

ETH ZÜRICH

MASTER THESIS

Boundary Element Method on Complex Screens

Author:
Lorenzo Giacomel

Supervisor:
Prof. Dr. Ralf HIPTMAIR

*A thesis submitted in fulfillment of the requirements
for the degree of M.Sc. in Computational Science and Engineering
in the*

Seminar for Applied Mathematics
Department of Mathematics

November 27, 2018

"If you can meet with triumph and disaster and treat those impostors just the same."

Rudyard Kipling

Acknowledgements

I would like to thank Prof. Dr. Ralf Hiptmair for providing me with such an interesting project and for his supervision during the past six months. I am really grateful to him not only for the help with the thesis, but also for having inspired me during the past months and for having always been a role-model showing me how a real scientist should work.

I also would like to thank Dr. Carolina Urzúa-Torres for her invaluable help and for giving me so many pieces of advice, which improved so much the quality of this thesis. Thanks in particular for dedicating me a lot of time even in such a busy period like the transition between the end of her PhD and the beginning of her postdoc.

Indeed, I would like to mention all the people who have supported, helped and inspired me during my studies.

The biggest part of the credit for my achievements goes to my parents who have taught me the importance of studying and who have always been exemplary hard workers. I owe everything to them.

Many thanks also to my brother, Luca, who has (probably subconsciously) taught me to never lay back and to keep pursuing greater goals. Moreover, back in 2015 he was the first suggesting me to apply for ETH Zürich. It was not such a bad idea after all.

I cannot forget to thank my CSE mates: Lars, Mathis, Philipp, Pierre and Sam. Studying at ETH without them would have been much more difficult and much less funny.

A special mention goes to all the people I had the pleasure to share the Rebhüsliweg 1D flat with. In particular, I would like to thank Federico, Tommaso and Marco for making me feel like in a family also here in Zürich.

I am also grateful to Stefano for the crazy nights spent together in Langstrasse and for all the fun we had during my staying in Zürich.

Finally, I would like to thank all of my *affascinanti ma un pizzico cafoni* friends back in Italy, who are fortunately too many to be mentioned one by one. It is so great that in these two years they always found some time for a little chat and a beer (or two) when I was back in Milano.

Contents

Acknowledgements	iii
Introduction	1
1 Theory for Acoustic Scattering by Complex Screens	3
1.1 Function Spaces	3
1.1.1 Geometry	3
1.1.2 Standard Trace Spaces	4
Standard Dirichlet traces	4
Standard Neumann Traces	4
1.1.3 Domain Based Function Spaces	5
1.1.4 Multi-Trace Spaces	6
1.1.5 Single-Trace Spaces	8
1.1.6 Jump Spaces	8
1.2 Boundary Integral Equations for acoustic scattering	9
1.2.1 Trace Operators	10
1.2.2 Layer Potentials and Representation Formula	10
1.2.3 BIOs and BIEs	11
1.3 Galerkin Discretization of the Boundary Integral Equations	13
1.3.1 Meshes on an closed Polygon	13
1.3.2 Triangular Meshes on Polyhedron	14
1.3.3 The Boundary Element Spaces	15
1.3.4 Linear System Construction	15
The Weakly Singular Operator System	15
The Hyper-Singular Operator System	16
1.3.5 The Variational Formulation of the Hypersingular Operator	17
2 2D Laplace Equation	19
2.1 Tests for the Kernel Dimensions	19
2.1.1 Kernel of the Weakly Singular operator	21
2.1.2 Kernel of the Hyper-Singular Operator	22
Couple of Vertices Away from the Junctions	22
Vertices Overlapping at the Junction	25
Vertices at the Tips	25
2.2 Tests for CG and GMRES Iterative Solvers	26
2.2.1 CG and GMRES for the Weakly-Singular BIO	26
2.2.2 CG and GMRES for the Hyper-Singular BIO	27
2.3 Validation on a segment	28
2.3.1 Validation for the Weakly-Singular BIO	28
2.3.2 Validation for the Hyper-Singular BIO	31
2.4 Elimination of the Kernels	31

3	3D Laplace Equation	35
3.1	Construction of the Multi-Trace Approach	35
3.2	Tests for the Kernel Dimensions	37
3.2.1	Kernel of the Weakly Singular Operator	37
3.2.2	Kernel of the Hypersingular Operator	37
	Couple of Vertices Away from the Junctions	40
	Vertices Overlapping at the Junctions	41
	Vertices at the Boundaries	42
3.3	Tests for CG and GMRES solvers	43
3.3.1	CG and GMRES for the Weakly Singular BIO	45
3.3.2	CG and GMRES for the Hypersingular BIO	45
3.4	Elimination of the kernels	46
4	Theory for Electromagnetic Scattering by Complex Screens	49
4.1	Function Spaces	49
4.1.1	Standard Tangential Trace Space	49
4.1.2	Function Spaces for Vector Fields in the Domain	50
4.1.3	Tangential Multi-Trace Space	50
4.1.4	Tangential Single-Trace Space	51
4.1.5	Tangential Jump Spaces	52
4.2	Boundary Integral Equations	52
4.2.1	Surface Differential Operators	52
4.2.2	Boundary Value Problem and Traces	54
	Boundary Value Problem	54
	Traces	55
4.2.3	Layer Potentials	55
4.2.4	The Representation Formula and the EFIE	56
4.3	Galerkin Discretization of the EFIE	57
4.3.1	The Boundary Element Spaces	57
4.3.2	Linear System Construction	59
5	3D Electromagnetic Scattering	61
5.1	Tests for the Kernel Dimensions	61
	Couples of Vertices Away from the Junctions	61
	Edges Overlapping at the Junctions	63
	Edges at the Boundaries	65
5.2	Test for CG and GMRES	66
5.2.1	The Static EFIE	67
5.2.2	The Indefinite EFIE	68
A	The Matlab Codes	73
B	The C++ Codes	103
	Bibliography	105

List of Figures

1.1	Infinitesimally thick screen	7
1.2	Multi-trace mesh on an infinitesimally thick screen	13
2.1	The screens used in our numerical simulations.	20
2.2	Eigenvalues of \mathbf{V} (left) and \mathbf{W} (right) on a hierarchy of nested meshes.	21
2.3	Multi-trace meshes in which equal DoFs are indicated with the same color	23
2.4	Multi-trace meshes in which equal DoFs are indicated with the same color	24
3.1	The screens used in our 3D numerical simulations	36
3.2	Rendering of the screens used in our numerical experiments as if they had a non-zero thickness	38
3.3	Eigenvalues of the BEM matrices on hierarchies of nested meshes. In the left column the dashed line indicates the machine precision.	39
3.4	Representation of the support of the basis functions associated with overlapping DoFs i and j and a generic DoF k	40
3.5	Representation of the support of the basis functions associated with two overlapping DoFs i and j and a generic DoF k	41
3.6	Representation of the support of the basis functions associated with three DoFs i , j and k which overlap at a triple junction and a generic DoF z	42
3.7	Representation of the support of the basis functions associated with a boundary DoF k and a generic DoF z	43
3.8	Two overlapping triangles with different local ordering of the nodes due to the opposite normals.	44
5.1	Representation of the support of the basis functions associated with the overlapping edges i and j and the generic edge k	62
5.2	Representation of the supports of the basis functions associated with the edges i, j and k , which overlap at the junction and the generic edge z	63
5.3	Representation of the support of the basis functions associated with a boundary edge k and a generic edge z	64
5.4	The magnitude of the eigenvalues of \mathbf{S} computed on a disk on a hierarchy of nested meshes.	66
A.1	Structure of the multiscreen_matlab Gitab repository	73
B.1	Structure of the BETL2 Gitab repository	104

List of Tables

2.1	Kernel dimensions of \mathbf{V} when using a mesh with N elements. The kernel dimension is always half as the total number of elements	22
2.2	Kernel dimensions of \mathbf{W} when using a mesh with N inner DoFs.	26
2.3	Condition number of \mathbf{V} with mesh thickness ε and N_e elements for each side of the fins.	27
2.4	Number of iterations for the CG solution of 2.5 with mesh thickness ε and system size and N_e elements for each side of the fins.	28
2.5	Number of iterations for the GMRES solution of 2.5 with mesh thickness ε and N_e elements for each side of the fins.	28
2.6	Condition number of \mathbf{W} with mesh thickness ε	29
2.7	Number of iterations for the CG solution of 2.6 with mesh thickness ε and system size N_f	29
2.8	Number of iterations for the GMRES solution of 2.6 with mesh thickness ε	30
2.9	Energy norm of the error and order of convergence for the weakly-singular operator on the segment $[-1, 1] \times \{0\}$ with N elements	30
2.10	Energy norm of the error and order of convergence for the hypersingular operator on the segment $[-1, 1] \times \{0\}$ with N elements	31
2.11	Comparison of the numbers of iterations to convergence for the weakly singular system (left) and the hypersingular system (right).	33
3.1	Dimension of the kernel of \mathbf{W} when using a mesh with N_T triangular elements	40
3.2	Dimension of the kernel of \mathbf{W} when using a mesh with N_T triangular elements	44
3.3	number of iterations to convergence for the hypersingular BIO system with condition number	45
3.4	Number of iterations to convergence for the hypersingular BIO system with condition number with non corrected entries.	46
3.5	Number of iterations to convergence for the discrete hypersingular BIO system with corrected entries.	47
5.1	Dimension of the kernel of \mathbf{S} when using a mesh with N_T triangular elements	67
5.2	Number of iterations to convergence for the static EFIE system with non corrected entries	68
5.3	Number of iterations to convergence for the static EFIE system with corrected entries	68
5.4	Number of iterations to convergence for the EFIE system with non corrected entries	69
5.5	Number of iterations to convergence for the corrected EFIE system with corrected entries	69

A Marina, Claudio e Luca

Introduction

The scattering of acoustic and electromagnetic waves is a topic of research which has been intensively studied by physicists, mathematicians and engineers over the last 150 years. The modelling of these phenomena involve partial differential equations whose solutions vary considerably depending on the geometry of the scatterer. Except in very few cases, it is too hard to derive analytic solutions to the scattering problems, therefore a big effort has been put in building numerical methods, which are able to approximate these solutions. For interior problems, in which case the domain is limited, Finite Element Method (FEM) is a really valid tool and its mathematical properties and practical applicability have been studied deeply. On the other hand, in many realistic scenarios scattering problems are actually posed in the whole space and FEM is not a suitable choice any more, as, in principle, its application would require the discretization of the whole \mathbb{R}^d space ($d = 2, 3$). Boundary Element Method (BEM), is instead much more attractive as it allows to reconstruct the solution in the whole space basing only on its traces on the boundary of the scatterer. As a result, to apply BEM one only needs to mesh the boundary of the scatterer. Moreover, given that only a discretization of the boundaries is required, the dimensionality of the problem is lowered, which is an attractive feature because the complexity of the algorithms is reduced.

From a mathematical point of view, BEM is a very interesting field of research, since a lot of effort is needed to prove its suitability when dealing with non-smooth geometries. In the beginning, the method has been derived for the case of smooth shapes but the mathematical justification of its extension to non-smooth or complex geometries is still one of the biggest challenges. Nowadays, the theoretical framework for the application of BEM to scattering by bodies with Lipschitz boundaries is well established [4] and the attention has moved towards complex geometries, such as composite objects or screens. In this thesis, in particular, we focus on the study of screen geometries, in which the thickness of the scatterer is much smaller than the other dimensions. The first applications of BEM to screen problems date back to the second half of the last century [13] but the issues related to non-smooth geometries have been solved at a much later time [2, 3]. In this thesis, we focus on a particular class of shapes, to which we refer as multi-screens. These geometries are composed by many smooth screens which intersect at junctions. At these points multi-screens are not locally-Lipschitz and not locally-orientable, for which reason the theory for Lipschitz screens cannot be applied. Moreover, at the junctions the screens have more than two sides, and, therefore, it is not clear how to practically define the jumps of the traces, which play a key role in the Boundary Integral Equations (BIE).

All these problems have been solved through a theory which considers the multi-screen as an infinitesimally thick bodies [6, 5]. This theoretical framework gives an abstract definition of the jumps, which can be shown to correspond with the standard definitions when considering simple geometries. On the other hand, it may

seem inconvenient for discretization purposes the presence of really abstract definition, but we show that the jumps do not show up in the equations which we discretize.

Our contribution to this theory is the discretization through a Galerkin approach of the BIE related to acoustic and electromagnetic scattering by complex screens in two and three dimensions. Moreover, we implement these ideas and report the numerical results for our experiments. The outline of the thesis is the following:

- **Chapter 1:** In the first chapter we summarize the theoretical background for the scattering of scalar fields by multi-screens and we show how to apply a Galerkin discretization to the BIEs which arise.
- **Chapter 2:** We test the ideas of BEM on multi-screens in the case of acoustic scattering in 2D, for the implementation we use a MATLAB code. The testing part also involves the comparison to a reference solution.
- **Chapter 3:** We implement a 3D C++ code using the BETL2 library [7] and we carry out the same discussion as in the previous chapter.
- **Chapter 4:** This chapter is devoted to recapitulate the theory for electromagnetic scattering at multi-screens. The BIE which arises, the so called Electric Field Integral Equation (EFIE), is again discretized using a Galerkin approach.
- **Chapter 5:** Using BETL2, we implement in C++ the Galerkin discretization of the EFIE and we discuss the obtained results.

Chapter 1

Theory for Acoustic Scattering by Complex Screens

In this chapter we discuss the mathematical background needed to deal with the Helmholtz equation in presence of complex screens. All the results exposed in this chapter are entirely derived in [6], to which we refer for the proofs and more detailed explanations. To begin with, in section 1.1 we introduce the function spaces which are useful for the formulation of the Boundary Integral Equations (BIE). In section 1.2, we present the Boundary Value Problems (BVP) which we want to tackle and enunciate the BIEs that can be used to solve those BVPs. Finally, in section 1.3 we show how to carry out a Galerkin discretization of our BIEs.

Remark: Throughout this chapter we will consider \mathbb{R}^d with $d = 2$ or 3 .

1.1 Function Spaces

First of all, it is fundamental to derive a suitable functional analysis framework to make sense of concepts like traces and jumps across a multi-screen. The standard definition of traces and jumps for screen problems cannot be directly extended to the case of complex screens, but we will enunciate some abstract definition, which in the case of a Lipschitz screen correspond to the classic definitions [2]. Interestingly, whenever considering a smooth surface most of the spaces have an interpretation as quotients of functions spaces defined on the domain. This observation plays a key role in the development of a functional analysis framework for complex screens, as it offers an alternative way to local chart mapping functions (which would do not work on complex screens) for the construction of spaces on the screen.

1.1.1 Geometry

In this paragraph we rigorously define the notion of a *Complex Screen*. Sloppily speaking, we consider geometries which are generated by gluing together many simple screens at a junction, and we refer to this class of shapes as *multi-screens*. For the definition of a multi-screen, it is useful to give the following auxiliary concept.

Definition 1 (Lipschitz Partition [6, Definition 2.2]). *A Lipschitz partition of \mathbb{R}^d is a finite collection of Lipschitz open sets $(\Omega_j)_{j=0\dots n}$ such that $\mathbb{R}^d = \cup_{j=0}^n \overline{\Omega}_j$ and $\Omega_j \cap \Omega_k = \emptyset$, if $j \neq k$.*

Now we are ready to define a multi-screen

Definition 2 (Multi-screen [6, Definition 2.3]). *A multi-screen is a subset $\Gamma \subset \mathbb{R}^d$ such that there exists a Lipschitz partition \mathbb{R}^d denoted $(\Omega_j)_{j=0\dots n}$ satisfying $\Gamma \subset \cup_{j=0}^n \overline{\Omega}_j$ and*

such that for each $j = 0 \dots n$, we have $\bar{\Gamma} \cap \partial\Omega_j = \bar{\Gamma}_j$ where $\bar{\Gamma}_j \subset \partial\Omega_j$ is some Lipschitz screen in the sense of Buffa-Christiansen [2, section 1.1].

It can be shown that multi-screens are a generalization of Lipschitz screens, which means that Lipschitz screens fit the definition of multi-screens (see the Möbius strip example in [6]).

Multi-screens pose two great difficulties: they are not necessarily locally orientable (alike Lipschitz screens) and, additionally, junction points where the screen is not two sided are admitted. At these points, It is not clear how to define jumps of traces, and this will require us to define more abstract trace spaces and trace operators than in the Lipschitz screen case.

1.1.2 Standard Trace Spaces

First of all, we summarize the standard functional analysis framework in which instead of Γ we consider traces on $\partial\Omega$, the boundary of a domain Ω . This will help us to find a new strategy for the construction of trace spaces.

To begin with, we have spaces of functions defined on the domain Ω :

- $H^1(\Omega) = \{v \in L^2(\Omega) : \|v\|_{H^1(\Omega)}^2 := \int_{\Omega} |v|^2 + |\nabla v|^2 dx < \infty\}$;
- $H_{0,\partial\Omega}^1(\Omega)$ is the closure of $C_{0,\partial\Omega}^\infty := \{\varphi \in C^\infty(\bar{\Omega}) : \varphi = 0 \text{ in a neighbourhood of } \partial\Omega\}$ with respect to the $\|\cdot\|_{H^1(\Omega)}$ norm;
- $\mathbf{H}(\text{div}, \Omega) = \{\mathbf{q} \in L^2(\Omega)^d : \|\mathbf{q}\|_{\mathbf{H}(\text{div}, \Omega)}^2 = \int_{\Omega} |\mathbf{q}|^2 + |\text{div}(\mathbf{q})|^2 dx < \infty\}$;
- $H_{0,\partial\Omega}(\text{div}, \Omega)$ is the closure of $C_{0,\partial\Omega}^\infty(\bar{\Omega})^d$ with respect to the $\|\cdot\|_{\mathbf{H}(\text{div}, \Omega)}$ norm.

Standard Dirichlet traces

We can define the Dirichlet trace of a function in $H^1(\Omega)$ as its restriction to Γ $\tau_{D,\Gamma} : v \mapsto v|_{\Gamma}$. Now the standard Dirichlet trace spaces are the following:

$$H^{\frac{1}{2}}(\partial\Omega) := \{u|_{\Gamma} : u \in H^1(\Omega)\} = \text{Range}(\tau_{D,\partial\Omega}).$$

We note that $H^{\frac{1}{2}}(\Gamma)$ can be alternatively identified as a quotient space. This is possible thanks to the fact that $H_{0,\partial\Omega}^1(\Omega) = \text{Ker}(\tau_{D,\partial\Omega})$. As a result, $\tau_{D,\partial\Omega}$ defines an isomorphism from $H^1(\Omega)/H_{0,\partial\Omega}^1(\Omega)$ to $H^{\frac{1}{2}}(\partial\Omega)$ which allows the identification

$$H^{\frac{1}{2}}(\partial\Omega) = H^1(\Omega)/H_{0,\partial\Omega}^1(\Omega), \quad (1.1)$$

as stated in [6, Equation 3].

Standard Neumann Traces

Also for the Neumann trace, it is possible to define a point trace operator. If we denote by \mathbf{n} the normal vector to $\partial\Omega$ (which is well-defined as now we are considering Ω to be a Lipschitz domain), we can define the normal component trace operator $\tau_{N,\Gamma} : \mathbf{q} \mapsto \mathbf{q} \cdot \mathbf{n}$. It can be seen that this operator induces a continuous and surjective mapping from $\mathbf{H}(\text{div}, \Omega)$ onto $H^{-\frac{1}{2}}(\partial\Omega) = (H^{\frac{1}{2}}(\partial\Omega))'$. Now we are ready to define the standard spaces

$$H^{-\frac{1}{2}}(\partial\Omega) := \{q|_{\partial\Omega} : q \in H^{-\frac{1}{2}}(\partial\Omega)\},$$

Also in this case, we can give a quotient space interpretation of $H^{-\frac{1}{2}}(\partial\Omega)$. By observing that $\mathbf{H}_{0,\Gamma}(\operatorname{div}, \Omega) = \operatorname{Ker}(\tau_{N,\Gamma})$, we have the identification

$$H^{-\frac{1}{2}}(\Gamma) = \mathbf{H}(\operatorname{div}, \Omega) / H_{0,\partial\Omega}(\operatorname{div}, \Omega) = \mathbf{H}(\operatorname{div}, \mathbb{R}^d \setminus \overline{\Omega}) / \mathbf{H}_{0,\partial\Omega}(\operatorname{div}, \mathbb{R}^d \setminus \overline{\Omega}). \quad (1.2)$$

For both $H^{\frac{1}{2}}(\partial\Omega)$ and $H^{-\frac{1}{2}}(\partial\Omega)$ it is possible to use the quotient space norms induced by the previously identifications.

Equation (1.1) and (1.2) suggest a strategy to define trace spaces on a multi-screen without using local chart mapping functions, and we will see that this approach works also on non locally orientable and non locally Lipschitz manifolds, like the multi-screens. Therefore, we will first define suitable spaces of functions defined on $\mathbb{R} \setminus \overline{\Gamma}$, where Γ is a multi-screen, and successively we will define the trace spaces as quotient spaces of these domain based function spaces.

1.1.3 Domain Based Function Spaces

In light of the previous discussion we concentrate on defining function spaces defined on $\mathbb{R} \setminus \overline{\Gamma}$, where Γ is a multi-screen.

First of all, we concentrate on $H^1(\mathbb{R}^d \setminus \overline{\Gamma})$. This space is the set of all functions $u \in L^2(\mathbb{R}^d)$ such that there exists a $\mathbf{p} \in L^2(\mathbb{R}^d)^d$ for which

$$\int_{\mathbb{R}^d \setminus \overline{\Gamma}} u \operatorname{div}(\mathbf{q}) dx = - \int_{\mathbb{R}^d \setminus \overline{\Gamma}} \mathbf{p} \cdot \mathbf{q} dx \quad \forall \mathbf{q} \in \mathcal{D}(\mathbb{R}^d \setminus \overline{\Gamma}),$$

where for an open set $\omega \subset \mathbb{R}^d$

$$\mathcal{D}(\omega) = \{\varphi \in C^\infty(\omega) \mid \operatorname{supp}(\varphi) \subset \omega\}.$$

It is important to notice that we can write $\mathbf{p} = \nabla u|_{\mathbb{R}^d \setminus \overline{\Gamma}}$ in the sense of distributions on $\mathbb{R}^d \setminus \overline{\Gamma}$ but we have $\mathbf{p} \neq \nabla u$ in the sense of distributions on \mathbb{R}^d . We equip $H^1(\mathbb{R}^d \setminus \overline{\Gamma})$ with the scalar product

$$(u, v)_{H^1(\mathbb{R}^d \setminus \overline{\Gamma})} := \int_{\mathbb{R}^d \setminus \overline{\Gamma}} u \overline{v} dx + \int_{\mathbb{R}^d \setminus \overline{\Gamma}} (\nabla u|_{\mathbb{R}^d \setminus \overline{\Gamma}}) \cdot (\nabla \overline{v}|_{\mathbb{R}^d \setminus \overline{\Gamma}}) dx,$$

which renders $H^1(\mathbb{R}^d \setminus \overline{\Gamma})$ a Hilbert space. Now we can endow this space with the norm induced by the $(u, v)_{H^1(\mathbb{R}^d \setminus \overline{\Gamma})}$ scalar product, which is

$$\|u\|_{H^1(\mathbb{R}^d \setminus \overline{\Gamma})}^2 := \|u\|_{L^2(\mathbb{R}^d)}^2 + \|\mathbf{p}\|_{L^2(\mathbb{R}^d)}^2 \quad \text{with } \mathbf{p} = \nabla u|_{\mathbb{R}^d \setminus \overline{\Gamma}}.$$

The difference between $H^1(\mathbb{R}^d \setminus \overline{\Gamma})$ and $H^1(\mathbb{R}^d)$ is that functions belonging to the former are allowed to jump across Γ , while functions belonging to the latter are not. Accordingly, it can be shown that $H^1(\mathbb{R}^d \setminus \overline{\Gamma})$ strictly contains $H^1(\mathbb{R}^d)$ as a non-trivial closed subspace.

In a very similar way, we can define the space $H^1(\operatorname{div}, \mathbb{R}^d \setminus \overline{\Gamma})$ with analogous scalar product and norm (obtained by simply replacing the gradients with divergences). We can observe that, like in the previous case, $H^1(\operatorname{div}, \mathbb{R}^d \setminus \overline{\Gamma})$ strictly contains $H^1(\operatorname{div}, \mathbb{R}^d)$ as a non-trivial closed subspace.

We also discuss spaces for functions that vanish in the neighbourhood of Γ . Notice that such functions are not allowed to jump across Γ .

In particular we define:

- $H_{0,\Gamma}^1(\mathbb{R}^d)$ to be the closure in $H^1(\mathbb{R}^d \setminus \overline{\Gamma})$ of the set $\mathcal{D}(\mathbb{R}^d \setminus \overline{\Gamma})$;

- $\mathbf{H}_{0,\Gamma}(\operatorname{div}, \mathbb{R}^d)$ to be the closure in $H^1(\operatorname{div}, \mathbb{R}^d \setminus \bar{\Gamma})$ of the set $\mathcal{D}(\mathbb{R}^d \setminus \bar{\Gamma})$;

The definitions of this section result in the two following chains of inclusions

$$\begin{aligned} H_{0,\Gamma}^1(\mathbb{R}) &\subset H^1(\mathbb{R}) \subset H^1(\mathbb{R} \setminus \bar{\Gamma}), \\ \mathbf{H}_{0,\Gamma}(\operatorname{div}, \mathbb{R}) &\subset \mathbf{H}(\operatorname{div}, \mathbb{R}) \subset \mathbf{H}(\operatorname{div}, \mathbb{R} \setminus \bar{\Gamma}). \end{aligned}$$

which make the quotient spaces in the next sections meaningful.

1.1.4 Multi-Trace Spaces

Having defined proper domain based function spaces, we are now ready to define trace spaces by taking quotients of them. Two types of trace spaces will arise: the multi-traces, which are allowed to have different values depending which side of the multi-screen is considered; the single-traces, which are special multi-traces taking the same value on each side of the screen.

In this section we concentrate on the first kind of traces and we leave the discussion about the single-traces to the next section.

Definition 3 (Multi-Trace Spaces). *The Dirichlet and Neumann multi-trace spaces are, respectively, defined as follows:*

$$\begin{aligned} \mathbb{H}^{+\frac{1}{2}}(\Gamma) &:= H^1(\mathbb{R}^d \setminus \bar{\Gamma}) / H_{0,\Gamma}^1(\mathbb{R}^d), \\ \mathbb{H}^{-\frac{1}{2}}(\Gamma) &:= \mathbf{H}(\operatorname{div}, \mathbb{R}^d \setminus \bar{\Gamma}) / \mathbf{H}_{0,\Gamma}(\operatorname{div}, \mathbb{R}^d). \end{aligned} \quad (1.3)$$

We will tag the elements of these spaces with a \cdot (e.g. \dot{u}, \dot{p}). This is done to emphasize the fact that elements of quotient spaces are classes of equivalence. Moreover, we equip the multi-trace spaces with the usual quotient space norms $\|\cdot\|_{\mathbb{H}^{\pm\frac{1}{2}}}$.

By virtue of the definition of multi-trace spaces as quotient spaces, it is easy to define trace-like operators for functions in $H^1(\mathbb{R}^d \setminus \bar{\Gamma})$ and $\mathbf{H}(\operatorname{div}, \mathbb{R}^d \setminus \bar{\Gamma})$ using the canonical surjections

$$\pi_D : H^1(\mathbb{R}^d \setminus \bar{\Gamma}) \rightarrow \mathbb{H}^{\frac{1}{2}}(\Gamma) \quad \text{and} \quad \pi_N : \mathbf{H}(\operatorname{div}, \mathbb{R}^d \setminus \bar{\Gamma}) \rightarrow \mathbb{H}^{-\frac{1}{2}}(\Gamma). \quad (1.4)$$

We can now build a special pairing with respect to which $\mathbb{H}^{\frac{1}{2}}(\Gamma)$ and $\mathbb{H}^{-\frac{1}{2}}(\Gamma)$ are dual to each other. We start with the observation that Green's formula does not necessarily hold for functions in $H^1(\mathbb{R}^d \setminus \bar{\Gamma})$ and $\mathbf{H}(\operatorname{div}, \mathbb{R}^d \setminus \bar{\Gamma})$. Indeed, for any $u \in H^1(\mathbb{R}^d \setminus \bar{\Gamma})$ and any $\mathbf{p} \in \mathbf{H}(\operatorname{div}, \mathbb{R}^d \setminus \bar{\Gamma})$ we have

$$\int_{\mathbb{R}^d \setminus \bar{\Gamma}} \mathbf{p} \cdot \nabla u + u \operatorname{div}(\mathbf{p}) dx \neq 0.$$

On the other hand, adding any function which vanishes on the screen to \mathbf{p} and v does not change the value of the integral:

$$\begin{aligned} \int_{\mathbb{R}^d \setminus \bar{\Gamma}} (\mathbf{p} + \mathbf{q}) \cdot \nabla (u + v) + (u + v) \operatorname{div}(\mathbf{p} + \mathbf{q}) dx &= \int_{\mathbb{R}^d \setminus \bar{\Gamma}} \mathbf{p} \cdot \nabla u + u \operatorname{div}(\mathbf{p}) dx \\ &\quad \forall v \in H_{0,\Gamma}^1(\mathbb{R}^d), \forall \mathbf{q} \in \mathbf{H}_{0,\Gamma}(\operatorname{div}, \mathbb{R}^d). \end{aligned}$$

This suggests that this formula represents a good bilinear pairing between the quotient spaces $\mathbb{H}^{\frac{1}{2}}(\Gamma)$ and $\mathbb{H}^{-\frac{1}{2}}(\Gamma)$.

For any $\dot{u} \in \mathbb{H}^{\frac{1}{2}}$ and $\dot{p} \in \mathbb{H}^{-\frac{1}{2}}$ we choose $u \in H^1(\mathbb{R} \setminus \bar{\Gamma})$ and $\mathbf{p} \in \mathbf{H}(\operatorname{div}, \mathbb{R} \setminus \bar{\Gamma})$ such

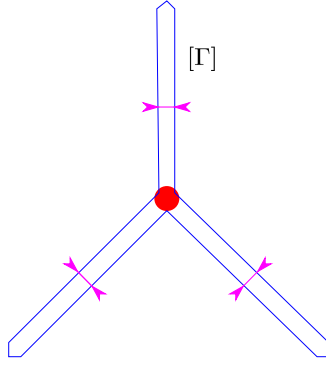


FIGURE 1.1: Infinitesimally thick screen

that $\pi_D(u) = \dot{u}$ and $\pi_N(\mathbf{p}) = \dot{\mathbf{p}}$ and construct the pairing

$$\ll \dot{u}, \dot{\mathbf{p}} \gg = \int_{\mathbb{R}^d \setminus \bar{\Gamma}} \mathbf{p} \cdot \nabla u + u \operatorname{div}(\mathbf{p}) \, dx. \quad (1.5)$$

Observation 1. *The bilinear form (1.5) is closely related to the usual L^2 -pairing $\langle \cdot, \cdot \rangle$ on the surface of a Lipschitz body. As a matter of fact, the right-hand side in equation (1.5) computes via integration by parts an integral on the boundary of the domain $\mathbb{R}^d \setminus \bar{\Gamma}$. We will see that we only consider radiating solutions, which satisfy decay conditions, and, as a result, the $\ll \cdot, \cdot \gg$ -pairing depends only on their value on the part of boundary neighboring the screen. It can be seen that computing $\ll \dot{u}, \dot{\mathbf{p}} \gg$ is equivalent to computing $\langle \dot{u}, \dot{\mathbf{p}} \rangle$ on the boundary of an imaginary infinitesimally thick screen, as depicted in figure 1.1. In order to convey this idea, we will use a special notation to indicate the integration on the boundary of the infinitesimally thick screen:*

$$\int_{[\Gamma]} \dot{u} \dot{\mathbf{p}} d\sigma = \int_{\mathbb{R}^d \setminus \bar{\Gamma}} \mathbf{p} \cdot \nabla u + u \operatorname{div}(\mathbf{p}) \, dx.$$

This notation is not theoretically rigorous and the integral should not be read as if it was with respect to the Lebesgue measure on Γ , however, it emphasizes the idea of the integration on the boundary of an infinitesimally thick screen $[\Gamma]$ which will be of practical use for the Galerkin discretization.

We observe immediately that the choice of the $\ll \cdot, \cdot \gg$ -pairing is particularly appropriate as, analogously to what happens for $H^{\pm\frac{1}{2}}(\partial\Omega)$ on smooth boundaries, we have that $\mathbb{H}^{\pm\frac{1}{2}}(\Gamma)$ are dual to each other via this pairing.

Proposition 1 ([6, Proposition 5.1]). *The pairing $\ll \cdot, \cdot \gg: \mathbb{H}^{+\frac{1}{2}}(\Gamma) \times \mathbb{H}^{-\frac{1}{2}}(\Gamma) \rightarrow \mathbb{C}$ defined by*

$$\ll \dot{v}, \dot{q} \gg = \int_{[\Gamma]} \dot{q} \dot{v} \, d\sigma \quad \forall \dot{v} \in \mathbb{H}^{+\frac{1}{2}}(\Gamma), \forall \dot{q} \in \mathbb{H}^{-\frac{1}{2}}(\Gamma)$$

induces and isometric duality between $\mathbb{H}^{+\frac{1}{2}}(\Gamma)$ and $\mathbb{H}^{-\frac{1}{2}}(\Gamma)$.

The last result of this section follows directly from the duality between $\mathbb{H}^{+\frac{1}{2}}(\Gamma)$ and $\mathbb{H}^{-\frac{1}{2}}(\Gamma)$.

Corollary 1 ([6, Corollary 5.2]). *Let $u \in H^1(\mathbb{R}^d \setminus \bar{\Gamma})$ and $\mathbf{p} \in \mathbf{H}(\text{div}, \mathbb{R}^d \setminus \bar{\Gamma})$. We have the following characterizations*

$$\begin{aligned} u \in H_{0,\Gamma}^1(\mathbb{R}^d) &\iff \int_{\mathbb{R}^d \setminus \bar{\Gamma}} \mathbf{q} \cdot \nabla u + u \text{div}(\mathbf{q}) dx = 0 \quad \forall \mathbf{q} \in \mathbf{H}(\text{div}, \mathbb{R}^d \setminus \bar{\Gamma}), \\ \mathbf{p} \in \mathbf{H}_{0,\Gamma}(\text{div}, \mathbb{R}^d) &\iff \int_{\mathbb{R}^d \setminus \bar{\Gamma}} \mathbf{p} \cdot \nabla v + v \text{div}(\mathbf{p}) dx = 0 \quad \forall v \in H^1(\mathbb{R}^d \setminus \bar{\Gamma}). \end{aligned}$$

This result shows that, through the \ll, \gg duality pairing, $H^1(\mathbb{R}^d \setminus \bar{\Gamma})$ and $\mathbf{H}_{0,\Gamma}(\text{div}, \mathbb{R}^d)$ are polar to each other, as well as $H_{0,\Gamma}^1(\mathbb{R}^d)$ and $\mathbf{H}(\text{div}, \mathbb{R}^d \setminus \bar{\Gamma})$.

