURBSComb(NURBSData* Data,double* wcoeffs, double t);
// returns sum_{l in Z} c_l*R_{l,p}(t), where wcoeffs[l+p-1]=w_l*c_l
// for l=(1-p),...,(N-#b+1)

double eval_NURBSCombDeriv(NURBSData* Data,double* wcoeffs, double t);
// returns sum_{l in Z} c_l*R_{l,p}'(t), where wcoeffs[l+p-1]=w_l*c_l
// for l=(1-p),...,(N-#b+1)


#endif
```

```
#include "Spline.h"


inline double knotseq(NURBSData* Data,int i){

    double a= get_NURBSData_a(Data);
    double* knots=get_NURBSData_knots(Data);
    int N=get_NURBSData_N(Data);
    double b=knots[N-1];

    // t_i=t_{i mod N}+(b-a)*floor(i/N)
    if (mod(i,N)!=0) {
        return knots[mod(i,N)-1]+(b-a)*floordiv(i,N);
    } else {
        return a+(b-a)*floordiv(i,N); // a=t_0
    }
}


int find_Span(NURBSData* Data, double t){
```

```
    int N=get_NURBSData_N(Data);
    int low=0, high=N+1; // t in [t_low,t_high)
    int index=(low+high)/2;
    double t_index=knotseq(Data,index); // t_index
    double t_indexp1=knotseq(Data,index+1); // t_{index+1}

    while (t<t_index || t_indexp1<=t) {
        if (t<0){
            return 0;
        }

        if (t<t_index) {
            high=index;
        } else {
            low=index;
        }
        index=(low+high)/2;
        t_index=knotseq(Data,index);
        t_indexp1=knotseq(Data,index+1);
    }

    return index;
}




int find_Span_nodes(NURBSData* Data, double t){

    int n=get_NURBSData_n(Data);
    double* nodes=get_NURBSData_nodes(Data);
    double a=get_NURBSData_a(Data);

    if (a<=t && t<nodes[0]){
        return 0;
    }

    int low=0, high=n-1; // t in [t_low,t_high)
    int index=(low+high)/2;

    while (t<nodes[index] || nodes[index+1]<=t) {
        if (t<nodes[index]) {
            high=index;
        } else {
            low=index;
        }
        index=(low+high)/2;
    }
    return index+1;
}




double deBoor_help(NURBSData* Data,int i, double* coeffdata,int coefftype,
                   int l,double t){

    int N=get_NURBSData_N(Data);
    int p=get_NURBSData_p(Data);
    int k,m;
    double beta; // beta_{k,p-m+1}
    double t_k,t_kppmmp1; // t_k, t_{k+p-m+1}
    double A[l+1]; // help vector

    // calculation of a_k^[0]=a_k=c_{k+1}=A[k+l-i]
    for (k=(i-l);k<=i;k=k+1){
        switch(coefftype){
            case 0:
                A[k+l-i]=coeffdata[k+p]; break;
```

```
            case 1:
                if (mod(k+1,N)!=0) {
                    A[k+l-i]=coeffdata[mod(k+1,N)-1];
                } else {
                    A[k+l-i]=coeffdata[N-1];
                } break;
            case 2:
                // c_{k+1}=delta_{k+1,coeffdata[0]}
                if ( ((int) coeffdata[0]) == (k+1) ){
                    A[k+l-i]=1;
                } else {
                    A[k+l-i]=0;
                }break;
        }
    }

    // calculation of a_k^[m]=A[k+l-i-m]
    for (m=1;m<=l;m=m+1){
        for (k=(i-l+m);k<=i;k=k+1){
            t_k=knotseq(Data,k);
            t_kppmmp1=knotseq(Data,k+p-m+1);
            // \beta_{k,p-m+1}
            if (!(nearly_equal(t_k,t_kppmmp1))) {
                beta=(t-t_k)/(t_kppmmp1-t_k);
            } else {
                beta=0;
            }
            // a_k^[m]=(1-\beta_{k,p-m+1})*a_{k-1}^[m-1]
            // + \beta_{k,p-m+1}*a_k^[m-1]
            A[k+l-i-m]=(1-beta)*A[k+l-i-m] + beta*A[k+l-i-m+1];
        }
    }

    return A[0];
}


double deBoorDeriv_help(NURBSData* Data, int i, double* coeffdata,
                        int coefftype,int l,double t){

    int N=get_NURBSData_N(Data);
    int p=get_NURBSData_p(Data);
    int k,m;
    double beta; // beta_{k,p-m}
    double t_k,t_kppmm,t_kpp; // t_k, t_{k+p-m}, t_{k+p}
    double tmp1,tmp2;
    double A_prime[l+1]; // help vector A'


    // calculation of a_k'^[0]=a_k'=c_{k+1}'=A'[k+l-i]
    for (k=(i-l);k<=i;k=k+1){
        t_k=knotseq(Data,k);
        t_kpp=knotseq(Data,k+p);
        switch(coefftype){
            case 0:
                if (nearly_equal(t_kpp,t_k)){
                    A_prime[k+l-i]= 0;
                }else{
                    A_prime[k+l-i]=p*(coeffdata[k+p]-coeffdata[k+p-1])
                    /(t_kpp-t_k);
                } break;
            case 1:
                if (mod(k,N)!=0) {
                    tmp1=coeffdata[mod(k,N)-1]; // c_k
                } else {
                    tmp1=coeffdata[N-1]; // c_k
                }
                if (mod(k+1,N)!=0) {
                    tmp2=coeffdata[mod(k+1,N)-1]; // c_{k+1}
                } else {
```

```
                    tmp2=coeffdata[N-1]; // c_{k+1}
                }
                if (nearly_equal(t_kpp,t_k)){
                    A_prime[k+l-i]= 0;
                } else {
                    A_prime[k+l-i]=p*(tmp2-tmp1)/(t_kpp-t_k);
                } break;
            case 2:
                // c_{k+1}=delta_{k+1,coeffdata[0]}
                if (((int) coeffdata[0]) == (k+1)){
                    if (nearly_equal(t_kpp,t_k)){
                        A_prime[k+l-i]=0;
                    }else{
                        A_prime[k+l-i]=p/(t_kpp-t_k);
                    }
                } else {
                    if (((int) coeffdata[0]) == k){
                        if (nearly_equal(t_kpp,t_k)){
                            A_prime[k+l-i]= 0;
                        }else{
                            A_prime[k+l-i]= -p/(t_kpp-t_k);
                        }
                    } else {
                        A_prime[k+l-i]=0;
                    }
                } break;
        }
    }

    // calculation of a_k'^[m]=A'[k+l-i-m]
    for (m=1;m<=l;m=m+1){
        for (k=(i-l+m);k<=i;k=k+1){
            t_k=knotseq(Data,k);
            t_kppmm=knotseq(Data,k+p-m);
            // \beta_{k,p-m}
            if (!(nearly_equal(t_k,t_kppmm))) {
                beta=(t-t_k)/(t_kppmm-t_k);
            } else {
                beta=0;
            }
            // a_k'^[m]=(1-\beta_{k,p-m})*a_{k-1}'^[m-1]
            // +\beta_{k,p-m}*a_k'^[m-1]
            A_prime[k+l-i-m]=(1-beta)*A_prime[k+l-i-m]
            + beta*A_prime[k+l-i-m+1];
        }
    }

    return A_prime[0];
}


double deBoor(NURBSData* Data, double* coeffdata, int coefftype,double t){

    int p=get_NURBSData_p(Data);
    double index=find_Span(Data,t);

    return deBoor_help(Data,index,coeffdata,coefftype,p,t);
}


double deBoorDeriv(NURBSData* Data,double* coeffdata,int coefftype,double t){

    int p=get_NURBSData_p(Data);
    double index=find_Span(Data,t);

    return deBoorDeriv_help(Data,index,coeffdata,coefftype,p-1,t);
}


double eval_NURBS(NURBSData* Data, int i, double t){
```

```
    int N=get_NURBSData_N(Data);
    double* weights=get_NURBSData_weights(Data);
    int is_rational=get_NURBSData_is_rational(Data);
    double weight, nominator,denominator;
    double coeffdata[1]={(double)i};

    if (is_rational==0){
        return deBoor(Data,coeffdata,2,t);
    }else{
        if (mod(i,N)!=0) {
            weight=weights[mod(i,N)-1]; // w_i
        } else {
            weight=weights[N-1]; // w_i
        }
        nominator=weight*deBoor(Data,coeffdata,2,t);
        denominator=deBoor(Data,weights,1,t);
        return nominator/denominator;
    }
}


double eval_NURBSDeriv(NURBSData* Data,int i, double t){

    int N=get_NURBSData_N(Data);
    double* weights=get_NURBSData_weights(Data);
    int is_rational=get_NURBSData_is_rational(Data);
    double weight, nominator, nominator_prime,denominator, denominator_prime;
    double coeffdata[1]={(double)i};

    if (is_rational==0){
        return deBoorDeriv(Data,coeffdata,2,t);
    }else{
        if (mod(i,N)!=0) {
            weight=weights[mod(i,N)-1]; // w_i
        } else {
            weight=weights[N-1]; // w_i
        }
        nominator=weight*deBoor(Data,coeffdata,2,t);
        nominator_prime=weight*deBoorDeriv(Data,coeffdata,2,t);
        denominator=deBoor(Data,weights,1,t);
        denominator_prime=deBoorDeriv(Data,weights,1,t);
        return (nominator_prime*denominator-nominator*denominator_prime)
        /(denominator*denominator);
    }
}


void eval_NURBSCurve(double* output, NURBSData* Data, double* wcpoints1,
                     double* wcpoints2, double t){

    int N=get_NURBSData_N(Data);
    double* weights=get_NURBSData_weights(Data);
    int is_rational=get_NURBSData_is_rational(Data);
    double nominator1, nominator2, denominator;

    if (is_rational==0){
        output[0]=deBoor(Data,wcpoints1,1,t);
        output[1]=deBoor(Data,wcpoints2,1,t);
    }else{
        nominator1=deBoor(Data,wcpoints1,1,t);
        nominator2=deBoor(Data,wcpoints2,1,t);
        denominator=deBoor(Data,weights,1,t);

        output[0]=nominator1/denominator;
        output[1]=nominator2/denominator;
    }
}
```

```
void eval_NURBSCurveDeriv(double* output, NURBSData* Data, double* wcpoints1,
                          double* wcpoints2, double t){

    int N=get_NURBSData_N(Data);
    double* weights=get_NURBSData_weights(Data);
    int is_rational=get_NURBSData_is_rational(Data);
    double nominator1,nominator2,denominator;
    double nominator1_prime,nominator2_prime,denominator_prime;

    if (is_rational==0){
        output[0]=deBoorDeriv(Data,wcpoints1,1,t);
        output[1]=deBoorDeriv(Data,wcpoints2,1,t);
    }else{
        nominator1=deBoor(Data,wcpoints1,1,t);
        nominator2=deBoor(Data,wcpoints2,1,t);
        denominator=deBoor(Data,weights,1,t);
        nominator1_prime=deBoorDeriv(Data,wcpoints1,1,t);
        nominator2_prime=deBoorDeriv(Data,wcpoints2,1,t);
        denominator_prime=deBoorDeriv(Data,weights,1,t);

        output[0]=(nominator1_prime*denominator-nominator1*denominator_prime)
        /(denominator*denominator);
        output[1]=(nominator2_prime*denominator-nominator2*denominator_prime)
        /(denominator*denominator);
    }
}


double eval_NURBSComb(NURBSData* Data,double* wcoeffs, double t) {

    double* knots=get_NURBSData_knots(Data);
    int N=get_NURBSData_N(Data);
    double* weights=get_NURBSData_weights(Data);
    int p=get_NURBSData_p(Data);
    int is_rational=get_NURBSData_is_rational(Data);
    double b=knots[N-1];
    int multb=0; // #b
    while (nearly_equal(knots[N-multb-1],b)) {multb=multb+1;}
    double weight,nominator,denominator;

    if (is_rational==0){
        return deBoor(Data,wcoeffs,0,t);
    }else{
        nominator=deBoor(Data,wcoeffs,0,t);
        denominator=deBoor(Data,weights,1,t);
        return nominator/denominator;
    }
}


double eval_NURBSCombDeriv(NURBSData* Data,double* wcoeffs, double t) {

    double* knots=get_NURBSData_knots(Data);
    int N=get_NURBSData_N(Data);
    double* weights=get_NURBSData_weights(Data);
    int p=get_NURBSData_p(Data);
    int is_rational=get_NURBSData_is_rational(Data);
    double b=knots[N-1];
    int multb=0; // #b
    while (nearly_equal(knots[N-multb-1],b)) {multb=multb+1;}
    double weight,nominator,denominator;

    if (is_rational==0){
        return deBoorDeriv(Data,wcoeffs,0,t);
    }else{
        nominator=deBoor(Data,weights,1,t)*deBoorDeriv(Data,wcoeffs,0,t)
                    -deBoor(Data,wcoeffs,0,t)*deBoorDeriv(Data,weights,1,t);
        denominator=deBoor(Data,weights,1,t)*deBoor(Data,weights,1,t);

        return nominator/denominator;
```

```
    }
}
```

A.4 phi_approx.h and phi_approx.c

These two files provide the functions to compute the approximation $\Pi_h \phi$ of the given Neumann boundary condition ϕ.