1.1.5 Single-Trace Spaces

The multi-trace spaces we described in the previous section allow the traces to have different values on opposite sides of the screen. Therefore, we can think of the two sides of the panel as independent surfaces on which the functions are free to assume different values. It is now possible to single out subspaces of $\mathbb{H}^{\frac{1}{2}}(\Gamma)$ and $\mathbb{H}^{-\frac{1}{2}}(\Gamma)$ that contain traces of single-valued functions. These spaces are obtained by simply replacing " $\mathbb{R}^d \setminus \bar{\Gamma}$ " with " \mathbb{R}^d " in (1.3), so we take the quotient of functions that are continuous across the screen.

Definition 4 (Single-trace spaces). *We define the single-trace spaces as the quotient spaces*

$$\begin{aligned} H^{+\frac{1}{2}}([\Gamma]) &:= H^1(\mathbb{R}^d) / H_{0,\Gamma}^1(\mathbb{R}^d), \\ H^{-\frac{1}{2}}([\Gamma]) &:= \mathbf{H}(\text{div}, \mathbb{R}^d) / \mathbf{H}_{0,\Gamma}(\text{div}, \mathbb{R}^d). \end{aligned} \tag{1.6}$$

The name "single-trace" is due to the fact that traces in those spaces either agree on opposite sides of Γ (in the case of $\mathbb{H}^{+\frac{1}{2}}(\Gamma)$) or they have opposite sign (in the case of $\mathbb{H}^{-\frac{1}{2}}(\Gamma)$). As desired, the single-trace spaces are subspaces of the multi-trace spaces.

Corollary 2 ([6, Proposition 6.2]). *The space $H^{+\frac{1}{2}}([\Gamma])$ (resp. $H^{-\frac{1}{2}}([\Gamma])$) is a closed subspace of $\mathbb{H}^{+\frac{1}{2}}(\Gamma)$ (resp. $\mathbb{H}^{-\frac{1}{2}}(\Gamma)$).*

Although $H^{\pm\frac{1}{2}}([\Gamma]) \subset \mathbb{H}^{\pm\frac{1}{2}}(\Gamma)$, $H^{\pm\frac{1}{2}}([\Gamma])$ are not dual to each other with respect to the \ll, \gg pairing, they are polar.

Proposition 2 ([6, Proposition 6.3]). *For $\dot{u} \in H^{+\frac{1}{2}}([\Gamma])$ and $\dot{p} \in H^{-\frac{1}{2}}([\Gamma])$ it holds true*

$$\begin{aligned} \dot{u} \in H^{+\frac{1}{2}}([\Gamma]) &\iff \int_{[\Gamma]} \dot{u} \dot{q} d\sigma = 0 \quad \forall \dot{q} \in H^{-\frac{1}{2}}([\Gamma]), \\ \dot{p} \in H^{-\frac{1}{2}}([\Gamma]) &\iff \int_{[\Gamma]} \dot{p} \dot{q} d\sigma = 0 \quad \forall \dot{q} \in H^{+\frac{1}{2}}([\Gamma]). \end{aligned}$$

1.1.6 Jump Spaces

As we have seen in the previous sections, the multi-traces are allowed to jump across a multi-screen. In this section we define the spaces to which these jumps belong as duals of the single-trace spaces.

Definition 5 (Jump Spaces). *We introduce respectively the Dirichlet and Neumann jump spaces as the dual spaces*

$$\begin{aligned} \tilde{H}^{+\frac{1}{2}}([\Gamma]) &= (H^{-\frac{1}{2}}([\Gamma]))', \\ \tilde{H}^{-\frac{1}{2}}([\Gamma]) &= (H^{+\frac{1}{2}}([\Gamma]))'. \end{aligned} \tag{1.7}$$

We endow these spaces with the dual norms

$$\|\varphi\|_{\tilde{H}^{+\frac{1}{2}}([\Gamma])} := \sup_{\substack{\dot{q} \in H^{-\frac{1}{2}}([\Gamma]) \\ \dot{q} \neq 0}} \frac{|\langle \varphi, \dot{q} \rangle|}{\|\dot{q}\|_{H^{-\frac{1}{2}}([\Gamma])}} \quad \text{and} \quad \|\psi\|_{\tilde{H}^{-\frac{1}{2}}([\Gamma])} := \sup_{\substack{\dot{v} \in H^{+\frac{1}{2}}([\Gamma]) \\ \dot{v} \neq 0}} \frac{|\langle \psi, \dot{v} \rangle|}{\|\dot{v}\|_{H^{+\frac{1}{2}}([\Gamma])}}$$

Now we can define jump operators as mappings between multi-trace functions and elements of the jump spaces.

Definition 6. We define continuous jump operators $[\] : \mathbb{H}^{+\frac{1}{2}}(\Gamma) \rightarrow \tilde{H}^{+\frac{1}{2}}([\Gamma])$ and $[\] : \mathbb{H}^{-\frac{1}{2}}(\Gamma) \rightarrow \tilde{H}^{-\frac{1}{2}}([\Gamma])$ as follows: for any $\dot{u} \in \mathbb{H}^{+\frac{1}{2}}(\Gamma)$ (resp. any $\dot{p} \in \mathbb{H}^{-\frac{1}{2}}(\Gamma)$), let $[\dot{u}]$ (resp. $[\dot{p}]$) be the unique element of $\tilde{H}^{+\frac{1}{2}}([\Gamma])$ (resp. $\tilde{H}^{-\frac{1}{2}}([\Gamma])$) satisfying

$$\begin{aligned} \langle [\dot{u}], \dot{q} \rangle &:= \int_{[\Gamma]} \dot{u} \dot{q} \, d\sigma \quad \forall \dot{q} \in H^{-\frac{1}{2}}([\Gamma]), \\ \langle \dot{v}, [\dot{p}] \rangle &:= \int_{[\Gamma]} \dot{v} \dot{p} \, d\sigma \quad \forall \dot{v} \in H^{+\frac{1}{2}}([\Gamma]). \end{aligned} \quad (1.8)$$

where $\langle \cdot, \cdot \rangle$ denotes the duality pairing either between $H^{+\frac{1}{2}}([\Gamma])$ and $\tilde{H}^{-\frac{1}{2}}([\Gamma])$, or between $\tilde{H}^{+\frac{1}{2}}([\Gamma])$ and $H^{-\frac{1}{2}}([\Gamma])$.

We have precise information about the kernels and the ranges of the jump operators. To understand the kernels, we can observe that single-valued traces do not jump so they have to belong to the kernel of the jump operators.

Corollary 3. (Kernels of jump operators, [6, Corollary 6.2]) For $v \in \mathbb{H}^{+\frac{1}{2}}(\Gamma)$ and $q \in \mathbb{H}^{-\frac{1}{2}}(\Gamma)$ it holds

$$\begin{aligned} v \in H^{+\frac{1}{2}}([\Gamma]) &\Leftrightarrow [v] = 0, \\ q \in H^{-\frac{1}{2}}([\Gamma]) &\Leftrightarrow [q] = 0. \end{aligned} \quad (1.9)$$

On the other hand, multi-trace functions must be associated with an element in the jump spaces via the jump operators.

Proposition 3. (Range of jump operators, [6, Proposition 6.7]) The jump operators $[\] : \mathbb{H}^{+\frac{1}{2}}(\Gamma) \rightarrow \tilde{H}^{+\frac{1}{2}}([\Gamma])$ and $[\] : \mathbb{H}^{-\frac{1}{2}}(\Gamma) \rightarrow \tilde{H}^{-\frac{1}{2}}([\Gamma])$ from Definition 6 are surjective.

We conclude this section by giving an interpretation of the newly introduced jump spaces as quotient spaces.

Proposition 4. [6, Proposition 6.8] The jump operators induce isometric isomorphisms

$$\tilde{H}^{+\frac{1}{2}}([\Gamma]) \cong \mathbb{H}^{+\frac{1}{2}}(\Gamma) / H^{+\frac{1}{2}}([\Gamma]) \quad \text{and} \quad \tilde{H}^{-\frac{1}{2}}([\Gamma]) \cong \mathbb{H}^{-\frac{1}{2}}(\Gamma) / H^{-\frac{1}{2}}([\Gamma])$$

1.2 Boundary Integral Equations for acoustic scattering

Throughout this whole thesis we will consider time-harmonic fields. In the scalar case they are of the form

$$\tilde{u}(\mathbf{x}, t) = u(\mathbf{x})e^{i\omega t}.$$

We consider the *acoustic wave equation*

$$\Delta_{\mathbf{x}} \tilde{u}(\mathbf{x}, t) - \frac{1}{c^2} \frac{\partial^2}{\partial t^2} \tilde{u}(\mathbf{x}, t) = 0,$$

with c being the speed of light, and, via the harmonic time dependence, we obtain the well known *Helmholtz equation*

$$-\Delta u(\mathbf{x}) - \kappa^2 u(\mathbf{x}) = 0, \quad (1.10)$$

in which $\kappa = \frac{\omega}{c}$ is the wavenumber. As usual, we can build BVPs by prescribing a Dirichlet or a Neumann data to (1.10). This requires the formal definition of the right Dirichlet and Neumann traces. Given that we deal with exterior problems, we will require the solution to satisfy suitable decay conditions. In the following, two more function spaces will be of use:

- $H^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma}) = \{u \in H^1(\mathbb{R} \setminus \bar{\Gamma}) : \nabla u \in \mathbf{H}(\text{div}, \mathbb{R} \setminus \bar{\Gamma})\}$
- $H_{loc}^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma}) = \{u \in L_{loc}^2(\mathbb{R}^d) : \varphi u \in H^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma}) \forall \varphi \in \mathcal{D}(\mathbb{R}^d \setminus \bar{\Gamma})\}$

1.2.1 Trace Operators

The first trace operator we introduce is the Dirichlet trace, which corresponds to the restriction of a function to the screen Γ :

$$\gamma_D(u) = \pi_D(u),$$

where π_D is the Dirichlet canonical surjection we defined in (1.4).

We have that $\gamma_D : H_{loc}^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma}) \rightarrow \mathbb{H}^{+\frac{1}{2}}(\Gamma)$ is continuous and if $u, v \in H_{loc}^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma})$ coincide in a neighbourhood of Γ then $\gamma_D(u) = \gamma_D(v)$, as pointed out in [6, section 7.1].

The second trace operator we consider is the Neumann trace operator, defined in the following way:

$$\gamma_N(u) = \pi_N(\nabla u),$$

where π_N is the Neumann canonical surjection from (1.4).

The Neumann trace operator has the same properties as the Dirichlet trace: $\gamma_N : H_{loc}^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma}) \rightarrow \mathbb{H}^{-\frac{1}{2}}(\Gamma)$ is continuous and if $u, v \in H_{loc}^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma})$ coincide in a neighbourhood of Γ then $\gamma_N(u) = \gamma_N(v)$ (see again [6, section 7.1]).

Together with the trace operators we can define their formal adjoints $\gamma'_D : \mathbb{H}^{-\frac{1}{2}} \rightarrow H_{loc}^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma})$ and $\gamma'_N : \mathbb{H}^{+\frac{1}{2}} \rightarrow H_{loc}^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma})$

$$\begin{aligned} \langle \gamma'_D \dot{p}, \varphi \rangle &= \ll \dot{p}, \gamma_D \varphi \gg \quad \forall \varphi \in \mathcal{D}(\mathbb{R}^d), \\ \langle \gamma'_N \dot{v}, \varphi \rangle &= \ll \dot{v}, \gamma_D \varphi \gg \quad \forall \varphi \in \mathcal{D}(\mathbb{R}^d). \end{aligned}$$

1.2.2 Layer Potentials and Representation Formula

The goal of this section is to find a representation formula for the solutions of our BVPs, as it is usually done in BEM. In order to do this, we first need to introduce two layer potentials called respectively the *single layer potential* and the *double layer potential*:

$$\begin{aligned} \text{SL}_\kappa(\dot{q})(\mathbf{x}) &= \mathcal{G}_\kappa * \gamma'_D(\dot{q}) = \int_{[\Gamma]} \gamma_D(\mathcal{G}_{\kappa, \mathbf{x}}) \dot{q} \, d\sigma \quad \forall \dot{q} \in \mathbb{H}^{-\frac{1}{2}}(\Gamma), \\ \text{DL}_\kappa(\dot{v})(\mathbf{x}) &= \mathcal{G}_\kappa * \gamma'_N(\dot{v}) = \int_{[\Gamma]} \gamma_N(\mathcal{G}_{\kappa, \mathbf{x}}) \dot{v} \, d\sigma \quad \forall \dot{v} \in \mathbb{H}^{+\frac{1}{2}}(\Gamma). \end{aligned}$$

where the convolution refers to a " $\int_{[\Gamma]}$ " integral and

$$\mathcal{G}_{\kappa, \mathbf{x}}(\mathbf{y}) = \mathcal{G}_{\kappa}(\mathbf{x} - \mathbf{y})$$

with \mathcal{G}_{κ} being the radiating fundamental solution of the Helmholtz equation:

$$\mathcal{G}_{\kappa}(\mathbf{x}) = \begin{cases} \frac{i}{4} \mathcal{H}_0^1(\kappa|\mathbf{x}|) & \text{if } d = 2, \\ \frac{e^{i\kappa x}}{4\pi x} & \text{if } d = 3, \end{cases}$$

where \mathcal{H}_0^1 is the zeroth order Hankel function of the first kind.

Before giving the representation formula we recall some important properties of the layer potentials. First of all, we have a continuity result.

Proposition 5 ([6, Propositions 8.3 and 8.4]). *The operator SL_{κ} continuously maps $\mathbb{H}^{-\frac{1}{2}}(\Gamma) \rightarrow$ into $H_{\text{loc}}^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma}) \cap H_{\text{loc}}^1(\mathbb{R}^d)$ and the operator DL_{κ} continuously maps $\mathbb{H}^{+\frac{1}{2}}(\Gamma) \rightarrow$ into $H_{\text{loc}}^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma})$.*

In a similar manner as the standard layer potentials, also the layer potentials for multi-screens satisfy some jump conditions, we summarize them in the following propositions.

Proposition 6 ([6, Proposition 8.5]).

$$\begin{aligned} [\gamma_D] \cdot DL_{\kappa}(\dot{u}) &= [\dot{u}] & [\gamma_N] \cdot DL_{\kappa}(\dot{u}) &= 0 & \forall \dot{u} \in \mathbb{H}^{+\frac{1}{2}}(\Gamma) \\ [\gamma_D] \cdot SL_{\kappa}(\dot{p}) &= 0 & [\gamma_N] \cdot SL_{\kappa}(\dot{p}) &= [\dot{p}] & \forall \dot{p} \in \mathbb{H}^{-\frac{1}{2}}(\Gamma) \end{aligned}$$

Interestingly, on the right-hand sides of the jump relations for multi-screens we find $[\dot{u}]$ and $[\dot{p}]$ instead of \dot{u} and \dot{p} as we would have with the standard geometries. Another important property of the layer potentials for multi-screens is that they are not injective when understood as mapping between multi-trace spaces:

Lemma 1 ([6, Lemma 8.6]). *We have $SL_{\kappa}\dot{p} = 0 \forall \dot{p} \in H^{-\frac{1}{2}}([\Gamma])$ and $DL_{\kappa}\dot{v} = 0 \forall \dot{v} \in H^{\frac{1}{2}}([\Gamma])$.*

As anticipated, solutions of the Helmholtz equation satisfy a representation formula in which the layer potentials play a key role, as shown in the following proposition.

Proposition 7 ([6, Lemma 8.1]). *For any $u \in H^1(\Delta, \mathbb{R}^d \setminus \bar{\Gamma})$ with bounded support, if $f = -\Delta u - \kappa^2 u$ in the sense of distributions in $\mathbb{R}^d \setminus \bar{\Gamma}$, we have the following formula*

$$u = \mathcal{G}_{\kappa} * f + SL_{\kappa} \cdot \gamma_N(u) + DL_{\kappa} \cdot \gamma_D(u) \quad \text{in } \mathbb{R}^d \setminus \Gamma.$$

1.2.3 BIOs and BIEs

Now that we have a representation formula we are able to derive BIEs by applying the trace operators to it. Let us first consider the exterior Helmholtz-Dirichlet BVP

$$\begin{cases} -\Delta u(\mathbf{x}) - \kappa^2 u(\mathbf{x}) = 0 & \text{in } \mathbb{R}^d \setminus \bar{\Gamma}, \\ \gamma_D u = g_D & \text{on } \Gamma, \\ \lim_{r \rightarrow \infty} r \left(\frac{\partial u}{\partial r}(\mathbf{x}) - i\kappa u(\mathbf{x}) \right) = 0, & r = \|\mathbf{x}\|. \end{cases}$$

If we plug the known Dirichlet trace in the representation formula, apply the Neumann trace and rearrange, we find

$$\gamma_D \cdot SL_\kappa \cdot \gamma_N(u) = g := g_D - \gamma_D DL_\kappa g_D.$$

Now we can define the *weakly singular* BIO as

$$V = \gamma_D \cdot SL_\kappa,$$

and we can find the unknown Neumann trace $\gamma_N(u) \in \mathbb{H}^{-\frac{1}{2}}(\Gamma)$ by solving

$$V\gamma_N(u) = g. \quad (1.11)$$

Then, by plugging $\gamma_N u$ into the representation formula it is possible to retrieve u in the whole space. The derived BIE can be cast in variational form as follows

$$\text{Find } \phi = \gamma_N u \in \mathbb{H}^{-\frac{1}{2}}(\Gamma) \text{ such that } \ll V\phi, \psi \gg = \ll g, \psi \gg \quad \forall \psi \in \mathbb{H}^{-\frac{1}{2}}(\Gamma). \quad (1.12)$$

Formulation (1.12) can be discretized using, for example, a Galerkin approach. We can proceed in a similar way for the Helmholtz-Neumann BVP

$$\begin{cases} -\Delta u(\mathbf{x}) - \kappa^2 u(\mathbf{x}) = 0 & \text{in } \mathbb{R}^d \setminus \bar{\Gamma}, \\ \gamma_N u = h_N & \text{on } \Gamma, \\ \lim_{r \rightarrow \infty} r \left(\frac{\partial u}{\partial r}(\mathbf{x}) - i\kappa u(\mathbf{x}) \right) = 0, & r = \|\mathbf{x}\|. \end{cases}$$

Indeed, if we take the Neumann trace of the representation formula and rearrange in the same way as before, we find

$$\gamma_N \cdot DL_\kappa \cdot \gamma_D(u) = h := h_N - \gamma_N SL_\kappa h_N.$$

Thus, we define the *hypersingular* BIO as

$$W = \gamma_N \cdot DL_\kappa.$$

Now the unknown Dirichlet trace $\gamma_D(u) \in \mathbb{H}^{\frac{1}{2}}(\Gamma)$ can be found by solving

$$W\gamma_D(u) = h. \quad (1.13)$$

Also this BIE can be cast in variational form and it results in the problem

$$\text{Find } v = \gamma_D u \in \mathbb{H}^{+\frac{1}{2}}(\Gamma) \text{ such that } \ll Wv, p \gg = \ll h, p \gg \quad \forall p \in \mathbb{H}^{-\frac{1}{2}}(\Gamma). \quad (1.14)$$

It would be desirable to have existence and uniqueness of the solutions of (1.12) and (1.14). Unfortunately, this can be proved only if we restrict the BIOs to map jump spaces into single trace spaces as stated in the following proposition.

Proposition 8 ([6, Proposition 8.8]). *For any wavenumber $\kappa \in \mathbb{C} \setminus \{0\}$ such that $\text{Im}\kappa \geq 0$ let $V : \tilde{H}^{-\frac{1}{2}}([\Gamma]) \rightarrow H^{+\frac{1}{2}}([\Gamma])$ and $W : \tilde{H}^{+\frac{1}{2}}([\Gamma]) \rightarrow H^{-\frac{1}{2}}([\Gamma])$ denote the weakly singular and hypersingular BIOs respectively. Then there exist compact operators $K_V : \tilde{H}^{-\frac{1}{2}}([\Gamma]) \rightarrow H^{+\frac{1}{2}}([\Gamma])$ and $K_W : \tilde{H}^{+\frac{1}{2}}([\Gamma]) \rightarrow H^{-\frac{1}{2}}([\Gamma])$ such that the following generalized Gårding*

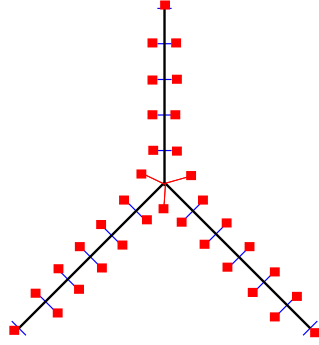


FIGURE 1.2: Multi-trace mesh on an infinitesimally thick screen

inequalities are satisfied

$$\operatorname{Re} \left\{ \int_{[\Gamma]} q(V + K_V) \bar{q} \, d\sigma \right\} \geq C \|q\|_{\tilde{H}^{-\frac{1}{2}}([\Gamma])}^2 \quad \forall q \in \tilde{H}^{-\frac{1}{2}}([\Gamma])$$

$$\operatorname{Re} \left\{ \int_{[\Gamma]} v(W + K_W) \bar{v} \, d\sigma \right\} \geq C \|v\|_{\tilde{H}^{+\frac{1}{2}}([\Gamma])}^2 \quad \forall v \in \tilde{H}^{+\frac{1}{2}}([\Gamma])$$

Thanks to proposition 4 it can be seen that the solutions of the BIEs posed on the multi-trace spaces exists but they are not unique

1.3 Galerkin Discretization of the Boundary Integral Equations

In this section we will show how to apply the Galerkin approach to the variational BIEs (1.12) and (1.14).

The discretization of the multi-trace spaces is possible as we have a concrete intuition about those spaces and the duality pairing between them. Indeed, recalling observation 1 and the fact that the trace spaces must allow a trace to have different values depending on which side of the screen is considered, we can imagine to have an infinitesimally thick screen and mesh its surface so that we get overlapping nodes and elements on opposite sides of the screen like in figure 1.2. Then we can build Boundary Element Spaces (BES) for the multi-trace spaces in the same way as we would build BES for the standard trace spaces on the boundary of a Lipschitz domain, which in our case is the infinitesimally thick screen.

Having chosen this strategy to build the BES, we first need to formally define 2D and 3D meshes \mathcal{G} on Lipschitz polygons or polyhedrons. Then, we can approximate the trace spaces on Γ using lowest order Lagrangian BES defined on \mathcal{G} . Finally, we derive the linear systems of equations which arise from the discretization.

1.3.1 Meshes on an closed Polygon

If Γ is a polygon, i.e. a piecewise straight curve, it can be partitioned as

$$\Gamma = \bar{\Gamma}_1 \cup \dots \cup \bar{\Gamma}_M \quad \text{with } \Gamma_i \cap \Gamma_j = \emptyset \quad \text{for } i \neq j.$$

where $\Gamma_j, j = 1, \dots, M$ are the edges of Γ with C^2 -parametrizations

$$\alpha_j : [-1, 1] \rightarrow \bar{\Gamma}_j, \quad j = 1, \dots, M$$

$$\alpha_j(1) = \alpha_{j+1}(-1), \quad j = 1, \dots, M-1 \quad \alpha_1(-1) = \alpha_M(1).$$

Now we are ready to define a mesh on a polygon.

Definition 7 (Mesh on a an Closed Polygon). *A mesh \mathcal{G} on a polygon is a decomposition*

$$\Gamma = \bigcup_{j=1}^M \bigcup_{i=1}^{N_j} \bar{\tau}_i^{(j)}, \quad \tau_i^{(j)} = \alpha_j([\xi_{i-1}^{(j)}, \xi_i^{(j)}]), \quad i = 1, \dots, N_j, N_j \in \mathbb{N}, j = 1, \dots, M$$

induced by grids of the parameter intervals $[-1, 1]$:

$$-1 =: \xi_0^{(j)} < \xi_1^{(j)} < \dots < \xi_{N_j-1}^{(j)} < \xi_{N_j}^{(j)} := 1$$

1.3.2 Triangular Meshes on Polyhedron

Suppose that Γ is a closed polyhedron which can be partitioned as

$$\Gamma = \bar{\Gamma}_1 \cup \dots \cup \bar{\Gamma}_N \quad \text{with } \Gamma_i \cap \Gamma_j = \emptyset \quad \text{for } i \neq j.$$

Each Γ_i has a C^2 -parameterization

$$\alpha_i : \Pi_i \rightarrow \Gamma_i,$$

where Π_i is a planar polygon. Now we can define a triangulation on a polygon in \mathbb{R}^2 .

Definition 8 (Triangulation of a Polygon). *A triangulation \mathcal{M} of a polygon $\Pi \subset \mathbb{R}^2$ is a finite collection $\{K_i\}_{i=1}^N$, $N \in \mathbb{N}$ of open non-degenerate triangles such that*

- $\bar{\Pi} = \bigcup \{\bar{K}_i, i = 1, \dots, M\}$,
- $K_i \cap K_j = \emptyset \leftrightarrow i \neq j$
- for all $i, j \in \{1, \dots, M\}$, $i \neq j$, the intersection $\bar{K}_i \cap \bar{K}_j$ is either empty or a vertex or edge both K_i and K_j

Now we are ready to define a triangular surface mesh

Definition 9 (Triangular Surface Mesh on a closed Polyhedron). *A triangular surface mesh \mathcal{G} is a partitioning*

$$\Gamma = \bar{\tau}_1 \cup \dots \cup \bar{\tau}_N, \quad \tau_i \cap \tau_j = \emptyset \quad \text{for } i \neq j,$$

such that

- every panel τ_j is contained in exactly one face Γ_i ,
- the pre-images of the panels contained in Γ_j under the parameterization α_j form a triangulation \mathcal{M}_j of Π_j according to previous definition.
- for all $\tau_i, \tau_j \in \mathcal{T}(\mathcal{G})$ the intersections $\bar{\tau}_i \cap \bar{\tau}_j$ are either empty, a common vertex, or a face of both panels.

In this definition and throughout the rest of this work we indicate with:

- $\mathcal{V}(\mathcal{G})$ the sets of vertices of the mesh \mathcal{G} with cardinality N_V
- $\mathcal{T}(\mathcal{G})$ the sets of panels of the mesh \mathcal{G} with cardinality N_T
- $\mathcal{E}(\mathcal{G})$ the sets of edges of the mesh \mathcal{G} with cardinality N_E

1.3.3 The Boundary Element Spaces

The strategy to build discrete Boundary Element spaces on the boundary is now to define classical finite element spaces on the domain of the parameterizations and to transfer them via pullback to Γ . The pullback of a function f is defined as $\alpha^* f = f \circ \alpha$.

Definition 10 (Lagrangian Boundary Element Spaces). *Let \mathcal{G} be a triangular mesh as defined in 9, we define the piecewise polynomial Boundary Element spaces as*

$$\mathcal{S}_p^0(\mathcal{G}) := \{v \in C^0(\Gamma) : \alpha_j^*(v|_\tau) \in \mathcal{P}_p(\mathbb{R}^q), \forall \tau \in \mathcal{G}, \tau \subset \Gamma_j, j = 1, \dots, M\}, p \geq 1,$$

$$\mathcal{S}_p^{-1}(\mathcal{G}) := \{v \in L^2(\Gamma) : \alpha_j^*(v|_\epsilon) \in \mathcal{P}_p(\mathbb{R}^q), \forall \tau \in \mathcal{G}, \tau \subset \Gamma_j, j = 1, \dots, M\}, p \geq 0,$$

where $q = 1$ for meshes on curves and $q = 2$ for meshes on surfaces.

Where the spaces \mathcal{P}_p are the spaces of polynomials of degree p . Since we carry out a lowest order Lagrangian discretization, we discretize $\mathbb{H}^{+\frac{1}{2}}(\Gamma)$ with $\mathcal{S}_1^0(\mathcal{G})$, while we discretize $\mathbb{H}^{-\frac{1}{2}}(\Gamma)$ with $\mathcal{S}_0^{-1}(\mathcal{G})$.

Keeping in mind the goal of Galerkin discretization we need to choose a basis for each of \mathcal{S}_1^0 and \mathcal{S}_0^{-1} . For the first we choose the nodal basis composed by tent functions. For each vertex $\mathbf{p}_i \in \mathcal{V}(\mathcal{G})$ we define the basis function

$$b_{N_V}^i \in \mathcal{S}_1^0(\mathcal{G}),$$

$$b_{N_V}^i(\mathbf{x}) = \begin{cases} \frac{1}{\Delta_i} & \text{if } \mathbf{x} = \mathbf{p}_i, \\ 0 & \text{if } \mathbf{x} \in \mathcal{V}(\mathcal{G}) \setminus \{\mathbf{p}_i\}, \end{cases}$$

where N_V is the number of vertices of the mesh and Δ_i is the Gram determinant when Γ is a polyhedron, or simply the panel length when Γ is a polygon. On the other hand, for \mathcal{S}_0^{-1} we chose characteristic functions of the panels. For each panel ϵ_i we define

$$\beta_{N_E}^i \in \mathcal{S}_0^{-1}(\mathcal{G}),$$

$$\beta_{N_E}^i = \begin{cases} \frac{1}{\Delta_i} & \text{if } \mathbf{x} \in \epsilon_i, \\ 0 & \text{elsewhere on } \Gamma. \end{cases}$$

Where N_E is the number of elements in the mesh.

In practice, the basis functions are built as usual parametrically via pullback using the parameterizations α_j .

1.3.4 Linear System Construction

Having the boundary element spaces and their basis functions, we are finally ready to build the linear systems of equations which arise from the variational formulations 1.12 and 1.14.

The Weakly Singular Operator System

In this paragraph we aim at discretizing the variational formulation (1.12). As we already anticipated, we approximate $\mathbb{H}^{-\frac{1}{2}}(\Gamma)$ with $\mathcal{S}_0^{-1}(\mathcal{G})$. Therefore, any function $\phi \in \mathbb{H}^{-\frac{1}{2}}(\Gamma)$ can be approximated as

$$\phi \approx \sum_{i=1}^{N_E} \phi_i \beta_{N_E}^i.$$

Every function ϕ we can now be associated to a vector of Degrees of Freedom (DoFs) $\vec{\phi} = (\phi_1, \dots, \phi_{N_E})^T$. On the other hand the function $g \in \mathbb{H}^{+\frac{1}{2}}(\Gamma)$ is approximated using $\mathcal{S}_1^0(\mathcal{G})$ as

$$g \approx \sum_{i=1}^{N_V} g_i b_{N_V}^i.$$

and it is associated to a DoF vector $\vec{g} = (g_1, \dots, g_{N_V})^T$. As a result, using the bilinearity properties of the involved operators, (1.12) becomes

$$\text{Find } \vec{\phi} \in \mathbb{R}^{N_E} \text{ such that } \sum_{i=1}^{N_E} \ll V \beta_{N_E}^i, \beta_{N_E}^j \gg \phi_i = \sum_{i=1}^{N_V} \ll b_{N_V}^i, \beta_{N_E}^j \gg g_i \quad \forall j \in \{1, \dots, N_E\}. \quad (1.15)$$

If now we define the matrices

$$\begin{aligned} [\mathbf{V}]_{i,j} &= \ll V \beta_{N_E}^i, \beta_{N_E}^j \gg \in \mathbb{R}^{N_E \times N_E}, \\ [\mathbf{M}]_{i,j} &= \ll b_{N_V}^i, \beta_{N_E}^j \gg \in \mathbb{R}^{N_V \times N_E}, \end{aligned} \quad (1.16)$$

formulation (1.15) can be compactly rewritten as

$$\text{Find } \vec{\phi} \text{ such that } \mathbf{V} \vec{\phi} = \vec{r}_V,$$

with $\vec{r}_V = \mathbf{M} \vec{g}$

The Hyper-Singular Operator System

We now take care of the variational formulation (1.14). Here we discretize the spaces $\mathbb{H}^{+\frac{1}{2}}(\Gamma)$ with $\mathcal{S}_1^0(\mathcal{G})$: a function $v \in \mathbb{H}^{+\frac{1}{2}}(\Gamma)$ can be approximated as

$$v \approx \sum_{i=1}^{N_V} v_i b_{N_V}^i,$$

and it is associated to a DoF vector $\vec{v} = \{v_1, \dots, v_{N_V}\}$. Instead the right-hand side function $h \in \mathbb{H}^{-\frac{1}{2}}(\Gamma)$ approximated with piecewise constants as

$$h \approx \sum_{i=1}^{N_E} h_i \beta_{N_E}^i,$$

and it is again associated with a DoF vector $\vec{h} = \{h_1, \dots, h_{N_E}\}$. Using once more the bilinearity of the operators, (1.14) becomes

$$\text{Find } \vec{v} \in \mathbb{R}^{N_V} \text{ such that } \sum_{i=1}^{N_V} \ll W b_{N_V}^i, b_{N_V}^j \gg v_i = \sum_{i=1}^{N_E} \ll \beta_{N_E}^i, b_{N_V}^j \gg h_i \quad \forall j \in \{1, \dots, N_V\}. \quad (1.17)$$

Now, by defining the matrix

$$[\mathbf{W}]_{i,j} = \ll W b_{N_V}^i, b_{N_V}^j \gg \in \mathbb{R}^{N_V \times N_V}, \quad (1.18)$$

the (1.17) can be rewritten as

$$\text{Find } \vec{h} \text{ such that } \mathbf{W} \vec{h} = \vec{r}_W,$$

with $r_{\mathcal{W}}^{\vec{h}} = \mathbf{M}^T \vec{h}$.

1.3.5 The Variational Formulation of the Hypersingular Operator

When written in its integral form, the hypersingular operator cannot be used in practice to compute the integrals (1.18) as they involve non-integrable kernels. Fortunately, through integration by parts it is possible to manipulate the hypersingular operator in variational form so that its kernel becomes integrable. The next formulae we report are rigorously proved in [12, Section 6.5]. In two dimensions we can write:

$$\ll Wb_{N_V}^i, b_{N_V}^j \gg = -\frac{1}{2\pi} \int_{[\Gamma]} \int_{[\Gamma]} \log(\|\mathbf{x} - \mathbf{y}\|) \frac{db_{N_V}^i}{ds} \frac{db_{N_V}^j}{ds} d\sigma(\mathbf{x}) d\sigma(\mathbf{y}),$$

where $\frac{d}{ds}$ indicates the arclength derivative. On the other hand, in three dimensions we have

$$\ll Wb_{N_V}^i, b_{N_V}^j \gg = \frac{1}{4\pi} \int_{[\Gamma]} \int_{[\Gamma]} \frac{1}{\|\mathbf{x} - \mathbf{y}\|} (\mathbf{grad}_{\Gamma} b_{N_V}^i \times \mathbf{n}) \cdot (\mathbf{grad}_{\Gamma} b_{N_V}^j \times \mathbf{n}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}).$$

Chapter 2

2D Laplace Equation

In this chapter we apply the ideas from chapter 1 to two-dimensional Laplace screen problems. In this case, the multi-screen is simply an open curve which is allowed to have junctions. The Laplace-Neumann and Laplace-Dirichlet problems are respectively

$$\left\{ \begin{array}{l} \Delta u = 0 \quad \text{in } \Omega \\ \gamma_D(u) = g \quad \text{on } \Gamma \end{array} \right. \quad \text{and} \quad \left\{ \begin{array}{l} \Delta u = 0 \quad \text{in } \Omega \\ \gamma_N(u) = h \quad \text{on } \Gamma \end{array} \right.$$

In the same way as it has been done for the Helmholtz equation in chapter 1, it can be shown that the problems are equivalent to the variational BIEs (??) and (??) with $\kappa = 0$).