```
#ifndef _phiapprox_
#define _phiapprox_

#include <math.h>
#include <stdio.h>
#include "Spline.h"

void build_phi_approx(double* output,int p, NURBSData* Data_Gamma,
            double* wcpoints1_gam, double* wcpoints2_gam,
            NURBSData* Data_Basis,QuadData* Data_Gauss);


#endif
```

```
#include "phi_approx.h"

//-----------------------------------------------------------------------------
// build_phi_approx
//-----------------------------------------------------------------------------

void build_phi_approx(double* output, int p, NURBSData* Data_Gamma,
            double* wcpoints1_gam, double* wcpoints2_gam,
            NURBSData* Data_Basis,QuadData* Data_Gauss){

    double* nodes_gauss=get_QuadData_nodes(Data_Gauss);
    double* weights_gauss=get_QuadData_weights(Data_Gauss);
    int n_gauss=get_QuadData_n(Data_Gauss);

    int N=get_NURBSData_N(Data_Basis);
    int n=get_NURBSData_n(Data_Basis);

    int j_tmp=0;
    int counter=0;
    int k1=0;
    int i=0;
    int j=0;
    int k=0;
    int q1=0;

    double t_k1m1=0;
    double t_k1=0;
    double H_k1=0;

    double intpoint1=0;

    double tmp[2];
    double tmp2[2];
    double y[p+1];
    double x[p+1];
    double lhs[n][p+1][p+1];
    double rhs[n][p+1];

    double L[p+1][p+1];
    double U[p+1][p+1];


    double knots_tmp[p+1];
```

```
    for (i=0;i<p+1;++i){
        knots_tmp[i]=1;
    }
    double weights_tmp[p+1];
    for (i=0;i<p+1;++i){
        weights_tmp[i]=1;
    }
    double nodes_tmp[1];
    nodes_tmp[0]=1;
    int is_rational_tmp=0;
    NURBSData* Basis_tmp=new_NURBSData(0,knots_tmp,p+1,weights_tmp,p,nodes_tmp,1,0);



    double sum=0;

    for (i=0;i<n*(p+1);++i){
        output[i]=0;
    }

    //build left hand side
    for (k1=1;k1<=N;++k1){
        t_k1m1=knotseq(Data_Basis,k1-1);
        t_k1=knotseq(Data_Basis,k1);
        H_k1=t_k1-t_k1m1;

        if (0<H_k1){

            for (j_tmp=0; j_tmp<p+1;++j_tmp){
                knots_tmp[j_tmp]=t_k1;
            }
            nodes_tmp[0]=t_k1;
            Basis_tmp=del_NURBSData(Basis_tmp);
            Basis_tmp=new_NURBSData(t_k1m1,knots_tmp,p+1,weights_tmp,p,nodes_tmp,1,0);

            for (i=0;i<=p;++i){
                for (j=0;j<=p;++j){
                    lhs[counter][i][j]=0;

                    for (q1=0;q1<n_gauss;++q1){
                        intpoint1=t_k1m1+H_k1*nodes_gauss[q1];
                        eval_NURBSCurveDeriv(tmp,Data_Gamma,wcpoints1_gam,
                                                wcpoints2_gam,intpoint1);

                        lhs[counter][i][j] += weights_gauss[q1]
                                    *eval_NURBS(Basis_tmp,i-p+1,intpoint1)
                                    *eval_NURBS(Basis_tmp,j-p+1,intpoint1)
                                    *norm(tmp);
                    }
                    lhs[counter][i][j] *= H_k1;
                }
            }
            ++counter;
        }
    }

    counter=0;
    //build right hand side
    for (k1=1;k1<=N;++k1){
        t_k1m1=knotseq(Data_Basis,k1-1);
        t_k1=knotseq(Data_Basis,k1);
        H_k1=t_k1-t_k1m1;

        if (0<H_k1){

            for (j_tmp=0; j_tmp<p+1;++j_tmp){
                knots_tmp[j_tmp]=t_k1;
            }
            nodes_tmp[0]=t_k1;
            Basis_tmp=del_NURBSData(Basis_tmp);
```

```
            Basis_tmp=new_NURBSData(t_k1m1,knots_tmp,p+1,weights_tmp,p,nodes_tmp,1,0);



            for (i=0;i<=p;++i){
                rhs[counter][i]=0;

                for (q1=0;q1<n_gauss;++q1){
                    intpoint1=t_k1m1+H_k1*nodes_gauss[q1];
                    eval_NURBSCurveDeriv(tmp,Data_Gamma,wcpoints1_gam,
                                                wcpoints2_gam,intpoint1);
                    eval_NURBSCurve(tmp2,Data_Gamma,wcpoints1_gam,
                                            wcpoints2_gam,intpoint1);

                    rhs[counter][i] += weights_gauss[q1]
                                    *eval_NURBS(Basis_tmp,i-p+1,intpoint1)
                                    *norm(tmp)*phi(tmp2);
                }
                rhs[counter][i] *= H_k1;
            }
            ++counter;
        }
    }

    counter=0;

    //solve system for each element via LU-decomposition
    for (k1=1;k1<=N;++k1){
        t_k1m1=knotseq(Data_Basis,k1-1);
        t_k1=knotseq(Data_Basis,k1);
        H_k1=t_k1-t_k1m1;
        if (0<H_k1){
            //compute L and U
            for (i=0;i<=p;++i){
                L[i][i]=1;
                for (k=i;k<=p;++k){
                    if (k>=i){
                        sum=0;
                        for (j=0;j<i;++j){
                            sum += L[i][j]*U[j][k];
                        }
                        U[i][k]=lhs[counter][i][k]-sum;
                    }
                    if (k>i){
                        sum=0;
                        for (j=0;j<i;++j){
                            sum += L[k][j]*U[j][i];
                        }
                        L[k][i]=(lhs[counter][k][i]-sum)/U[i][i];
                    }
                }
            }

            //solve lhs*x=LU*x=L*y=rhs
            for (j=0;j<=p;++j){
                sum=0;
                for(k=0;k<j;++k){
                    sum += L[j][k]*y[k];
                }
                y[j]=rhs[counter][j]-sum;
            }

            //solve U*x=y
            for (j=p;j>=0;--j){
                sum=0;
                for (k=p;k>j;--k){
                    sum += U[j][k]*x[k];
                }
                x[j]=(y[j]-sum)/U[j][j];
```

```
            }


            for (i=0;i<=p;++i){
                output[(counter)*(p+1)+i]=x[i];
            }
            ++counter;
        }
    }
    Basis_tmp=del_NURBSData(Basis_tmp);
}
```

A.5 WMatrix.h and WMatrix.c

These two files contain everything to compute the mass matrix W_h from Section 6.

```
#ifndef _WMatrix_
#define _WMatrix_

#include <math.h>
#include <stdio.h>
#include "Spline.h"

double W_h1_integrand_smooth(NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis, double s,
            double t, int i, int k, double denominator, double Jdet);

double W_h1_integrand_log(NURBSData* Data_Basis, double s,double t,int i,
            int j, double Jdet);

double W_h1_integral_identical(NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis,
            QuadData* Data_Gauss, QuadData* Data_LogGauss,
            int i, int j, int k);

double W_h1_integral_adjacent(NURBSData* Data_Gamma,double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis,
            QuadData* Data_Gauss,QuadData* Data_LogGauss,
            int i, int j, int k1, int k2);

void build_W_h1(double* output, NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis,
            QuadData* Data_Gauss, QuadData* Data_LogGauss);

void build_W_h2(double* output, NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis,
            QuadData* Data_Gauss,QuadData* Data_LogGauss);

void build_W(double* output, NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis,
            QuadData* Data_Gauss, QuadData* Data_LogGauss);

#endif
```

```
#include "WMatrix.h"


//-------------------------------------------------------------------------
// W_h1_integrand_smooth
//-------------------------------------------------------------------------


double W_h1_integrand_smooth(NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis, double t,
            double s, int i, int j, double denominator, double Jdet){
```

```
  double tmp1[2];
  double tmp2[2];
  double diff_gam[2];
  double R_st_j=0; // R_j^\prime(s)
  double R_st_i=0; // R_i^\prime(t)

  //gamma(s)
  eval_NURBSCurve(tmp1, Data_Gamma, wcpoints1_gam, wcpoints2_gam,t);

  //gamma(t)
  eval_NURBSCurve(tmp2, Data_Gamma, wcpoints1_gam, wcpoints2_gam,s);

  //gamma(s)-gamma(t)
  diff_gam[0]=tmp1[0]-tmp2[0];
  diff_gam[1]=tmp1[1]-tmp2[1];

  //R'_j(t)
  R_st_j=eval_NURBSDeriv(Data_Basis, j, t);

  //R'_i(s)
  R_st_i=eval_NURBSDeriv(Data_Basis, i, s);

  return -log(norm(diff_gam)/fabs(denominator))/(2*M_PI)*R_st_j*R_st_i*Jdet;
}


//-----------------------------------------------------------------------------
// W_h1_integrand_log
//-----------------------------------------------------------------------------


double W_h1_integrand_log(NURBSData* Data_Basis, double t, double s, int i,
            int j, double Jdet){

    double R_st_i=0; // R_i^\prime(s)
    double R_st_j=0; // R_j^\prime(t)

    R_st_j = eval_NURBSDeriv(Data_Basis, j, t);
    R_st_i = eval_NURBSDeriv(Data_Basis, i, s);

    return R_st_j *R_st_i*Jdet/(2*M_PI);
}


//-----------------------------------------------------------------------------
// W_h1_integral_identical
//-----------------------------------------------------------------------------


double W_h1_integral_identical(NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis,
            QuadData* Data_Gauss, QuadData* Data_LogGauss,
            int i, int j, int k){

    double* nodes_gauss=get_QuadData_nodes(Data_Gauss);
    double* weights_gauss=get_QuadData_weights(Data_Gauss);
    int n_gauss=get_QuadData_n(Data_Gauss);

    double* nodes_loggauss=get_QuadData_nodes(Data_LogGauss);
    double* weights_loggauss=get_QuadData_weights(Data_LogGauss);
    int n_loggauss=get_QuadData_n(Data_LogGauss);

    int q1=0;
    int q2=0;

    double t_km1=knotseq(Data_Basis,k-1); // t_{k-1}
    double t_k=knotseq(Data_Basis,k); // t_k
    double H_k=t_k-t_km1; // H_k

```