We analyze 3 different geometries for the numerical tests: a segment screen (figure 2.1a), a triple junction (figure 2.1b) and a quadruple junction (figure 2.1c).

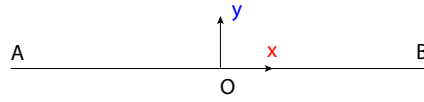
We first investigate the kernels of the discretized BIEs, which will turn out to be non-empty, (section 2.1). In section 2.2 we empirically show that iterative solvers, such as Conjugate Gradient (CG) method and Generalized Minimal Residual (GMRES) method, are still able to solve the linear systems despite the non-empty kernels, by virtue to the structure of the involved matrices. Moreover, we validate our code in the segment case using some known analytical solutions (section 2.3). To conclude, we show how the kernels can be suppressed by incorporating into the problems the knowledge we have about the multi-trace spaces (section 2.4). The codes used in this chapter are briefly discussed in Appendix A.¹

2.1 Tests for the Kernel Dimensions

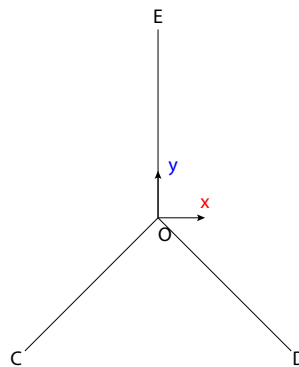
The matrices corresponding to the discrete versions of the weakly- and hyper-singular BIEs are supposed to become singular when the thickness of the screen tends to zero. From an abstract theoretical point of view, the singularity of the discrete BIEs is inherited from the single and double layer potentials (recall lemma 1). From a practical point of view, when the thickness of screen is zero, some of mesh elements and vertices coincide. Therefore, we have that the rows and columns corresponding to these DoFs couples are exactly equal, as we will prove in the next section. Accordingly, in the following we verify that kernel dimensions of the Galerkin matrices corresponding to the two BIEs match the expectations.

In MATLAB the kernel dimension of a matrix \mathbf{A} can be obtained using the commands `size(null(A),2)`. Another option would be to check how many eigenvalues of the matrices are numerically null (of order of 10^{-14} or smaller). This number gives the kernel dimension. Figure 2.2 shows, for example the eigenvalues of \mathbf{V} and \mathbf{W} in the segment case for different system sizes. In the pictures it can be clearly seen

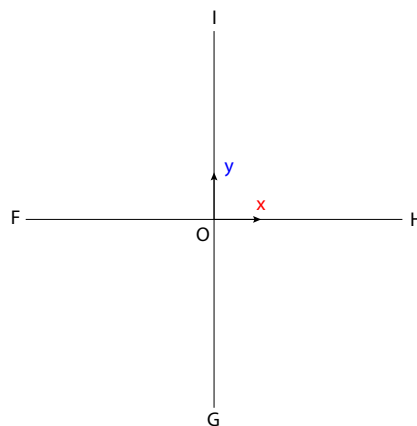
¹For all the implementations of this chapter we used MATLAB R2017b



(A) Segment screen.
Coordinates: $A = (-1, 0)^T$, $B = (1, 0)^T$



(B) Triple screen.
Coordinates:
 $C = (-\sqrt{2}/2, -\sqrt{2}/2)^T$,
 $D = (\sqrt{2}/2, -\sqrt{2}/2)^T$,
 $E = (0, 1)^T$



(C) Quadruple screen.
Coordinates:
 $F = (-1, 0)^T$, $G = (0, -1)^T$,
 $H = (1, 0)^T$, $I = (0, 1)^T$

FIGURE 2.1: The screens used in our numerical simulations.

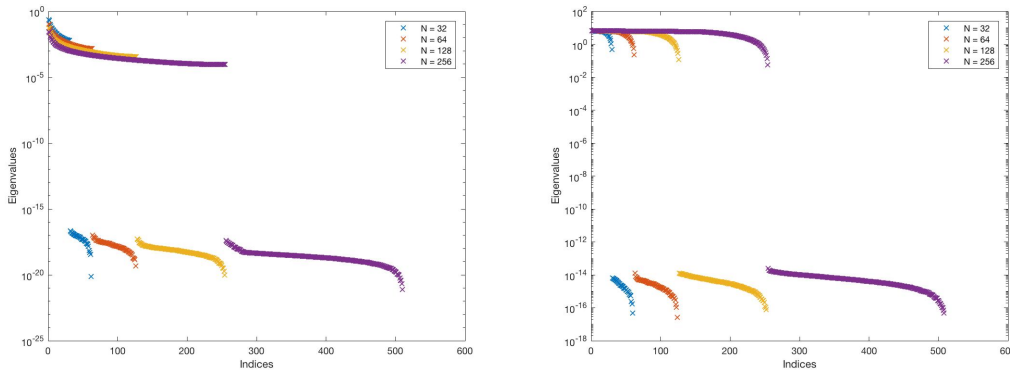


FIGURE 2.2: Eigenvalues of \mathbf{V} (left) and \mathbf{W} (right) on a hierarchy of nested meshes.

that in each case half of the eigenvalues are numerically negligible with respect to the others.

2.1.1 Kernel of the Weakly Singular operator

In the case of the weakly singular operator we have DoFs associated with the elements of the mesh, since we carry out a Galerkin discretization based on piecewise linear basis functions.

As it is shown in figure 2.3a, the mesh elements overlap in couples and the number of overlapping couples is equal to the total number of elements divided by two.

We can prove that each of these couples brings a contribution to the kernel of \mathbf{V} . If we consider two overlapping elements $e_i \equiv e_j$, then for any other element e_k we have

$$\begin{aligned} V_{ki} &= -\frac{1}{2\pi} \int_{e_k} \int_{e_i} \log \frac{1}{\|\mathbf{x}-\mathbf{y}\|} \frac{1}{|e_k|} \frac{1}{|e_i|} d\sigma(\mathbf{x})d\sigma(\mathbf{y}) = \\ &= -\frac{1}{2\pi} \int_{e_k} \int_{e_j} \log \frac{1}{\|\mathbf{x}-\mathbf{y}\|} \frac{1}{|e_k|} \frac{1}{|e_j|} d\sigma(\mathbf{x})d\sigma(\mathbf{y}) = V_{kj} \end{aligned} \quad (2.1)$$

The condition $V_{ki} = V_{kj}$ results in a kernel contribution since we can build a vector $\boldsymbol{\phi}^{ij} \in \mathbb{R}^{N_e}$ with

$$\phi_l^{ij} = \begin{cases} 1 & \text{if } l = i \\ -1 & \text{if } l = j \\ 0 & \text{otherwise,} \end{cases} \quad (2.2)$$

and we will have $\mathbf{V}\boldsymbol{\phi}^{ij} = \mathbf{0}$, since, using 2.1, we have by direct computation:

$$\sum_{l=1}^{N_e} V_{kl} \phi_l^{ij} = V_{ki} \phi_i^{ij} + V_{kj} \phi_j^{ij} = 0 \quad \forall k = 1 \dots N_e.$$

Now if we chose any other couple of overlapping elements $e_{i'} \equiv e_{j'}$ we can build a vector $\boldsymbol{\phi}^{i'j'}$ in the same way as in 2.2. Given that $i' \neq i$ and $j' \neq j$, $\boldsymbol{\phi}^{i'j'}$ will always be linearly independent from $\boldsymbol{\phi}^{ij}$, and it will add a further dimension to the kernel of \mathbf{V} . Repeating the same reasoning for every couple of overlapping elements, we observe a new kernel contribution for each of those couples, getting in total $\frac{N_e}{2}$ contributions. Moreover, it seems clear that, in general, for couples of non overlapping elements there is no relation similar to 2.1 therefore those couples do not carry any new contribution to the kernel of \mathbf{V} and the $\boldsymbol{\phi}$ vectors constitute a base for the kernel.

Segment		Triple Junction		Quadruple Junction	
N	$\dim(\ker(\mathbf{V}))$	N	$\dim(\ker(\mathbf{V}))$	N	$\dim(\ker(\mathbf{V}))$
4	2	12	6	16	8
10	5	30	15	40	20
22	11	66	33	88	44
46	23	138	69	184	92
94	47	282	141	376	188

TABLE 2.1: Kernel dimensions of \mathbf{V} when using a mesh with N elements. The kernel dimension is always half as the total number of elements

Observation 2. Each vector $\boldsymbol{\phi}$ corresponds to a Neumann trace whose jump across the screen is null. Moreover, using the basis constituted by the $\boldsymbol{\phi}$ vectors we can only build Neumann traces which do not jump across the screen. Therefore, the kernel of \mathbf{V} is composed by Neumann traces which do not jump across the screen. This practical result confirms the theoretical expectations from [6, Lemma 8.6].

Table 2.1 shows the numerical results of the tests for the kernel size of \mathbf{V} , which confirm our expectations.

2.1.2 Kernel of the Hyper-Singular Operator

For the hyper-singular operator the mesh vertices are the geometric entities associated with the DoFs. Figure 2.3b shows how the vertices overlap when the mesh thickness tends to zero. We observe that the nodes overlap in couples when they are away from the junction, while we have many overlapping vertices at the junction. We divide the analysis of the kernel of \mathbf{W} in three cases: dependencies for couples of overlapping vertices away from the junction, dependencies of nodes overlapping at the junction and nodes at the tips.

Couple of Vertices Away from the Junctions

We will consider a couple of vertices overlapping on a flat region of the screen as in figure 2.4a. First of all, we observe that there is a specific relation for the arc length derivative of overlapping basis functions, namely:

$$\frac{db_i}{ds} = -\frac{db_j}{ds}.$$

By virtue of this formula and the fact that $e_i \equiv e_{j-1}$ $e_j \equiv e_{i-1}$, we find that for any vertex k we have

$$\begin{aligned} W_{ki} &= -\frac{1}{2\pi} \int_{e_k \cup e_{k-1}} \int_{e_i \cup e_{i-1}} \log \frac{1}{\|\mathbf{x}-\mathbf{y}\|} \frac{db_k}{ds} \frac{db_i}{ds} d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\ &= \frac{1}{2\pi} \int_{e_k \cup e_{k-1}} \int_{e_j \cup e_{j-1}} \log \frac{1}{\|\mathbf{x}-\mathbf{y}\|} \frac{db_k}{ds} \frac{db_j}{ds} d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = -W_{kj} \end{aligned} \quad (2.3)$$

Similarly to what we have seen in the previous section, this relation implies that the vector Ψ^{ij} defined as

$$\Psi_l^{ij} = \begin{cases} 1 & \text{if } l = i \text{ or } l = j \\ 0 & \text{otherwise} \end{cases}$$

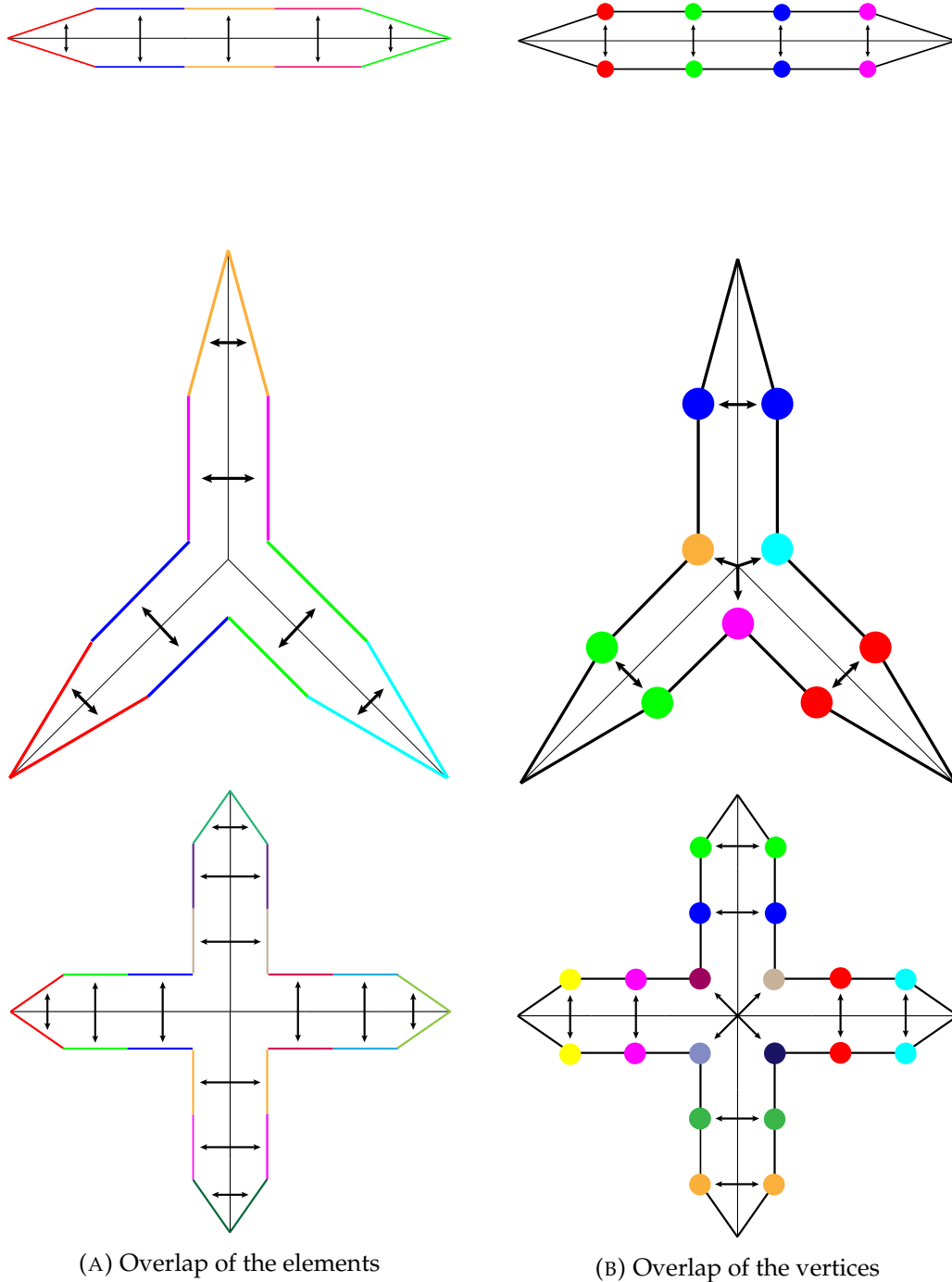
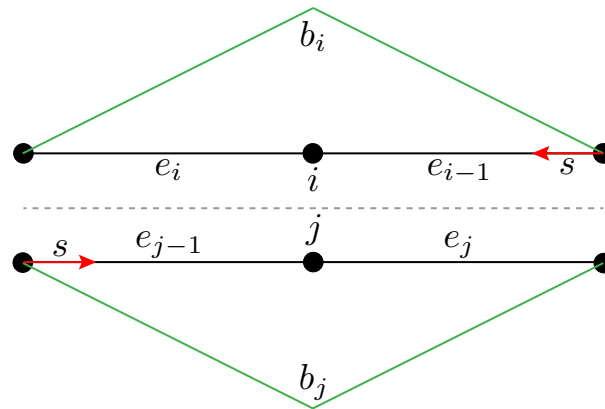
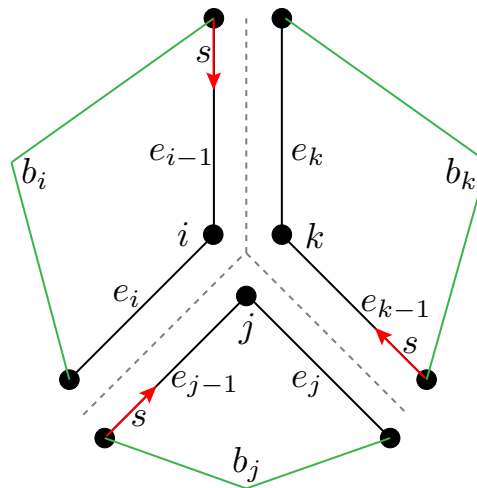


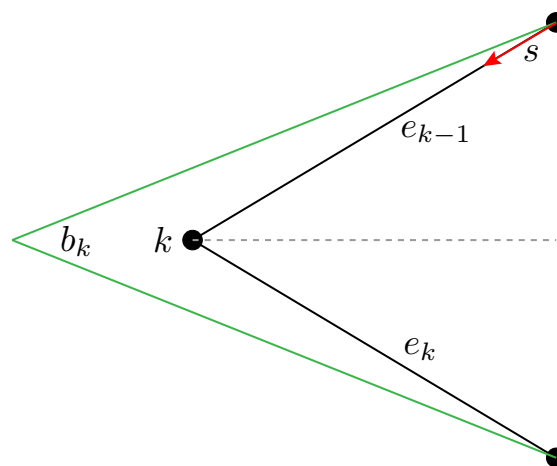
FIGURE 2.3: Multi-trace meshes in which equal DoFs are indicated with the same color



(A) Overlapping vertices away from the junction



(B) Overlapping vertices at the junction



(C) Tip vertex

FIGURE 2.4: Multi-trace meshes in which equal DoFs are indicated with the same color

belongs to the kernel of \mathbf{W} . On the other hand, in the hypersingular case there also other basis functions that contribute to the kernel of \mathbf{W} , therefore the Ψ^{ij} vectors are not enough to form a basis of $\ker W$

Vertices Overlapping at the Junction

As it can be seen in 2.4b for a triple junction, thanks to the opposite orientations of the elements which overlap we have the following relation

$$\frac{db_i}{ds} + \frac{db_j}{ds} + \frac{db_k}{ds} = 0,$$

from which we find easily that

$$W_{zi} + W_{zj} + W_{zk} = 0.$$

For a general n-uple junction we have that

$$\sum_{t=1}^n W_{zt} = 0.$$

We can observe that any vector μ of the form

$$\mu_l = \begin{cases} 1 & \text{if } l \text{ is a junction DoF} \\ 0 & \text{otherwise} \end{cases}$$

lies in the kernel of \mathbf{W} and it is linearly independent from the vectors Ψ^{ij}

Vertices at the Tips

From figure 2.4c we observe that for any tip DoF k we have that $e_k \equiv e_{k-1}$ and, therefore,

$$\begin{aligned} W_{zk} &= -\frac{1}{2\pi} \int_{e_z \cup e_{z-1}} \int_{e_k \cup e_{k-1}} \log \frac{1}{\|\mathbf{x}-\mathbf{y}\|} \frac{db_z}{ds} \frac{db_k}{ds} d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\ &= -\frac{1}{2\pi} \int_{e_z \cup e_{z-1}} \int_{e_k} \log \frac{1}{\|\mathbf{x}-\mathbf{y}\|} \frac{db_z}{ds} \frac{1}{|e_k|} d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) + \\ &\quad -\frac{1}{2\pi} \int_{e_z \cup e_{z-1}} \int_{e_{k-1}} \log \frac{1}{\|\mathbf{x}-\mathbf{y}\|} \frac{db_z}{ds} \left(-\frac{1}{|e_{k-1}|}\right) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = 0, \quad \forall z \in \{1, \dots, N_v\}. \end{aligned} \quad (2.4)$$

As a result, the vectors η^k such that

$$\eta_l^k = \begin{cases} 1 & \text{if } l = i \\ 0 & \text{otherwise} \end{cases}$$

belong to the kernel of \mathbf{W} . Moreover, they are linearly independent from the vectors Ψ^{ij} and μ found in the previous sections.

Observation 3. *The vectors $(\Psi^{ij}, \mu^i, \eta^k)$ form a basis of $\ker(\mathbf{W})$. Therefore the functions in this space are Dirichlet traces which cannot jump across the screen, not even at the junction. This observation complies with [6, Lemma 8.6]*

Given that the tips DoFs lead to rows and columns of \mathbf{W} which are populated only by zeros, we decided to neglect them leaving the associated mesh vertices out of the computations.

Segment		Triple Junction		Quadruple Junction	
N	$\dim(\ker(\mathbf{V}))$	N	$\dim(\ker(\mathbf{V}))$	N	$\dim(\ker(\mathbf{V}))$
2	1	9	4	12	5
8	4	27	13	36	17
20	10	63	31	84	41
44	22	135	69	180	89
92	46	279	139	372	185

TABLE 2.2: Kernel dimensions of \mathbf{W} when using a mesh with N inner DoFs.

We tested the kernel dimensionalities of the \mathbf{W} matrices computed with our numerical codes and we report them in table 2.2. For the segment screen the kernel of \mathbf{W} has number of dimensions equal to half the total number of DoFs. For the triple junction we observe that \mathbf{W} has always $(N - 3)/2$ dimensions, where N is the total number of DoFs, while for the quadruple junction it has $(N - 3)/2$ dimensions.

2.2 Tests for CG and GMRES Iterative Solvers

In this section we show that the CG and GMRES iterative solvers are still able to solve the linear systems that involve \mathbf{V} and \mathbf{W} , although those matrices have non-empty kernels. For each of them we pick a right hand-side vector which belongs to the column space of the matrices and we show that both of the solvers converge to a solution in a number of iterations much smaller than the system size.

As it has been shown in [8] and [1], CG and GMRES are still supposed to converge thanks to the fact that the matrices \mathbf{V} and \mathbf{W} are symmetric and semi-positive definite. Moreover the solution given by those solvers is the same as the one we would get using a pseudo-inverse approach. From the point of view of the convergence of CG and GMRES it seems like the null eigenvalues of \mathbf{V} and \mathbf{W} do not influence the number of iterations needed to converge. For this reason for all the numerical experiments of this thesis we will use an unusual definition of the condition number, defined as the ratio between the highest eigenvalue and the lowest **non-zero** eigenvalue.

2.2.1 CG and GMRES for the Weakly-Singular BIO

To test the behaviour of iterative solvers for the discrete weakly-singular BIO, we solve the system

$$\mathbf{V}\mathbf{x} = \mathbf{r}_V. \quad (2.5)$$

where $\mathbf{r}_V = \mathbf{V}\mathbf{z}$ and \mathbf{z} is a random vector. This choice of the right-hand side vector ensures the solvability of the system. The results show that, as predicted, both CG and GMRES are able to solve the underdetermined system and they perform nicely even when the mesh thickness is exactly zero. In table 2.3 we reported the condition number of \mathbf{V} for different number of elements and varying the mesh thickness. It is interesting to observe that with $\epsilon = 0$ the condition number grows linearly with the number of mesh elements in all the three cases. In the case in which $\epsilon = 0$ many eigenvalues of \mathbf{V} are null and we ignored them computing the condition number as the ratio between the largest singular value and the smallest non-zero eigenvalue. The results show that the condition number of \mathbf{V} increases both when we decrease the mesh thickness and when we increase the number of elements. As predicted

Segment				
$\epsilon \backslash N_e$	32	64	128	256
0.1	7.0035e+1	1.8061e+2	5.4803e+2	1.9051e+3
0.01	2.1623e+2	2.2381e+2	3.0677e+2	6.0394e+2
0.001	3.8852e+3	3.0436e+3	2.5584e+3	2.2446e+3
0.0001	2.5804e+5	1.2999e+5	6.9615e+4	4.3079e+4
0.00001	2.5630e+7	1.2649e+7	6.2879e+6	3.1444e+6
0	3.7019e+1	7.5657e+1	1.5281e+2	3.0706e+2
Triple Junction				
$\epsilon \backslash N_e$	32	64	128	256
0.1	2.7429e+2	7.5706e+2	2.3743e+3	8.7757e+3
0.01	3.0054e+2	4.1453e+2	8.2605e+2	1.6853e+3
0.001	4.0731e+3	3.4149e+3	2.9923e+3	2.9503e+3
0.0001	4.4801e+5	1.0108e+5	5.7493e+4	4.3186e+4
0.00001	1.7123e+7	8.4414e+6	4.2039e+6	2.2845e+6
0	1.0363e+2	2.1083e+2	4.2520e+2	8.5393e+2
Quadruple Junction				
$\epsilon \backslash N_e$	32	64	128	256
0.1	2.3279e+2	8.5931e+2	3.3656e+3	1.3387e+4
0.01	5.1285e+2	7.5803e+2	1.0277e+3	2.0579e+3
0.001	5.2136e+3	4.4840e+3	4.1865e+3	4.4894e+3
0.0001	2.2064e+5	1.1721e+5	7.2541e+4	5.5144e+4
0.00001	2.1490e+7	1.0593e+7	5.2750e+6	2.6640e+6
0	1.4842e+2	3.0190e+2	6.0883e+2	1.2227e+3

TABLE 2.3: Condition number of \mathbf{V} with mesh thickness ϵ and N_e elements for each side of the fins.

by the theory, in our numerical experiments both CG and GMRES have been able to solve system 2.5. The performances of the two methods in terms of number of iterations needed to reach the convergence, with stopping criterion for the relative residual equal to $1e-6$, are shown in tables 2.4 and 2.5.

2.2.2 CG and GMRES for the Hyper-Singular BIO

We now consider the system

$$\mathbf{W}\mathbf{x} = \mathbf{r}_w \quad (2.6)$$

In analogy to the what has been done in the previous section, we take $\mathbf{r}_w = \mathbf{W}\mathbf{z}$ with \mathbf{z} being a random vector. By repeating the same experiments as for \mathbf{V} we obtain the results reported in tables 2.6, 2.7 and 2.8. The stopping criterion for the relative residual is again equal to $1e-6$. Also in this case, when $\epsilon = 0$ the condition number grows linearly with the number of elements.

Observation 4. *Both in the weakly singular and hypersingular cases, when decreasing ϵ the number of iterations to convergence first increases and then decreases. The reason is that both \mathbf{V} and \mathbf{W} have bunch of eigenvalues whose order of magnitude decreases when ϵ becomes smaller, while the other eigenvalues are not affected. When these eigenvalues are small the*

$\varepsilon \backslash N_e$	Segment				Triple Junction				Quadruple Junction			
	32	64	128	256	32	64	128	256	32	64	128	256
0.1	23	32	42	59	39	54	80	109	33	48	70	106
0.01	29	31	36	48	36	44	55	71	29	42	55	82
0.001	35	53	64	73	61	75	84	87	24	33	51	78
0.0001	26	36	60	73	56	70	89	109	22	31	42	58
0.00001	17	23	29	39	27	34	44	103	22	31	40	52
0	17	22	30	37	27	34	46	54	22	31	40	52

TABLE 2.4: Number of iterations for the CG solution of 2.5 with mesh thickness ε and system size and N_e elements for each side of the fins.

$\varepsilon \backslash N$	Segment				Triple Junction				Quadruple Junction			
	32	64	128	256	32	64	128	256	32	64	128	256
0.1	21	27	33	40	31	40	49	56	28	39	50	61
0.01	25	26	29	36	30	33	40	49	26	36	44	58
0.001	28	39	46	49	45	52	55	54	23	30	41	57
0.0001	23	30	36	51	42	41	58	68	21	28	36	45
0.00001	16	21	25	31	23	29	35	42	21	28	34	43
0	16	20	25	25	30	29	35	41	21	28	34	43

TABLE 2.5: Number of iterations for the GMRES solution of 2.5 with mesh thickness ε and N_e elements for each side of the fins.

matrices become ill-conditioned and iterative solvers struggle more to find the solution. On the other hand, if the magnitude of these eigenvalues becomes small enough they become irrelevant and the iterative solvers converge as if these eigenvalues did not exist.

2.3 Validation on a segment

In this section we show the validation results for our implementation of the BIOs when the screen is the horizontal segment $\Gamma = [-1, 1] \times 0$. We compare our numerical solutions to the ones prescribed by analytical formulae as in [14, section 4.1.1-4.1.2] and we compute the errors in the discrete energy norms

$$\|\phi_h\|_V^2 = \phi_h^T \mathbf{V}_h \phi_h, \quad (2.7)$$

$$\|\psi_h\|_W^2 = \psi_h^T \mathbf{W}_h \psi_h, \quad (2.8)$$

where ϕ_h and ψ_h are the DoFs vectors corresponding to ϕ_h and ψ_h respectively and \mathbf{V}_h and \mathbf{W}_h are the matrices encoding the discretized BIOs.

Remark: in this analysis we ignored the multiplicative constant of the fundamental solution of the Laplace equation, but it can be easily plugged back in).

2.3.1 Validation for the Weakly-Singular BIO

For the weakly singular operator we have the analytic formula

$$V\phi(x) = \pi x \quad \text{with} \quad \phi(x) = \frac{x}{\sqrt{1-x^2}},$$

Segment				
$\varepsilon \backslash N_f$	32	64	128	256
0.1	6.4553e+1	1.3921e+2	3.1517e+2	7.1894e+2
0.01	7.5600e+2	1.0988e+3	1.7015e+3	3.0155e+3
0.001	1.9488e+4	2.5552e+4	3.6117e+4	5.4028e+4
0.0001	1.3563e+6	1.2599e+6	1.2857e+6	1.5129e+6
0.00001	1.3477e+8	1.2283e+8	1.1713e+8	1.1466e+8
0	1.3763e+1	2.7969e+1	5.6380e+1	1.1320e+2
Triple Junction				
$\varepsilon \backslash N_f$	32	64	128	256
0.1	2.4482e+2	3.2596e+2	7.6596e+2	1.8223e+4
0.01	1.1912e+3	1.8971e+3	3.3397e+3	6.7633e+3
0.001	2.7537e+4	3.9101e+4	5.8863e+4	9.2623e+4
0.0001	1.3579e+6	1.3992e+6	1.6419e+6	2.2452e+6
0.00001	1.3346e+8	1.2676e+8	1.6419e+8	1.2033e+8
0	3.5654e+1	7.2014e+1	1.4475e+2	2.9022e+2
Quadruple Junction				
$\varepsilon \backslash N_f$	32	64	128	256
0.1	1.4495e+2	3.5456e+2	8.3689e+2	1.9331e+3
0.01	1.2153e+3	1.8430e+2	3.1206e+3	6.2508e+3
0.001	3.0710e+4	4.3156e+4	6.3957e+4	9.8094e+4
0.0001	1.5883e+6	1.5922e+6	1.8559e+6	2.5206e+6
0.00001	3.8214e+7	1.9110e+8	4.5429e+8	1.9708e+8
0	4.9820e+1	1.0114e+2	2.0378e+2	4.0907e+2

TABLE 2.6: Condition number of \mathbf{W} with mesh thickness ε .

$\varepsilon \backslash N_f$	Segment				Triple Junction				Quadruple Junction			
	32	64	128	256	32	64	128	256	32	64	128	256
0.1	23	32	46	66	37	56	83	112	33	48	70	106
0.01	51	66	81	108	84	85	111	157	29	42	55	82
0.001	60	116	251	372	173	289	386	490	24	33	51	78
0.0001	48	109	194	493	75	368	693	1519	22	31	42	58
0.00001	20	40	118	289	21	46	52	125	22	31	40	52
0	14	21	29	40	26	36	47	65	22	31	40	52

TABLE 2.7: Number of iterations for the CG solution of 2.6 with mesh thickness ε and system size N_f .

$\varepsilon \backslash N_f$	Segment				Triple Junction				Quadruple Junction			
	32	64	128	256	32	64	128	256	32	64	128	256
0.1	23	31	45	62	36	54	75	95	40	54	71	103
0.01	51	65	80	104	78	83	110	152	72	97	115	152
0.001	60	119	211	310	147	250	364	479	164	263	377	426
0.0001	51	104	212	396	142	272	526	793	166	337	566	887
0.00001	20	35	70	129	60	98	177	318	49	97	179	321
0	14	21	29	40	60	98	177	318	25	34	46	63

TABLE 2.8: Number of iterations for the GMRES solution of 2.6 with mesh thickness ε .

Single-Trace			Multi-Trace		
N	$\ \phi - \phi_h\ _V$	Order	N	$\ \phi - \phi_h\ _V$	Order
4	7.7816e-1	/	4	1.5563	/
8	5.2962e-1	0.5551	8	1.0592	0.5551
16	3.6619e-1	0.5324	16	7.3238e-1	0.5324
32	2.5520e-1	0.5210	32	5.1039e-1	0.5210
64	1.7853e-1	0.5154	64	3.5706e-1	0.5154
128	1.2513e-1	0.5128	128	2.5026e-1	0.5128
256	8.7777e-2	0.5114	256	1.7555e-1	0.5114
512	6.1600e-2	0.5109	512	1.2320e-1	0.5108
1024	4.3235e-2	0.5107	1024	8.6496e-2	0.5103

TABLE 2.9: Energy norm of the error and order of convergence for the weakly-singular operator on the segment $[-1, 1] \times \{0\}$ with N elements

as it is stated in [14, Section 4.1.1].

For the energy norm of the solution we have

$$\begin{aligned}
\|\phi - \phi_h\|_V^2 &= \langle V(\phi - \phi_h), (\phi - \phi_h) \rangle = \\
&= \langle V\phi, \phi \rangle - \langle V\phi_h, \phi_h \rangle = \\
&= \left\langle \pi x, \frac{x}{\sqrt{1-x^2}} \right\rangle - \langle V\phi_h, \phi_h \rangle = \\
&= \frac{\pi^2}{2} - \boldsymbol{\phi}_h^T \mathbf{V} \boldsymbol{\phi}_h.
\end{aligned}$$

The results for the single-trace approach are summarized in the left part of table 2.9. For the multi-trace approach we carried out the same same simulations with some small changes in the analytical solution. In this case the integration path is twice as long, therefore the error energy norm becomes

$$\|\phi - \phi_h\|_V^2 = 2\pi^2 - \boldsymbol{\phi}_h^T \mathbf{V} \boldsymbol{\phi}_h$$

we summarize the results of the simulations in the right part of table 2.9. For both cases we observe order of convergence $\frac{1}{2}$ due to the singularity of the solution.

Single-Trace			Multi-Trace		
N	$\ \psi - \psi_h\ _W$	Order	N	$\ \psi - \psi_h\ _W$	Order
4	7.7816e-1	/	4	1.5563	/
8	5.2962e-1	0.5551	8	1.0592	0.5551
16	3.6619e-1	0.5324	16	7.3238e-1	0.5324
32	2.5520e-1	0.5210	32	5.1039e-1	0.5210
64	1.7853e-1	0.5154	64	3.5706e-1	0.5154
128	1.2513e-1	0.5128	128	2.5026e-1	0.5128
256	8.7777e-2	0.5114	256	1.7555e-1	0.5114
512	6.1600e-2	0.5109	512	1.2320e-1	0.5108
1024	4.3235e-2	0.5107	1024	8.6496e-2	0.5103

TABLE 2.10: Energy norm of the error and order of convergence for the hyper-singular operator on the segment $[-1, 1] \times \{0\}$ with N elements

2.3.2 Validation for the Hyper-Singular BIO

Also for the hyper-singular BIO we have an analytic formula which gives a reference solution. From [14, Section 4.1.2] we know that

$$W\psi(x) = \pi \quad \text{with} \quad \psi(x) = \sqrt{1-x^2}$$

In the same way as stated before we can compute the error in the energy norm for the single-trace approach as

$$\|\psi - \psi_h\|_W^2 = \pi^2/2 - \boldsymbol{\psi}_h^T \mathbf{W} \boldsymbol{\psi}_h,$$

and for the multi-trace approach as

$$\|\psi - \psi_h\|_W^2 = 2\pi^2 - \boldsymbol{\psi}_h^T \mathbf{W} \boldsymbol{\psi}_h$$

The convergence results are reported in table 2.10. Also here we observe order of convergence $\frac{1}{2}$ as the derivative of the solution is singular in $[-1, 1]$.

2.4 Elimination of the Kernels

It is possible to incorporate the knowledge we have about the relations between the overlapping DoFs into the matrices \mathbf{V} and \mathbf{W} in order to suppress their kernels. In order to do so we create new DoFs vectors (in the following example $\hat{\mathbf{u}}$) which are obtained from the old DoFs (\mathbf{u}) through a linear transformation \mathbf{T} which encodes the prior knowledge about the DoFs.

For example in the weakly-singular operator case it is known that the old DoFs are equal in pairs when they are overlapping. This has to happen by virtue of the fact that the jump of a Dirichlet trace across the screen (away from junctions) has to be null. Therefore, on a segment mesh with N elements the new DoFs must be such that

$$\hat{\mathbf{u}}_i = \mathbf{u}_i = \mathbf{u}_j \quad \text{if } i = j \text{ or } i + j = 2N - 1.$$

Hence, the components of the matrix \mathbf{T} are

$$\mathbf{T}_{ij} = \begin{cases} 1 & \text{if } i = j \text{ or } i + j = 2N - 1 \\ 0 & \text{otherwise} \end{cases}$$

The new DoFs are obtained as

$$\hat{\mathbf{u}} = \mathbf{T}^T \mathbf{u}$$

With this new DoFs vector we also need to modify the Galerkin matrix corresponding to the weakly-singular BIO through the formula

$$\hat{\mathbf{V}} = \mathbf{T}^T \mathbf{V} \mathbf{T}.$$

The matrix $\hat{\mathbf{V}}$ is now non-singular and the system

$$\hat{\mathbf{V}} \hat{\boldsymbol{\phi}} = \hat{\mathbf{g}}$$

can be directly solved.

The solution expressed in the old DoFs system can be then retrieved as

$$\boldsymbol{\phi} = \mathbf{T} \hat{\boldsymbol{\phi}}.$$

When we consider the hyper-singular operator we know that DoFs associated with overlapping vertices have to be opposite in sign. Again this fact has an interpretation related to jumps of traces: the Neumann jump across the screen away from junctions must be null. Accordingly, the entries of \mathbf{T} for a segment are

$$\mathbf{T}_{ij} = \begin{cases} 1 & \text{if } i = j, \\ -1 & \text{if } i + j = 2N - 1, \\ 0 & \text{otherwise.} \end{cases}$$

The same consideration done in the previous case apply and the matrix

$$\hat{\mathbf{W}} = \mathbf{T}^T \mathbf{W} \mathbf{T}$$

is full-ranked.

It is possible to build \mathbf{T} matrices also for the cases with junctions by analogous considerations. In particular for the hyper-singular operator we also need to enforce the constrain 2.1.2 in the \mathbf{T} matrix. In order to do this, we reorder the DoFs to place the junction DoFs at the end of our vectors and matrices. Now, the matrix \mathbf{T} needs to have this block structure:

$$\mathbf{T} = \left[\begin{array}{c|c} \mathbf{T}' & \\ \hline & \mathbf{J} \end{array} \right]$$

Where \mathbf{T}' is analogous to the \mathbf{T} matrix in the segment case, while \mathbf{J} for a triple junction is

$$\mathbf{J} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

	Segment			
	Original		Reduced	
N	CG	GMRES	CG	GMRES
32	17	16	16	16
64	22	20	22	20
128	30	25	29	25
256	38	31	38	31
Triple Junction				
	Original		Reduced	
	N	CG	GMRES	CG
32	26	23	26	23
64	34	29	34	29
128	46	34	46	34
256	54	41	57	41
Quadruple				
	Original		Reduced	
	N	CG	GMRES	CG
32	26	24	26	24
64	34	30	34	30
128	42	35	42	35
256	54	43	54	53

	Segment			
	Original		Reduced	
N	CG	GMRES	CG	GMRES
32	14	14	14	14
64	21	21	21	21
128	28	27	28	27
256	40	38	40	38
Triple Junction				
	Original		Reduced	
	N	CG	GMRES	CG
32	24	24	28	28
64	35	35	35	34
128	47	46	48	47
256	54	60	64	60
Quadruple				
	Original		Reduced	
	N	CG	GMRES	CG
32	24	24	25	25
64	34	34	35	34
128	48	47	49	47
256	64	55	64	53

TABLE 2.11: Comparison of the numbers of iterations to convergence for the weakly singular system (left) and the hypersingular system (right).

and for a quadruple function it is

$$\mathbf{J} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$

Unfortunately, for the discrete hypersingular operator the nodes reduction requires detailed knowledge about the geometric situation at the junction.

An advantage of solving the reduced system is that it is much smaller than the original one, and, therefore, the computational burden of CG and GMRES is considerably smaller. This is because in the reduced case the matrix by vector multiplications require much less floating point operations and each iteration of both the two methods is much faster.

On the other hand, we can see how the null eigenvalues which show up in the non-reduced case do not influence the convergence of the iterative solver. By comparing the numbers of iterations to convergence for the original and the reduced systems, which are reported in table 2.11, it is possible to observe that the numbers of iterations are always really similar, which means that the null eigenvalues are irrelevant for the convergence of CG and GMRES.

Chapter 3

3D Laplace Equation

In this chapter we work with the Laplace equation posed in three dimensions. In principle, this case is not really different from the previous one, but some new difficulties arise. In particular, the treatment of the singular integrands in the numerical quadrature schemes is in general much more difficult and the entries of the BEM matrices have to be computed using the complex schemes from [11, Chapter 5]. These methods involve transformations which introduce small numerical inaccuracies which are different depending on the orientation of the involved elements. This phenomenon is visible in our numerical results and in the case of the hyper-singular operator we have to deal with matrices which are not exactly singular, but still really ill-conditioned. Although these new problems with numerical quadrature, the CG and GMRES iterative solvers are able to compute a solution for the linear systems involved within a reasonably small number of iterations. On the other hand, we are more interested in analysing the possibility of solving the singular linear systems, therefore we correct the matrices to recover the non-empty kernels and study the convergence of GMRES and CG.

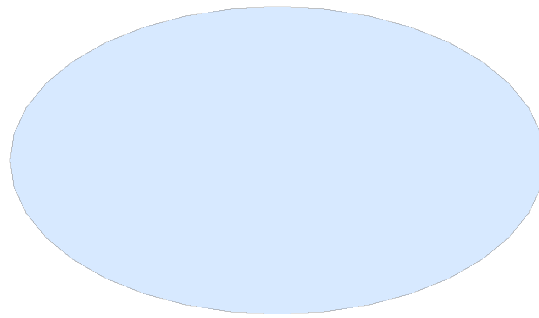
This chapter is organized similarly to the previous one: in section 3.1 we explain the approach used to build suitable multi-trace boundary element spaces, in section 3.2 we analyse the dimension of the kernels of the BEM matrices, in section 3.3 we show that the CG and GMRES solvers are able to solve the linear systems despite the non-empty kernels, finally in section 3.4 we show how prior knowledge about the jumps of the traces can be incorporated in our computations in order to eliminate the kernels of the BEM matrices.

All the numerical experiments of this chapter are carried out using BETL2, a well established C++ library for Boundary Element Method [7].

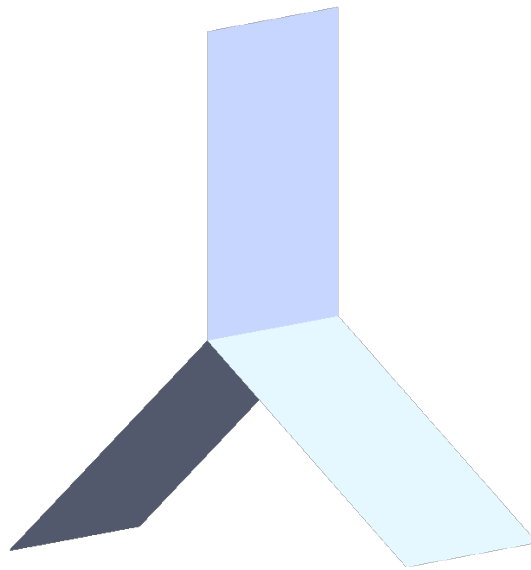
3.1 Construction of the Multi-Trace Approach

There are different strategies which could be used to construct the BES defined on a three-dimensional multi-screen, therefore in this section we would like to describe our choice. In our numerical experiments we deal with a disk screen (figure 3.1a), a triple multi-screen (figure 3.1b) and a quadruple multi-screen (figure 3.1c). The junctions of our multi-screens form in both cases a simple line. More complicated geometries with articulated junctions are allowed and they can be treated exactly in the same way as we did for our simpler junctions.

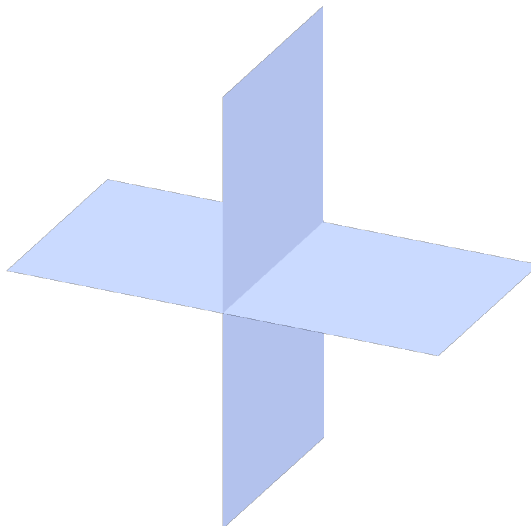
Similarly to the what we did in the previous chapter, also for the three-dimensional case we built multi-trace meshes which tightly wrap the screens. This means that all the mesh nodes which are not on the boundary of the screen are duplicated and they form two perfectly overlapping surfaces for each of the flat parts of the multi-screen. To understand better the structure of the multi-trace meshes we show in figure 3.2



(A) Disk screen



(B) Triple junction



(C) Quadruple junction

FIGURE 3.1: The screens used in our 3D numerical simulations

how the surface of the multi-screen would look like if it had a finite thickness, to obtain the multi-trace mesh we then set the thickness to be null. If now we mesh such a surface we obtain couples of overlapping nodes on the flat parts and triplets or quadruplets of overlapping nodes at the junctions.

After the mesh is created, the BES can be generated in the same way as for a closed mesh.

An alternative strategy to generate the necessary BES could have been to use a classic single trace mesh on the open surfaces and manually build BES with duplicated DoFs.

3.2 Tests for the Kernel Dimensions

Also in 3D, when dealing with multi-trace meshes, the BEM matrices corresponding to the two BIOs have non empty kernels. As explained in the previous chapter, couple, triplets or quadruplets of overlapping elements or nodes can yield kernel contributions. In this section we show why overlapping DoFs yield a kernel contribution and we verify the theoretical expectations with numerical experiments. We will see that in the numerical experiments for the hypersingular operator, due to numerical errors which appear in the computation of double integrals by quadrature, the \mathbf{W} matrices are not exactly singular but they still have a certain number of eigenvalues which are smaller than the others. We report in figure 3.3 the plots of the eigenvalues of \mathbf{V} and \mathbf{W} for the three kind of meshes we considered.

3.2.1 Kernel of the Weakly Singular Operator

When discretizing the weakly singular operator, we use piecewise constant basis functions, each of which is associated with an element.

In the following, we show how every couple of overlapping elements yields a kernel contribution to \mathbf{V} .

Let us consider the situation depicted in figure 3.4. The DoF k is associated with the element τ_k and the DoFs i and j are associated with the overlapping elements $\tau_i \equiv \tau_j$. We observe that

$$\begin{aligned} \mathbf{V}_{ki} &= \frac{1}{4\pi} \int_{\tau_k} \int_{\tau_i} \frac{1}{\|\mathbf{x}-\mathbf{y}\|} \frac{1}{\Delta_k} \frac{1}{\Delta_i} d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\ &= \frac{1}{4\pi} \int_{\tau_k} \int_{\tau_j} \frac{1}{\|\mathbf{x}-\mathbf{y}\|} \frac{1}{\Delta_k} \frac{1}{\Delta_j} d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \mathbf{V}_{kj} \quad \forall k \in (1, \dots, N_T), \end{aligned} \quad (3.1)$$

where N_T is the total number of elements in the mesh. The discussion about the kernel contribution from section 2.1.1 applies also here. We empirically verified our theoretical expectations by computing the \mathbf{V} matrix using the BETL2 library on the meshes previously presented, we report the results in table 3.1. The dimension of the kernel has been computed by importing the matrices in Matlab and using the command `size(null(V),1)`. As expected the dimensionality of the kernel of \mathbf{V} is always half as the total number of elements, one for each couple.

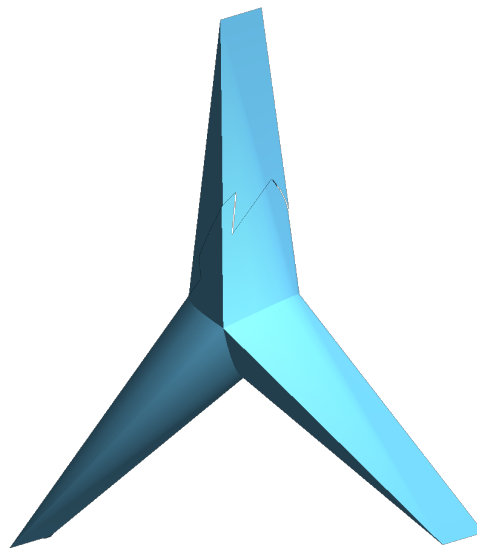
3.2.2 Kernel of the Hypersingular Operator

In the same way as done in section 2.1.2, we analyse in which way overlapping DoFs yield kernel contributions to \mathbf{W} .

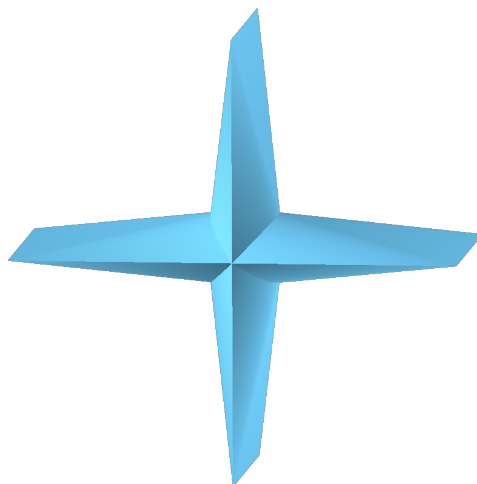
For the discretization of the hypersingular operator we use nodal piecewise linear



(A) Thick disk screen

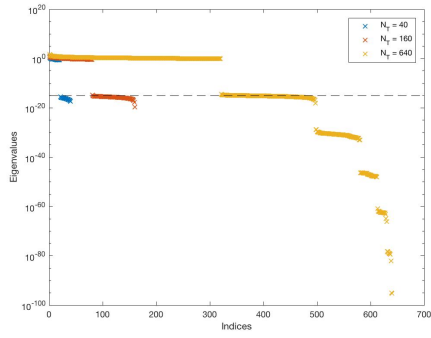


(B) Thick triple junction

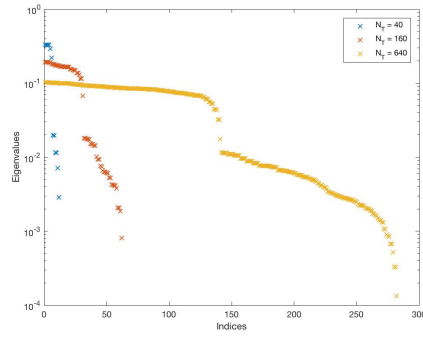


(C) Thick quadruple junction

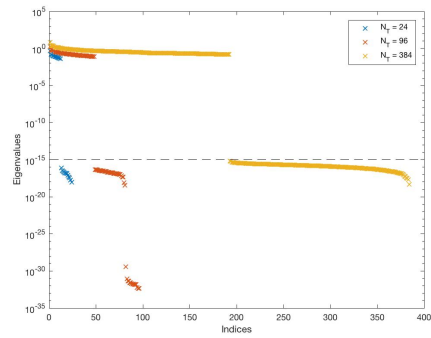
FIGURE 3.2: Rendering of the screens used in our numerical experiments as if they had a non-zero thickness



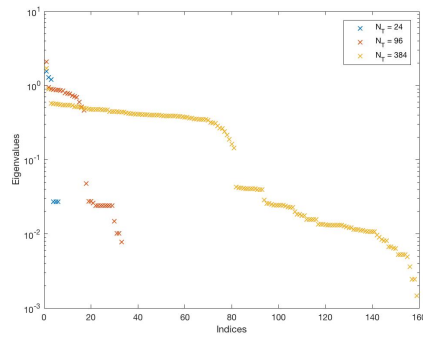
(A) Eigenvalues of V computed on a disk



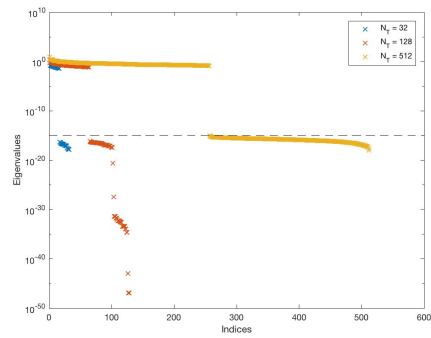
(D) Eigenvalues of W computed on a disk



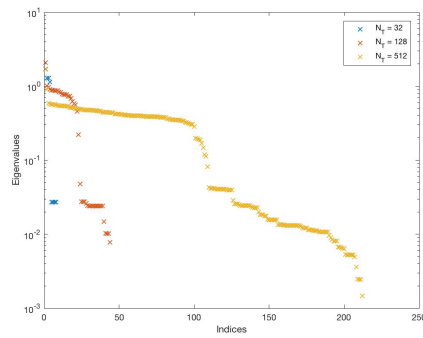
(B) Eigenvalues of V computed on a triple junction



(E) Eigenvalues of W computed on a triple junction



(C) Eigenvalues of V computed on a quadruple junction



(F) Eigenvalues of W computed on a quadruple junction

FIGURE 3.3: Eigenvalues of the BEM matrices on hierarchies of nested meshes. In the left column the dashed line indicates the machine precision.

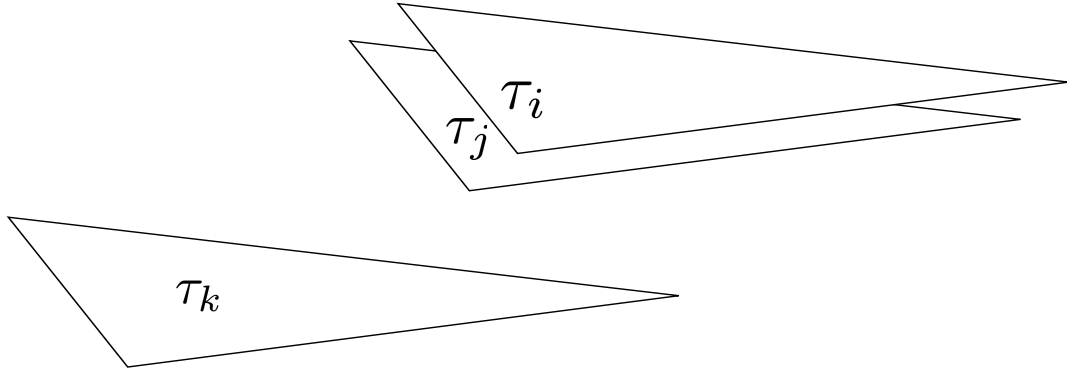


FIGURE 3.4: Representation of the support of the basis functions associated with overlapping DoFs i and j and a generic DoF k .

Disk			
N_T	40	160	640
$\dim(\ker(\mathbf{V}))$	20	80	320
Triple Junction			
N_T	24	96	384
$\dim(\ker(\mathbf{V}))$	12	48	192
Quadruple Junction			
N_T	32	128	512
$\dim(\ker(\mathbf{V}))$	16	64	256

TABLE 3.1: Dimension of the kernel of \mathbf{W} when using a mesh with N_T triangular elements

basis functions, associated with mesh vertices. Hence, we have to check what happens when the nodes overlap.

Following the same procedure of 2.1.2 we analyze separately the three cases of nodes away from the junctions, nodes at the junctions and nodes at the tips.

Couple of Vertices Away from the Junctions

We will analyze the situation depicted in figure 3.5. Although we pretend that the nodes are located at the intersection between three triangles, our analysis can be straightforwardly extended if more triangles are involved.

The DoFs i and j are associated to basis functions with overlapping supports $\text{supp}\{b_i\} \equiv \text{supp}\{b_j\}$ which have opposite normals $\mathbf{n}_i = -\mathbf{n}_j$ while the k -th DoF is any other DoF.

To keep the formulae short we write

$$f_{st}(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi} \frac{1}{\|\mathbf{x} - \mathbf{y}\|} (\mathbf{grad}_\Gamma b_{N_v}^s(\mathbf{x}) \times \mathbf{n}_s) \cdot (\mathbf{grad}_\Gamma b_{N_v}^t(\mathbf{y}) \times \mathbf{n}_t),$$

and

$$f_{st}^{rq}(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi} \frac{1}{\|\mathbf{x} - \mathbf{y}\|} \Big|_{\{\tau_q \times \tau_r\}} (\mathbf{grad}_\Gamma b_{N_v}^s(\mathbf{x})|_{\tau_q} \times \mathbf{n}_s) \cdot (\mathbf{grad}_\Gamma b_{N_v}^t(\mathbf{y})|_{\tau_r} \times \mathbf{n}_t).$$

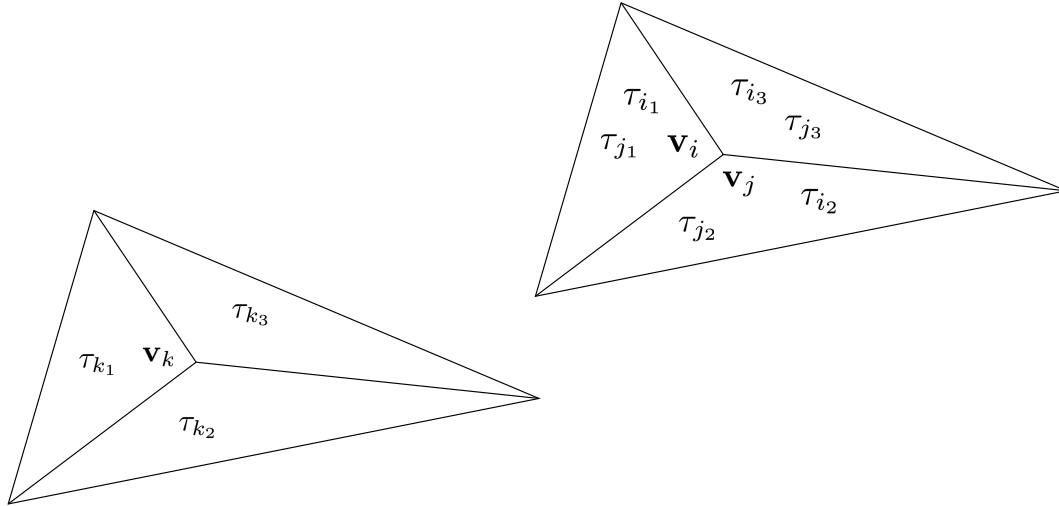


FIGURE 3.5: Representation of the support of the basis functions associated with two overlapping DoFs i and j and a generic DoF k .

We observe that in our case for every couple of overlapping triangles $\tau_m \equiv \tau_n$ it holds $f_{ki}^{lm}(\mathbf{x}, \mathbf{y}) = -f_{kj}^{ln}(\mathbf{x}, \mathbf{y})$ since we have $\mathbf{grad}_{\Gamma} b_{NV}^i|_{\tau_m} = \mathbf{grad}_{\Gamma} b_{NV}^j|_{\tau_n}$, but the normals are opposite. Consequently, we find that

$$\begin{aligned}
 \mathbf{W}_{ki} &= \int_{\tau_{k_1} \cup \tau_{k_2} \cup \tau_{k_3}} \int_{\tau_{i_1} \cup \tau_{i_2} \cup \tau_{i_3}} f_{ki}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\
 &= \sum_{q=\{k_1, k_2, k_3\}} \sum_{p=\{i_1, i_2, i_3\}} \int_{\tau_q} \int_{\tau_p} f_{ki}^{qp}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\
 &= - \sum_{q=\{k_1, k_2, k_3\}} \sum_{r=\{j_1, j_2, j_3\}} \int_{\tau_q} \int_{\tau_r} f_{kj}^{qr}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\
 &= - \int_{\tau_{k_1} \cup \tau_{k_2} \cup \tau_{k_3}} \int_{\tau_{j_1} \cup \tau_{j_2} \cup \tau_{j_3}} f_{kj}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = -\mathbf{W}_{kj} \quad \forall k \in (1, \dots, N_V),
 \end{aligned} \tag{3.2}$$

Vertices Overlapping at the Junctions

We analyze the case of a triple junction as depicted in figure 3.6. The same reasoning can be extended to any kind of junction with a small effort. Using the same notations from the previous section we find that

$$\begin{aligned}
 \mathbf{W}_{zi} + \mathbf{W}_{zj} + \mathbf{W}_{zk} &= \sum_{q=\{z_1, z_2, z_3\}} \sum_{r=\{i_1, i_2, i_3\}} \int_{\tau_q} \int_{\tau_r} f_{zi}^{qr}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) + \\
 &+ \sum_{q=\{z_1, z_2, z_3\}} \sum_{s=\{j_1, j_2, j_3\}} \int_{\tau_q} \int_{\tau_s} f_{zj}^{qs}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) + \\
 &\sum_{q=\{z_1, z_2, z_3\}} \sum_{t=\{k_1, k_2, k_3\}} \int_{\tau_q} \int_{\tau_t} f_{zk}^{qt}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\
 &= \sum_{q=\{z_1, z_2, z_3\}} \int_{\tau_q} \left[\underbrace{\sum_{r=\{i_1, i_2\}} \int_{\tau_r} f_{zi}^{qr}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) + \sum_{t=\{k_3, k_4\}} \int_{\tau_t} f_{zk}^{qt}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x})}_{=0 \text{ since } \tau_r \equiv \tau_t \text{ and } f_{zi}^{qr} = -f_{zk}^{qt}} \right. \\
 &+ \underbrace{\sum_{s=\{j_1, j_2\}} \int_{\tau_s} f_{zj}^{qs}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) + \sum_{r=\{i_3, i_4\}} \int_{\tau_r} f_{zi}^{qr}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x})}_{=0 \text{ since } \tau_s \equiv \tau_r \text{ and } f_{zj}^{qs} = -f_{zi}^{qr}} \\
 &\left. + \sum_{t=\{k_1, k_2\}} \int_{\tau_t} f_{zk}^{qt}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) + \sum_{s=\{j_3, j_4\}} \int_{\tau_s} f_{zj}^{qs}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) \right] d\sigma(\mathbf{y}) = 0, \\
 &= 0 \text{ since } \tau_t \equiv \tau_s \text{ and } f_{zk}^{qt} = -f_{zj}^{qs}
 \end{aligned}$$

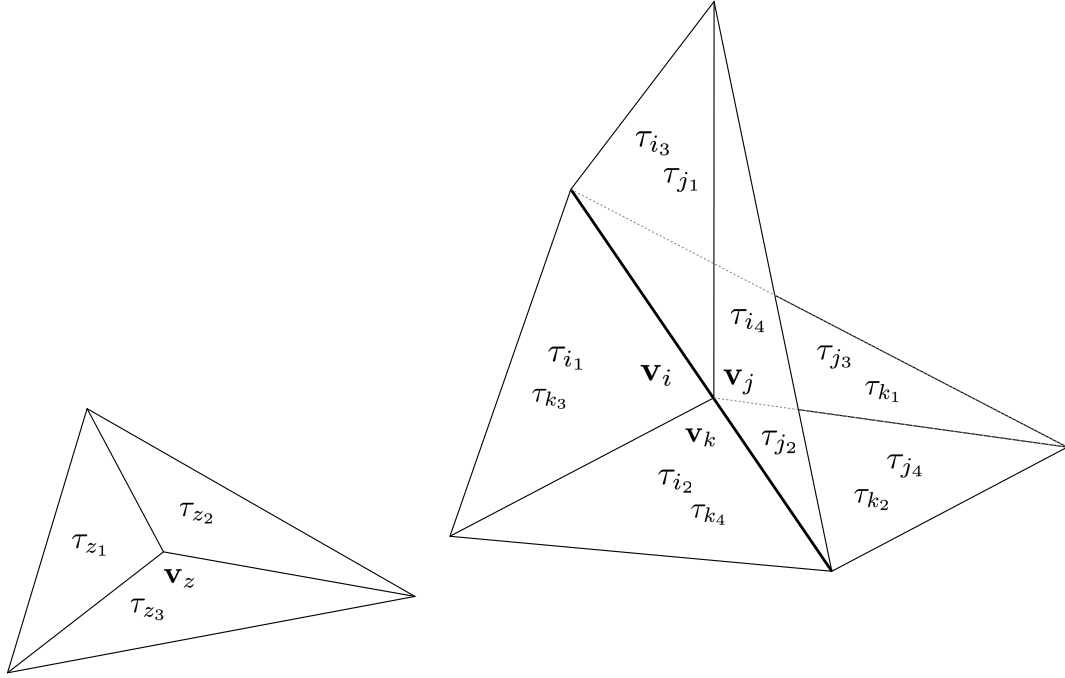


FIGURE 3.6: Representation of the support of the basis functions associated with three DoFs i, j and k which overlap at a triple junction and a generic DoF z .

that is

$$\mathbf{W}_{zi} + \mathbf{W}_{zj} + \mathbf{W}_{zk} = 0. \quad (3.3)$$

Vertices at the Boundaries

The last special case we analyze is the situation in which the DoF k is located on the boundary of the screen and z is any other DoF, as depicted in figure 3.7.

$$\begin{aligned} \mathbf{W}_{zk} &= \sum_{p=\{z_1, z_2, z_3\}} \sum_{q=\{k_1, k_2, k_3, k_4\}} \int_{\tau_p} \int_{\tau_q} f_{zk}^{pq}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\ &= \sum_{p=\{z_1, z_2, z_3\}} \int_{\tau_p} \left[\sum_{q=\{k_1, k_2, k_3, k_4\}} \int_{\tau_q} f_{zk}^{pq}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) \right] d\sigma(\mathbf{y}) = \\ &= \sum_{p=\{z_1, z_2, z_3\}} \int_{\tau_p} \left[\underbrace{\int_{\tau_{k_1}} f_{zk}^{pk_1}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) + \int_{\tau_{k_3}} f_{zk}^{pk_3}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x})}_{=0 \text{ since } \tau_{k_1} \equiv \tau_{k_3} \text{ and } f_{zk}^{pk_1} = -f_{zk}^{pk_3}} \right. \\ &\quad \left. + \underbrace{\int_{\tau_{k_2}} f_{zk}^{pk_2}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) + \int_{\tau_{k_4}} f_{zk}^{pk_4}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x})}_{=0 \text{ since } \tau_{k_2} \equiv \tau_{k_4} \text{ and } f_{zk}^{pk_2} = -f_{zk}^{pk_4}} \right] d\sigma(\mathbf{y}) = 0. \end{aligned} \quad (3.4)$$

Given equations (3.2), (3.3) and (3.4) we can repeat all the observations about the kernel contributions from section 2.1.2. In particular, we excluded the DoFs at the boundary of the screen from the computations as they would lead to rows and columns in \mathbf{W} which are populated by zeros only.

Observation 5. When it comes to numerical experiments we incur an undesired effect. The relations (3.2) and (3.3) are not precisely satisfied by some of the entries of \mathbf{W} . We investigated the reasons for these discrepancies between the theoretical expectations and the

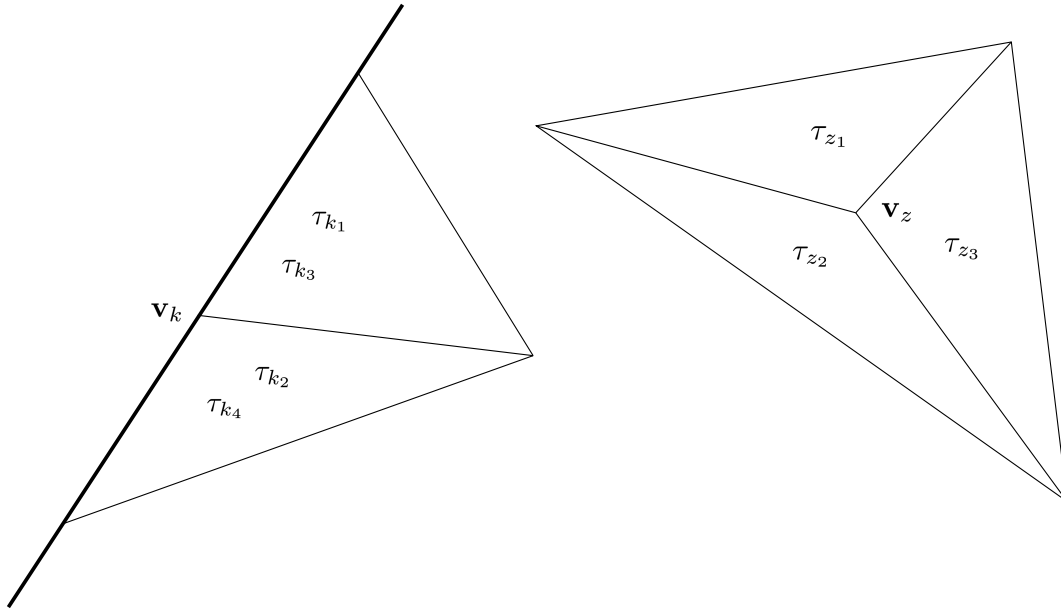


FIGURE 3.7: Representation of the support of the basis functions associated with a boundary DoF k and a generic DoF z .

numerical experiments, and we eventually found out that this effect is caused by the particular techniques used to compute the double integrals with singular integrands (details can be found in [11, Chapter 5]). In particular, the quadrature seems to produce slightly different results when changing the local ordering of the nodes in the involved triangles. In the meshes used for computing \mathbf{W} we needed to take into account the different orientations of the normals of overlapping triangles, which implies that they must have different local ordering of the nodes (see figure 3.8). This was not happening in the case of \mathbf{V} , as there we did not need to take into account the orientation of overlapping elements, which could then have the the same local ordering of the nodes.