```
    double squareint=0;
    double intpoint1=0;
    double intpoint2=0;

    double denominator;
    double Jdet;

    //smooth integrals
    for (q1=0; q1<n_gauss; q1++){
        for (q2=0; q2<n_gauss; q2++){
            intpoint1=t_km1+H_k*(1-nodes_gauss[q1]);
            intpoint2=t_km1+H_k*(1-nodes_gauss[q1]
                        +nodes_gauss[q1]*nodes_gauss[q2]);
            denominator=nodes_gauss[q1]*nodes_gauss[q2];
            Jdet=nodes_gauss[q1];

            squareint += weights_gauss[q1]*weights_gauss[q2]
                        *W_h1_integrand_smooth(Data_Gamma, wcpoints1_gam,
                                wcpoints2_gam, Data_Basis, intpoint1,
                                intpoint2, i, j, denominator, Jdet);

            intpoint1=t_km1+H_k*nodes_gauss[q1];
            intpoint2=t_km1+H_k*(nodes_gauss[q1]
                        -nodes_gauss[q1]*nodes_gauss[q2]);
            denominator=nodes_gauss[q1]*nodes_gauss[q2];
            Jdet=nodes_gauss[q1];

            squareint += weights_gauss[q1]*weights_gauss[q2]
                        *W_h1_integrand_smooth(Data_Gamma, wcpoints1_gam,
                                wcpoints2_gam, Data_Basis, intpoint1,
                                intpoint2, i, j, denominator, Jdet);
        }
    }

    //integrals with t-logarithmic singularity
    for (q1=0;q1<n_loggauss;q1++) {
        for (q2=0;q2<n_gauss;q2++) {
            intpoint1=t_km1+H_k*(1-nodes_gauss[q2]);
            intpoint2=t_km1+H_k*(1-nodes_gauss[q2]
                        +nodes_loggauss[q1]*nodes_gauss[q2]);
            Jdet=nodes_gauss[q2];

            squareint += weights_loggauss[q1]*weights_gauss[q2]
                        *W_h1_integrand_log(Data_Basis, intpoint1,
                                intpoint2, i, j, Jdet);

            intpoint1=t_km1+H_k*(nodes_gauss[q2]);
            intpoint2=t_km1+H_k*(nodes_gauss[q2]
                        -nodes_loggauss[q1]*nodes_gauss[q2]);
            Jdet=nodes_gauss[q2];

            squareint += weights_loggauss[q1]*weights_gauss[q2]
                        *W_h1_integrand_log(Data_Basis, intpoint1,
                                intpoint2, i, j, Jdet);
        }
    }

    // integrals with s-logarithmic singularity
    for (q1=0;q1<n_gauss;q1++) {
        for (q2=0;q2<n_loggauss;q2++) {
            intpoint1=t_km1+H_k*(1-nodes_loggauss[q2]);
            intpoint2=t_km1+H_k*(1-nodes_loggauss[q2]
                        +nodes_gauss[q1]*nodes_loggauss[q2]);
            Jdet=nodes_loggauss[q2];

            squareint += weights_gauss[q1]*weights_loggauss[q2]
                        *W_h1_integrand_log(Data_Basis, intpoint1,
                                intpoint2, i, j, Jdet);

            intpoint1=t_km1+H_k*nodes_loggauss[q2];
```

```
            intpoint2=t_km1+H_k*(nodes_loggauss[q2]
                        -nodes_gauss[q1]*nodes_loggauss[q2]);
            Jdet=nodes_loggauss[q2];

            squareint += weights_gauss[q1]*weights_loggauss[q2]
                        *W_h1_integrand_log(Data_Basis, intpoint1,
                                intpoint2, i, j, Jdet);
        }
    }
    return squareint;
}


//-----------------------------------------------------------------------------
// W_h1_integral_adjacent
//-----------------------------------------------------------------------------


double W_h1_integral_adjacent(NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis,
            QuadData* Data_Gauss, QuadData* Data_LogGauss,
            int i, int j, int k1, int k2){

    double* nodes_gauss=get_QuadData_nodes(Data_Gauss);
    double* weights_gauss=get_QuadData_weights(Data_Gauss);
    int n_gauss=get_QuadData_n(Data_Gauss);

    double* nodes_loggauss=get_QuadData_nodes(Data_LogGauss);
    double* weights_loggauss=get_QuadData_weights(Data_LogGauss);
    int n_loggauss=get_QuadData_n(Data_LogGauss);

    int q1=0;
    int q2=0;

    double t_k1m1=knotseq(Data_Basis,k1-1); // t_{k_1-1} == t_{k-1}
    double t_k1=knotseq(Data_Basis,k1); // t_{k_1} == t_k
    double H_k1=t_k1-t_k1m1; // H_{k_1} == H_k

    double t_k2m1=knotseq(Data_Basis,k2-1); // t_{k_2-1} == t_{m-1}
    double t_k2=knotseq(Data_Basis,k2); // t_{k_2} == t_m
    double H_k2=t_k2-t_k2m1; // H_{k_2} == H_m

    double squareint=0; // integral over square
    double intpoint1=0; // first and second integration point
    double intpoint2=0;
    double denominator;
    double Jdet; // Jacobi determinant of Duffy transformation

    // smooth integrals
    for (q1=0;q1<n_gauss;q1++) {
        for (q2=0;q2<n_gauss;q2++) {
            intpoint1=t_k1m1+H_k1*nodes_gauss[q1];
            intpoint2=t_k2m1 + H_k2*(1-nodes_gauss[q1]*nodes_gauss[q2]);
            denominator=nodes_gauss[q1];
            Jdet=nodes_gauss[q1];

            squareint += weights_gauss[q1]*weights_gauss[q2]
                        *W_h1_integrand_smooth(Data_Gamma, wcpoints1_gam,
                                wcpoints2_gam, Data_Basis, intpoint1,
                                intpoint2, i, j, denominator,Jdet);

            intpoint1 = t_k1m1+H_k1*nodes_gauss[q1]*nodes_gauss[q2];
            intpoint2 = t_k2m1+H_k2*(1-nodes_gauss[q2]);
            denominator = nodes_gauss[q2];
            Jdet = nodes_gauss[q2];

            squareint += weights_gauss[q1]*weights_gauss[q2]
                        *W_h1_integrand_smooth(Data_Gamma, wcpoints1_gam,
                                wcpoints2_gam, Data_Basis, intpoint1,
                                intpoint2, i, j, denominator,Jdet);
```

```
        }
    }

    // integral with t-logarithmic singularity
    for (q1=0;q1<n_loggauss;q1++){
        for (q2=0;q2<n_gauss;q2++){
            intpoint1 = t_k1m1+H_k1*nodes_loggauss[q1];
            intpoint2 = t_k2m1+H_k2*(1-nodes_loggauss[q1]*nodes_gauss[q2]);
            Jdet = nodes_loggauss[q1];

            squareint += weights_loggauss[q1]*weights_gauss[q2]
                        *W_h1_integrand_log(Data_Basis, intpoint1,
                                intpoint2, i, j, Jdet);
        }
    }

    // integral with t-logarithmic singularity
    for (q1=0;q1<n_gauss;q1++){
        for (q2=0;q2<n_loggauss;q2++){
            intpoint1 = t_k1m1+H_k1*nodes_gauss[q1]*nodes_loggauss[q2];
            intpoint2 = t_k2m1+H_k2*(1-nodes_loggauss[q2]);
            Jdet = nodes_loggauss[q2];

            squareint += weights_gauss[q1]*weights_loggauss[q2]
                        *W_h1_integrand_log(Data_Basis, intpoint1,
                                intpoint2, i, j, Jdet);
        }
    }

    return squareint;
}


//-----------------------------------------------------------------------------
// build_W_h1
//-----------------------------------------------------------------------------


void build_W_h1(double* output, NURBSData* Data_Gamma, double* wcpoints1_gam,
        double* wcpoints2_gam, NURBSData* Data_Basis, QuadData* Data_Gauss,
        QuadData* Data_LogGauss){

    double* nodes_gauss=get_QuadData_nodes(Data_Gauss);
    double* weights_gauss=get_QuadData_weights(Data_Gauss);
    int n_gauss=get_QuadData_n(Data_Gauss);

    int N=get_NURBSData_N(Data_Basis);
    int p=get_NURBSData_p(Data_Basis);

    double tmp[2];

    int i=0;
    int j=0;
    int k1=0;
    int k2=0;
    int q1=0;
    int q2=0;

    int multb=1+p; // #b

    double R_st [N][p+1][n_gauss];
    // R_st [i-1+p][k1-i][q1] = R_{i,k1}^\prime(nodes_gauss[q1])

    double gamma1[N][n_gauss];
    // gamma1[k1-1][q1] is first component of
    // gamma(t_{k1-1}+H_k1*nodes_gauss[q1])

    double gamma2[N][n_gauss];
    // gamma2[l1-1][q1] is second component of
    // gamma(t_{l1-1}+H_l1*nodes_gauss[q1])
```

```
    double squareint=0; // integral over square
    double t_k1m1=0;
    double t_k1=0;
    double H_k1=0;
    double t_k2m1;
    double t_k2=0;
    double H_k2=0;

    // t_{k1-1},t_k1,H_k1,t_{k2-1},t_k2,H_k2
    double intpoint; // integration point


    // calculation of R_st
    // R_i_st
    for (i=1-p;i<=(N-p);i++){
        for (k1=max(i,1);k1<=min(i+p,N);k1++){
        // elements with nonemty intersection with support of R_i
            // quadrature points
            for (q1=0;q1<n_gauss;q1++){
                t_k1m1 = knotseq(Data_Basis,k1-1);
                t_k1 = knotseq(Data_Basis,k1);
                H_k1 = t_k1-t_k1m1;
                intpoint = t_k1m1+H_k1*nodes_gauss[q1];

                R_st[i-1+p][k1-i][q1] = eval_NURBSDeriv(Data_Basis, i, intpoint);

            }
        }
    }

    // calculation of gamma1, gamma2
    for (k1=1;k1<=N;k1++){
        for (q1=0;q1<n_gauss;q1++){
            t_k1m1 = knotseq(Data_Basis,k1-1);
            t_k1 = knotseq(Data_Basis,k1);
            H_k1 = t_k1-t_k1m1;
            intpoint=t_k1m1+H_k1*nodes_gauss[q1];

            eval_NURBSCurve(tmp, Data_Gamma, wcpoints1_gam, wcpoints2_gam,
                            intpoint);
            gamma1[k1-1][q1] = tmp[0];
            gamma2[k1-1][q1] = tmp[1];
        }
    }


    // calculation of W_h1_matrix
    // R_i
    for (i=1-p;i<=(N-p);i++) {
        // R_j
        for (j=1-p;j<=i;j++) {
            output[i+p-1+(j+p-1)*(N)]=0;
            // elements with nonemty intersection with support of R_j
            for (k1=max(j,1);k1<=min(j+p,N);k1++) {
                // elements with nonemty intersection with support of R_i
                for (k2=max(i,1);k2<=min(i+p,N);k2++) {
                    t_k1m1 = knotseq(Data_Basis,k1-1);
                    t_k1 = knotseq(Data_Basis,k1);
                    t_k2m1 = knotseq(Data_Basis,k2-1);
                    t_k2 = knotseq(Data_Basis,k2);
                    H_k1 = t_k1-t_k1m1;
                    H_k2 = t_k2-t_k2m1;
                    // quadrature
                    if (0<min(H_k1,H_k2)){
                        // elements with no intersection
                        squareint=0;
                        if ((!nearly_equal(t_k1m1,t_k2m1))
                            && (!nearly_equal(t_k1m1,t_k2))
                            && (!nearly_equal(t_k1,t_k2m1))
```