For these reasons, some of the entries of \mathbf{W} which should have the same value differ by small error. Consequently, if we were to compute the kernel of \mathbf{W} using again the Matlab command `null(W)` we would always find that the kernel of \mathbf{W} only consists of the zero vector. On the other hand, we can observe that some eigenvalues of \mathbf{W} are way smaller than the others. Those eigenvalues correspond to the rows/columns of \mathbf{W} which are similar but not exactly equal. This phenomenon is observed in all the three considered cases as it is shown in figures 3.3d, 3.3e and 3.3f. Therefore, we can still identify an "approximate kernel dimensionality" by counting the cardinality of this subset of eigenvalues which are much smaller than the others. In table 3.2 we report the results we obtain in our numerical experiments by following this policy to compute the approximate kernel dimensionality of \mathbf{W} . The results again match with the expectation in term of number of couples, triplets or quadruplets of overlapping DoFs.

3.3 Tests for CG and GMRES solvers

In this section we aim at showing that, in the same way as in the two-dimensional case, iterative solvers such as CG and GMRES are able to solve systems of the form

$$\mathbf{V}\mathbf{x} = \mathbf{r}_v, \quad (3.5)$$

Disk			
N_T	40	160	640
DoFs away from the junction	12	62	282
DoFs at the junction	0	0	0
Expected kernel dimensions	6	31	141
Approximate kernel dimensions	6	31	141
Triple Junction			
N_T	24	96	384
DoFs away from the junction	6	30	150
DoFs at the junction	0	3	9
Expected kernel dimensions	3	16	78
Approximate kernel dimensions	3	16	78
Quadruple Junction			
N_T	32	256	512
DoFs away from the junction	8	40	200
DoFs at the junction	0	4	12
Expected kernel dimensions	4	21	103
Approximate kernel dimensions	4	21	103

TABLE 3.2: Dimension of the kernel of \mathbf{W} when using a mesh with N_T triangular elements

$$\mathbf{W}\mathbf{x} = \mathbf{r}_W, \quad (3.6)$$

despite the presence of the non-empty kernels. We would like to remind that for the CG this is only possible thanks to the particular structure of the BEM matrices which are always symmetric and positive semi-definite.

Given that our BEM matrices are not full-ranked, a generic right hand-side vector will not be always in the column space of \mathbf{V} or \mathbf{W} . Therefore, the choice of the right hand-side vector is important in order for the system to be solvable.

For our numerical experiments we chose to use

$$\begin{aligned} \mathbf{r}_V &= \mathbf{V}\mathbf{z}, \\ \mathbf{r}_W &= \mathbf{W}\mathbf{z}, \end{aligned} \quad (3.7)$$

where \mathbf{z} is a random vector. Notice that in any practical case the solubility of the system would be ensured by the fact that the right hand-side vectors would be the result of the multiplication of a mass matrix as defined in (1.16) times another vector. In this section we decided to use the right hand-side vectors as defined in (3.7) for the sake of simplicity. In both of the cases the stopping criterion of CG and GMRES is based on the relative residual and the tolerance is 10^{-5} .

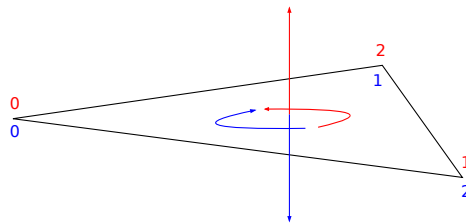


FIGURE 3.8: Two overlapping triangles with different local ordering of the nodes due to the opposite normals.

3.3.1 CG and GMRES for the Weakly Singular BIO

As already discussed in the previous section the \mathbf{V} matrix computed on a multi-screen mesh has a bunch of zero eigenvalues. Anyways, the matrix is still symmetric positive semi-definite and, thus, both CG and GMRES are still able to find a solution of system 3.5. In table 3.3 we report the results for our numerical tests together with the condition numbers of the \mathbf{V} matrices. As in the previous chapter, null eigenvalues have been neglected in the computation of the condition number, which therefore does not blow up even though the matrices are singular. The results show how both the methods converge with a number of iterations which is much smaller than the system size.

Disk			
N_T	40	160	640
size(\mathbf{V})	40×40	160×160	640×640
condition number	1.3040e+01	2.8897e+01	6.0478e+01
#iterations for CG	9	15	21
#iterations for GMRES	10	12	16
Triple Junction			
N_T	24	96	384
size(\mathbf{V})	24×24	96×96	384×384
condition number	1.0944e+01	2.3755e+01	4.8850e+01
#iterations for CG	8	13	20
#iterations for GMRES	9	15	18
Quadruple Junction			
N_T	32	128	512
size(\mathbf{V})	32×32	128×128	512×512
condition number	1.5405e+01	3.2453e+01	6.6592e+01
#iterations for CG	9	16	21
#iterations for GMRES	10	16	18

TABLE 3.3: number of iterations to convergence for the hypersingular BIO system with condition number

3.3.2 CG and GMRES for the Hypersingular BIO

We already showed that, in all of the three cases of interest, \mathbf{W} has a group of eigenvalues which are small but not really null. The numerical results from table 3.4 show how using CG and GMRES the systems are still solvable although, with respect to the previous case, the number of iterations to convergence is much higher for comparable system sizes. Together with the number of iterations we reported again also the condition number, but this time we did not neglect any eigenvalue as the small but non-zero eigenvalues of \mathbf{W} play a role in the convergence of the CG and GMRES methods. Indeed, we can infer that the higher number of iterations required for convergence with respect to the weakly singular case is caused by the presence of these small but not null eigenvalues. This result comply with Observation 4. As a matter of fact, it can be observed that our 3D \mathbf{W} matrices and the BEM matrices computed in the previous chapter with a small but non zero mesh thickness have a similar eigenvalue structure and like in that case we have a slow convergence. Since it would still be interesting to observe what happens when getting rid of the numerical inaccuracies explained in Observation 5, we decided to post-process the

computed BEM matrices so that the relations (3.2) and (3.3) are perfectly satisfied. The corrected BEM matrices have non empty kernels and some null-eigenvalues. In table 3.5 we report the convergence results for CG and GMRES and we can observe how those two methods converge much faster now. Indeed, it would be desirable to have a BEM code which does not suffer from these inaccuracies, but these results still give a confirmation that the convergence of CG and GMRES is not affected by the null eigenvalues also in this case.

Disk			
N_T	40	160	640
size(\mathbf{W})	12×12	62×62	282×282
condition number	1.1530e+02	2.3774e+02	7.7688e+02
#iterations for CG	12	40	99
#iterations for GMRES	11	55	157
Triple Junction			
N_T	24	96	384
size(\mathbf{W})	6×6	33×33	159×159
condition number	6.2836e+01	2.7002e+02	1.1539e+03
#iterations for CG	3	23	80
#iterations for GMRES	4	23	139
Quadruple Junction			
N_T	32	128	512
size(\mathbf{W})	8×8	44×44	212×212
condition number	6.2836e+01	2.7029e+02	1.1451e+03
#iterations for CG	3	27	96
#iterations for GMRES	4	28	173

TABLE 3.4: Number of iterations to convergence for the hypersingular BIO system with condition number with non corrected entries.

3.4 Elimination of the kernels

Analogously to what has been done in section 2.4, we can include our prior knowledge about the jumps of the traces into the BEM matrices to eliminate the kernels. The procedure is exactly equal to the one used in the two-dimensional case: we apply a transformation \mathbf{T} to the DoFs in order to obtain a reduced DoFs system. The \mathbf{T} transformations can be used to obtain the reduced matrices

$$\hat{\mathbf{V}} = \mathbf{T}^T \mathbf{V} \mathbf{T} \quad \text{and} \quad \hat{\mathbf{W}} = \mathbf{T}^T \mathbf{W} \mathbf{T},$$

which are full-ranked. The \mathbf{T} transformation can be built exactly how we stated in section 2.4 with the difference that with a two-dimensional mesh we cannot rely on the DoFs ordering to understand which nodes are overlapping and which DoFs should be eliminated. This is because in triangular meshes in which some DoFs have been eliminated it is not easy to establish a priori the nodes ordering. In fact, the BETL2 library produces a different ordering every time that the code is run. For the weakly singular operator we can write

$$\mathbf{T}_{ij} = \begin{cases} 1 & \text{if } \tau_i \equiv \tau_j, \\ 0 & \text{otherwise,} \end{cases}$$

Disk			
N_T	40	160	640
size(\mathbf{W})	12×12	62×62	282×282
#iterations for CG	3	6	9
#iterations for GMRES	4	7	10
Triple Junction			
N_T	24	96	384
size(\mathbf{W})	6×6	33×33	159×159
#iterations for CG	2	7	11
#iterations for GMRES	3	8	12
Quadruple Junction			
N_T	32	128	512
size(\mathbf{W})	8×8	44×44	212×212
#iterations for CG	2	8	33
#iterations for GMRES	3	9	34

TABLE 3.5: Number of iterations to convergence for the discrete hypersingular BIO system with corrected entries.

where τ_i and τ_j elements of the mesh ordered in the same way as the original DoFs. Using this transformation we associate to every couple of overlapping elements the value of the Neumann jump at the centroid of the elements. Similarly for \mathbf{W} , we can write that if the nodes \mathbf{p}_i and \mathbf{p}_j overlap away from a junction we have $\mathbf{T}_{ij} = 1$ and $\mathbf{T}_{ji} = -1$. Here the idea is to associate to every couple of overlapping nodes the value of the Dirichlet jump at those nodes.

Chapter 4

Theory for Electromagnetic Scattering by Complex Screens

The theoretical and practical ideas used in the previous chapters can be applied also to the computation of the electromagnetic field scattered by a complex screen. In this chapter we summarize the theory as derived in [5], which allows to extend our approach to the electromagnetic scattering.

Although some new complications arise, we can follow a similar scheme to the one used in chapter 1 and derive a multi-trace space to then single out a single-trace space. To complete the picture, the jump space will be again the dual of the single-trace space. The definition of the multi-trace space as quotient space will allow to redefine the commonly used traces using the canonical surjection. Having traces and suitable layer potentials, it will be possible to define the BIEs which will show up in a representation formula for the solution of electromagnetic scattering. From this representation formula we will derive the Electric Field Integral Equation (EFIE), which is the only BIE we consider in this case.

4.1 Function Spaces

Using the same strategy as in Chapter 1, we define a tangential trace space as quotient of spaces of functions defined on the domain $\mathbb{R}^3 \setminus \Gamma$. Before that we need to recall the standard functional analysis framework on smooth boundaries, then we define function spaces on a domain which contains a multi-screens and, taking the quotients of them, we define a multi-trace space

4.1.1 Standard Tangential Trace Space

In this section we recall the main definitions used in the standard functional analysis framework as they are stated in [2]. In the following Ω is a generic Lipschitz domain and $\partial\Omega$ is its boundary.

To begin with, we define the function spaces for functions defined in the domain Ω :

$$\begin{aligned} \mathbf{H}(\mathbf{curl}, \Omega) &= \{\mathbf{u} \in L^2(\Omega) : \mathbf{curl}(\mathbf{u}) \in (L^2(\Omega))^3\}, \\ \mathbf{H}_{0,\partial\Omega}(\mathbf{curl}, \Omega) &= \{\mathbf{u} \in L^2(\Omega) : \mathbf{curl}(\mathbf{u}) = \mathbf{0}\}. \end{aligned}$$

For functions in $\mathbf{H}(\mathbf{curl}, \Omega)$ we define the tangential trace

$$\gamma_T(\mathbf{u}) = \mathbf{n} \times (\mathbf{u} \times \mathbf{n}).$$

The definition of the tangential trace space is now simply

$$\mathbf{H}^{-\frac{1}{2}}(\text{curl}_{\partial\Omega}, \partial\Omega) = \{\gamma_T(\mathbf{u}) : \mathbf{u} \in \mathbf{H}(\text{curl}, \Omega)\}$$

. With these definitions we have that the tangential trace induces a surjective continuous trace operator $\gamma_T : \mathbf{H}(\text{curl}, \Omega) \rightarrow \mathbf{H}^{-\frac{1}{2}}(\text{curl}_{\partial\Omega}, \partial\Omega)$ and $\mathbf{H}_{0,\partial\Omega}(\text{curl}, \partial\Omega) = \ker(\gamma_T)$. Once more the trace space can be identified as a quotient space

$$\mathbf{H}^{-\frac{1}{2}}(\text{curl}_{\partial\Omega}, \partial\Omega) = \mathbf{H}(\text{curl}, \Omega) / \mathbf{H}_{0,\partial\Omega}(\text{curl}, \Omega) \quad (4.1)$$

and the trace γ_T is the canonical projection associated with the quotient space.

4.1.2 Function Spaces for Vector Fields in the Domain

Now we consider Γ as being a multiscreen and we define function spaces for functions defined on $\mathbb{R}^3 \setminus \Gamma$.

The space $\mathbf{H}(\text{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$ denotes the set of functions $\mathbf{u} \in (L^2(\mathbb{R}^3))^3$ such that there exists $\mathbf{p} \in (L^2(\mathbb{R}^3))^3$ satisfying

$$\int_{\mathbb{R}^3 \setminus \bar{\Gamma}} \mathbf{u} \cdot \text{curl}(\mathbf{v}) dx = \int_{\mathbb{R}^3 \setminus \bar{\Gamma}} \mathbf{p} \cdot \mathbf{v} dx \quad \forall \mathbf{v} \in (\mathcal{D}(\mathbb{R}^3 \setminus \Gamma))^3.$$

This is an Hilbert space when equipped with the scalar product

$$(\mathbf{u}, \mathbf{v})_{\mathbf{H}(\text{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})} := \int_{\mathbb{R}^3 \setminus \bar{\Gamma}} \mathbf{u} \cdot \bar{\mathbf{v}} dx + \int_{\mathbb{R}^3 \setminus \bar{\Gamma}} (\text{curl} \mathbf{u}|_{\mathbb{R}^3 \setminus \bar{\Gamma}}) \cdot (\text{curl} \bar{\mathbf{v}}|_{\mathbb{R}^3 \setminus \bar{\Gamma}}) dx.$$

where $\text{curl} \mathbf{u}|_{\mathbb{R}^3 \setminus \bar{\Gamma}} = \mathbf{p}$.

This scalar product induces a norm $\| \cdot \|_{\mathbf{H}(\text{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})}$ and we can define $\mathbf{H}_{0,\Gamma}(\text{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$ as the closure of $\mathcal{D}(\mathbb{R} \setminus \bar{\Gamma})$ with respect to the $\| \cdot \|_{\mathbf{H}(\text{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})}$ norm.

It is important for the following to observe that both $\mathbf{H}(\text{curl}, \mathbb{R}^3)$ and $\mathbf{H}_{0,\Gamma}(\text{curl}, \mathbb{R}^3)$ are closed subspaces of $\mathbf{H}(\text{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$ and we have the following chain of inclusions:

$$\mathbf{H}_{0,\Gamma}(\text{curl}, \mathbb{R}^3) \subset \mathbf{H}(\text{curl}, \mathbb{R}^3) \subset \mathbf{H}(\text{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$$

4.1.3 Tangential Multi-Trace Space

Exactly in the same way as in chapter 1 we define tangential multi-trace space by taking the cue from equation 4.1:

$$\mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma) := \mathbf{H}(\text{curl}, \mathbb{R}^3 \setminus \bar{\Gamma}) / \mathbf{H}_{0,\Gamma}(\text{curl}, \mathbb{R}^3 \setminus \bar{\Gamma}).$$

Like the multi-trace spaces defined in chapter 1, this space should be read as a generalization of the trace spaces to a multi-screen. Traces in this space do not enjoy any tangential continuity across the screen.

The tangential multi-trace space is equipped with the quotient norm and the canonical surjection

$$\pi_T : \mathbf{H}(\text{curl}, \mathbb{R}^3 \setminus \bar{\Gamma}) \rightarrow \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma),$$

which plays the same role as the tangential trace in the Lipschitz case.

We define the duality pairing for functions $\dot{\mathbf{u}}, \dot{\mathbf{v}} \in \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$

$$\ll \dot{\mathbf{u}}, \dot{\mathbf{v}} \gg_{\times} = \int_{\mathbb{R}^3 \setminus \bar{\Gamma}} \mathbf{curl}(\mathbf{u}) \cdot \mathbf{v} - \mathbf{u} \cdot \mathbf{curl}(\mathbf{v}) dx,$$

where $\mathbf{u}, \mathbf{v} \in \mathbf{H}(\mathbf{curl}, \mathbb{R} \setminus \bar{\Gamma})$ are such that $\pi_T(\mathbf{u}) = \dot{\mathbf{u}}$ and $\pi_T(\mathbf{v}) = \dot{\mathbf{v}}$.

In [5, Section 4.4] the authors show concrete formulae for the computation of this pairing in some particular situations. Using their reasoning it can be seen that this pairing is equivalent to the $\langle \cdot, \cdot \rangle_{\tau, \Gamma}$ pairing from [4, Equation 10] when computed on the boundary of the infinitesimally thick screen just like in observation 1. To convey this idea and according to the notation of Chapter 1 we will denote the integral on the boundary of the infinitesimally thick screen as

$$\ll \dot{\mathbf{u}}, \dot{\mathbf{v}} \gg_{\times} = \int_{[\Gamma]} (\dot{\mathbf{u}} \times \mathbf{n}) \cdot \dot{\mathbf{v}} d\sigma. \quad (4.2)$$

The most important property about the $\ll \cdot, \cdot \gg_{\times}$ -pairing is that it puts $\mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma)$ in duality with itself.

Proposition 9 (Self-duality of $\mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma)$, [5, Proposition 4.2]). *For any continuous linear form $\phi : \mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma) \rightarrow \mathbb{C}$ there exists a unique $\dot{\mathbf{u}} \in \mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma)$ such that $\phi(\dot{\mathbf{v}}) = \ll \dot{\mathbf{u}}, \dot{\mathbf{v}} \gg_{\times}$ for all $\dot{\mathbf{v}} \in \mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma)$ and $\|\phi\|_{(\mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma))'} = \|\dot{\mathbf{u}}\|_{\mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma)}$.*

Moreover, we have a characterization of $\mathbf{H}_{0,\Gamma}(\mathbf{curl}, \mathbb{R}^3)$ as kernel of the bilinear form $\ll \cdot, \cdot \gg_{\times}$.

Lemma 2 ([5, Lemma 4.3]). *For any $\mathbf{u} \in \mathbf{H}(\mathbf{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$, we have $\mathbf{u} \in \mathbf{H}_{0,\Gamma}(\mathbf{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$ if and only if*

$$\int_{\mathbb{R}^3 \setminus \bar{\Gamma}} \mathbf{curl}(\mathbf{u}) \cdot \mathbf{v} - \mathbf{u} \cdot \mathbf{curl}(\mathbf{v}) dx = 0 \quad \forall \mathbf{v} \in \mathbf{H}(\mathbf{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$$

4.1.4 Tangential Single-Trace Space

In this section we single out a subspace of $\mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma)$ to which functions which do not jump across Γ belong.

Definition 11 (Tangential single-trace space). *The tangential single-trace space is defined as the quotient space*

$$\mathbf{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, [\Gamma]) := \mathbf{H}(\mathbf{curl}, \mathbb{R}^3) / \mathbf{H}_{0,\Gamma}(\mathbf{curl}, \mathbb{R}^3)$$

.

$\mathbf{H}(\mathbf{curl}, \mathbb{R}^3) \subset \mathbf{H}(\mathbf{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$ implies that $\mathbf{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, [\Gamma]) \subset \mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma)$, which means that single-traces are special multi-traces which additionally satisfy transmission conditions across Γ .

In the same way as in the scalar case, when dealing with single traces, polarity takes duality's place.

Proposition 10 ([5, Proposition 4.5]). *For $\dot{\mathbf{u}} \in \mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma)$, we have*

$$\dot{\mathbf{u}} \in \mathbf{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, [\Gamma]) \iff \ll \cdot, \cdot \gg_{\times} = 0 \quad \forall \dot{\mathbf{v}} \in \mathbf{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, [\Gamma])$$

4.1.5 Tangential Jump Spaces

Lastly, we define the tangential jump space as dual of spaces of the tangential single-trace space in analogy to Section 1.1.6.

Definition 12. *The tangential jump space on the multi-screen Γ is defined as the dual space*

$$\tilde{\mathbf{H}}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma]) = (\mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma]))'$$

The $\tilde{\mathbf{H}}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])$ space is endowed with the dual norm

$$\|\dot{\mathbf{u}}\|_{\tilde{\mathbf{H}}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])} := \sup_{\dot{\mathbf{v}} \in \mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])} \frac{|\ll \dot{\mathbf{u}}, \dot{\mathbf{v}} \gg_\Gamma|}{\|\dot{\mathbf{v}}\|_{\mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])}}$$

Also the jump spaces allow an interpretation as quotient space, as stated in the following lemma.

Lemma 3 ([5, Lemma 4.7]). *The tangential jump space $\tilde{\mathbf{H}}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])$ is isometrically isomorphic to the quotient space $\mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma) / \mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])$.*

Thanks to this interpretation, it is possible to see that each element of $\mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$ induces an element of $\tilde{\mathbf{H}}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])$ via the duality pairing $\ll \cdot, \cdot \gg_\times$. This observation justifies the following definition of jumps at multiscreens.

Definition 13 (Jump operator, [5, Definition 4.8]). *We define the jump operator $[\] : \mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma) \rightarrow \tilde{\mathbf{H}}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])$ through*

$$\ll [\dot{\mathbf{u}}], \dot{\mathbf{v}} \gg_\times := \ll \dot{\mathbf{u}}, \dot{\mathbf{v}} \gg_\times, \quad \forall \dot{\mathbf{v}} \in \mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma]).$$

An important consequence of the self-polarity of the tangential single-trace space is the following property about the kernel of the jump operators.

Lemma 4 ([5, Lemma 4.9]). *A trace $\dot{\mathbf{u}} \in \mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$ belongs to $\mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])$ if and only if $[\dot{\mathbf{u}}] = 0$.*

4.2 Boundary Integral Equations

Now that we have suitable function spaces defined on a multi-screen, we are ready to derive the BIE which solve the electromagnetic scattering problem, the EFIE. However, to derive such equations it is necessary to define some surface differential operators for multi-screens. Those operators are crucial to define proper traces on multi-screens, using the canonical surjections given by the quotient spaces, and proper layer potentials. Combining traces and layer potentials we obtain the BIOs which show up in the representation formula for the solution of the electromagnetic scattering problem.

4.2.1 Surface Differential Operators

First, we can define the surface gradient on multi-screen. We notice that if $p \in H^1(\mathbb{R}^3 \setminus \bar{\Gamma})$ then in $\mathbb{R}^3 \setminus \bar{\Gamma}$ we have $\text{curl}(\nabla p) = 0$ and, therefore, $\nabla p \in \mathbf{H}(\text{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$

and its tangential trace is well defined. Given these considerations we can write for all $\mathbf{u} \in \mathbf{H}(\mathbf{curl}, \mathbb{R}^3 \setminus \Gamma)$ and $p \in H^1(\mathbb{R}^3 \setminus \Gamma)$

$$\int_{\mathbb{R}^3 \setminus \Gamma} \mathbf{curl}(\mathbf{u}) \cdot \nabla p \, dx = \ll \pi_T(\mathbf{u}), \pi_T(\nabla p) \gg_{\times}. \quad (4.3)$$

Roughly speaking, the left-hand side only depends on the boundary values of \mathbf{u} and p , therefore nothing changes if we substitute p with $p + q$ for any $q \in H_{0,\Gamma}^1(\mathbb{R}^3)$. This implies that we can define a surface gradient $\nabla_{\Gamma} : \mathbb{H}^{\frac{1}{2}}(\Gamma) \rightarrow \mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma)$ through the formula

$$\nabla_{\Gamma}(\pi_D(p)) := \pi_T(\nabla p) \quad \forall p \in H^1(\mathbb{R}^3 \setminus \bar{\Gamma}).$$

Repeating the same reasoning for $p \in H^1(\mathbb{R}^3)$, which implies $\nabla p \in \mathbf{H}(\mathbf{curl}, \mathbb{R}^3)$ and this time we get $\nabla_{\Gamma} \dot{p} \in \mathbf{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, [\Gamma])$ for all $\dot{p} \in H^{\frac{1}{2}}([\Gamma])$, which means that the surface gradient as defined above also maps $H^{\frac{1}{2}}([\Gamma])$ into $\mathbf{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, [\Gamma])$.

With a similar strategy we can build also a surface curl operator, which so far was only used as a abstract notation for the trace spaces.

To begin with, we observe that for any $\mathbf{u} \in \mathbf{H}(\mathbf{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$ we have $\operatorname{div}(\mathbf{curl}(\mathbf{u})) = 0$ in $\mathbb{R}^3 \setminus \bar{\Gamma}$, which implies that $\mathbf{curl}(\mathbf{u}) \in H(\operatorname{div}, \mathbb{R}^3 \setminus \bar{\Gamma})$. Then, recalling the definition of the $\ll \cdot, \cdot \gg$ pairing between $\mathbb{H}^{\frac{1}{2}}(\Gamma)$ and $\mathbb{H}^{\frac{1}{2}}(\Gamma)$, we have for all $\mathbf{u} \in \mathbf{H}(\mathbf{curl}, \mathbb{R}^3 \setminus \Gamma)$ and $p \in H^1(\mathbb{R}^3 \setminus \Gamma)$

$$\int_{\mathbb{R}^3 \setminus \Gamma} \mathbf{curl}(\mathbf{u}) \cdot \nabla p \, dx = \ll \pi_D(p), \pi_N(\mathbf{curl}(\mathbf{u})) \gg, \quad (4.4)$$

by inspecting the left-hand side of this equation, we can observe that $\pi_N(\mathbf{curl}(\mathbf{u}))$ only depends on $\pi_T(\mathbf{u})$, therefore we can define the surface curl through the formula

$$\operatorname{curl}_{\Gamma}(\pi_T(\mathbf{u})) := \pi_N(\mathbf{curl}(\mathbf{u})) \quad \forall \mathbf{u} \in \mathbf{H}(\mathbf{curl}, \mathbb{R}^3 \setminus \bar{\Gamma}).$$

It is possible to make an interesting observation about the relation between the surface gradient and the surface curl. If $\pi_T(\mathbf{u})$ for some $\dot{p} = \pi_D(p) \in \mathbb{H}^{\frac{1}{2}}(\Gamma)$, then $\operatorname{curl}_{\Gamma}(\nabla_{\Gamma} \dot{p}) = \operatorname{curl}_{\Gamma}(\pi_T(\nabla p)) = \pi_N(\mathbf{curl}(\nabla p)) = 0$, that is $\mathbf{curl}_{\Gamma} \cdot \nabla_{\Gamma} = 0$.

Moreover in the same way as for the surface gradient we can see that the surface curl maps single traces to single traces.

The definitions of the surface differential operators allow to formulate a surface Green's formula on multi-screens. By observing that $\pi_D : H^1(\mathbb{R}^3) \setminus \bar{\Gamma} \rightarrow \mathbb{H}^{\frac{1}{2}}(\Gamma)$ and $\pi_T : \mathbf{H}(\mathbf{curl}, \mathbb{R}^3) \setminus \bar{\Gamma} \rightarrow \mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma)$ and, using equations (4.3) and (4.4), we obtain the formula

$$\ll \dot{p}, \operatorname{curl}_{\Gamma}(\dot{\mathbf{v}}) \gg = \ll \dot{\mathbf{v}}, \nabla_{\Gamma}(\dot{p}) \gg_{\times} \quad \forall \dot{p} \in \mathbb{H}^{\frac{1}{2}}(\Gamma), \quad \dot{\mathbf{v}} \in \mathbb{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, \Gamma).$$

Using this formula we can extend the definition of the surface operators to the tangential jump spaces. For the surface gradient we can define $\nabla_{\Gamma} : \tilde{H}^{\frac{1}{2}}([\Gamma]) \rightarrow \tilde{\mathbf{H}}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, [\Gamma])$ as adjoint to $\operatorname{curl}_{\Gamma}$ by the formula

$$\ll \dot{\mathbf{v}}, \nabla_{\Gamma} \dot{u} \gg_{\times} := \ll \dot{u}, \operatorname{curl}_{\Gamma} \dot{\mathbf{v}} \gg \quad \forall \dot{\mathbf{v}} \in \mathbf{H}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, [\Gamma]),$$

for all $\dot{u} \in \tilde{H}^{\frac{1}{2}}([\Gamma])$. Similarly one can also define an operator $\operatorname{curl}_{\Gamma} : \tilde{\mathbf{H}}^{-\frac{1}{2}}(\mathbf{curl}_{\Gamma}, [\Gamma]) \rightarrow \tilde{H}^{-\frac{1}{2}}([\Gamma])$.

4.2.2 Boundary Value Problem and Traces

Along the same lines as in Chapter 1, after having defined the abstract functional analysis framework needed to extend the notions of jumps and traces to the case of a complex screen, we are ready to extend also the results about the BVPs and the traces.

Boundary Value Problem

First of all we show how to derive the electric wave equation starting from the Maxwell Equations. In the same way as in Chapter 1 we consider time-harmonic vector fields, which are of the form

$$\tilde{\mathbf{F}}(\mathbf{x}, t) = \mathbf{F}(\mathbf{x})e^{i\omega t}.$$

With this time dependency the Maxwell's equation become

$$\begin{cases} \operatorname{div} \mathbf{E} &= \frac{\rho}{\epsilon_0}, \\ \operatorname{div} \mathbf{B} &= 0, \\ \operatorname{curl} \mathbf{E} &= -i\omega t \mathbf{B}, \\ \operatorname{curl} \mathbf{B} &= \mu_0 (\mathbf{J} + \epsilon_0 i\omega t \mathbf{E}). \end{cases}$$

where \mathbf{E} is the amplitude of the electric field, \mathbf{B} is the amplitude of the magnetic field, ρ is the electric charge density, \mathbf{J} is the electric current density, ϵ_0 and μ_0 are respectively the permittivity and permeability of free space.

If now we impose $\mathbf{J} = 0$ (i.e. there are no free currents), by taking the curl of the third equation and plugging the fourth into it we obtain

$$\operatorname{curl} \operatorname{curl} \mathbf{E} - \kappa^2 \mathbf{E} = 0 \quad \text{in } \Gamma \setminus \mathbb{R}^3,$$

with $\kappa = \omega \sqrt{\epsilon_0 \mu_0}$. Moreover, we will suppose the tangential trace of the solution to be known and we will look for a solution which satisfies the Silver-Müller radiation conditions

$$\lim_{r \rightarrow \infty} \int_{\partial B_r} |\operatorname{curl}(\mathbf{E}) \times \mathbf{n}_r - i\kappa \mathbf{u}|^2 d\sigma = 0.$$

As we will see later, this BVP is well-posed on a multi-screen.

In order to prove the well-posedness of the BVPs a restriction of the considered geometries is needed because the general notion of a multi-screen as in Definition 2 would pose some problems regarding compact embeddings. We therefore restrict ourselves to considering piecewise smooth multi-screens.

Definition 14 (Piecewise Smooth Multi-screen, [5, Definition 6.3]). *We call a multi-screen piecewise smooth, if the adjacent Lipschitz domains Ω_j , $j = 0, \dots, n$, stipulated by definition 2, are curved Lipschitz polyhedra and $\bar{\Gamma} \cap \partial \Omega_j$ is the union of smooth faces of Ω_j*

According to [5, Section 6.1], this class of multi-screens does not pose any technical issues in the development of the theory for the BVPs.

It is important to notice that the screens of our interest (for example the ones used in the previous chapter) belong to this class and we can keep performing computations on them because the following existence and uniqueness result is valid.

Proposition 11 (Existence and Uniqueness of Solutions of the Exterior Dirichlet Problem, [5, Proposition 6.6]). *Assume that Γ is a piecewise smooth multi-screen and $\mathbb{R}^3 \setminus \Gamma$ is*

connected, then for any tangential multi-trace $\mathbf{g} \in \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$ there exists a unique vector field $\mathbf{E} \in \mathbf{H}_{\text{loc}}(\mathbf{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$ that satisfies

$$\mathbf{curlcurlE} - \kappa^2 \mathbf{E} = \mathbf{0} \quad \text{in } \mathbb{R}^3 \setminus \bar{\Gamma}, \quad \pi_T(\mathbf{E}) = \mathbf{g} \quad \text{on } \Gamma$$

and the Silver-Müller radiation condition. Moreover, \mathbf{E} depends continuously on \mathbf{g} .

Traces

The last ingredient needed for the derivation of BIEs for the electromagnetic scattering are the trace operators. In the same way as in the scalar case the trace operators are directly related to the canonical surjection onto the tangential multi-trace space. In particular, for sufficiently smooth vector fields the traces are defined as

$$\gamma_T(\mathbf{E}) := \pi_T(\mathbf{E}) \quad \text{and} \quad \gamma_R(\mathbf{E}) := \pi_T(\mathbf{curl}(\mathbf{E}))$$

By denoting $\mathbf{curl}^2 := \mathbf{curlcurl}$ we can now define the Hilbert space

$$\mathbf{H}(\mathbf{curl}^2, \mathbb{R}^3 \setminus \bar{\Gamma}) := \{\mathbf{v} \in \mathbf{H}(\mathbf{curl}, \mathbb{R}^3 \setminus \bar{\Gamma}) : \mathbf{curl}^2(\mathbf{v}) \in (L^2(\Gamma))^3\},$$

equipped with the natural norm $\sum_{j=0}^2 \|\mathbf{curl}^j \mathbf{v}\|_{L^2(\mathbb{R}^3 \setminus \bar{\Gamma})}$. Then it is possible to derive the following result:

Lemma 5 ([5, Lemma 7.1]). *The operators $\gamma_T, \gamma_R : \mathbf{H}(\mathbf{curl}^2, \mathbb{R}^3 \setminus \bar{\Gamma}) \rightarrow \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$ are both continuous and they admit a continuous right-inverse.*

In the following section we will also make use of the operators

$$\begin{aligned} \gamma'_D &: \mathbb{H}^{-\frac{1}{2}}(\Gamma) \rightarrow H_{\text{loc}}^1(\mathbb{R}^3 \setminus \bar{\Gamma})', \\ \gamma'_T &: \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma) \rightarrow \mathbf{H}_{\text{loc}}(\mathbb{R}^3 \setminus \bar{\Gamma})', \\ \gamma'_R &: \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma) \rightarrow \mathbf{H}_{\text{loc}}(\mathbb{R}^3 \setminus \bar{\Gamma})', \end{aligned}$$

which are defined as the formal adjoint operators of γ_D, γ_T and γ_R :

$$\begin{aligned} \langle \gamma'_D \dot{p}, \varphi \rangle &= \ll \dot{p}, \gamma_D \varphi \gg & \forall \varphi \in \mathcal{D}(\mathbb{R}^3), \\ \langle \gamma'_T \dot{\mathbf{u}}, \boldsymbol{\varphi} \rangle &= \ll \dot{\mathbf{u}}, \gamma_T \boldsymbol{\varphi} \gg & \forall \boldsymbol{\varphi} \in (\mathcal{D}(\mathbb{R}^3))^3, \\ \langle \gamma'_R \dot{\mathbf{u}}, \boldsymbol{\varphi} \rangle &= \ll \dot{\mathbf{u}}, \gamma_R \boldsymbol{\varphi} \gg & \forall \boldsymbol{\varphi} \in (\mathcal{D}(\mathbb{R}^3))^3. \end{aligned}$$

4.2.3 Layer Potentials

We can now define the layer potentials which show up in the representation formula for the solution of the considered BVP. As usual, we have a single layer potential SL_κ and a double layer potential DL_κ for any $\dot{\mathbf{u}} \in \mathbf{H}(\mathbf{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$:

$$\begin{aligned} SL_\kappa(\dot{\mathbf{u}}) &= -\mathcal{G}_\kappa * \gamma'_T(\dot{\mathbf{u}}) + \kappa^{-2} \nabla(\mathcal{G}_\kappa * \gamma'_D \cdot \text{curl}_\Gamma(\dot{\mathbf{u}})), \\ DL_\kappa(\dot{\mathbf{u}}) &= -\mathcal{G}_\kappa * \gamma'_R(\dot{\mathbf{u}}). \end{aligned}$$

In the rest of this section we will list the most important properties of the layer potentials.

To begin with, we report a continuity result for the single layer potentials.

Lemma 6 ([5, Lemma 7.4] and [5, Corollary 7.7]). *The single layer potential SL_κ maps continuously the space $\mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$ into $\mathbf{H}_{\text{loc}}(\mathbf{curl}, \mathbb{R}^3)$. The double layer potential DL_κ maps continuously the space $\mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$ into $\mathbf{H}_{\text{loc}}(\mathbf{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$.*

In addition, the layer potentials can be shown to be solutions of the Maxwell equations.

Corollary 4 ([5, Corollary 7.7]). *It holds $\mathbf{curl}^2 SL_\kappa(\dot{\mathbf{u}}) - \kappa^2 SL_\kappa(\dot{\mathbf{u}}) = -\gamma'_T(\dot{\mathbf{u}})$ and $\mathbf{curl}^2 DL_\kappa(\dot{\mathbf{u}}) - \kappa^2 DL_\kappa(\dot{\mathbf{u}}) = -\mathbf{curl}(\gamma'_T(\dot{\mathbf{u}}))$ in the sense of the distributions in \mathbb{R}^3 . In addition $SL_\kappa(\dot{\mathbf{u}})$ and $DL_\kappa(\dot{\mathbf{u}})$ both satisfy the Silver Müller radiation conditions at ∞ for any $\dot{\mathbf{u}} \in \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$.*

Moreover, the layer potentials satisfy certain jump relations.

Proposition 12 (Jump Relations, [5, Proposition 7.8]). *For all the tangential traces $\dot{\mathbf{u}} \in \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$ we have*

$$\begin{aligned} [\gamma_T] \cdot DL_\kappa(\dot{\mathbf{u}}) &= [\dot{\mathbf{u}}] & [\gamma_T] \cdot SL_\kappa(\dot{\mathbf{u}}) &= 0, \\ [\gamma_R] \cdot DL_\kappa(\dot{\mathbf{u}}) &= 0 & [\gamma_R] \cdot SL_\kappa(\dot{\mathbf{u}}) &= [\dot{\mathbf{u}}] \end{aligned}$$

It is really important to observe how, exactly like in the scalar case, the layer potentials are not injective.

Lemma 7 (Kernels of layer potentials, [5, Lemma 7.9]).

$$\text{Ker}(SL_\kappa) = \text{Ker}(DL_\kappa) = \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$$

As it has been seen in Lemma 3 the tangential jump space $\tilde{\mathbf{H}}^{-\frac{1}{2}}(\mathbf{curl}_\Gamma, [\Gamma])$ can be interpreted as a quotient space. This observation together with the previous lemma show that the layer potential operators induce injective continuous maps defined on the jump space:

$$SL_\kappa, DL_\kappa : \tilde{\mathbf{H}}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma]) \rightarrow \mathbf{H}_{\text{loc}}(\mathbf{curl}^2, \mathbb{R}^3 \setminus \bar{\Gamma})$$

4.2.4 The Representation Formula and the EFIE

The last ingredient needed to derive the EFIE is a representation formula for the solution of the electromagnetic scattering problem, which we state in the following proposition.

Proposition 13 (Representation formula, [5, Proposition 7.3]). *Assume that $\mathbf{u} \in \mathbf{H}(\mathbf{curl}, \mathbb{R}^3 \setminus \bar{\Gamma})$ is a radiating vector field satisfying $\mathbf{curl}^2 \mathbf{u} - \kappa^2 \mathbf{u} = 0$ in \mathbb{R}^3 . Then it can be represented as*

$$\mathbf{u}(\mathbf{x}) = DL_\kappa(\gamma_T(\mathbf{u}))(\mathbf{x}) + SL_\kappa(\gamma_R(\mathbf{u}))(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^3 \setminus \bar{\Gamma}.$$

Now, when \mathbf{u} a solution of the BVP 11, by applying Lemma 7 and the representation formula we find

$$\gamma_T \cdot SL_\kappa(\gamma_R(\mathbf{u})) = \gamma_T \cdot SL_\kappa([\gamma_R(\mathbf{u})]) = \mathbf{f} := \mathbf{g} - \gamma_T \cdot DL_\kappa(\mathbf{g})$$

If we solve this equation and retrieve $\mathbf{p} = [SL_\kappa(\gamma_R(\mathbf{u}))]$ we can find the corresponding \mathbf{u} by plugging \mathbf{p} and \mathbf{g} into the representation formula. Since equation (4.2.4) is posed in $\mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])$ we can obtain an equivalent variational form by testing

with arbitrary functions in $\tilde{\mathbf{H}}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])$:

Find $\mathbf{p} \in \tilde{\mathbf{H}}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma])$ such that

$$\ll \gamma_T \cdot SL_\kappa(\mathbf{p}), \mathbf{q} \gg_\times = \ll \mathbf{f}, \mathbf{q} \gg_\times \quad \forall \mathbf{q} \in \tilde{\mathbf{H}}^{-\frac{1}{2}}(\text{curl}_\Gamma, [\Gamma]).$$

It is possible to give a more explicit form to the left-hand side of the EFIE by plugging into it the definition of the single layer potential:

$$\begin{aligned} \ll \gamma_T \cdot SL_\kappa(\mathbf{p}), \mathbf{q} \gg_\times &= \\ &= \kappa^{-2} \ll \gamma_D \cdot \mathcal{G}_\kappa * \gamma'_D(\text{curl}_\Gamma \mathbf{p}), \text{curl}_\Gamma \mathbf{q} \gg - \ll \gamma_T \cdot \mathcal{G}_\kappa * \gamma'_T(\mathbf{p}), \mathbf{q} \gg_\times \end{aligned} \quad (4.5)$$

In the notation from equation (4.2) the two terms become:

$$\begin{aligned} \ll \gamma_D \cdot \mathcal{G}_\kappa * \gamma'_D(\text{curl}_\Gamma \mathbf{p}), \text{curl}_\Gamma \mathbf{q} \gg &= \\ &= \int_{[\Gamma]} \int_{[\Gamma]} \mathcal{G}_\kappa(\mathbf{x} - \mathbf{y}) \text{curl}_\Gamma \mathbf{p}(\mathbf{x}) \text{curl}_\Gamma \mathbf{q}(\mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) \end{aligned} \quad (4.6)$$

$$\begin{aligned} \ll \gamma_T \cdot \mathcal{G}_\kappa * \gamma'_T(\mathbf{p}), \mathbf{q} \gg_\times &= \\ &= \int_{[\Gamma]} \int_{[\Gamma]} \mathcal{G}_\kappa(\mathbf{x} - \mathbf{y}) (\mathbf{n}(\mathbf{x}) \times \mathbf{p}(\mathbf{x})) \cdot (\mathbf{n}(\mathbf{y}) \times \mathbf{q}(\mathbf{y})) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) \end{aligned} \quad (4.7)$$

The EFIE can be shown to be elliptic, since it can be proved that it satisfies a generalized Gårding inequality. The proof is out of the scope of this thesis, but it can be found in [5, Section 9].

4.3 Galerkin Discretization of the EFIE

We can apply the same ideas as in Section 1.3 to discretize the EFIE using a Galerkin approach and turn it into a linear system of equations. In order to do so, we will need to specify which BES we will use to discretize the $\mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$, namely the Nédélec lowest order BES [9]. It will turn out that we need a second BES which is a rotated version of the first, to which we refer as the Raviart-Thomas BES [10].

4.3.1 The Boundary Element Spaces

Let us consider a triangulation \mathcal{G} of the infinitesimally thick screen, as in definition 9. We define the reference triangle as $\hat{\tau} := \{\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2)^T \in (0, 1)^2 : \hat{x}_2 < \hat{x}_1\}$ and the local to global mapping $\Phi_\tau : \hat{\tau} \mapsto \tau$. Now we are able to formally define the Nédélec BES. Here and in the following $\mathcal{E}(\mathcal{G})$ denotes the set of the panels of the mesh \mathcal{G} . Moreover, since we will deal again with an infinitesimally thick screen, in the next definitions the spaces $\mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$ and $\mathbf{H}^{-\frac{1}{2}}(\text{div}_\Gamma, \Gamma)$ are the same as in [4], where the authors deal with closed Lipschitz surfaces.

Definition 15 (Nédélec Edge Elements). *For $k \in \mathbb{N}_0$, we define the Nédélec boundary element space by*

$$\mathcal{N}_{\mathcal{T}(\mathcal{G})}^k = \{\mathbf{v} \in \mathbf{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma) : \mathbf{v}|_\tau \in \mathbf{F}_{t,\tau}(\mathcal{N}^k(\hat{\tau})), \forall \tau \in \mathcal{T}(\mathcal{G})\},$$

where $\mathbf{F}_{\mathbf{t},\tau}$ denotes the following (rotated) Piola transform

$$\begin{aligned}\mathbf{F}_{\mathbf{t},\tau}\{\mathbf{p}\}(\mathbf{x}) &:= \mathbf{D}\Phi_\tau(\hat{\mathbf{x}})(\mathbf{G}_\tau(\hat{\mathbf{x}}))^{-1}\mathbf{p}(\hat{\mathbf{x}}), \\ \mathbf{G}_\tau(\mathbf{x}) &:= (\mathbf{D}\Phi_\tau(\hat{\mathbf{x}}))^T\mathbf{D}\Phi_\tau(\hat{\mathbf{x}}), \quad \hat{\mathbf{x}} = \Phi_\tau^{-1}(\mathbf{x}), \quad \mathbf{x} \in \tau,\end{aligned}$$

and

$$\begin{aligned}\mathcal{N}^k(\hat{\tau}) &:= (\mathcal{P}_k(\hat{\tau}))^2 \oplus \mathcal{S}_k(\hat{\tau}), \\ \mathcal{S}_k(\hat{\tau}) &:= \{\mathbf{x} \in (\tilde{\mathcal{P}}_{k+1}(\hat{\tau}))^2 : \mathbf{v}(\hat{\mathbf{x}}) \cdot \hat{\mathbf{x}} = 0 \quad \forall \hat{\mathbf{x}} \in \hat{\tau}\},\end{aligned}$$

and $\tilde{\mathcal{P}}_k(\hat{\tau})$ represents all homogeneous multivariate polynomials of total degree $k \in \mathbb{N}_0$. $\mathcal{P}_k(\hat{\tau})$ contains all multivariate polynomials of total degree $k \in \mathbb{N}_0$ on $\hat{\tau}$. For $k = 0$, which we will consider for our numerical experiments, $\mathcal{P}_0(\hat{\tau})$ contains all constant functions on $\hat{\tau}$.

We will apply a lowest-order discretization, i.e. we will consider the zeroth order Nédélec space $\mathcal{N}_{\mathcal{T}(\mathcal{G})}^0(\mathcal{G})$. For this space the local shape functions on the reference triangle are:

$$\hat{\eta}_1^{\mathcal{N}^0}(\hat{\mathbf{x}}) = \begin{bmatrix} -\hat{x}_2 \\ \hat{x}_1 \end{bmatrix}, \quad \hat{\eta}_2^{\mathcal{N}^0}(\hat{\mathbf{x}}) = \begin{bmatrix} -\hat{x}_2 \\ \hat{x}_1 - 1 \end{bmatrix}, \quad \hat{\eta}_3^{\mathcal{N}^0}(\hat{\mathbf{x}}) = \begin{bmatrix} 1 - \hat{x}_2 \\ \hat{x}_1 \end{bmatrix}.$$

The second BES which is of interest for us is the Raviart-Thomas space which is defined as follows.

Definition 16 (Raviart-Thomas Edge Elements). For $k \in \mathbb{N}_0$, we define the Raviart-Thomas boundary element space by

$$\mathcal{RT}_{\mathcal{T}(\mathcal{G})}^k(\mathcal{G}) = \{\mathbf{v} \in \mathbf{H}^{-\frac{1}{2}}(\text{div}_\Gamma, \Gamma) : \mathbf{v}|_\tau \in \mathbf{F}_{\times,\tau}(\mathcal{RT}^k(\hat{\tau})), \forall \tau \in \mathcal{T}(\mathcal{G})\},$$

where $\mathbf{F}_{\times,\tau}$ is the so-called Piola transform

$$\begin{aligned}\mathbf{F}_{\times,\tau}\{\mathbf{p}\}(\mathbf{x}) &:= \frac{1}{\sqrt{\det \mathbf{G}_\tau(\hat{\mathbf{x}})}}\mathbf{D}\Phi_\tau(\hat{\mathbf{x}})\mathbf{p}(\hat{\mathbf{x}}), \\ \mathbf{G}_\tau(\mathbf{x}) &:= (\mathbf{D}\Phi_\tau(\hat{\mathbf{x}}))^T\mathbf{D}\Phi_\tau(\hat{\mathbf{x}}), \quad \hat{\mathbf{x}} = \Phi_\tau^{-1}(\mathbf{x}), \quad \mathbf{x} \in \tau,\end{aligned}$$

and

$$\begin{aligned}\mathcal{RT}^k(\hat{\tau}) &:= (\mathcal{P}_k(\hat{\tau}))^2 \oplus (\hat{\mathbf{x}}\tilde{\mathcal{P}}_k(\hat{\tau})) \\ &= \{\hat{\mathbf{x}} \mapsto \mathbf{p}(\hat{\mathbf{x}}) + q(\hat{\mathbf{x}})\hat{\mathbf{x}}, \quad \hat{\mathbf{x}} \in \hat{\tau} : \mathbf{p} \in (\mathcal{P}_k(\hat{\tau}))^2, q \in \tilde{\mathcal{P}}_k(\hat{\tau})\},\end{aligned}$$

and $\mathcal{P}_k(\hat{\tau})$ contains all multivariate polynomials of total degree $k \in \mathbb{N}_0$ on $\hat{\tau}$ and $\tilde{\mathcal{P}}_k(\hat{\tau})$ represents all homogeneous multivariate polynomials of total degree $k \in \mathbb{N}_0$.

We will consider only the lowest order Raviart Thomas BES $\mathcal{RT}_{\mathcal{T}(\mathcal{G})}^0(\mathcal{G})$, for which the local shape functions on the reference triangle are

$$\hat{\eta}_1^{\mathcal{RT}^0}(\hat{\mathbf{x}}) = \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix}, \quad \hat{\eta}_2^{\mathcal{RT}^0}(\hat{\mathbf{x}}) = \begin{bmatrix} \hat{x}_1 - 1 \\ \hat{x}_2 \end{bmatrix}, \quad \hat{\eta}_3^{\mathcal{RT}^0}(\hat{\mathbf{x}}) = \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 - 1 \end{bmatrix}.$$

At this stage it is important to observe that the Nédélec local shape functions can be obtained transforming the Raviart-Thomas local shape functions by applying to

them a rotation by 90 degrees rotation about the normal vector to the surface represented by the matrix

$$\mathbf{R} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

Therefore, we get that

$$\widehat{\boldsymbol{\eta}}_i^{\mathcal{RT}^0}(\hat{\mathbf{x}}) = -\mathbf{R} \widehat{\boldsymbol{\eta}}_i^{\mathcal{N}^0}(\hat{\mathbf{x}}) = \mathbf{n} \times \widehat{\boldsymbol{\eta}}_i^{\mathcal{N}^0}(\hat{\mathbf{x}}) \quad i = \{1, 2, 3\}, \quad (4.8)$$

which is due to the fact that the Nédélec BES is a rotated version of the Raviart-Thomas space (and viceversa). Each of the global basis functions are associated with the edges of the mesh and the total number of DoFs of the BES is equal to the total number of edges. We denote the global basis functions as $\boldsymbol{\eta}_i^{\mathcal{N}^0}$ and $\boldsymbol{\eta}_i^{\mathcal{RT}^0}$.

4.3.2 Linear System Construction

In this final section of this chapter we derive the typical linear system of equations which we obtain with a BE discretization. The BIE we would like to discretize is the EFIE posed in the multi-trace space. Also in the EFIE we substitute the jump spaces with multi-trace spaces for the same reasons as in section 1.3 and we obtain the following variational problem:

$$\begin{aligned} &\text{Find } \mathbf{p} \in \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma) \text{ s.t.} \\ &\ll \gamma_T \cdot SL_\kappa(\mathbf{p}, \mathbf{q}) \gg_\times = \ll \mathbf{f}, \mathbf{q} \gg_\times \quad \forall \mathbf{q} \in \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma). \end{aligned} \quad (4.9)$$

Now we approximate the $\mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$ using the $\mathcal{N}_{\mathcal{E}(\mathcal{G})}^k(\mathcal{G})$ BES. Therefore, a function $\mathbf{p} \in \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma)$ is approximated on the mesh \mathcal{G} as

$$\mathbf{p} \approx \sum_{i=1}^{N_E} p_i \boldsymbol{\eta}_i^{\mathcal{N}^0},$$

where N_E is the number of edges of \mathcal{G} . Therefore, the variational problem (4.9) can be rewritten in the form

$$\ll \gamma_T \cdot SL_\kappa\left(\sum_{i=1}^{N_E} p_i \boldsymbol{\eta}_i^{\mathcal{N}^0}, \mathbf{q}\right) \gg_\times = \ll \sum_{i=1}^{N_E} f_i \boldsymbol{\eta}_i^{\mathcal{N}^0}, \mathbf{q} \gg_\times \quad \forall \mathbf{q} \in \mathcal{N}_{\mathcal{E}(\mathcal{G})}^k(\mathcal{G}),$$

which results in

$$\ll \gamma_T \cdot SL_\kappa\left(\sum_{i=1}^{N_E} p_i \boldsymbol{\eta}_i^{\mathcal{N}^0}, \boldsymbol{\eta}_j^{\mathcal{N}^0}\right) \gg_\times = \ll \sum_{i=1}^{N_E} f_i \boldsymbol{\eta}_i^{\mathcal{N}^0}, \boldsymbol{\eta}_j^{\mathcal{N}^0} \gg_\times \quad \forall j = 1, \dots, N_E.$$

By defining the matrices

$$\begin{aligned} [\mathbf{S}]_{i,j} &:= \ll \gamma_T \cdot SL_\kappa(\boldsymbol{\eta}_i^{\mathcal{N}^0}, \boldsymbol{\eta}_j^{\mathcal{N}^0}) \gg_\times, & \mathbf{S} &\in \mathbb{R}^{N_E \times N_E} \\ [\mathbf{M}_\times]_{i,j} &:= \ll \boldsymbol{\eta}_i^{\mathcal{N}^0}, \boldsymbol{\eta}_j^{\mathcal{N}^0} \gg_\times, & \mathbf{M} &\in \mathbb{R}^{N_E \times N_E} \end{aligned}$$

and the vectors

$$\vec{p}_i = (p_1, \dots, p_{N_E})^T, \quad \vec{f}_i = (f_1, \dots, f_{N_E})^T,$$

we finally get the linear system

$$\mathbf{S}\vec{p} = \mathbf{M}\vec{f} \quad (4.10)$$

from which we can obtain the DoFs μ associated with the solution \mathbf{p} .

The entries of \mathbf{S} cannot be practically computed straightforwardly and they require a more detailed analysis. The idea is to decompose \mathbf{S} using equation (4.5) and finally compute the entries using the formulae (4.6) and (4.7). Then \mathbf{S} is computed as

$$\mathbf{S} = \kappa^{-2}\mathbf{S}_d - \mathbf{S}_{edge},$$

where

$$\begin{aligned} [\mathbf{S}_{edge}]_{i,j} &= \int_{[\Gamma]} \int_{[\Gamma]} \mathcal{G}_\kappa(\mathbf{x} - \mathbf{y}) (\mathbf{n}(\mathbf{x}) \times \boldsymbol{\eta}_i^{N^0}(\mathbf{x})) \cdot (\mathbf{n}(\mathbf{y}) \times \boldsymbol{\eta}_j^{N^0}(\mathbf{y})) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}), \\ [\mathbf{S}_d]_{i,j} &= \int_{[\Gamma]} \int_{[\Gamma]} \mathcal{G}_\kappa(\mathbf{x} - \mathbf{y}) \operatorname{curl}_\Gamma \boldsymbol{\eta}_i^{N^0}(\mathbf{x}) \operatorname{curl}_\Gamma \boldsymbol{\eta}_j^{N^0}(\mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}). \end{aligned}$$

Now we can observe that $\forall \mathbf{p} \in \mathbf{H}^{-\frac{1}{2}}(\operatorname{curl}, \Gamma)$ we have

$$\operatorname{curl}_\Gamma \mathbf{p} = \operatorname{div}_\Gamma(\mathbf{n} \times \mathbf{p}),$$

which, together with equation (4.8), allows us to compute the entries of the two matrices using the formulae

$$\begin{aligned} [\mathbf{S}_{edge}]_{i,j} &= \int_{[\Gamma]} \int_{[\Gamma]} \mathcal{G}_\kappa(\mathbf{x} - \mathbf{y}) \boldsymbol{\eta}_i^{\mathcal{RT}^0}(\mathbf{x}) \cdot \boldsymbol{\eta}_j^{\mathcal{RT}^0}(\mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}), \\ [\mathbf{S}_d]_{i,j} &= \int_{[\Gamma]} \int_{[\Gamma]} \mathcal{G}_\kappa(\mathbf{x} - \mathbf{y}) \operatorname{div}_\Gamma \boldsymbol{\eta}_i^{\mathcal{RT}^0}(\mathbf{x}) \operatorname{div}_\Gamma \boldsymbol{\eta}_j^{\mathcal{RT}^0}(\mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}). \end{aligned}$$

When using BETL2 it is easy to compute the \mathbf{S}_d matrix by using a discretized surface divergence. The discretization is carried out by observing that we can write

$$\operatorname{div}_\Gamma \boldsymbol{\eta}_i^{\mathcal{RT}^0} = \sum_{j=1}^{N_V} d_{ij} \beta_{N_E}^j$$

and if we construct the matrix $[\mathbf{D}]_{i,j} = d_{i,j}$, $\mathbf{D} \in \mathbb{R}^{N_E \times N_V}$, we can decompose \mathbf{S}_d as

$$\mathbf{S}_d = \mathbf{D}\mathbf{V}\mathbf{D}^T$$

where denotes \mathbf{V} same matrix as in Chapter 1

$$[\mathbf{V}]_{i,j} = \int_{[\Gamma]} \int_{[\Gamma]} \mathcal{G}_\kappa(\mathbf{x} - \mathbf{y}) \beta_{N_E}^i(\mathbf{x}) \beta_{N_E}^j(\mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}), \quad \mathbf{V} \in \mathbb{R}^{N_V \times N_V}.$$

To summarize, the linear system 4.10 becomes

$$(\kappa^{-2}\mathbf{S}_d - \mathbf{S}_{edge})\vec{p} = \mathbf{M}\vec{f}$$

Chapter 5

3D Electromagnetic Scattering

In this last chapter we will show the numerical results for our discretization of the EFIE. This chapter follows closely the structure of chapter 3: in section 5.1 we show that the kernel dimensions of the computed matrices match our theoretical expectations and in section 5.2 we show that iterative solvers are still able to solve the singular systems within a small number of iterations.

The construction of the boundary element spaces is done in the same way as for the 3D-Laplace equation: we build close meshes which wrap the screen and the BES are simply the usual trace spaces on a closed mesh.

Also in this case we observe the impact of numerical quadrature due to the different local ordering of overlapping DoFs, in particular in this case it is not always possible to observe a subset of much smaller eigenvalues. The convergence of the iterative solvers deteriorates because of these inaccuracies but we will show how, when eliminating them by correcting the matrices, fast convergence is recovered.

We test our codes on the same screens as in chapter 3 and we compute the BEM matrices using the BETL2 library [7].

5.1 Tests for the Kernel Dimensions

In this section we illustrate which combinations of DOFs yield a kernel contribution to the discrete EFIE operator \mathbf{S} and we show that the results of our numerical experiments match the theoretical expectations. In this case, we have degrees of freedom associated with the edges of the mesh so we need to analyze particular situations in which edges overlap. As in sections 2.1.2 and 3.2.2 there are three kinds of DoFs combinations which yield kernel contributions: couple of edges overlapping away from the junction, edges overlapping at a junction and edges at the boundary of the screen.

Couples of Vertices Away from the Junctions

To shorten the formulae we define two auxiliary functions:

$$g_{i,j}(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi} \frac{1}{\|\mathbf{x} - \mathbf{y}\|} (\mathbf{n}(\mathbf{x}) \times \boldsymbol{\eta}_i^{\mathcal{N}_0}(\mathbf{x})) \cdot (\mathbf{n}(\mathbf{y}) \times \boldsymbol{\eta}_j^{\mathcal{N}_0}(\mathbf{y})),$$

$$h_{i,j}(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi} \frac{1}{\|\mathbf{x} - \mathbf{y}\|} \text{curl}_{\Gamma} \boldsymbol{\eta}_i^{\mathcal{N}_0}(\mathbf{x}) \text{curl}_{\Gamma} \boldsymbol{\eta}_j^{\mathcal{N}_0}(\mathbf{y}),$$

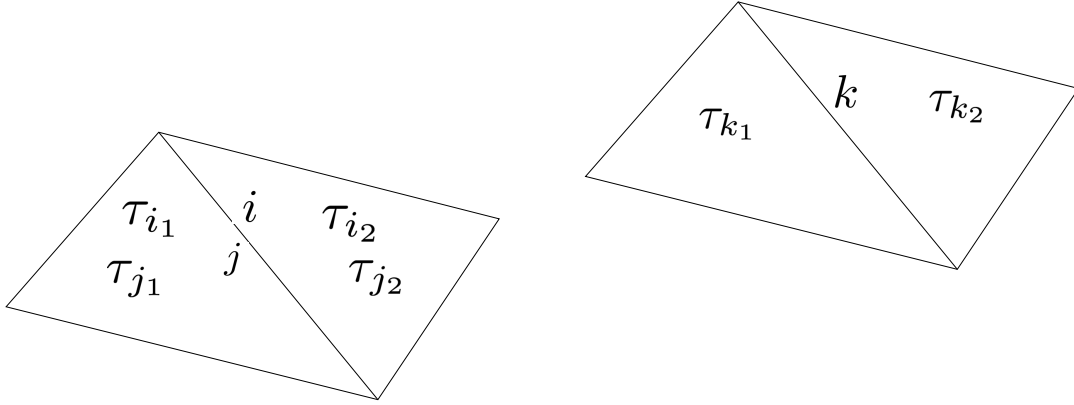


FIGURE 5.1: Representation of the support of the basis functions associated with the overlapping edges i and j and the generic edge k .

and their restrictions

$$g_{i,j}^{s,t}(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi} \frac{1}{\|\mathbf{x} - \mathbf{y}\|} \Big|_{\{\tau_s \times \tau_t\}} (\mathbf{n}(\mathbf{x}) \times \boldsymbol{\eta}_i^{\mathcal{N}_0}(\mathbf{x}))|_{\tau_s} \cdot (\mathbf{n}(\mathbf{y}) \times \boldsymbol{\eta}_j^{\mathcal{N}_0}(\mathbf{y}))|_{\tau_t},$$

$$h_{i,j}^{s,t}(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi} \frac{1}{\|\mathbf{x} - \mathbf{y}\|} \Big|_{\{\tau_s \times \tau_t\}} \text{curl}_{\Gamma} \boldsymbol{\eta}_i^{\mathcal{N}_0}(\mathbf{x}) \Big|_{\tau_s} \cdot \text{curl}_{\Gamma} \boldsymbol{\eta}_j^{\mathcal{N}_0}(\mathbf{y}) \Big|_{\tau_t}.$$

If we suppose that two overlapping edges i and j have the same orientations then we find that for two overlapping, but opposite triangles, i.e. $\tau_m \equiv \tau_n$ and $\mathbf{n}|_{\tau_s} = -\mathbf{n}|_{\tau_t}$, we have $g_{k,i}^{l,m} = -g_{k,j}^{l,n}$ and $h_{k,i}^{l,m} = -h_{k,j}^{l,n}$. Now we consider the situation depicted in figure 5.1, in which we have the overlapping edges i and j respectively at the interfaces between the couples of triangles (τ_{i_1}, τ_{i_2}) and (τ_{j_1}, τ_{j_2}) and the generic edge k at the interface of the triangles (τ_{k_1}, τ_{k_2}) .

In this case we find

$$\begin{aligned} [\mathbf{S}_{edge}]_{k,i} &= \int_{\tau_{k_1} \cup \tau_{k_2}} \int_{\tau_{i_1} \cup \tau_{i_2}} g_{i,j}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\ &= \sum_{q=\{k_1, k_2\}} \sum_{p=\{i_1, i_2\}} \int_{\tau_q} \int_{\tau_p} g_{i,j}^{q,p} d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\ &= - \sum_{q=\{k_1, k_2\}} \sum_{r=\{j_1, j_2\}} \int_{\tau_q} \int_{\tau_r} g_{i,j}^{q,p} d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\ &= - \int_{\tau_{k_1} \cup \tau_{k_2}} \int_{\tau_{j_1} \cup \tau_{j_2}} g_{i,j}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = -[\mathbf{S}_{edge}]_{k,j}. \end{aligned}$$

and

$$\begin{aligned} [\mathbf{S}_d]_{k,i} &= \int_{\tau_{k_1} \cup \tau_{k_2}} \int_{\tau_{i_1} \cup \tau_{i_2}} h_{i,j}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\ &= \sum_{q=\{k_1, k_2\}} \sum_{p=\{i_1, i_2\}} \int_{\tau_q} \int_{\tau_p} h_{i,j}^{q,p} d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\ &= - \sum_{q=\{k_1, k_2\}} \sum_{r=\{j_1, j_2\}} \int_{\tau_q} \int_{\tau_r} h_{i,j}^{q,p} d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = \\ &= - \int_{\tau_{k_1} \cup \tau_{k_2}} \int_{\tau_{j_1} \cup \tau_{j_2}} h_{i,j}(\mathbf{x}, \mathbf{y}) d\sigma(\mathbf{x}) d\sigma(\mathbf{y}) = -[\mathbf{S}_d]_{k,j}. \end{aligned}$$

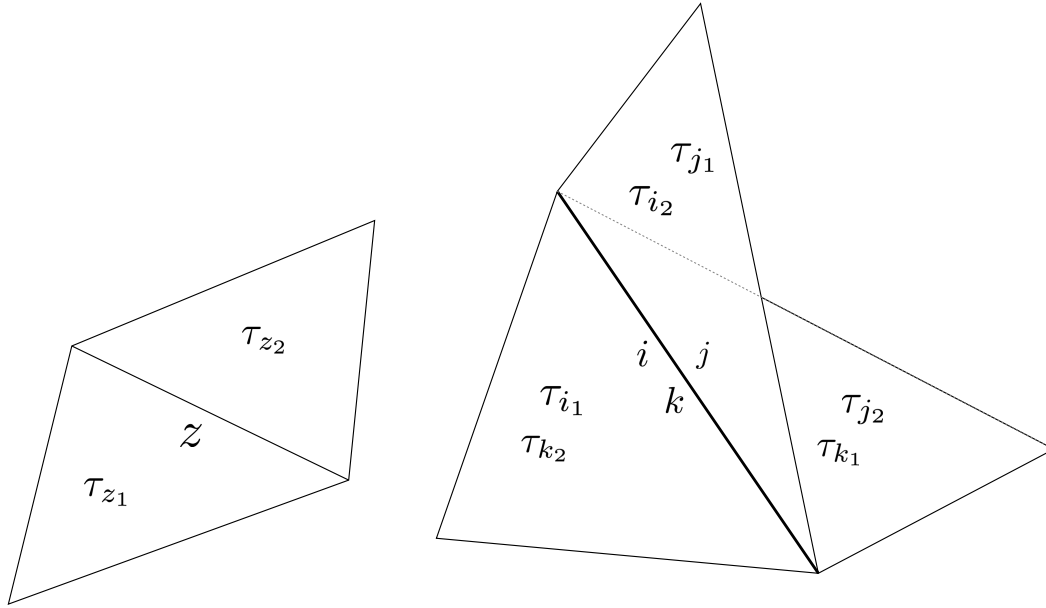


FIGURE 5.2: Representation of the supports of the basis functions associated with the edges i, j and k , which overlap at the junction and the generic edge z .

Thus, summing up it holds

$$[\mathbf{S}]_{k,i} = -[\mathbf{S}]_{k,j} \quad (5.1)$$

As discussed already in sections 2.1.2 and 3.2.2 this relation means that there is a kernel contribution for every couple of overlapping edges.

Edges Overlapping at the Junctions

We can also prove that there exists a linear dependence among the DoFs related to edges overlapping at a junction. We will consider a triple junction, but the reasoning can be extended to any kind junction. As it is depicted in figure 5.2 we suppose that the three edges i, j and k overlap at the junction and they are the interfaces between the couples of panels (τ_{i_1}, τ_{i_2}) , (τ_{j_1}, τ_{j_2}) and (τ_{k_1}, τ_{k_2}) respectively. Moreover, z is a generic edge and it is the interface between the couple of triangles (τ_{z_1}, τ_{z_2}) .