```
                            && (!nearly_equal(t_k1,t_k2))
                            && ((k1!=(N-p)) || (k2!=1))
                            && ((k2!=(N-p)) || (k1!=1))) {
                            for (q1=0;q1<n_gauss;q1++) {
                                for (q2=0;q2<n_gauss;q2++) {
                                    tmp[0] = gamma1[k1-1][q1] - gamma1[k2-1][q2];
                                    tmp[1] = gamma2[k1-1][q1] - gamma2[k2-1][q2];

                                    squareint += -weights_gauss[q1]
                                                *weights_gauss[q2]*log(norm(tmp))
                                                *R_st[j-1+p][k1-j][q1]
                                                *R_st[i-1+p][k2-i][q2]/(2*M_PI);
                                }
                            }
                        }
                        // elements with intersection
                        else {
                            // identical elements
                            if (k1==k2){
                                squareint = W_h1_integral_identical(Data_Gamma,
                                                wcpoints1_gam, wcpoints2_gam,
                                                Data_Basis, Data_Gauss,
                                                Data_LogGauss, i, j, k1);
                            }
                            // adjacent elements
                            else{
                                // singularity at t=0,s=1
                                if (nearly_equal(t_k1m1,t_k2)
                                    || ((k2==(N-p)) && (k1==1))){
                                    squareint = W_h1_integral_adjacent(Data_Gamma,
                                                wcpoints1_gam, wcpoints2_gam,
                                                Data_Basis, Data_Gauss,
                                                Data_LogGauss, i, j, k1, k2);
                                }
                                // singularity at t=1,s=0
                                else{
                                    squareint = W_h1_integral_adjacent(Data_Gamma,
                                                wcpoints1_gam, wcpoints2_gam,
                                                Data_Basis, Data_Gauss,
                                                Data_LogGauss, j, i, k2, k1);
                                }
                            }
                        }
                        output[i+p-1+(j+p-1)*(N-multb+1+p)]+=H_k1*H_k2*squareint;
                    }
                }
            }
            if (i!=j){
                // W symmetric
                output[j+p-1+(i+p-1)*(N-multb+1+p)] = output[i+p-1+(j+p-1)
                                                        *(N-multb+1+p)];
            }
        }
    }
    output[0]+=2*output[N-1]+output[N*N-1];
    for(i=2-p;i<=N-p-1;i++){
        output[i+p-1]+=output[i+p-1+(N-1)*N];
        output[(i+p-1)*N]+=output[N-1+(i+p-1)*N];
    }
}


//-----------------------------------------------------------------------------
// build_W_h2
//-----------------------------------------------------------------------------


void build_W_h2(double* output,NURBSData* Data_Gamma,double* wcpoints1_gam,
                double* wcpoints2_gam,NURBSData* Data_Basis,
                QuadData* Data_Gauss,QuadData* Data_LogGauss){
```

```
    double* nodes_gauss=get_QuadData_nodes(Data_Gauss);
    double* weights_gauss=get_QuadData_weights(Data_Gauss);
    int n_gauss=get_QuadData_n(Data_Gauss);

    //double* knots=get_NURBSData_knots(Data_Basis);
    int N=get_NURBSData_N(Data_Basis);
    int p=get_NURBSData_p(Data_Basis);
    double tmp[2];
    int i,j,k1,k2,q1,q2;

    //double b=knots[N-1];
    int multb=p+1; // #b

    double R_til[N-multb+1+p][p+1][n_gauss];
    // R_til[i-1+p][k1-i][q1]=\tilde{R}_{i,k1}(nodes_gauss[q1])

    double squareint; // integral over square
    double t_k1m1,t_k1,H_k1,t_k2m1,t_k2,H_k2;
    // t_{l1-1},t_l1,H_l1,t_{l2-1},t_l2,H_l2
    double intpoint; // integration point


    // calculation of R_til
    // R_i
    for (i=1-p;i<=(N-p);i++) {
        // elements with nonemty intersection with support of R_i
        for (k1=max(i,1);k1<=min(i+p,N);k1++) {
            // quadrature points
            for (q1=0;q1<n_gauss;q1++) {
                t_k1m1=knotseq(Data_Basis,k1-1);
                t_k1=knotseq(Data_Basis,k1);
                H_k1=t_k1-t_k1m1;
                intpoint=t_k1m1+H_k1*nodes_gauss[q1];
                eval_NURBSCurveDeriv(tmp,Data_Gamma,wcpoints1_gam,
                                     wcpoints2_gam,intpoint);
                R_til[i-1+p][k1-i][q1]=eval_NURBS(Data_Basis, i,intpoint)
                *norm(tmp);
            }
        }
    }

    // calculation of W_h2_matrix
    // R_i
    for (i=1-p;i<=(N-multb+1);i++) {
        // R_j
        for (j=1-p;j<=i;j++) {
            // elements with nonemty intersection with support of R_i
            if (i==N-p || j==N-p){
                output[i+p-1+(j+p-1)*N]=0;
            }
            for (k1=max(i,1);k1<=min(i+p,N);k1++) {
                // elements with nonemty intersection with support of R_j
                for (k2=max(j,1);k2<=min(j+p,N);k2++) {
                    t_k1m1 = knotseq(Data_Basis,k1-1);
                    t_k1 = knotseq(Data_Basis,k1);
                    t_k2m1 = knotseq(Data_Basis,k2-1);
                    t_k2 = knotseq(Data_Basis,k2);
                    H_k1 = t_k1-t_k1m1;
                    H_k2 = t_k2-t_k2m1;
                    // quadrature
                    if (0<min(H_k1,H_k2)){
                        // elements with no intersection
                        squareint=0;
                        for (q1=0;q1<n_gauss;q1++) {
                            for (q2=0;q2<n_gauss;q2++) {
                                squareint += weights_gauss[q1]*weights_gauss[q2]
                                             *R_til[i-1+p][k1-i][q1]
                                             *R_til[j-1+p][k2-j][q2];
                            }
```

```
                        }
                    }

                    output[i+p-1+(j+p-1)*(N-multb+1+p)] += H_k1*H_k2
                                                          *squareint;
                }
            }

            if (i!=j){
                // W_h2 symmetric
                output[j+p-1+(i+p-1)*(N-multb+1+p)] = output[i+p-1+(j+p-1)
                                                        *(N-multb+1+p)];
            }
        }
    }
    output[0]+=2*output[N-1]+output[N*N-1];

    for(i=2-p;i<=N-p-1;i++){
        output[i+p-1] += output[i+p-1+(N-1)*N];
        output[(i+p-1)*N] += output[N-1+(i+p-1)*N];
    }
}


//-----------------------------------------------------------------------------
// build_W_h
//-----------------------------------------------------------------------------


void build_W(double* output, NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis,
            QuadData* Data_Gauss,QuadData* Data_LogGauss){

    build_W_h1(output, Data_Gamma, wcpoints1_gam, wcpoints2_gam,
            Data_Basis, Data_Gauss, Data_LogGauss);
    build_W_h2(output, Data_Gamma, wcpoints1_gam, wcpoints2_gam,
            Data_Basis, Data_Gauss, Data_LogGauss);
}
```

A.6 FVector.h and FVector.c

These two files are used to compute the right-hand side vector F_h from Section 6.

```
#ifndef _Fvector_h
#define _Fvector_h

#include <math.h>
#include <stdio.h>
#include "Spline.h"
#include "phi_approx.h"

double F_integrand_K(NURBSData* Data_Gamma,double* wcpoints1_gam,
            double* wcpoints2_gam,NURBSData* Data_Basis,NURBSData* Basis_tmp,
            int j,double t,double s,double Jdet,int p_phi_approx,double* coeff);


double F_integral_K_identical(NURBSData* Data_Gamma,double* wcpoints1_gam,
            double* wcpoints2_gam,NURBSData* Data_Basis,NURBSData* Basis_tmp,
            QuadData* Data_Gauss,double* coeff,
            int j,int k,int p_phi_approx);


double F_integral_K_Adjacent(NURBSData* Data_Gamma,double* wcpoints1_gam,
            double* wcpoints2_gam,NURBSData* Data_Basis,NURBSData* Basis_tmp,
            QuadData* Data_Gauss,double* coeff,
            int j,int k1,int k2,int singtype,int p_phi_approx);

```

```
void build_Fvector(double* output,NURBSData* Data_Gamma,double* wcpoints1_gam,
            double* wcpoints2_gam,NURBSData* Data_Basis,
            QuadData* Data_Gauss,QuadData* Data_Gauss_small,
            QuadData* Data_Gauss_phi_approx,
            int with_K,int p_phi_approx, double* coeff);

#endif
```

```
#include "Fvector.h"


//-----------------------------------------------------------------------------
// F_integrand_K
//-----------------------------------------------------------------------------


double F_integrand_K(NURBSData* Data_Gamma,double* wcpoints1_gam,
        double* wcpoints2_gam, NURBSData* Data_Basis, NURBSData* Basis_tmp,
        int j, double t, double s, double Jdet, int p_phi_approx, double* coeff){

    double a=get_NURBSData_a(Data_Basis);
    int n=get_NURBSData_n(Data_Basis);
    double* nodes=get_NURBSData_nodes(Data_Basis);
    double nodesa[n+1];

    nodesa[0]=a;
    int j_tmp=0;

    for (j_tmp=1;j_tmp<=n;++j_tmp){
        nodesa[j_tmp]=nodes[j_tmp-1];
    }

    double sum=0;
    int index=0;

    index=find_Span_nodes(Data_Basis,t);

    double knots_tmp[p_phi_approx+1];
    double nodes_tmp[1];

    double weights_tmp[p_phi_approx+1];

    for (j_tmp=0;j_tmp<p_phi_approx+1;j_tmp++){
        knots_tmp[j_tmp]=nodesa[index+1];
        weights_tmp[j_tmp]=1;
    }
    nodes_tmp[0]=nodesa[index+1];

    NURBSData* Data_Basis_tmp=new_NURBSData(nodesa[index], knots_tmp,
                p_phi_approx+1, weights_tmp, p_phi_approx, nodes_tmp, 1, 0);



    double tmp1[2];
    double tmp2[2];
    double diff_gam[2]; // gamma(t)-gamma(s)
    double phi_til,R_nu1,R_nu2;

    eval_NURBSCurve(tmp1,Data_Gamma,wcpoints1_gam,wcpoints2_gam,t);
    // gamma(t)
    eval_NURBSCurve(tmp2,Data_Gamma,wcpoints1_gam,wcpoints2_gam,s);
    // gamma(s)
    sum=0;
    for (j_tmp=0;j_tmp<=p_phi_approx;++j_tmp){
        sum+=coeff[index*(p_phi_approx+1)+j_tmp]
                            *eval_NURBS(Data_Basis_tmp,j_tmp-p_phi_approx+1,t);
    }

```

```
    phi_til=sum;

    diff_gam[0] = tmp1[0] - tmp2[0];
    diff_gam[1] = tmp1[1] - tmp2[1];

    R_nu1 = eval_NURBS(Data_Basis, j, s);
    R_nu2 = eval_NURBS(Data_Basis, j, s);

    eval_NURBSCurveDeriv(tmp1,Data_Gamma,wcpoints1_gam,wcpoints2_gam,t);
    // gamma'(t)
    eval_NURBSCurveDeriv(tmp2,Data_Gamma,wcpoints1_gam,wcpoints2_gam,s);
    // gamma'(s)

    phi_til *= norm(tmp1);
    R_nu1 *= tmp2[1];
    R_nu2 *= -tmp2[0];

    Data_Basis_tmp=del_NURBSData(Data_Basis_tmp);

    return ((diff_gam[0]*R_nu1 + diff_gam[1]*R_nu2)
                    / (2*M_PI*norm(diff_gam)))*(Jdet/norm(diff_gam)) * phi_til;
}