Now, with the same notation as in the previous case, we can write

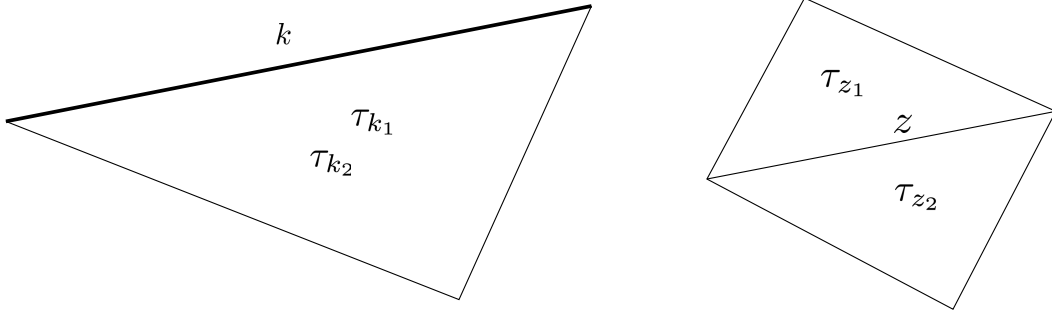


FIGURE 5.3: Representation of the support of the basis functions associated with a boundary edge k and a generic edge z .

$$\begin{aligned}
& [\mathbf{S}_{edge}]_{zi} + [\mathbf{S}_{edge}]_{zj} + [\mathbf{S}_{edge}]_{zk} = \sum_{q=\{z_1, z_2\}} \sum_{r=\{i_1, i_2\}} \int_{\tau_q} \int_{\tau_r} g_{zi}^{qr}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) dS(\mathbf{y}) + \\
& + \sum_{q=\{z_1, z_2\}} \sum_{s=\{j_1, j_2\}} \int_{\tau_q} \int_{\tau_s} g_{zj}^{qs}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) dS(\mathbf{y}) + \\
& \sum_{q=\{z_1, z_2\}} \sum_{t=\{k_1, k_2\}} \int_{\tau_q} \int_{\tau_t} g_{zk}^{qt}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) dS(\mathbf{y}) = \\
& = \sum_{q=\{z_1, z_2\}} \int_{\tau_q} \left[\underbrace{\int_{\tau_{i_1}} g_{zi}^{qi_1}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) + \int_{\tau_{k_2}} g_{zk}^{qk_2}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x})}_{=0 \text{ since } \tau_{i_1} \equiv \tau_{k_2} \text{ and } g_{zi}^{qi_1} = -g_{zk}^{qk_2}} \right. \\
& \left. + \int_{\tau_{j_1}} g_{zj}^{qj_1}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) + \int_{\tau_{i_2}} g_{zi}^{qi_2}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) + \right. \\
& \left. \underbrace{\int_{\tau_{k_1}} g_{zk}^{qk_1}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) + \int_{\tau_{j_2}} g_{zj}^{qj_2}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x})}_{=0 \text{ since } \tau_{j_1} \equiv \tau_{i_2} \text{ and } g_{zj}^{qj_1} = -g_{zi}^{qi_2}} \right] dS(\mathbf{y}) = 0, \\
& \underbrace{\int_{\tau_{k_1}} g_{zk}^{qk_1}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) + \int_{\tau_{j_2}} g_{zj}^{qj_2}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x})}_{=0 \text{ since } \tau_{k_1} \equiv \tau_{j_2} \text{ and } g_{zk}^{qk_1} = -g_{zj}^{qj_2}}
\end{aligned}$$

and

$$\begin{aligned}
[\mathbf{S}_d]_{zi} + [\mathbf{S}_d]_{zj} + [\mathbf{S}_d]_{zk} &= \sum_{q=\{z_1, z_2\}} \sum_{r=\{i_1, i_2\}} \int_{\tau_q} \int_{\tau_r} h_{zi}^{qr}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) dS(\mathbf{y}) + \\
&+ \sum_{q=\{z_1, z_2\}} \sum_{s=\{j_1, j_2\}} \int_{\tau_q} \int_{\tau_s} h_{zj}^{qs}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) dS(\mathbf{y}) + \\
&\sum_{q=\{z_1, z_2\}} \sum_{t=\{k_1, k_2\}} \int_{\tau_q} \int_{\tau_t} h_{zk}^{qt}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) dS(\mathbf{y}) = \\
&= \sum_{q=\{z_1, z_2\}} \int_{\tau_q} \left[\underbrace{\int_{\tau_{i_1}} h_{zi}^{qi_1}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) + \int_{\tau_{k_2}} h_{zk}^{qk_2}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x})}_{=0 \text{ since } \tau_{i_1} \equiv \tau_{k_2} \text{ and } h_{zi}^{qi_1} = -h_{zk}^{qk_2}} \right. \\
&+ \underbrace{\int_{\tau_{j_1}} h_{zj}^{qj_1}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) + \int_{\tau_{i_2}} h_{zi}^{qi_2}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x})}_{=0 \text{ since } \tau_{j_1} \equiv \tau_{i_2} \text{ and } h_{zj}^{qj_1} = -h_{zi}^{qi_2}} \\
&\left. + \underbrace{\int_{\tau_{k_1}} h_{zk}^{qk_1}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) + \int_{\tau_{j_2}} h_{zj}^{qj_2}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x})}_{=0 \text{ since } \tau_{k_1} \equiv \tau_{j_2} \text{ and } h_{zk}^{qk_1} = -h_{zj}^{qj_2}} \right] dS(\mathbf{y}) = 0.
\end{aligned}$$

Together these two formulae result in

$$[\mathbf{S}]_{zi} + [\mathbf{S}]_{zj} + [\mathbf{S}]_{zk} = 0, \quad (5.2)$$

which means that we expect a kernel contribution from every group of edges that overlap at a junction.

Edges at the Boundaries

Finally, we analyze the situation of the edges which are located at the boundaries of the screen. We consider an edge k which is located at the boundary of the screen and a generic edge z . The two are respectively at the interfaces between the triangles (τ_{k_1}, τ_{k_2}) and (τ_{z_1}, τ_{z_2}) (see figure 5.3). Then we have

$$\begin{aligned}
[\mathbf{S}_{edge}]_{zk} &= \sum_{p=\{z_1, z_2\}} \sum_{q=\{k_1, k_2\}} \int_{\tau_p} \int_{\tau_q} g_{zk}^{pq}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) dS(\mathbf{y}) = \\
&= \sum_{p=\{z_1, z_2\}} \int_{\tau_p} \left[\sum_{q=\{k_1, k_2\}} \int_{\tau_q} g_{zk}^{pq}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) \right] dS(\mathbf{y}) = \\
&= \sum_{p=\{z_1, z_2\}} \int_{\tau_p} \left[\underbrace{\int_{\tau_{k_1}} g_{zk}^{pk_1}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) + \int_{\tau_{k_2}} g_{zk}^{pk_2}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x})}_{=0 \text{ since } \tau_{k_1} \equiv \tau_{k_2} \text{ and } g_{zk}^{pk_1} = -g_{zk}^{pk_2}} \right] = 0,
\end{aligned}$$

and

$$\begin{aligned}
[\mathbf{S}_d]_{zk} &= \sum_{p=\{z_1, z_2\}} \sum_{q=\{k_1, k_2\}} \int_{\tau_p} \int_{\tau_q} h_{zk}^{pq}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) dS(\mathbf{y}) = \\
&= \sum_{p=\{z_1, z_2\}} \int_{\tau_p} \left[\sum_{q=\{k_1, k_2\}} \int_{\tau_q} h_{zk}^{pq}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) \right] dS(\mathbf{y}) = \\
&= \sum_{p=\{z_1, z_2\}} \int_{\tau_p} \left[\underbrace{\int_{\tau_{k_1}} h_{zk}^{pk_1}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x}) + \int_{\tau_{k_2}} h_{zk}^{pk_2}(\mathbf{x}, \mathbf{y}) dS(\mathbf{x})}_{=0 \text{ since } \tau_{k_1} \equiv \tau_{k_2} \text{ and } h_{zk}^{pk_1} = -h_{zk}^{pk_2}} \right] = 0,
\end{aligned}$$

from which we see that

$$[\mathbf{S}]_{zk} = 0.$$

Analogously to the previous cases, due to this last observation we will exclude from the computations the DoFs at the boundaries, as the relative entries are always null. All things considered, in our numerical experiments we expect to obtain a kernel

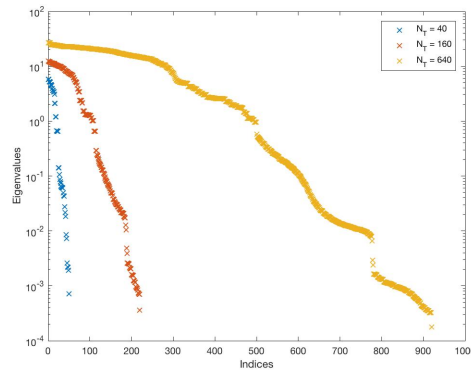


FIGURE 5.4: The magnitude of the eigenvalues of \mathbf{S} computed on a disk on a hierarchy of nested meshes.

contribution from each couple of overlapping DoFs and for each triplet or quadruplet of edges which overlap at a junction.

At this point it is important to observe that in this case counting the "approximated kernel dimensions" as done in section 3.2.2 is not possible. The reason is that due to the impact of numerical quadrature which we already discussed in chapter 3 and, due to the peculiar eigenvalue structure of \mathbf{S} which is indefinite, we are not able to distinguish two clusters of eigenvalues. In figure 5.4 we show the magnitude of the eigenvalues of \mathbf{S} , when using a disk mesh. We clearly see that the eigenvalues are uniformly distributed and there is no way to distinguish the approximate kernel dimensionalities by inspecting the eigenvalues. For this reason the approximate kernel dimensionality has to be counted using a different strategy based on the fact that the impact of the numerical quadrature is only relevant when the entries are computed by approximating integrals with singular integrands. This only happens when the integrals are performed on neighbouring or overlapping panels. For most of the entries, then, the integrals are accurate and relations (5.1) and (5.2) are satisfied up to machine precision. Therefore, the approximate kernel dimensionality is the number of rows or columns which satisfy the relations (5.1) and (5.2) up to a small number of entries which are those involving the inaccurate singular integrals. With this strategy we are still able to count the approximate kernel dimensionalities and the results, which confirm the theoretical expectations in terms of kernel dimensions, are reported in table 5.1.

5.2 Test for CG and GMRES

In this section we want to test the capability of iterative solvers to find the solution of the discrete EFIE. In first we will consider a static version of the EFIE, which involves matrices which are positive semi-definite, and after that we will consider the regular discrete EFIE, for which the system matrix is indefinite. In all the tests of this section the stopping criterion is based on the relative residual with a tolerance of 10^{-5} .

Disk			
N_T	40	160	640
DoFs away from the junction	50	220	920
DoFs at the junction	0	0	0
Expected kernel dimensions	25	110	460
Approximate kernel dimensions	25	110	460
Triple Junction			
N_T	24	96	384
DoFs away from the junction	24	120	540
DoFs at the junction	3	6	12
Expected kernel dimensions	13	62	268
Approximate kernel dimensions	13	62	268
Quadruple Junction			
N_T	32	256	512
DoFs away from the junction	32	168	704
DoFs at the junction	4	8	16
Expected kernel dimensions	17	82	356
Approximate kernel dimensions	17	82	356

TABLE 5.1: Dimension of the kernel of \mathbf{S} when using a mesh with N_T triangular elements

5.2.1 The Static EFIE

First we will set $\kappa = 0$ and consider the static version of the EFIE, which reads

$$\begin{aligned} \ll \gamma_T \cdot \mathcal{G}_0 * \gamma'_T(\mathbf{p}), \mathbf{q} \gg_x + \ll \gamma_D \cdot \mathcal{G}_0 * \gamma'_D(\text{curl}_\Gamma \mathbf{p}), \text{curl}_\Gamma \mathbf{q} \gg = \ll \mathbf{f} \mathbf{q} \gg_x \\ \forall \mathbf{p} \in \mathbb{H}^{-\frac{1}{2}}(\text{curl}_\Gamma, \Gamma). \end{aligned} \quad (5.3)$$

Equation (5.3) can be discretized using the same ideas as for the regular EFIE, giving

$$\underbrace{(\mathbf{S}_{edge} + \mathbf{S}_d)}_{\mathbf{S}_0} \vec{p} = \mathbf{M} \vec{f}.$$

Unlike the regular EFIE system, the system matrix for the bilinear form of the static EFIE is positive definite, and, thus, we can solve the system using both CG and GMRES as done in chapters 2 and 3. From the point of view of the kernel dimensions nothing changes and we could apply the ideas of the previous section to the system matrix of the static EFIE and we would obtain the same results. To prove the solvability of the systems associated with \mathbf{S}_{edge} , we construct a right-hand side vector in the image of \mathbf{S}_{edge} as

$$\mathbf{r}_{\mathbf{S}_0} = \mathbf{S}_0 \mathbf{z}$$

with \mathbf{z} being a random vector.

Then we solve the system

$$\mathbf{S}_0 \mathbf{x} = \mathbf{r}_{\mathbf{S}_0}$$

which has infinitely many solutions.

We report condition number of \mathbf{S}_0 together with the number of iterations to convergence for CG and GMRES in table 5.2.

We can observe that for both the solvers the number of iterations is not as small as expected. We can explain this by observing once more that \mathbf{S}_0 does not have any null

Disk			
N_T	40	160	640
condition number	8.1370e+03	3.4933e+04	1.5111e+05
#iterations for CG	82	635	3361
#iterations for GMRES	47	180	617
Triple Junction			
N_T	24	96	384
condition number	5.1662e+02	1.5610e+03	6.6056e+03
#iterations for CG	21	129	759
#iterations for GMRES	20	90	330
Quadruple Junction			
N_T	32	128	512
condition number	5.4696e+02	1.6086e+03	6.6781e+03
#iterations for CG	27	154	901
#iterations for GMRES	25	110	422

TABLE 5.2: Number of iterations to convergence for the static EFIE system with non corrected entries

Disk			
N_T	40	160	640
#iterations for CG	17	38	62
#iterations for GMRES	18	39	63
Triple Junction			
N_T	24	96	384
#iterations for CG	10	25	53
#iterations for GMRES	11	26	49
Quadruple Junction			
N_T	32	128	512
#iterations for CG	12	77	399
#iterations for GMRES	13	71	238

TABLE 5.3: Number of iterations to convergence for the static EFIE system with corrected entries

eigenvalue due to the numerical inaccuracies. Moreover in this case the eigenvalues which should be null are negative and for this reason CG struggles even more than GMRES to find a solution.

In order to observe what would happen when eliminating the inaccuracies, we corrected the \mathbf{S}_0 by forcing their entries to satisfy the relations (5.1) and (5.2) and repeated the numerical experiments. As we can see from the results reported in table 5.3, the number of iterations required to find a solution is now much smaller.

5.2.2 The Indefinite EFIE

We now consider the indefinite version of the EFIE, and we set $\kappa = 1$. The linear system we aim to solve is

$$\underbrace{(\mathbf{S}_{edge} - \mathbf{S}_d)}_{\mathbf{S}} \mathbf{x} = \mathbf{r}_s.$$

Disk			
N_T	40	160	640
condition number	8.0738e+03	3.4855e+04	1.5103e+05
#iterations for GMRES	46	155	458
Triple Junction			
N_T	24	96	384
condition number	4.4801e+03	2.6702e+03	9.9557e+03
#iterations for GMRES	19	105	311
Quadruple Junction			
N_T	32	128	512
condition number	4.8026e+03	2.8053e+03	1.0428e+04
#iterations for GMRES	27	129	396

TABLE 5.4: Number of iterations to convergence for the EFIE system with non corrected entries

Disk			
N_T	40	160	640
#iterations for GMRES	24	75	197
Triple Junction			
N_T	24	96	384
#iterations for GMRES	14	63	207
Quadruple Junction			
N_T	32	128	512
#iterations for GMRES	19	80	279

TABLE 5.5: Number of iterations to convergence for the corrected EFIE system with corrected entries

The matrix \mathbf{S} is indefinite and CG is not supposed to converge in this case. On the other hand, GMRES should still converge to a solution in a number of iterations smaller than the system size since \mathbf{S} is symmetric. We tested the convergence of the GMRES on our numerically computed matrices, the results are reported in table 5.4 together with the condition number of \mathbf{S} . Also in this case the computed \mathbf{S} is polluted by numerical inaccuracies in the integration by quadrature, which slows down the convergence of GMRES. Also in this case we post-processed the matrices and tested GMRES with the corrected \mathbf{S} . With this procedure also in this case we have a faster convergence of the iterative solver, as shown by the results in table 5.5. Anyway, the number of iterations is higher than in the static case as GMRES is slower when dealing with indefinite matrices.

Conclusion

To conclude, we would like to recapitulate the content of the thesis. We have applied a Galerkin discretization to the BIEs which arise in the acoustic and the electromagnetic scattering in presence of multi-screens. Due to the use of multi-trace spaces instead of jump spaces in the BIEs, the uniqueness of the solutions to these BIEs is lost, but we have shown that GMRES and CG are still able to find a solution of these BIEs. In addition we have shown with numerical examples how the convergence iterative solvers is not influenced by the null eigenvalues of the BEM matrices.

Moreover, we have shown that it is possible to suppress the redundant DoFs and reduce the computational burden of CG and GMRES.

Unfortunately, it has not been possible to solve some issues related to the numerical quadrature in presence of singular integrands in BETL2 and some further investigations should be done to understand how the inaccuracies can be avoided.

Future research will aim at extending our approach to composite geometries formed by the union of many junctions and generic domains.

Appendix A

The Matlab Codes

In this appendix we give an overview of the routines and scripts we implemented to obtain the results reported in Chapter 2. These codes have been included to the Git-Lab repository https://gitlab.math.ethz.ch/curzuato/matlab_multiscreen.git currently in the branch `multiscreen_tests`.

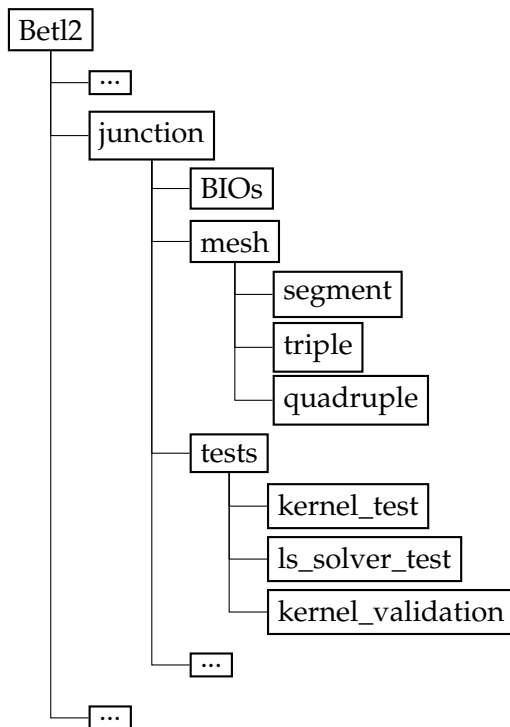


FIGURE A.1: Structure of the multiscreen_matlab Gitab repository

Computation of V and W

The two routines which we use to compute the matrices V and W are the following.

LISTING A.1: `BIOs/varV0_ms.m`

```

1 function [MVQ] = varV0_ms( X, elems, dofs, Nq)
2 % Carolina A. Urzua Torres - SAM, ETH, Zuerich.
3 %
4 % varV0_ms - Compute the Galerkin matrix corresponding to the
5 %           variational weakly singular for Laplace eq. on a triple
6 %           multi-screen using Gauss quadrature. Test and trial
7 %           functions correspond to p.w.c. Implementation works for

```

```

 8 %           both single and multi trace approach.
 9 %           ENFORCES SYMMETRY.
10 %
11 % INPUT:
12 % - X       : vector containing the coordinates of the mesh points.
13 % - elems   : indices of the vertices of each mesh element.
14 % - dofs    :
15 % - Nq      : Number of quadrature points to be used
16 %
17 % OUTPUT:
18 % - MVQ     : Galerkin matrix corresponding to variational V.
19
20 N = size(dofs,1); %number of elements
21
22 MVQ = zeros(N); % coefficient matrix L1_ij = < L1 psi_i, psi_j>
23
24 % Get quadrature points and weights
25 [xx, w1]=GL(Nq);
26 [yy, w2]=GL(Nq+1);
27
28 % create function handle with the kernel
29 f=@(x,y) -log(norm(x-y));
30
31 for i=1:N
32     %get indices of elements in which phi_i is supported
33     dofs_i = dofs{i};
34     for di=1:size(dofs_i,2)
35         %define element e_i
36         e_i = X(elems(dofs_i(di),1:2),:);
37         for j=1:i
38             %get indices of elements in which phi_j is supported
39             dofs_j = dofs{j};
40
41             for dj=1:size(dofs_j,2)
42                 %define element e_j
43                 e_j = X(elems(dofs_j(dj),1:2),:);
44
45                 MVQ(i,j) = MVQ(i,j) + VDGLQ(e_i(1,:), e_i(2,:), ...
46                     e_j(1,:), e_j(2,:),xx, yy, w1, w2, f);
47                 MVQ(j,i) = MVQ(i,j);
48             end
49         end
50     end
51 end
52
53
54 end

```

LISTING A.2: BIOs/varW0_ms.m

```

1 function [ W ] = varW0_ms( X, elems, dofs, Nq)
2 % Carolina A. Urzua Torres - SAM, ETH, Zuerich.
3 %
4 % varW0_ms - Compute the Galerkin matrix corresponding to the
5 %           variational hypersingular for Laplace eq. on a triple
6 %           multi-screen using Gauss quadrature. Test and trial
7 %           functions correspond to p.w.lin Implementation works for
8 %           both single and multi trace approach.
9 %           ENFORCES SYMMETRY.
10 %

```

```

11 % INPUT:
12 % - X      : vector containing the coordinates of the mesh points.
13 % - elems  : indices of the vertices of each mesh element.
14 % - dofs   :
15 % - Nq     : Number of quadrature points to be used
16 %
17 % OUTPUT:
18 % - W      : Galerkin matrix corresponding to variational W.
19
20 N = size(dofs,1); %number of points
21 %Nl = (length(X)+2)/3; % number of points per leg
22
23 % Since we are working with p.w.linear test and trial spaces, u_i' and
24 % u_j' are constants. Then, we calculate E_ij entries by
25 %
26 %           W_ij = < L1 u_i', v_j' > = u_i' * v_j' * M1_ij,
27 %
28 % where M1 is the Galerkin Matrix corresponding to the variational
29 % form of the weakly singular operator V.
30 W = zeros(N); % coefficient matrix L1_ij = < L1 psi_i, psi_j >
31 % zero bc at tip points
32
33 % Get quadrature points and weights
34 [xx, w1]=GL(Nq);
35 [yy, w2]=GL(Nq+1);
36
37 % create function handle with the kernel
38 f=@(x,y) -log(norm(x-y));
39
40 for i=1:N
41     %get indices of elements in which phi_i is supported
42     dofs_i = dofs{i};
43     %get indices of vertices that compose each element in the support
44     %of phi_i
45     els_i = [elems(dofs_i(1),1:2); %e_{i-1}
46             elems(dofs_i(2),1:2)]; %e_{i}
47     %compute derivatives
48     dphi_i = [ 1/norm(X(els_i(1,2),:) - X(els_i(1,1),:));
49              -1/norm(X(els_i(2,2),:) - X(els_i(2,1),:))];
50
51     % if i=1 and we are with ST, we actually have 3 elements. Complete!
52     if(i==1 && size(dofs{i},2)>2)
53         %disp(['ST'])
54         %correct sign
55         dphi_i(1) = -dphi_i(1);
56         %add extra element (and derivative)
57         els_i = [els_i; elems(dofs_i(3),1:2)];
58         dphi_i = [dphi_i; -1/norm(X(els_i(3,2),:) - X(els_i(3,1),:))];
59     end
60
61     for j=1:i
62         %get indices of elements in which phi_i is supported
63         dofs_j = dofs{j};
64         %get indices of vertices that compose each element in the
65         %support of phi_j
66         els_j = [elems(dofs_j(1),1:2); %e_{j-1}
67                 elems(dofs_j(2),1:2)]; %e_{j}
68         %compute derivatives
69         dphi_j = [ 1/norm(X(els_j(1,2),:) - X(els_j(1,1),:));
70                  -1/norm(X(els_j(2,2),:) - X(els_j(2,1),:))];
71
72         % if j=1 and we are with ST, we actually have 3 elements. Complete!
73         if(j==1 && size(dofs{j},2)>2)

```

```

74     %correct sign
75     dphi_j(1) = -dphi_j(1);
76     %add extra element (and derivative)
77     els_j = [els_j; elems(dofs_j(3),1:2)];
78     dphi_j = [dphi_j; -1/norm(X(els_j(3,2),:) - X(els_j(3,1),:))];
79     end
80
81     I1 = 0;
82     %compute and add all contributions!
83     for ki=1:size(els_i,1)
84         for kj=1:size(els_j,1)
85
86             I1 = I1 + dphi_i(ki)*dphi_j(kj)*VDGLQ( ...
87                 X(els_i(ki,1),:), ...
88                 X(els_i(ki,2),:), X(els_j(kj,1),:), ...
89                 X(els_j(kj,2),:), ...
90                 xx, yy, w1, w2, f);
91         end
92     end
93     W(i,j) = I1;
94     W(j,i) = I1;
95
96     end
97 end
98
99
100 end

```

The Meshes

We wrote the routines which build the meshes used in the numerical experiments. We needed both single-trace and a multi-trace meshes for the segment and only multi-trace meshes for the junctions.

LISTING A.3: mesh/segment/segmentST_mesh.m

```

1 function [ X, elems, dofs] = segmentST_mesh( a, b, N, order, plotmesh)
2 % Lorenzo Giacomel - SAM, ETH, Zuerich.
3 %
4 % segmentST_mesh - Compute Mesh for a simple flat screen. It uses ...
5 % single trace approach and it assumes that the screen is ...
6 % located on the x axis.
7 %
8 % %INPUT:
9 % - a : x coordinate of the left tip.
10 % - b : x coordinate of the right tip.
11 % - N : Number of points.
12 % - order : 0 if primal mesh is discretized by p.w.constants.
13 % : 1 if discretized by p.w.linear
14 %
15 % %OUTPUT:
16 % - X : (3N1-2)x2 vector containing primal mesh points ...
17 % coordinates for standard screen
18 % - elems : 3(N1-1)x3 vector containing elements' information. The ...
19 % two columns

```



```

18 %           give the indices of the vertices that conform the ...
    current element.
19 % - dofs   : cell-array returning dofs indices.
20 %           In case of order 0, it gives the indices of the vertices
21 %           composing the element.
22 %           In case of order 1, it gives the indices of the elements
23 %           composing the support of the p.w.linear function.
24
25
26 X = [linspace(a, b, N)' zeros(N, 1)];
27
28 elems = [[1 : N - 1]', [2 : N]'];
29
30 if(order == 0)
31     dofs = mat2cell([1 : N - 1]', ones(1, N - 1), 1);
32
33 elseif(order == 1)
34     aux = (1 : length(X))';
35     doftmp = [aux - 1 aux];
36     % General dofs formula
37     dofs = mat2cell(doftmp, ones(1, length(X)), 2);
38     % Correct first and last row
39     dofs{1} = [1];
40     dofs{N} = [N - 1];
41
42 else
43     disp("Only order 0 and 1 implemented")
44     elems = 0;
45     X = 0;
46 end
47
48 if(plotmesh)
49     figure;
50     plot(X(: , 1), X(: , 2), 'ob')
51     for i = 1 : 1 : length(X)
52         t(i) = text(X(i, 1) + 0.02, X(i, 2) , num2str(i));
53     end
54     for i = 1 : 1 : length(elems)
55         text((X(elems(i, 1), 1) + X(elems(i, 2), 1)) / 2, ...
              (X(elems(i, 1), 2) + X(elems(i, 2), 2)) / 2, ...
              num2str(i), 'fontw', 'b')
56     end
57     set(t(:), 'FontSize', 8, 'color', 'blue');
58 end
59
60
61 end

```

LISTING A.4: mesh/segment/segmentMT_mesh.m

```

1 function [ X, elems, dofs] = segmentMT_mesh( a, b, N, order, eps, ...
    plotmesh)
2 % Lorenzo Giacomel - SAM, ETH, Zuerich.
3 %
4 % segmentMT_mesh - Compute Mesh for a simple flat screen. It uses ...
    multi trace
5 %           approach and it assumes that the screen is ...
    located on the x axis.
6 %
7 %INPUT:
8 % - a       : x coordinate of the left tip.

```

```

 9 % - b      : x coordinate of the right tip.
10 % - N      : Number of points.
11 % - order  : 0 if primal mesh is discretized by p.w.constants.
12 %           1 if discretized by p.w.linear
13 % - eps    : thickness of the mesh
14 %
15 %OUTPUT:
16 % - X      : (3N1-2)x2 vector containing primal mesh points ...
              coordinates for
17 %           standard screen
18 % - elems  : 3(N1-1)x3 vector containing elements' information. The ...
              two columns
19 %           give the indices of the vertices that conform the ...
              current element.
20 % - dofs   : cell-array returning dofs indices.
21 %           In case of order 0, it gives the indices of the vertices
22 %           composing the element.
23 %           In case of order 1, it gives the indices of the elements
24 %           composing the support of the p.w.linear function.
25
26 aux1 = linspace(a + (b - a) / (N - 1), b - (b - a) / (N - 1), N ...
              - 2)';
27 X = [a,      0;
28      aux1,  -eps * ones(N - 2, 1);
29      b,      0;
30      flipud(aux1)  eps * ones(N - 2, 1)];
31 Ntot = 2 * N - 2;
32 elems = [[1 : Ntot - 1]', [2 : Ntot]'];
33         Ntot, 1];
34
35 if(order == 0)
36     dofs = mat2cell([1 : Ntot]', ones(1, Ntot), 1);
37
38 elseif(order == 1)
39     aux2 = (1 : length(X))';
40     doftmp = [aux2 - 1 aux2];
41     % General dofs formula
42     dofs = mat2cell(doftmp, ones(1, length(X)), 2);
43     dofs{1} = [Ntot, 1];
44 else
45     disp("Only order 0 and 1 implemented")
46     elems = 0;
47     X = 0;
48 end
49
50 if(plotmesh)
51     figure;
52     plot([X( : , 1); X(1, 1)], [X( : , 2); X(1, 2)], '-ob')
53     for i = 1 : 1 : length(X)
54         t(i) = text(X(i, 1) + 0.02, X(i, 2) ,num2str(i));
55     end
56     for i=1 : 1 : length(elems)
57         text((X(elems(i, 1), 1) + X(elems(i, 2), 1)) / 2, ...
              (X(elems(i, 1), 2) + X(elems(i, 2), 2)) / 2, ...
              num2str(i), 'fontw', 'b')
58     end
59     set(t(:), 'FontSize', 8, 'color', 'blue');
60 end
61
62
63 end

```

LISTING A.5: mesh/triple/tripleJMT_mesh.m

```

1 function [X, elems, dofs] = tripleJMT_mesh( tips_c, Nl, order, e, ...
    plotmesh)
2 % Carolina A. Urzua Torres - SAM, ETH, Zuerich.
3 % Compute Mesh for triple multiscreen. It uses multi-trace approach ...
    (fat screen)
4 % and it assumes that the junction point is located in the origin.
5 %
6 %INPUT:
7 % - tips_c : 3x2 vector containing (x,y) coordinates of the three ...
    tips/end points
8 % - Nl      : Number of points per leg.
9 % - order   : 0 if primal mesh is discretized by p.w.constants.
10 %           : 1 if discretized by p.w.linear
11 % - e       : thickness of inflated multi-screen
12 %
13 %OUTPUT:
14 % - X       : (3(2Nl-1))x2 vector containing primal mesh points ...
    coordinates for
15 %           "inflated" multi-screen
16 % - elems   : (6Nl-4)x3 vector containing elements' information. ...
    First two columns
17 %           give the indices of the vertices that conform the ...
    current element,
18 %           and the third column states to which leg the element ...
    belongs.
19 % - dofs    : cell-array returning dofs indices.
20 %           In case of order 0, it gives the indices of the vertices
21 %           composing the element.
22 %           In case of order 1, it gives the indices of the elements
23 %           composing the support of the p.w.linear function.
24
25 if(nargin < 4)
26     e = 0;
27 end
28
29 if(nargin < 5)
30     plotmesh = false;
31 end
32 %{
33 disp(['Computing inflated multi-screen mesh with ', num2str(Nl), ...
34     ' points per leg.'])
35 disp(['This means ', num2str(6*(Nl-1)), ' mesh points in total.'])
36 %}
37
38 %auxiliary vector
39 scaling = linspace(0+e,1,Nl)';
40
41 % get tips' angles
42 theta = atan2(tips_c(:,2),tips_c(:,1));
43
44 % direction vector (to construct legs)
45 %dir = repmat([norm(tips_c(1,:)); norm(tips_c(2,:)); ...
46 %           norm(tips_c(3,:))],1,2).*[cos(theta) sin(theta)]
47 dir = tips_c;
48 % get mesh nodes with origins (un-inflated)
49 P = [scaling*dir(1,:) scaling*dir(2,:) scaling*dir(3,:)];
50
51 % compute direction tangent to leg (to inflate screen)
52 dirtan = zeros(Nl-2,6);
53 for i=1:1:3

```

```

54     dirtan(:,2*i-1:2*i) = e*repmat([-sin(theta(i)) ...
        cos(theta(i))],Nl-2,1);
55 end
56
57 %create vertices for inflated screen
58 X = [ %inflate first leg
59     e*[cos(theta(2)-pi) sin(theta(2)-pi)]
60     P(2:Nl-1,1:2)-dirtan(:,1:2);
61     P(Nl,1:2); %endpoint is not duplicated
62     flipud(P(2:Nl-1,1:2))+dirtan(:,1:2);
63     %inflate second leg
64     e*[cos(theta(3)-pi) sin(theta(3)-pi)]
65     P(2:Nl-1,3:4)-dirtan(:,3:4);
66     P(Nl,3:4) %endpoint is not duplicated
67     flipud(P(2:Nl-1,3:4))+dirtan(:,3:4);
68     %inflate third leg
69     e*[cos(theta(1)-pi) sin(theta(1)-pi)]
70     P(2:Nl-1,5:6)-dirtan(:,5:6)
71     P(Nl,5:6) %endpoint is not duplicated
72     flipud(P(2:Nl-1,5:6))+dirtan(:,5:6)] ;
73
74 elems = [[1:2*(Nl-1)]'      [2:(2*Nl-1)]'      repmat(1,2*Nl-2,1);
75          [(2*Nl-1):4*(Nl-1)]' [2*Nl:(4*Nl-3)]'      repmat(2,2*Nl-2,1);
76          [(4*Nl-3):(6*Nl-7)]' [(4*Nl-2):6*(Nl-1)]' repmat(3,2*Nl-3,1);
77          6*(Nl-1)           1           3           ];
78
79 % compute dofs vector
80 if(order == 0)
81     % p.w.c.: dofs contains the vertices indices for each element
82     % i.e. e_i=[x(i), x(i+1)], such that it avoids fictional elements
83     % (tips, etc)
84     %not really necessary but implemented for consistency
85     N = size(elems,1);
86     dofs = mat2cell([1:N]',ones(1,N),1);
87
88     %{
89     disp([num2str(length(elems)),' p.w.constants dofs have been ...
90         created.'])
91     %}
92 else if(order ==1)
93     % p.w.linear
94     aux = (1:length(X))';
95     doftmp = [aux-1 aux];
96     dofs = mat2cell(doftmp,ones(1,length(doftmp)),2);
97     %correct first row
98     dofs{1} = [length(X) 1];
99
100    %{
101    disp([num2str(length(dofs)),' p.w.linear dofs have been created.'])
102    %}
103    else
104        disp('Sorry, not implemented')
105        elems=0;
106    end
107 end
108 if(plotmesh)
109     figure;
110     plot([X(:,1);X(1,1)], [X(:,2);X(1,2)], '-ob')
111     hold on
112     for i=1:length(X)
113         t(i) = text(X(i,1)+0.02, X(i,2) ,num2str(i));
114     end