//-----------------------------------------------------------------------------
// F_integral_K_identical
//-----------------------------------------------------------------------------


double F_integral_K_identical(NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis, NURBSData* Basis_tmp,
            QuadData* Data_Gauss,double* coeff, int j, int k, int p_phi_approx){

    double* nodes_gauss=get_QuadData_nodes(Data_Gauss);
    double* weights_gauss=get_QuadData_weights(Data_Gauss);
    int n_gauss=get_QuadData_n(Data_Gauss);
    int q1,q2;

    double t_km1=knotseq(Data_Basis,k-1); // t_{k-1}
    double t_k=knotseq(Data_Basis,k); // t_{k}
    double H_k=t_k-t_km1; // H_k
    double squareint=0; // integral over square
    double intpoint1, intpoint2; // first and second integration point
    double Jdet; // Jacobi determinant for Duffy transformation

    for (q1=0;q1<n_gauss;q1++) {
        for (q2=0;q2<n_gauss;q2++) {
            intpoint1=t_km1+H_k*nodes_gauss[q1];
            intpoint2=t_km1+H_k*nodes_gauss[q1]*(1-nodes_gauss[q2]);
            Jdet=nodes_gauss[q1];

            squareint+=weights_gauss[q1]*weights_gauss[q2]
                        *F_integrand_K(Data_Gamma, wcpoints1_gam, wcpoints2_gam,
                                    Data_Basis, Basis_tmp, j, intpoint1,
                                    intpoint2, Jdet, p_phi_approx,coeff);

            intpoint1=t_km1+H_k*(1-nodes_gauss[q1]);
            intpoint2=t_km1+H_k*(1-nodes_gauss[q1]+nodes_gauss[q1]*nodes_gauss[q2]);
            Jdet=nodes_gauss[q1];

            squareint+=weights_gauss[q1]*weights_gauss[q2]
                        *F_integrand_K(Data_Gamma, wcpoints1_gam, wcpoints2_gam,
                                    Data_Basis, Basis_tmp, j, intpoint1,
                                    intpoint2, Jdet, p_phi_approx,coeff);
        }
    }
    return squareint;
}


```

```
//-----------------------------------------------------------------------------
// F_integral_K_Adjacent
//-----------------------------------------------------------------------------


double F_integral_K_Adjacent(NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis, NURBSData* Basis_tmp,
            QuadData* Data_Gauss, double* coeff, int j,int k1,
            int k2, int singtype, int p_phi_approx){

    double* nodes_gauss=get_QuadData_nodes(Data_Gauss);
    double* weights_gauss=get_QuadData_weights(Data_Gauss);
    int n_gauss=get_QuadData_n(Data_Gauss);
    int q1,q2;
    double t_k1m1=knotseq(Data_Basis,k1-1); // t_{l_1-1}
    double t_k1=knotseq(Data_Basis,k1); // t_{l_1}
    double H_k1=t_k1-t_k1m1; // H_{l_1}
    double t_k2m1=knotseq(Data_Basis,k2-1); // t_{l_2-1}
    double t_k2=knotseq(Data_Basis,k2); // t_{l_2}
    double H_k2=t_k2-t_k2m1; // H_{l_1}
    double squareint=0; // integral over square
    double intpoint1, intpoint2; // first and second integration point
    double Jdet; // Jacobi determinant of Duffy transformation

    if (singtype==0){
        for (q1=0;q1<n_gauss;q1++) {
            for (q2=0;q2<n_gauss;q2++) {
                // first double integral
                intpoint1=t_k1m1+H_k1*nodes_gauss[q1];
                intpoint2=t_k2m1+H_k2*(1-nodes_gauss[q1]*nodes_gauss[q2]);
                Jdet=nodes_gauss[q1];
                squareint+=weights_gauss[q1]*weights_gauss[q2]
                            *F_integrand_K(Data_Gamma, wcpoints1_gam, wcpoints2_gam,
                                   Data_Basis, Basis_tmp, j, intpoint1,
                                   intpoint2, Jdet, p_phi_approx,coeff);

                // second double integral
                intpoint1=t_k1m1+H_k1*nodes_gauss[q1]*nodes_gauss[q2];
                intpoint2=t_k2m1+H_k2*(1-nodes_gauss[q2]);
                Jdet=nodes_gauss[q2];
                squareint+=weights_gauss[q1]*weights_gauss[q2]
                            *F_integrand_K(Data_Gamma, wcpoints1_gam,
                                    wcpoints2_gam, Data_Basis, Basis_tmp,
                                    j, intpoint1, intpoint2, Jdet,
                                    p_phi_approx,coeff);
            }
        }
    } else {
        for (q1=0;q1<n_gauss;q1++) {
            for (q2=0;q2<n_gauss;q2++) {
                // first double integral
                intpoint1=t_k1m1+H_k1*(1-nodes_gauss[q1]*nodes_gauss[q2]);
                intpoint2=t_k2m1+H_k2*nodes_gauss[q2];
                Jdet=nodes_gauss[q2];
                squareint+=weights_gauss[q1]*weights_gauss[q2]
                            *F_integrand_K(Data_Gamma, wcpoints1_gam, wcpoints2_gam,
                                    Data_Basis, Basis_tmp, j, intpoint1,
                                    intpoint2, Jdet, p_phi_approx, coeff);

                // second double integral
                intpoint1=t_k1m1+H_k1*(1-nodes_gauss[q1]);
                intpoint2=t_k2m1+H_k2*nodes_gauss[q1]*nodes_gauss[q2];
                Jdet=nodes_gauss[q1];
                squareint+=weights_gauss[q1]*weights_gauss[q2]
                            *F_integrand_K(Data_Gamma, wcpoints1_gam, wcpoints2_gam,
                                    Data_Basis, Basis_tmp, j, intpoint1,
                                    intpoint2, Jdet, p_phi_approx, coeff);
            }
        }
    }
```

```
    return squareint;
}


//-----------------------------------------------------------------------------
// build_Fvector
//-----------------------------------------------------------------------------


void build_Fvector(double* output, NURBSData* Data_Gamma, double* wcpoints1_gam,
            double* wcpoints2_gam, NURBSData* Data_Basis, QuadData* Data_Gauss,
            QuadData* Data_Gauss_small, QuadData* Data_Gauss_phi_approx,
            int with_K, int p_phi_approx, double* coeff){

    double* nodes_gauss=get_QuadData_nodes(Data_Gauss);
    double* weights_gauss=get_QuadData_weights(Data_Gauss);
    int n_gauss=get_QuadData_n(Data_Gauss);

    double* knots_gam=get_NURBSData_knots(Data_Gamma);
    int N_gam=get_NURBSData_N(Data_Gamma);

    int N=get_NURBSData_N(Data_Basis);
    int p=get_NURBSData_p(Data_Basis);

    double tmp[2];
    int j,k1,k2,q1,q2;

    int multb=p+1; // #b

    double R_til[N-multb+1+p][p+1][n_gauss];
    // R_til[i-1+p][k1-i][q1]=\tilde{R}_{i,k1}(nodes_gauss[q1])

    double gamma1[N][n_gauss];
    // gamma1[k1-1][q1] is first component of
    // gamma(t_{k1-1}+H_k1*nodes_gauss[q1])

    double gamma2[N][n_gauss];
    // gamma2[k1-1][q1] is second component of
    // gamma(t_{k1-1}+H_k1*nodes_gauss[q1])

    double phi_til[N][n_gauss];
    //phi[k1-1][q1] is \phi(\gamma(t_{k-1}+
    //H_k*nodes_gauss[q1]))*|\gamma'(t_{k-1}+H_k*nodes_gauss[q1])|

    double R_nu1[N][p+1][n_gauss];
    // R_nu1[j+p-1][k2-j][q2] is first component of
    // {{0,1},{-1,0}}*gamma'(t_{k2-1}+H_k2*nodes_gauss[q2])
    // *R_j(t_{k2-1}+H_k2*nodes_gauss[q2])

    double R_nu2[N][p+1][n_gauss];
    // R_nu2[j+p-1][k2-j][q2] is second component of
    // {{0,1},{-1,0}}*gamma'(t_{k2-1}+H_k2*nodes_gauss[q2])
    // *R_j(t_{k2-1}+H_k2*nodes_gauss[q2])

    double squareint; // integral over square

    //double Jdet; // Jacobi determinant of Duffy transformation

    //double diff_gam[2];

    double t_k1m1,t_k1,H_k1,t_k2m1,t_k2,H_k2;
    // t_{k1-1},t_k1,H_k1,t_{k2-1},t_k2,H_k2

    double intpoint1, intpoint2; // first and second integration point
    int index; // help index

    int i=0;

    double knots_tmp[p_phi_approx+1];
    for (i=0;i<p_phi_approx+1;++i){
```

```
        knots_tmp[i]=1;
    }
    double weights_tmp[p_phi_approx+1];
    for (i=0;i<p_phi_approx+1;++i){
        weights_tmp[i]=1;
    }
    double nodes_tmp[1];
    nodes_tmp[0]=1;
    int is_rational_tmp=0;
    NURBSData* Basis_tmp=new_NURBSData(0, knots_tmp, p_phi_approx+1,
                                weights_tmp, p_phi_approx, nodes_tmp, 1, 0);

    int n=get_NURBSData_n(Data_Basis);
    double* nodes=get_NURBSData_nodes(Data_Basis);
    double sum=0;
    int j_tmp=0;
    int index2=0;

    // calculation of R_til, R_nu1, R_nu2
    // R_i
    for (j=1-p;j<=(N-multb+1);j++) {
        // elements with nonemty intersection with support of R_i
        for (k2=max(j,1);k2<=min(j+p,N);k2++) {
            // quadrature points
            for (q2=0;q2<n_gauss;q2++) {
                t_k2m1=knotseq(Data_Basis,k2-1);
                t_k2=knotseq(Data_Basis,k2);
                H_k2=t_k2-t_k2m1;
                intpoint2=t_k2m1+H_k2*nodes_gauss[q2];

                R_nu1[j-1+p][k2-j][q2] = eval_NURBS(Data_Basis,j,intpoint2);
                R_nu2[j-1+p][k2-j][q2] = eval_NURBS(Data_Basis,j,intpoint2);

                eval_NURBSCurveDeriv(tmp, Data_Gamma, wcpoints1_gam,
                                wcpoints2_gam, intpoint2);

                R_nu1[j-1+p][k2-j][q2] *=  tmp[1];
                R_nu2[j-1+p][k2-j][q2] *= -tmp[0];
                R_til[j-1+p][k2-j][q2]  =  eval_NURBS(Data_Basis,j,intpoint2)
                                        *norm(tmp);
            }
        }
    }

    if (with_K==1){

        // calculation of gamma1, gamma2, phi_til
        for (k1=1;k1<=N;k1++){

            t_k1m1=knotseq(Data_Basis,k1-1);
            t_k1=knotseq(Data_Basis,k1);
            H_k1=t_k1-t_k1m1;

            if(0<H_k1){

                index2=find_Span_nodes(Data_Basis,t_k1m1);

                for (j_tmp=0; j_tmp<p_phi_approx+1;++j_tmp){
                    knots_tmp[j_tmp]=t_k1;
                }
                nodes_tmp[0]=t_k1;
                Basis_tmp=del_NURBSData(Basis_tmp);
                Basis_tmp=new_NURBSData(t_k1m1, knots_tmp, p_phi_approx+1,
                                    weights_tmp, p_phi_approx, nodes_tmp, 1, 0);

                for (q1=0;q1<n_gauss;q1++){



                    intpoint1=t_k1m1+H_k1*nodes_gauss[q1];
```

```
                    eval_NURBSCurve(tmp,Data_Gamma,wcpoints1_gam,
                                    wcpoints2_gam,intpoint1);
                    gamma1[k1-1][q1]=tmp[0];
                    gamma2[k1-1][q1]=tmp[1];

                    sum=0;
                    for (j_tmp=0;j_tmp<=p_phi_approx;++j_tmp){
                        sum+=coeff[index2*(p_phi_approx+1)+j_tmp]
                                    *eval_NURBS(Basis_tmp,j_tmp-p_phi_approx+1,
                                                intpoint1);
                    }
                    phi_til[k1-1][q1]=sum;
                    eval_NURBSCurveDeriv(tmp,Data_Gamma, wcpoints1_gam,
                                    wcpoints2_gam,intpoint1);
                    phi_til[k1-1][q1] *= norm(tmp);
                }
            }
        }


        // calculation of <phi,K\hat{R}_j>_{L_2(Gamma)}
        // R_j
        for (j=1-p;j<=(N-multb+1);j++) {
            output[j+p-1]=0;
            // elements with nonemty intersection with support of R_j
            for (k1=1;k1<=N;k1++) {
                // all elements
                for (k2=max(j,1);k2<=min(j+p,N);k2++) {
                    t_k1m1=knotseq(Data_Basis,k1-1);
                    t_k1=knotseq(Data_Basis,k1);
                    t_k2m1=knotseq(Data_Basis,k2-1);
                    t_k2=knotseq(Data_Basis,k2);
                    H_k1=t_k1-t_k1m1;
                    H_k2=t_k2-t_k2m1;
                    // quadrature
                    if (0<min(H_k1,H_k2)){
                        squareint=0;
                        // elements with no intersection
                        if ((!nearly_equal(t_k1m1,t_k2m1))
                            && (!nearly_equal(t_k1m1,t_k2))
                            && (!nearly_equal(t_k1,t_k2m1))
                            && (!nearly_equal(t_k1,t_k2))
                            && ((k1!=(N-multb+1)) || (k2!=1))
                            && ((k2!=(N-multb+1)) || (k1!=1))) {
                            for (q1=0;q1<n_gauss;q1++) {
                                for (q2=0;q2<n_gauss;q2++) {
                                    tmp[0] = gamma1[k1-1][q1] - gamma1[k2-1][q2];
                                    tmp[1] = gamma2[k1-1][q1] - gamma2[k2-1][q2];
                                    squareint += (weights_gauss[q1]*weights_gauss[q2]
                                                *(tmp[0]*R_nu1[j-1+p][k2-j][q2]
                                                +tmp[1]*R_nu2[j-1+p][k2-j][q2])
                                                /norm(tmp))/norm(tmp)
                                                *phi_til[k1-1][q1]/(2*M_PI);
                                }
                            }
                        }
                        // elements with intersection
                        else {
                            // identical elements
                            if (k1==k2){
                                squareint=F_integral_K_identical(Data_Gamma,
                                                wcpoints1_gam, wcpoints2_gam,
                                                Data_Basis, Basis_tmp,
                                                Data_Gauss_small, coeff,
                                                j, k1, p_phi_approx);
                            }
                            // elements with point intersection
                            else{
                                // singularity at s=0,t=1
                                if (nearly_equal(t_k1m1,t_k2)
```

```
                                        || ((k2==(N-multb+1)) && (k1==1))){
                                        index=1;
                                        while(!nearly_equal(t_k2,knots_gam[index-1])){
                                            index=index+1;
                                            if (index==(N_gam+1)){break;}
                                        }
                                        // t_k2 no knot of Gamma
                                        if (index==(N_gam+1)){
                                            squareint=F_integral_K_Adjacent(Data_Gamma,
                                                        wcpoints1_gam, wcpoints2_gam,
                                                        Data_Basis, Basis_tmp,
                                                        Data_Gauss_small, coeff,
                                                        j, k1, k2, 0, p_phi_approx);
                                        }
                                        // t_k2 knot of Gamma
                                        else {
                                            squareint=F_integral_K_Adjacent(Data_Gamma,
                                                        wcpoints1_gam, wcpoints2_gam,
                                                        Data_Basis, Basis_tmp,
                                                        Data_Gauss, coeff, j,
                                                        k1, k2, 0, p_phi_approx);
                                        }
                                    }
                                    // singularity at s=1,t=0
                                    else{
                                        index=1;
                                        while(!nearly_equal(t_k1,knots_gam[index-1])){
                                            index=index+1;
                                            if (index==(N_gam+1)){break;}
                                        }
                                        // t_k1 no knot of Gamma
                                        if (index==(N_gam+1)){
                                            squareint=F_integral_K_Adjacent(Data_Gamma,
                                                        wcpoints1_gam, wcpoints2_gam,
                                                        Data_Basis, Basis_tmp,
                                                        Data_Gauss_small, coeff,
                                                        j, k1, k2, 1, p_phi_approx);
                                        }
                                        // t_k1 knot of Gamma
                                        else {
                                            squareint=F_integral_K_Adjacent(Data_Gamma,
                                                        wcpoints1_gam,wcpoints2_gam,
                                                        Data_Basis, Basis_tmp,
                                                        Data_Gauss, coeff, j,
                                                        k1, k2, 1, p_phi_approx);
                                        }
                                    }
                                }
                            }
                            output[j+p-1] += H_k1 * H_k2 * squareint;
                        }
                    }
                }
                output[j+p-1] = -output[j+p-1];
            }
        }


        // calculation of <phi,R_hat_j/2-K R_hat_j>_{L_2(Gamma)}
        // R_j
        for (j=1-p;j<=(N-multb+1);j++) {
            if (with_K==0){
                output[j+p-1]=0;
            }
            // elements with nonemty intersection with support of R_j
            for (k1=max(j,1);k1<=min(j+p,N);k1++) {
                t_k1m1=knotseq(Data_Basis,k1-1); // t_{l1-1}
                t_k1=knotseq(Data_Basis,k1); // t_l1
                H_k1=t_k1-t_k1m1;
                // quadrature
```

```
            if (0<H_k1){
                index2=find_Span_nodes(Data_Basis,t_k1m1);

                for (j_tmp=0; j_tmp<p_phi_approx+1;++j_tmp){
                    knots_tmp[j_tmp]=t_k1;
                }
                nodes_tmp[0]=t_k1;
                Basis_tmp=del_NURBSData(Basis_tmp);
                Basis_tmp=new_NURBSData(t_k1m1,knots_tmp,p_phi_approx
                                +1,weights_tmp,p_phi_approx,nodes_tmp,1,0);

                for (q1=0;q1<n_gauss;q1++) {
                    eval_NURBSCurve(tmp,Data_Gamma,wcpoints1_gam,wcpoints2_gam,
                                    t_k1m1+H_k1*nodes_gauss[q1]);


                    sum=0;
                    for (j_tmp=0;j_tmp<=p_phi_approx;++j_tmp){
                        sum+=coeff[index2*(p_phi_approx+1)+j_tmp]
                                *eval_NURBS(Basis_tmp, j_tmp-p_phi_approx+1,
                                t_k1m1+H_k1*nodes_gauss[q1]);
                    }
                    output[j+p-1] += H_k1*weights_gauss[q1]*sum
                                    /2*R_til[j-1+p][k1-j][q1];
                }
            }
        }
    }
    output[0] += output[N-1];
    Basis_tmp=del_NURBSData(Basis_tmp);
}
```

A.7 res_RHS.h and res_RHS.c

These two files are used to evaluate the right-hand side $(1/2 - K')\phi_h(x)$ for $x \in \Gamma \setminus \{z_1, \ldots, z_n\}$.

```
#ifndef _eval_RHS_h
#define _eval_RHS_h

#include <math.h>
#include <stdio.h>
#include "Spline.h"
#include "phi_approx.h"


double Integrand_K_pr(NURBSData* Data_Gamma,double* wcpoints1_gam,
            double* wcpoints2_gam,double s,double t, NURBSData* Data_Basis,
            int p_phi_approx, double* coeff);

double eval_RHS(NURBSData* Data_Gamma,double* wcpoints1_gam,
            double* wcpoints2_gam,NURBSData* Data_Basis,
            QuadData* Data_Gauss,double s,int with_K, int p_phi_approx, double* coeff);


#endif
```

```
#include "res_RHS.h"


//-----------------------------------------------------------------------------
// Integrand_K_pr
//-----------------------------------------------------------------------------


double Integrand_K_pr(NURBSData* Data_Gamma,double* wcpoints1_gam,
```

```
        double* wcpoints2_gam, double s, double t, NURBSData* Data_Basis,
        int p_phi_approx, double* coeff){

    double a=get_NURBSData_a(Data_Basis);
    int n=get_NURBSData_n(Data_Basis);
    double* nodes=get_NURBSData_nodes(Data_Basis);
    double nodesa[n+1];
    nodesa[0]=a;
    int j_tmp=0;

    for (j_tmp=1;j_tmp<=n;++j_tmp){
        nodesa[j_tmp]=nodes[j_tmp-1];
    }
    double sum=0;
    int index=find_Span_nodes(Data_Basis,t);
    double knots_tmp[p_phi_approx+1];
    double nodes_tmp[1];
    double weights_tmp[p_phi_approx+1];

    for (j_tmp=0;j_tmp<p_phi_approx+1;j_tmp++){
        knots_tmp[j_tmp]=nodesa[index+1];
        weights_tmp[j_tmp]=1;
    }
    nodes_tmp[0]=nodesa[index+1];
    NURBSData* Data_Basis_tmp=new_NURBSData(nodesa[index],knots_tmp,
                                p_phi_approx+1,weights_tmp,p_phi_approx,
                                nodes_tmp,1,0);
    sum=0;
    for (j_tmp=0;j_tmp<=p_phi_approx;++j_tmp){
        sum+=coeff[index*(p_phi_approx+1)+j_tmp]
                        *eval_NURBS(Data_Basis_tmp,j_tmp-p_phi_approx+1,t);
    }

    double phi_nu[2];
    double s_hat[2];
    double t_hat[2];
    double tmp[2];
    double tmp2[2];
    double tmp3[2];
    eval_NURBSCurve(s_hat,Data_Gamma,wcpoints1_gam,wcpoints2_gam,s);
    eval_NURBSCurve(t_hat,Data_Gamma,wcpoints1_gam,wcpoints2_gam,t);

    eval_NURBSCurveDeriv(tmp,Data_Gamma,wcpoints1_gam,wcpoints2_gam,s);
    // \gamma'(s)
    eval_NURBSCurveDeriv(tmp2,Data_Gamma,wcpoints1_gam,wcpoints2_gam,t);
    // \gamma'(t)
    phi_nu[0] = sum * tmp[1];
    phi_nu[1] = -sum * tmp[0];
    tmp3[0] = s_hat[0] - t_hat[0]; // \gamma(s)-\gamma(t)
    tmp3[1] = s_hat[1] - t_hat[1];

    Data_Basis_tmp=del_NURBSData(Data_Basis_tmp);

    return -(tmp3[0]*phi_nu[0] + tmp3[1]*phi_nu[1])*norm(tmp2)/norm(tmp)
                                        / (2*M_PI*pow(norm(tmp3),2));
}


//-----------------------------------------------------------------------------
// eval_RHS
//-----------------------------------------------------------------------------


double eval_RHS(NURBSData* Data_Gamma,double* wcpoints1_gam,
        double* wcpoints2_gam,NURBSData* Data_Basis,
        QuadData* Data_Gauss,double s,int with_K,
        int p_phi_approx, double* coeff){

    double a=get_NURBSData_a(Data_Gamma);
    double* nodes=get_NURBSData_nodes(Data_Basis);
```

```
    int n=get_NURBSData_n(Data_Basis);
    double* nodes_gauss=get_QuadData_nodes(Data_Gauss);
    double* weights_gauss=get_QuadData_weights(Data_Gauss);
    int n_gauss=get_QuadData_n(Data_Gauss);
    int j,q;
    double output=0;
    double x_ch_jm1,x_ch_j,h_j;

    double intpoint;
    double s_hat[2];
    eval_NURBSCurve(s_hat,Data_Gamma,wcpoints1_gam,wcpoints2_gam,s);


    double nodesa[n+1];
    nodesa[0]=a;
    int j_tmp=0;
    for (j_tmp=1;j_tmp<=n;++j_tmp){
        nodesa[j_tmp]=nodes[j_tmp-1];
    }
    double sum=0;
    int index=find_Span_nodes(Data_Basis,s);
    double knots_tmp[p_phi_approx+1];
    double nodes_tmp[1];
    double weights_tmp[p_phi_approx+1];

    for (j_tmp=0;j_tmp<p_phi_approx+1;j_tmp++){
        knots_tmp[j_tmp]=nodesa[index+1];
        weights_tmp[j_tmp]=1;
    }
    nodes_tmp[0]=nodesa[index+1];
    NURBSData* Data_Basis_tmp=new_NURBSData(nodesa[index],knots_tmp,
                                            p_phi_approx+1,weights_tmp,
                                            p_phi_approx,nodes_tmp,1,0);
    sum=0;
    for (j_tmp=0;j_tmp<=p_phi_approx;++j_tmp){
        sum+=coeff[index*(p_phi_approx+1)+j_tmp]
                        *eval_NURBS(Data_Basis_tmp,j_tmp-p_phi_approx+1,s);
    }




    if (with_K==1){
        // calculation of K'phi(\gamma(s))
        for (j=1;j<=n;j=j+1){
            if (j==1) {x_ch_jm1=a;} else {x_ch_jm1=nodes[j-2];}
            x_ch_j=nodes[j-1];
            h_j=x_ch_j-x_ch_jm1;

           if ((!nearly_leq(x_ch_jm1,s)) || (!nearly_leq(s,x_ch_j))){
                for (q=0;q<n_gauss;q=q+1){
                    intpoint = x_ch_jm1 + h_j * nodes_gauss[q];
                    output += h_j * weights_gauss[q] * Integrand_K_pr(Data_Gamma,
                                                    wcpoints1_gam,wcpoints2_gam,s,
                                                    intpoint,Data_Basis,
                                                    p_phi_approx,coeff);
                }
            }else{
                // first integral
                if (!nearly_equal(s,x_ch_jm1)){
                    for (q=0;q<n_gauss;q=q+1){
                        intpoint = s - (s - x_ch_jm1) * nodes_gauss[q];
                        output += (s - x_ch_jm1) * weights_gauss[q]
                                    * Integrand_K_pr(Data_Gamma,wcpoints1_gam,
                                            wcpoints2_gam,s,intpoint,
                                            Data_Basis,p_phi_approx,coeff);
                    }
                }
                //second integral
                if (!nearly_equal(s,x_ch_j)){
```

```
                    for (q=0;q<n_gauss;q=q+1){
                        intpoint = s + (x_ch_j - s) * nodes_gauss[q];
                        output += (x_ch_j - s) * weights_gauss[q]
                                    * Integrand_K_pr(Data_Gamma,wcpoints1_gam,
                                            wcpoints2_gam,s,intpoint,
                                            Data_Basis,p_phi_approx,coeff);
                    }
                }
            }
        }
    }

    Data_Basis_tmp=del_NURBSData(Data_Basis_tmp);
    return sum*0.5-output;
}
```

A.8 mark.m

This file uses Doerfler marking to mark nodes resp. elements, cf. [Gan14, Appendix B].

```
function [marked_nodes,marked_elements]=mark(indicators,theta,a,knots,p,is_open_gam,href)
% input:
%   indicators...column vector of error indicators
%   theta...constant for Doerfler marking
%   a...left interval boundary of [a,b]
%   knots...column vector of knots in (a,b] with mult<=p+1
%   p...polynomial degree
%   is_open_gam...1 if Gamma is open, 0 else
%   href...1 for pure h-refinement, 0 for new proposed algorithm
% output:
%   marked_nodes...column vector of indices of (finally) marked nodes
%   marked_elements...column vector of indices of (finally) marked elements


% Doerfler marking for nodes
if theta<1
    [indicators_tmp,tmp] = sort(indicators,'descend');
    sum_indicatorssq = cumsum(indicators_tmp.^2);
    index = find(sum_indicatorssq >= (sum_indicatorssq(end)*theta),1);
    marked_nodes = sort(tmp(1:index))-is_open_gam;
else
    marked_nodes=(1:length(indicators))'-is_open_gam;
end

% my modulo
if is_open_gam==0
    my_mod=@(x,y) mod(x,y)+y*(mod(x,y)==0);
else
    my_mod=@(x,y) min(x,y);
end

% resulting marked nodes and elements
nodes0=[a;unique(knots)];
n=length(nodes0)-1;
marked_elements=[];
del_nodes=[];
% h-refinement of elements whose nodes are marked
for i=1:(length(marked_nodes)-is_open_gam)
    if marked_nodes(my_mod(i+1,length(marked_nodes))) == my_mod(marked_nodes(i)+1,n)
        marked_elements=[marked_elements;my_mod(marked_nodes(i)+1,n)];
        del_nodes=[del_nodes;marked_nodes(i);my_mod(marked_nodes(i)+1,n)];
    end
end
marked_nodes=setdiff(marked_nodes,del_nodes);
del_nodes=[];
% h-refinement for elements with marked intersection node and (full multiplicity
% or href==1)
```

```
for i=1:length(marked_nodes)
    if (sum(knots==nodes0(marked_nodes(i)+1))==(p)...
                || sum(knots==nodes0(marked_nodes(i)+1))==(p+1)...
                || marked_nodes(i)==0 || href==1)
        marked_elements=...
            [marked_elements;max(1,marked_nodes(i));my_mod(marked_nodes(i)+1,n)];
        del_nodes=[del_nodes;marked_nodes(i)];
    end
end
marked_nodes=setdiff(marked_nodes,del_nodes);
marked_elements=unique(marked_elements);
```

A.9 insert_cpoint.m, increase_Mult and refine_BoundaryMesh

The file **insert_cpoint.m** computes the new control points of a NURBS curve, if one knot is added, such that the curve does not change. The file **increase_Mult** computes the new knot resp. weight vector and the new control points, if the multiplicity of a given set of nodes is increased by one. The file **refine_BoundaryMesh** refines a given boundary mesh.

```
function [ cpoints_new ] = insert_cpoint(a,knots,weights,cpoints,p,knot_added)

knots_a=[a*ones(p+1,1);knots];
weights=[weights(end-p+1:end);weights(1:end-p)];
cpoints=[cpoints(:,end-p+1:end),cpoints(:,1:end-p)];

ind=find(knots_a<=knot_added,1,'last');

cpoints_added=[];
alpha=[];
P=[];
Q=[];

for k=0:p
    P=[[cpoints(:,ind-k).*weights(ind-k);weights(ind-k)],P];
end

for k=0:p-1
    alpha=[(knot_added-knots_a(ind-k))/(knots_a(ind-k+p)-knots_a(ind-k)),alpha];
end

for k=0:p-1
    Q=[P(:,p-k).*(1-alpha(p-k))+P(:,p-k+1).*alpha(p-k),Q];
end

for k=1:p
    cpoints_added=[cpoints_added,Q(1:2,k)./Q(3,k)];
end

cpoints_new=[cpoints(:,1:(ind-p)),cpoints_added,cpoints(:,ind:end)];

cpoints_new=[cpoints_new(:,p+1:end),cpoints_new(:,1:p)];
end
```

```
function [knots_fine,weights_fine,cpoints_fine]=increase_Mult_cpoints(a,...
    knots,weights,cpoints,p,marked)
% input:
%   a...left interval boundary of [a,b]
%   knots...column vector of knots in (a,b] with mult<=p+1 and mult=p+1 of b
%   weights...column vector of positive weights corresponding to knots
%   marked...column vector with indices of marked nodes
% output:
%   knots_fine...column vector of refined knots (via knot insertion),
%   weights_fine...column vector of new weights corresponding to knots_fine
% comments:
% we assume that
```

```
%   -) number of knots N>=p+1
%   -) number of nodes n>=4

b=knots(end);

% start data
weights_current=weights;
knots_current=knots;

weights_current_tmp=weights;
cpoints_current=cpoints;

N_current=length(knots_current);
nodes=unique(knots);
marked_current=marked; % marked nodes of current mesh

% loop, in each step one insertion
while ~isempty(marked_current)
    incr=marked_current(1); % node whose mult is increased
    tmp=find(knots_current==nodes(incr));
    l=tmp(1);
    mult=length(tmp);
    % marked node = t_l
    t_l=knotseq(a,knots_current,l);

    % insert knot
    t_pr=t_l; % t'
    weights_tmp=zeros(N_current+1,1); % w'''
    weights_tmp(1:(l-p+mult))=weights_current(1:(l-p+mult));
    for i=max(1,(l-p+mult+1)):l
        t_im1=knotseq(a,knots_current,i-1);
        t_im1pp=knotseq(a,knots_current,i-1+p);
        if t_im1<t_im1pp
            beta=(t_pr-t_im1)/(t_im1pp-t_im1);
        else
            beta=0;
        end
        weights_tmp(i)=(1-beta)*weights_current(i-1+N_current*(i==1))...
            +beta*weights_current(i);
    end
    weights_tmp((l+1):(N_current+1))=weights_current(l:N_current);
    weights_current(1:min(N_current+1,l+N_current+1+mult-p))=...
        weights_tmp(1:min(N_current+1,l+N_current+1+mult-p));
    for i=(l+N_current+mult+2-p):(N_current+1)
        t_im2=knotseq(a,knots_current,i-2);
        t_im2pp=knotseq(a,knots_current,i-2+p);
        if t_im2<t_im2pp
            beta=(t_pr+b-a-t_im2)/(t_im2pp-t_im2);
        else
            beta=0;
        end
        weights_current(i)=(1-beta)*weights_tmp(i-1)...
            +beta*weights_tmp(i);
    end

    % update data
    cpoints_current=insert_cpoint(a,knots_current,weights_current_tmp,...
                                  cpoints_current,p,t_pr);
    weights_current_tmp=weights_current;

    knots_current=[knots_current(1:(l-1));t_pr;knots_current(l:end)];
    N_current=N_current+1;
    marked_current=marked_current(2:end);
end

cpoints_fine=cpoints_current;

knots_fine=knots_current;
weights_fine=weights_current;
end
```

```
function output=knotseq(a,knots,i)
% returns t_i
b=knots(end);
N=length(knots);
if (mod(i,N)~=0)
    output=knots(mod(i,N))+(b-a)*floor(i/N);
else
    output=a+(b-a)*floor(i/N);
end
end
```

```
function [knots_fine,knots_added,weights_fine,cpoints_fine]=refine_BoundaryMesh_cpoints(...
    a,knots,weights,cpoints,p,marked,kappa_max,is_open_gam)
% input:
%   a...left interval boundary of [a,b]
%   knots...column vector of knots in (a,b] with mult<=p+1
%       where kappa(\check{\mathcal{T}})<=kappa_max
%   weights...column vector of positive weights corresponding to knots
%   p...polynomial degree
%   marked...column vector with indices of marked elements
%   kappa_max...maximal allowed mesh constant
%   is_open_gam...1 if Gamma is open, 0 else
% output:
%   knots_fine...column vector of refined knots (via knot insertion),
%       where kappa(\check{\mathcal{T}}_fine)<=kappa_max, at least all
%       marked elements are
%       refined, no element is refined more than one time
%   knots_added...column vector of all added knots
%   weights_fine...column vector of new weights corresponding to knots_fine
% comments:
% we assume that
%   -) number of knots N>=p+1
%   -) number of nodes n>=4

b=knots(end);

% start data
weights_current=weights;
knots_current=knots;

weights_current_tmp=weights;
cpoints_current=cpoints;

knots_added=[];
N_current=length(knots_current);
nodes_current=unique(knots_current);
n_current=length(nodes_current);
nodes0_current=[a;nodes_current];
marked_current=marked; % marked elements of current mesh

% loop, in each step one refinement
while ~isempty(marked_current)
    refine=marked_current(1); % element to be refined
    l=find(knots_current==nodes_current(refine),1);
    % marked element = [t_{l-1},t_l]
    t_lm1=knotseq(a,knots_current,l-1);
    t_l=knotseq(a,knots_current,l);

    % length of first marked element and its neighbours
    if (refine~=1)
        h_ch_left=nodes0_current(refine)-nodes0_current(refine-1);
    else
        if is_open_gam==0
            h_ch_left=b-nodes0_current(n_current);
        end
    end
    h_ch=nodes0_current(refine+1)-nodes0_current(refine);
```

```
    if (refine~=n_current)
        h_ch_right=nodes0_current(refine+2)-nodes0_current(refine+1);
    else
        if is_open_gam==0
            h_ch_right=nodes0_current(2)-a;
        end
    end
    if is_open_gam==1
        if refine==1
            h_ch_left=h_ch;
        elseif refine==n_current
            h_ch_right=h_ch;
        end
    end
    % mark neighbours of first marked element if necessary
    % to guarantee kappa<=kappa_max (+1e-3)
    if max((h_ch/2)/h_ch_left,h_ch_left/(h_ch/2))>(kappa_max+1e-3)
        marked_current=unique([refine-1+n_current*(refine==1);...
            marked_current]);
    end
    if max((h_ch/2)/h_ch_right,h_ch_right/(h_ch/2))>(kappa_max+1e-3)
        marked_current=unique([marked_current;...
            refine+1-n_current*(refine==n_current)]);
    end

    % refine element
    t_pr=(t_lm1+t_l)/2; % t'
    weights_tmp=zeros(N_current+1,1); % w'''
    weights_tmp(1:(l-p))=weights_current(1:(l-p));
    for i=max(1,(l+1-p)):l
        t_im1=knotseq(a,knots_current,i-1);
        t_im1pp=knotseq(a,knots_current,i-1+p);
        if t_im1<t_im1pp
            beta=(t_pr-t_im1)/(t_im1pp-t_im1);
        else
            beta=0;
        end
        weights_tmp(i)=(1-beta)*weights_current(i-1+N_current*(i==1))...
            +beta*weights_current(i);
    end
    weights_tmp((l+1):(N_current+1))=weights_current(l:N_current);
    weights_current(1:min(N_current+1,l+N_current+1-p))=...
        weights_tmp(1:min(N_current+1,l+N_current+1-p));
    for i=(l+N_current+2-p):(N_current+1)
        t_im2=knotseq(a,knots_current,i-2);
        t_im2pp=knotseq(a,knots_current,i-2+p);
        if t_im2<t_im2pp
            beta=(t_pr+b-a-t_im2)/(t_im2pp-t_im2);
        else
            beta=0;
        end
        weights_current(i)=(1-beta)*weights_tmp(i-1)...
            +beta*weights_tmp(i);
    end

    % update data
    cpoints_current=insert_cpoint(a,knots_current,weights_current_tmp,...
                                          cpoints_current,p,t_pr);
    weights_current_tmp=weights_current;

    knots_current=[knots_current(1:(l-1));t_pr;knots_current(l:end)];
    knots_added=[knots_added;t_pr];


    N_current=N_current+1;
    nodes_current=unique(knots_current);
    n_current=length(nodes_current);
    nodes0_current=[a;nodes_current];
    marked_current=[marked_current(marked_current<refine);...
        1+marked_current(marked_current>refine)];
```

```
end

cpoints_fine=cpoints_current;

knots_fine=knots_current;
knots_added=sort(knots_added);
weights_fine=weights_current;
end

function output=knotseq(a,knots,i)
% returns t_i
b=knots(end);
N=length(knots);
if (mod(i,N)~=0)
    output=knots(mod(i,N))+(b-a)*floor(i/N);
else
    output=a+(b-a)*floor(i/N);
end
end
```

A.10 MEX file

The following example of a MEX file can be used to compute the Matrix W from Section 6.

```
#include "mex.h"
#include "Spline.c"
#include "WMatrix.c"

void mexFunction(int nlhs, mxArray* plhs[], int nrhs, const mxArray* prhs[]) {
    double a=*mxGetPr(prhs[0]);
    double* knots_gam=mxGetPr(prhs[1]);
    double* weights_gam=mxGetPr(prhs[2]);
    int p_gam=(int)*mxGetPr(prhs[3]);
    double* nodes_gam=mxGetPr(prhs[4]);
    int is_rational_gam=(int)*mxGetPr(prhs[5]);
    double* cpoints_gam=mxGetPr(prhs[6]);
    double* knots=mxGetPr(prhs[7]);
    double* weights=mxGetPr(prhs[8]);
    int p=(int)*mxGetPr(prhs[9]);
    double* nodes=mxGetPr(prhs[10]);
    int is_rational=(int)*mxGetPr(prhs[11]);
    double* nodes_gauss=mxGetPr(prhs[12]);
    double* weights_gauss=mxGetPr(prhs[13]);
    double* nodes_loggauss=mxGetPr(prhs[14]);
    double* weights_loggauss=mxGetPr(prhs[15]);
    double* output;
    int l;

    int N_gam=mxGetM(prhs[1]);
    int n_gam=mxGetM(prhs[4]);
    int N=mxGetM(prhs[7]);
    int n=mxGetM(prhs[10]);
    int n_gauss=mxGetM(prhs[12]);
    int n_loggauss=mxGetM(prhs[14]);
    double b;
    int multb=0; // #b
    NURBSData* Data_Basis;
    NURBSData* Data_Gamma;
    QuadData* Data_Gauss;
    QuadData* Data_LogGauss;
    double wcpoints1_gam[N_gam];
    double wcpoints2_gam[N_gam];

    b=knots[N-1];
    while (nearly_equal(knots[N-multb-1],b)) {
        multb=multb+1;
```

```
        if (multb==N){break;}
    }
    plhs[0]=mxCreateDoubleMatrix(N-multb+1+p,N-multb+1+p,mxREAL);
    output=mxGetPr(plhs[0]);

    if (is_rational_gam==0){
        for (l=0;l<N_gam;l=l+1){
            wcpoints1_gam[l]=cpoints_gam[0+2*l];
            wcpoints2_gam[l]=cpoints_gam[1+2*l];
        }
    }else{
        for (l=0;l<N_gam;l=l+1){
            wcpoints1_gam[l]=weights_gam[l]*cpoints_gam[0+2*l];
            wcpoints2_gam[l]=weights_gam[l]*cpoints_gam[1+2*l];
        }
    }

    Data_Basis=new_NURBSData(a,knots,N,weights,p,nodes,n,is_rational);
    Data_Gamma=new_NURBSData(a,knots_gam,N_gam,weights_gam,p_gam,nodes_gam,
                             n_gam,is_rational_gam);
    Data_Gauss=new_QuadData(nodes_gauss,weights_gauss,n_gauss);
    Data_LogGauss=new_QuadData(nodes_loggauss,weights_loggauss,n_loggauss);

    build_W(output,Data_Gamma,wcpoints1_gam,wcpoints2_gam,Data_Basis,
                  Data_Gauss,Data_LogGauss);

    Data_Basis=del_NURBSData(Data_Basis);
    Data_Gamma=del_NURBSData(Data_Gamma);
    Data_Gauss=del_QuadData(Data_Gauss);
    Data_LogGauss=del_QuadData(Data_LogGauss);
}
```

References

[AFF+13] M. Aurada, M. Feischl, T. Führer, M. Karkulik, and D. Praetorius. Efficiency and optimality of some weighted-residual error estimator for adaptive 2D boundary element methods. *Comput. Methods Appl. Math.*, 13(3):305–332, 2013.

[AFF+15] M. Aurada, M. Feischl, T. Führer, M. Karkulik, J. M. Melenk, and D. Praetorius. Local inverse estimates for non-local boundary integral operators. *arXiv:1504.04394*, April 2015.

[BdVBSV14] L. Beirão da Veiga, A. Buffa, G. Sangalli, and R. Vázquez. Mathematical analysis of variational isogeometric methods. *Acta Numer.*, 23:157–287, 2014.

[Car97] C. Carstensen. An a posteriori error estimate for a first-kind integral equation. *Math. Comp.*, 66(217):139–155, 1997.

[CHB09] J. Cottrell, T. Hughes, and Y. Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons, 2009.

[CP06] C. Carstensen and D. Praetorius. Averaging techniques for the effective numerical solution of Symm's integral equation of the first kind. *SIAM J. Sci. Comput.*, 27(4):1226–1260, 2006.

[CS95] C. Carstensen and E. P. Stephan. A posteriori error estimates for boundary element methods. *Math. Comp.*, 64(210):483–500, 1995.

[de 86] C. de Boor. *B(asic)-spline basics*. Mathematics Research Center, University of Wisconsin-Madison, 1986.

[Fae00] B. Faermann. Localization of the Aronszajn-Slobodeckij norm and application to adaptive boundary element methods. I: The two-dimensional case. *IMA J. Numer. Anal.*, 20(2):203–234, 2000.

[FFK+15] M. Feischl, T. Führer, M. Karkulik, J. M. Melenk, and D. Praetorius. Quasi-optimal convergence rates for adaptive boundary element methods with data approximation. Part II: Hyper-singular integral equation. *Electron. Trans. Numer. Anal.*, 44:153–176, 2015.

[FGHP16a] M. Feischl, G. Gantner, A. Haberl, and D. Praetorius. Adaptive 2D IGA boundary element methods. *Eng. Anal. Bound. Elem.*, 62:141–153, 2016.

[FGHP16b] M. Feischl, G. Gantner, A. Haberl, and D. Praetorius. Optimal convergence for adaptive IGA boundary element methods for weakly-singular integral equations. *Numer. Math.*, pages 1–36, 2016.

[FGP] T. Führer, G. Gantner, and D. Praetorius. Optimal convergence for adaptive IGA boundary element methods with knot multiplicity decrease. Unpublished.

[FGP15] M. Feischl, G. Gantner, and D. Praetorius. Reliable and efficient a posteriori error estimation for adaptive IGA boundary element methods for weakly-singular integral equations. *Comput. Methods Appl. Mech. Engrg.*, 290:362–386, 2015.

[Gan14] G. Gantner. Adaptive isogeometric BEM. Master's thesis, Institute for Analysis and Scientific Computing, TU Wien, Wien, 2014.

[HCB05] T. Hughes, J. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Methods Appl. Mech. Engrg.*, 194(39):4135–4195, 2005.

[PGK+09] C. Politis, A. Ginnis, P. Kaklis, K. Belibassakis, and C. Feurer. An isogeometric BEM for exterior potential-flow problems in the plane. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 349–354. ACM, 2009.

[Pra06] D. Praetorius. *Numerische Mathematik*. Lecture Notes, Institute for Analysis and Scientific Computing, TU Wien, Wien, 2006.

[Pra07] D. Praetorius. *Boundary Element Method*. Lecture Notes, Institute for Analysis and Scientific Computing, TU Wien, Wien, 2007.

[SS11] S. A. Sauter and C. Schwab. *Boundary element methods.* Springer, Berlin, 2011.

[SSE+13] M. Scott, R. Simpson, J. Evans, S. Lipton, S. Bordas, T. Hughes, and T. Sederberg. Isogeometric boundary element analysis using unstructured t-splines. *Comput. Methods Appl. Mech. Engrg.*, 254:197–221, 2013.

[Ste08] O. Steinbach. *Numerical approximation methods for elliptic boundary value problems.* Springer, New York, 2008.