```

```

115     for i=1:1:length(elems)
116         text((X(elems(i,1),1)+X(elems(i,2),1))/2, ...
              (X(elems(i,1),2)+X(elems(i,2),2))/2 ...
              ,num2str(i),'fontw','b')
117     end
118         set(t(:),'FontSize',8, 'color', 'blue');
119 end
120
121 end

```

LISTING A.6: mesh/quadruple/quadrupleJMT_mesh.m

```

1 function [X, elems, dofs] = quadrupleJMT_mesh( l, Nl, order, eps, ...
      plotmesh)
2 % Lorenzo Giacomel - SAM, ETH, Zuerich.
3 %
4 % quadrupleJMT_mesh - Compute Mesh for quadruple multiscreen. It uses
5 % multi-trace approach (fat screen) and it assumes
6 % that the junction point is located at the origin.
7 %
8 %INPUT:
9 % - l      : length of each leg of the cross
10 % - Nl     : Number of points per leg.
11 % - order  : 0 if primal mesh is discretized by p.w.constants.
12 %           1 if discretized by p.w.linear
13 % - e      : thickness of inflated multi-screen
14 %
15 %OUTPUT:
16 % - X      : (8Nl-8)x2 vector containing primal mesh points ...
      coordinates for
17 %           "inflated" multi-screen
18 % - elems  : (8Nl-8)x3 vector containing elements' information. The ...
      two columns
19 %           give the indices of the vertices that conform the ...
      current element.
20 % - dofs   : cell-array returning dofs indices.
21 %           In case of order 0, it gives the indices of the vertices
22 %           composing the element.
23 %           In case of order 1, it gives the indices of the elements
24 %           composing the support of the p.w.linear function.
25
26 if(nargin < 4)
27     e = 0;
28 end
29
30 if(nargin < 5)
31     plotmesh = false;
32 end
33 %{
34 disp(['Computing inflated multi-screen mesh with ',num2str(Nl), ...
35      ' points per leg.'])
36 disp(['This means ', num2str(8*Nl-8),' mesh points in total.'])
37 %}
38 %auxiliary vector
39 aux_x = [linspace(1 / (Nl - 1), 1 - 1 / (Nl - 1), Nl - 2)' , ...
      zeros(Nl - 2, 1)];
40 aux_y = [zeros(Nl - 2, 1), linspace(1 / (Nl - 1), 1 - 1 / (Nl - ...
      1), Nl - 2)'];
41 % compute direction tangent to leg (to inflate screen)
42
43

```

```

44 %create vertices for inflated screen
45 X = [ %inflate first leg
46     -eps , eps;
47     -aux_x + repmat([-eps, eps], N1 - 2, 1);
48     -1, 0; %endpoint is not duplicated
49     -flipud(aux_x) + repmat([-eps, -eps], N1 - 2, 1);
50     %inflate second leg
51     -eps, -eps;
52     -aux_y + repmat([-eps,-eps],N1 - 2, 1);
53     0, -1;%endpoint is not duplicated
54     -flipud(aux_y) + repmat([eps, -eps], N1 - 2, 1);
55     %inflate third leg
56     eps, -eps;
57     aux_x + repmat([eps, -eps], N1 - 2, 1);
58     1, 0;%endpoint is not duplicated
59     flipud(aux_x) + repmat([eps, eps], N1 - 2, 1);
60     %inflate fourth leg
61     eps , eps;
62     aux_y + repmat([eps, eps], N1 - 2, 1);
63     0, 1;%endpoint is not duplicated
64     flipud(aux_y) + repmat([-eps, eps], N1 - 2, 1)];
65
66 elems = [(1:2*(N1-1))'      (2:2*N1-1)'      ones(2*N1-2,1);
67          (2*N1-1:4*N1-4)'  (2*N1:4*N1-3)'  repmat(2,2*N1-2,1);
68          (4*N1-3:6*N1-6)'  (4*N1-2:6*N1-5)'  repmat(3,2*N1-2,1);
69          (6*N1-5:8*N1-9)'  (6*N1-4:8*N1-8)'  repmat(4,2*N1-3,1);
70          8*N1-8            1                        4];
71
72 % compute dofs vector
73 if(order == 0)
74     % p.w.c.: dofs contains the vertices indices for each element
75     % i.e. e_i=[x(i), x(i+1)], such that it avoids fictional elements
76     % (tips, etc)
77     % not really necessary but implemeted for consistency
78     N = size(elems, 1);
79     dofs = mat2cell([1:N]', ones(1, N), 1);
80
81     %{
82     disp([num2str(length(elems)),' p.w.constants dofs have been ...
83         created.'])
84     %}
85 else if(order ==1)
86     % p.w.linear
87     aux = (1 : length(X))';
88     doftmp = [aux - 1 aux];
89     dofs = mat2cell(doftmp, ones(1, length(doftmp)), 2);
90     %correct first row
91     dofs{1} = [length(X) 1];
92     %{
93     disp([num2str(length(dofs)),' p.w.linear dofs have been created.'])
94     %}
95     else
96         disp('Sorry, not implemented')
97         elems = 0;
98     end
99 end
100
101 if(plotmesh)
102     figure;
103     plot([X( : , 1); X(1, 1)], [X( : , 2); X(1, 2)], '-ob')
104     hold on
105     for i = 1 : 1 : length(X)

```

```

106         t(i) = text(X(i, 1) + 0.02, X(i, 2), num2str(i));
107     end
108     for i = 1 : 1 : length(elems)
109         text((X(elems(i, 1), 1) + X(elems(i, 2), 1)) / 2, ...
              (X(elems(i, 1), 2) + X(elems(i, 2), 2)) / 2, ...
              num2str(i), 'fontw', 'b')
110     end
111     set(t(:), 'FontSize', 8, 'color', 'blue');
112 end
113
114 end

```

For the hyper-singular operator we excluded the exterior DoFs from the computations. We, therefore, implemented the following routines which prune the cells corresponding to the tip DoFs and return the reduced array of cells containing the new DoFs.

LISTING A.7: mesh/segment/segmentST_innerDofs.m

```

1 function [dofs] = segmentST_innerDofs( dofs, N, order)
2 % Lorenzo Giacomel - SAM, ETH, Zuerich.
3 %
4 % segmentST_innerDofs - Extract the interior DoFs from the DoFs of a
5 %                       single-trace segment mesh.
6 %INPUT:
7 % - dofs    : cell-array from which to extract the interior DoFs
8 % - N       : Number of points.
9 % - order   : 0 if primal mesh is discretized by p.w.constants.
10 %            1 if discretized by p.w.linear
11 %OUTPUT:
12 % - dofs    : cell-array returning interior DoFs indices.
13
14     if (N <= 3)
15         disp('Impossible to retrieve inner dofs for N <= 3')
16     end
17     if (order == 1)
18         %remove dofs cells corresponding to vertices on tips
19         dofs(N) = [];
20         dofs(1) = [];
21     else
22         disp('Implemented only for first order')
23     end
24
25 end

```

LISTING A.8: mesh/segment/segmentMT_innerDofs.m

```

1 function [idofs] = segmentMT_innerDofs( dofs, N, order)
2 % Lorenzo Giacomel - SAM, ETH, Zuerich.
3 %
4 % segmentMT_innerDofs - Extract the interior DoFs from the DoFs of a
5 %                       multi-trace segment mesh.
6 %INPUT:
7 % - dofs    : cell-array from which to extract the interior DoFs
8 % - N       : Number of points.
9 % - order   : 0 if primal mesh is discretized by p.w.constants.
10 %            1 if discretized by p.w.linear
11 %OUTPUT:
12 % - dofs    : cell-array returning interior DoFs indices.
13

```

```

14     if (N <= 2)
15         disp('Impossible to retrieve inner dofs for N <= 3')
16     else
17         idofs = dofs;
18         if (order == 1)
19             %remove dofs cells corresponding to vertices on tips
20             idofs(N) = [];
21             idofs(1) = [];
22         else
23             disp('Implemented only for first order')
24         end
25     end
26
27 end

```

LISTING A.9: mesh/triple/tripleJMT_innerDofs.m

```

1 function [ dofs] = tripleJMT_innerDofs( dofs, N1, order)
2 % Carolina A. Urzua Torres - SAM, ETH, Zuerich.
3
4 if (order==1)
5     %remove dofs cells corresponding to vertices on tips
6     dofs(5*N1-4) = [];
7     dofs(3*N1-2) = [];
8     dofs(N1) = [];
9
10    %since new cell array has one elements less than original
11    %since new cell array has two elements less than original
12
13 else
14     disp('Sorry, not implemented.')
15 end
16
17 end

```

LISTING A.10: mesh/quadruple/quadrupleJMT_mesh.m

```

1 function [ dofs] = quadrupleJMT_innerDofs( dofs, N1, order)
2 % Lorenzo Giacomel - SAM, ETH, Zuerich.
3 %
4 % quadrupleJMT_innerDofs - Extract the interior DoFs from the DoFs ...
5 %   of a
6 %   multi-trace quadruple junction mesh.
7 %INPUT:
8 % - dofs : cell-array from which to extract the interior DoFs
9 % - N : Number of points.
10 % - order : 0 if primal mesh is discretized by p.w.constants.
11 %           1 if discretized by p.w.linear
12 %OUTPUT:
13 % - dofs : cell-array returning interior DoFs indices.
14
15 if (order==1)
16     %remove dofs cells corresponding to vertices on tips
17     dofs(7*N1 - 6) = [];
18     dofs(5*N1 - 4) = [];
19     dofs(3*N1 - 2) = [];
20     dofs(N1) = [];
21 else
22     disp('Sorry, not implemented.')
23 end

```



```

23
24 end

```

Tests for the Kernel Dimensions

The results from section 2.1 have been obtained by running the following scripts.

LISTING A.11: tests/kernel_test/test_kernel_V0_segment.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Segment limits [a,b]
3  a = -1; b = 1;
4  % Number of mesh vertices per side of the mesh
5  Nl = [3, 6, 12, 24, 48];
6  % Auxiliary matrix for storing results
7  res = zeros(length(Nl), 2);
8  % Piecewise order of the Boundary Element Space (constant for V)
9  order = 0;
10 % Number of quadrature points
11 Nq = 4;
12 % Mesh thickness
13 e = 0;
14
15 disp('|-----|')
16 out = strcat('|', 9, '#elements', 9, '|', '|', 9, '#dim(ker(V))', 9, ...
17           '|');
18 disp(out)
19
20 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Compute kernels %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21 for k = 1 : length(Nl)
22     % Compute the mesh
23     [ mX, melems, mdofs] = segmentMT_mesh( a, b, Nl(k), order, 0, ...
24         false);
25     % Compute V0 on the mesh
26     mV = varV0_ms(mX, melems, mdofs, Nq);
27     % Number of elements in the mesh
28     res(k, 1) = 2 * (Nl(k) - 1);
29     % Kernel size
30     res(k, 2) = size(null(mV), 2);
31
32     out = strcat('|', 9, num2str(res(k, 1)), 9, 9, '|', '|', 9, ...
33         num2str(res(k, 2)), 9, 9, '|');
34     disp(out)
35 end
36
37 disp('|-----|')

```

LISTING A.12: tests/kernel_test/test_kernel_V0_triple.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Tip coordinates
3  tips = [-sqrt(2)/2 -sqrt(2)/2; sqrt(2)/2 -sqrt(2)/2; 0 1];
4  % Number of mesh vertices per side of the mesh for each fin
5  Nl = [3, 6, 12, 24, 48];
6  % Auxiliary matrix for storing results
7  res = zeros(length(Nl), 2);
8  % Piecewise order of the Boundary Element Space (constant for V)
9  order = 0;

```

```

10 % Number of quadrature points
11 Nq = 4;
12 % Mesh thickness
13 e = 0;
14
15 disp('|-----|')
16 out = strcat('|', 9, '#elements', 9, '|', '|', 9, '#dim(ker(V))', ...
17           9, '|');
18 disp(out)
19 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Compute kernels %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
20 for k=1 : length(Nl)
21     % Compute the mesh
22     [mX, melems, mdofs] = tripleJMT_mesh(tips, Nl(k), order, 0, false);
23     % Compute V0 on the mesh
24     mV = varV0_ms(mX, melems, mdofs, Nq);
25     % Number of elements in the mesh
26     res(k, 1) = 6 * (Nl(k) - 1);
27     % Kernel size
28     res(k, 2) = size(null(mV), 2);
29
30     out = strcat('|', 9, num2str(res(k, 1)), 9, 9, '|', '|', 9, ...
31             num2str(res(k, 2)), 9, 9, '|');
32     disp(out)
33 end
34 disp('|-----|')

```

LISTING A.13: tests/kernel_test/test_kernel_V0_quadruple.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Length of each fin
3 l=1;
4 % Number of mesh vertices per side of the mesh for each flat part ...
5   of the mesh
6 Nl = [3, 6, 12, 24, 48];
7 % Auxiliary matrix for storing results
8 res = zeros(length(Nl), 2);
9 % Piecewise order of the Boundary Element Space (constant for V)
10 order = 0;
11 % Number of quadrature points
12 Nq = 4;
13 % Mesh thickness
14 e = 0;
15
16 disp('|-----|')
17 out = strcat('|', 9, '#elements', 9, '|', '|', 9, '#dim(ker(V))', ...
18           9, '|');
19 disp(out)
20 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Compute kernels %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21 for k = 1 : length(Nl)
22     % Compute the mesh
23     [mX, melems, mdofs] = quadrupleJMT_mesh(l, Nl(k), order, 0, false);
24     % Compute V0 on the mesh
25     mV = varV0_ms(mX, melems, mdofs, Nq);
26     % Number of elements in the mesh
27     res(k, 1) = 8 * (Nl(k) - 1);
28     % Kernel size
29     res(k, 2) = size(null(mV), 2);
30

```

```

30     out = strcat('|', 9, num2str(res(k, 1)), 9, 9, '|', '|', 9, ...
        num2str(res(k, 2)), 9, 9, '|');
31     disp(out)
32 end
33
34 disp('|-----|')

```

LISTING A.14: tests/kernel_test/test_kernel_W0_segment.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Segment limits [a,b]
3  a = -1; b = 1;
4  % Number of mesh vertices per side of the mesh
5  N1 = [32, 64, 128, 256];
6  % Auxiliary matrix for storing results
7  res = zeros(length(N1), 2);
8  % Piecewise order of the Boundary Element Space (linear for W)
9  order = 1;
10 % Number of quadrature points
11 Nq = 4;
12 % Mesh thickness
13 e = 0;
14
15 disp('|-----|')
16 out = strcat('|', 9, '#elements', 9, '|', '|', 9, '#dim(ker(W))', ...
        9, '|');
17 disp(out)
18
19 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Compute kernels %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
20 for k = 1 : length(N1)
21     % Compute the mesh
22     [mX, melems, mdofs] = segmentMT_mesh( a, b, N1(k), order, 0 , ...
        false);
23     % Extract the interior DoFs
24     [idofs] = segmentMT_innerDofs(mdofs, N1(k), order);
25     % Compute W0 on the mesh
26     mW = varW0_ms(mX, melems, idofs, Nq);
27     % Number of vertices in the mesh
28     res(k, 1) = size(idofs, 1);
29     % Kernel size
30     res(k, 2) = size(null(mW), 2);
31
32     out = strcat('|', 9, num2str(res(k, 1)), 9, 9, '|', '|', 9, ...
        num2str(res(k, 2)), 9, 9, '|');
33     disp(out)

```

LISTING A.15: tests/kernel_test/test_kernel_W0_triple.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Tip coordinates
3  tips = [-sqrt(2)/2 -sqrt(2)/2; sqrt(2)/2 -sqrt(2)/2; 0 1];
4  % Number of mesh vertices per side of the mesh for each fin
5  N1 = [3, 6, 12, 24, 48];
6  % Auxiliary matrix for storing results
7  res = zeros(length(N1), 2);
8  % Piecewise order of the Boundary Element Space (linear for W)
9  order = 1;
10 % Number of quadrature points
11 Nq = 4;
12 % Mesh thickness

```

```

13 e = 0;
14
15 disp('|-----|')
16 out = strcat('|', 9, '#elements', 9, '|', '|', 9, '#dim(ker(V))', ...
17           9, '|');
18 disp(out)
19 %%%%%%%%%% Compute kernels %%%%%%%%%%
20 for k = 1 : length(Nl)
21     % Compute the mesh
22     [ mX, melems, mdofs] = tripleJMT_mesh(tips, Nl(k), order, 0 , ...
23           false);
24     % Extract the interior DoFs
25     [idofs] = tripleJMT_innerDofs(mdofs, Nl(k), order);
26     % Compute W0 on the mesh
27     mW = varW0_ms(mX, melems, idofs, Nq);
28     % Number of vertices in the mesh
29     res(k, 1) = size(idofs,1);
30     % Kernel size
31     res(k, 2) = size(null(mW), 2);
32
33     out = strcat('|', 9, num2str(res(k,1)), 9, 9, '|', '|', 9, ...
34           num2str(res(k,2)), 9 , 9, '|');
35     disp(out)
36 end

```

LISTING A.16: tests/kernel_test/test_kernel_W0_quadruple.m

```

1 %%%%%%%%%% Define parameters %%%%%%%%%%
2 % Length of each fin
3 l = 1;
4 % Number of mesh vertices per side of the mesh for each fin
5 Nl = [3, 6, 12, 24, 48];
6 % Auxiliary matrix for storing results
7 res = zeros(length(Nl), 2);
8 % Piecewise order of the Boundary Element Space
9 order = 1;
10 % Number of quadrature points
11 Nq = 4;
12 % Mesh thickness
13 e = 0;
14
15 disp('|-----|')
16 out = strcat('|', 9, '#elements', 9, '|', '|', 9, '#dim(ker(W))', 9, '|');
17 disp(out)
18
19 %%%%%%%%%% Compute kernels %%%%%%%%%%
20 for k = 1 : length(Nl)
21     % Compute the mesh
22     [ mX, melems, mdofs] = quadrupleJMT_mesh(l, Nl(k), order, 0, ...
23           false);
24     % Extract the interior DoFs
25     [idofs] = quadrupleJMT_innerDofs(mdofs, Nl(k), order);
26     % Compute W0 on the mesh
27     mW = varW0_ms(mX, melems, idofs, Nq);
28     % Number of vertices in the mesh
29     res(k, 1) = size(idofs, 1);
30     % Kernel size
31     res(k, 2) = size(null(mW), 2);
32
33
34
35
36
37
38
39
40
41

```

```

32     out = strcat('|', 9, num2str(res(k, 1)), 9, 9, '|', '|', 9, ...
        num2str(res(k, 2)), 9, 9, '|');
33     disp(out)
34 end

```

Tests CG and GMRES Iterative Solvers

The tests for the iterative solvers from section 2.2 are implemented in the following scripts.

LISTING A.17: tests/ls_solver_test/segment_V0_ls.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Segment limits [a,b]
3  a = -1; b = 1;
4  % Number of mesh elements per side of the mesh
5  Nl = [32, 64, 128, 256];
6  % Number of quadrature points
7  Nq = 4;
8  % Mesh thickness
9  ev = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0];
10 % Auxiliary matrices for storing results
11 cn = zeros(length(ev), length(Nl));
12 iter_pcg = zeros(length(ev), length(Nl));
13 iter_gmres = zeros(length(ev), length(Nl));
14 % Piecewise order of the Boundary Element Space (constant for V)
15 order = 0;
16
17 out1 = strcat('|', 'eps', 9, '|');
18 out2 = strcat('|', 9, '|');
19
20 for i = 1 : length(Nl)
21     out1 = strcat(out1, 9, 9, 'N=', num2str(Nl(i)), 9, 9, '|');
22     out2 = strcat(out2, 'cond(V)', 9, '|', 'CG', 9, '|', 'GMRES', ...
        9, '|');
23 end
24 disp(out1)
25 disp(out2)
26
27 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Solve System %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
28 for i = 1 : length(ev)
29     out3 = strcat('|', num2str(ev(i)), 9, '|');
30     for j = 1 : length(Nl)
31         % Compute the mesh
32         [X, elems, dofs] = segmentMT_mesh( a, b, Nl(j), order, ...
            ev(i), false);
33         % Compute V0 on the mesh
34         V = varV0_ms(X, elems, dofs, Nq);
35         % Compute random rhs
36         rhs = V * rand(size(V, 1), 1);
37         % Solve with CG
38         [~, ~, ~, iter_pcg(i, j)] = pcg(V, rhs, 1e-6, size(V, 1));
39         % Solve with GMRES
40         [~, ~, ~, iteraux] = gmres(V, rhs, [], 1e-6, size(V, 1));
41         % Store and output results
42         iter_gmres(i, j) = iteraux(2);
43         cn(i, j) = cond(V);
44         if(i == length(ev))
45             eg = eig(V);

```

```
46         cn(i, j) = max(eig(V)) / min(eg(eg > 1e-16));
```

LISTING A.18: tests/ls_solver_test/tripleJ_V0_ls.m

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Tip coordinates
3  tips = [-sqrt(2)/2 -sqrt(2)/2; sqrt(2)/2 -sqrt(2)/2; 0 1];
4  % Number of mesh vertices for each fin per side of the mesh
5  Nl = [32, 64, 128, 256];
6  % Number of quadrature points
7  Nq = 4;
8  % Mesh thickness
9  ev = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0];
10 % Auxiliary matrices for storing results
11 cn = zeros(length(ev), length(Nl));
12 iter_pcg = zeros(length(ev), length(Nl));
13 iter_gmres = zeros(length(ev), length(Nl));
14 % Piecewise order of the Boundary Element Space (constant for V)
15 order = 0;
16
17 out1 = strcat('|', 'eps', 9, '|');
18 out2 = strcat('|', 9, '|');
19
20 for i = 1 : length(Nl)
21     out1 = strcat(out1, 9, 9, 'N=', num2str(Nl(i)), 9, 9, '|');
22     out2 = strcat(out2, 'cond(V)', 9, '|', 'CG', 9, '|', 'GMRES', ...
23                 9, '|');
24 end
25 disp(out1)
26 disp(out2)
27
28 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Solve System %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29 for i = 1 : length(ev)
30     out3 = strcat('|', num2str(ev(i)), 9, '|');
31     for j = 1 : length(Nl)
32         % Compute the mesh
33         [X, elems, dofs] = tripleJMT_mesh(tips, Nl(j), order, ...
34         ev(i), false);
35         % Compute V0 on the mesh
36         V = varV0_ms(X, elems, dofs, Nq);
37         % Compute random rhs
38         rhs = V * rand(size(V, 1), 1);
39         % Solve with CG
40         [~, ~, ~, iter_pcg(i,j)] = pcg(V, rhs, 1e-6, size(V, 1));
41         % Solve with GMRES
42         [~, ~, ~, iteraux] = gmres(V, rhs, [], 1e-6, size(V, 1));
43         % Store and output results
44         iter_gmres(i, j) = iteraux(2);
45         cn(i, j) = cond(V);
46         if(i == length(ev))
47             eg = eig(V);
48             cn(i, j) = max(eig(V)) / min(eg(eg > 1e-16));
```

LISTING A.19: tests/ls_solver_test/quadrupleJ_V0_ls.m

```
1
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % Length of each fin
4  l=1;
5  % Number of mesh elements for each fin per side of the mesh
```

```

6 N1 = [32, 64, 128, 256];
7 % Number of quadrature points
8 Nq = 4;
9 % Mesh thickness
10 ev = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0];
11 % Auxiliary matrices for storing results
12 cn = zeros(length(ev), length(N1));
13 iter_pcg = zeros(length(ev), length(N1));
14 iter_gmres = zeros(length(ev), length(N1));
15 % Piecewise order of the Boundary Element Space (constant for V)
16 order = 0;
17
18 out1 = strcat(' ', 'eps', 9, ' ');
19 out2 = strcat(' ', 9, ' ');
20
21 for i = 1 : length(N1)
22     out1 = strcat(out1, 9, 9, 'N=', num2str(N1(i)), 9, 9, ' ');
23     out2 = strcat(out2, 'cond(V)', 9, ' ', 'CG', 9, ' ', 'GMRES', ...
24                 9, ' ');
25 end
26 disp(out1)
27 disp(out2)
28 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Solve System %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29 for i = 1 : length(ev)
30     out3 = strcat(' ', num2str(ev(i)), 9, ' ');
31     for j = 1 : length(N1)
32         % Compute the mesh
33         [X, elems, dofs] = quadrupleJMT_mesh(1, N1(j), order, ...
34         ev(i), false);
35         % Compute V0 on the mesh
36         V = varV0_ms(X, elems, dofs, Nq);
37         % Compute random rhs
38         rhs = V * rand(size(V, 1), 1);
39         % Solve with CG
40         [~, ~, ~, iter_pcg(i,j)] = pcg(V, rhs, 1e-6, size(V, 1));
41         % Solve with GMRES
42         [~, FLAG, relres, iteraux] = gmres(V, rhs, [], 1e-6, size(V, ...
43         1));
44         % Store and output results
45         iter_gmres(i, j) = iteraux(2);
46         cn(i, j) = cond(V);
47         if(i == length(ev))
48             eg = eig(V);

```

LISTING A.20: tests/ls_solver_test/segment_W0_ls.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Segment limits [a,b]
3 a = -1; b = 1;
4 % Number of mesh elements per side of the mesh
5 N1 = [32, 64, 128, 256];
6 % Number of quadrature points
7 Nq = 4;
8 % Mesh thickness
9 ev = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0];
10 % Auxiliary matrices for storing results
11 cn = zeros(length(ev), length(N1));
12 iter_pcg = zeros(length(ev), length(N1));
13 iter_gmres = zeros(length(ev), length(N1));
14 % Piecewise order of the Boundary Element Space (linear for W)

```

```

15 order = 1;
16
17 out1 = strcat('||', 'eps', 9, '|');
18 out2 = strcat('||', 9, '|');
19
20 for i = 1 : length(Nl)
21     out1 = strcat(out1, 9, 9, 'N=', num2str(Nl(i)), 9, 9, '|');
22     out2 = strcat(out2, 'cond(W)', 9, '|', 'CG', 9, '|', 'GMRES', ...
23         9, '|');
24 end
25 disp(out1)
26 disp(out2)
27
28 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Solve System %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29 for i = 1 : length(ev)
30     out3 = strcat('||', num2str(ev(i)), 9, '|');
31     for j = 1 : length(Nl)
32         % Compute the mesh
33         [X, elems, dofs] = segmentMT_mesh( a, b, Nl(j), ...
34             order, ev(i), false);
35         % Extract the interior DoFs
36         [idofs] = segmentMT_innerDofs(dofs, Nl(j), order);
37         % Compute W0 on the mesh
38         W = varW0_ms(X, elems, idofs, Nq);
39         % Compute random rhs
40         rhs = W * rand(size(W, 1), 1);
41         % Solve with CG
42         [~, ~, ~, iter_pcg(i,j)] = pcg(W, rhs, 1e-6, size(W, 1));
43         % Solve with GMRES
44         [~, ~, ~, iteraux] = gmres(W, rhs, [], 1e-6, size(W, 1));
45         % Store and output results
46         iter_gmres(i, j) = iteraux(2);
47         cn(i, j) = cond(W);
48         if(i == length(ev))

```

LISTING A.21: tests/ls_solver_test/tripleJ_W0_ls.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Tips coordinates [a,b]
3 tips = [-sqrt(2)/2 -sqrt(2)/2; sqrt(2)/2 -sqrt(2)/2; 0 1];
4 % Number of mesh elements per side of the mesh
5 Nl = [32, 64, 128, 256];
6 % Number of quadrature points
7 Nq = 4;
8 % Mesh thickness
9 ev = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0];
10 % Auxiliary matrices for storing results
11 cn = zeros(length(ev), length(Nl));
12 iter_pcg = zeros(length(ev), length(Nl));
13 iter_gmres = zeros(length(ev), length(Nl));
14 % Piecewise order of the Boundary Element Space (linear for W)
15 order = 1;
16
17 out1 = strcat('||', 'eps', 9, '|');
18 out2 = strcat('||', 9, '|');
19
20 for i = 1 : length(Nl)
21     out1 = strcat(out1, 9, 9, 'N=', num2str(Nl(i)), 9, 9, '|');
22     out2 = strcat(out2, 'cond(W)', 9, '|', 'CG', 9, '|', 'GMRES', ...
23         9, '|');
24 end

```



```

24 disp(out1)
25 disp(out2)
26
27 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Solve System %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
28 for i = 1 : length(ev)
29     out3 = strcat('|', num2str(ev(i)), 9, '|');
30     for j = 1 : length(Nl)
31         % Compute the mesh
32         [X, elems, dofs] = tripleJMT_mesh(tips, Nl(j), order, ev(i), ...
33             false);
34         % Extract the interior DoFs
35         [idofs] = tripleJMT_innerDofs(dofs, Nl(j), order);
36         % Compute W0 on the mesh
37         W = varW0_ms(X, elems, idofs, Nq);
38         % Compute random rhs
39         rhs = W * rand(size(W, 1), 1);
40         % Solve with CG
41         [~, ~, ~, iter_pcg(i, j)] = pcg(W, rhs, 1e-6, size(W, 1));
42         % Solve with GMRES
43         [~, ~, ~, iteraux] = gmres(W, rhs, [], 1e-6, size(W, 1));
44         % Store and output results
45         iter_gmres(i, j) = iteraux(2);
46         cn(i, j) = cond(W);
47         if(i == length(ev))

```

LISTING A.22: tests/ls_solver_test/quadrupleJ_W0_ls.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Length of each fin
3 l = 1;
4 % Number of mesh elements per side of the mesh
5 Nl = [32, 64, 128, 256];
6 % Number of quadrature points
7 Nq = 4;
8 % Mesh thickness
9 ev = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0];
10 % Auxiliary matrices for storing results
11 cn = zeros(length(ev), length(Nl));
12 iter_pcg = zeros(length(ev), length(Nl));
13 iter_gmres = zeros(length(ev), length(Nl));
14 % Piecewise order of the Boundary Element Space (linear for W)
15 order = 1;
16
17 out1 = strcat('|', 'eps', 9, '|');
18 out2 = strcat('|', 9, '|');
19
20 for i = 1:length(Nl)
21     out1 = strcat(out1, 9, 9, 'N=', num2str(Nl(i)), 9, 9, '|');
22     out2 = strcat(out2, 'cond(W)', 9, '|', 'CG', 9, '|', 'GMRES', ...
23         9, '|');
24 end
25 disp(out1)
26 disp(out2)
27
28 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Solve System %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29 for i = 1 : length(ev)
30     out3 = strcat('|', num2str(ev(i)), 9, '|');
31     for j = 1 : length(Nl)
32         % Compute the mesh
33         [X, elems, dofs] = quadrupleJMT_mesh(l, Nl(j), order, ev(i), ...
34             false);

```

```

33     % Extract the interior DoFs
34     [idofs] = quadrupleJMT_innerDofs(dofs, N1(j), order);
35     % Compute W0 on the mesh
36     W = varW0_ms(X, elems, idofs, Nq);
37     % Compute random rhs
38     rhs = W * rand(size(W, 1),1);
39     % Solve with CG
40     [~, ~, ~, iter_pcg(i,j)] = pcg(W, rhs, 1e-6, size(W, 1));
41     % Solve with GMRES
42     [~, ~, ~, iteraux] = gmres(W, rhs, [], 1e-6, size(W, 1));
43     % Store and output results
44     iter_gmres(i, j) = iteraux(2);
45     cn(i, j) = cond(W);
46     if(i == length(ev))

```

Validation on a Segment

The tests for the validation of our BEM matrices on a segment from section 2.3 are implemented in the scripts

LISTING A.23: tests/segment_validation/segment_validation_V0_ST.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Segment limits [a,b]
3  a = -1; b = 1;
4  % Number of mesh vertices
5  N1 = [4, 8, 16, 32, 64, 128, 256, 512, 1024]+1;
6  % Auxiliary matrix for storing results
7  res = zeros(length(N1), 2);
8  % Piecewise order of the Boundary Element Space (constant for V)
9  order = 0;
10 % Number of quadrature points
11 Nq = 30;
12 % Mesh thickness
13 e = 0;
14 errv = [];
15
16 out = strcat('|', 9, 'N', 9, '|', 9, 'L2-error', 9, 9, '|', '|', 9, ...
17             'order', 9, 9, '|');
18 disp(out)
19
20 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21 for i = 1 : length(N1)
22     % Compute the mesh
23     [ X, elems, dofs] = segmentST_mesh(a, b, N1(i), order, false);
24     Ne = size(elems, 1);
25
26     % Compute right-hand side vector
27     rhs = zeros(Ne, 1);
28     for j = 1 : Ne
29         rhs(j) = pi / 2 * (X(j + 1, 1) ^ 2 - X(j, 1) ^ 2);
30     end
31     % Compute V0 on the mesh
32     V = varV0_ms(X, elems, dofs, Nq);
33     % Solve the linear system
34     [sol, ~, ~, ~] = pcg(V, rhs, 1e-6, size(V, 1));
35
36     % Compute L2 error and estimated order

```

```

36     errv = [errv sqrt(abs(pi ^ 2 / 2 - sol' * V * sol))];
37     if i > 1
38         p = log2(errv(end - 1) ./ errv(end));
39     else
40         p = 0;
41     end
42
43     out = strcat('|', 9, num2str(Ne), 9, '|', 9, num2str(errv(end), ...
44         '%e'), 9, 9, '|', '|', 9, num2str(p), 9, 9, '|');
45     disp(out)
46 end

```

LISTING A.24: tests/segment_validation/segment_validation_V0_MT.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Segment limits [a,b]
3  a = -1; b = 1;
4  % Number of mesh vertices
5  Nl = [4, 8, 16, 32, 64, 128, 256, 512, 1024]+1;
6  % Auxiliary matrix for storing results
7  res = zeros(length(Nl), 2);
8  % Piecewise order of the Boundary Element Space (constant for V)
9  order = 0;
10 % Number of quadrature points
11 Nq = 30;
12 % Mesh thickness
13 e = 0;
14 errv = [];
15
16 out = strcat('|', 9, 'N', 9, '|', 9, 'L2-error', 9, 9, '|', '|', 9, ...
17     'order', 9, 9, '|');
18 disp(out)
19 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
20 for i = 1 : length(Nl)
21     % Compute the mesh
22     [ X, elems, dofs] = segmentMT_mesh(a, b, Nl(i), order, 0, false);
23     Ne = size(elems, 1);
24
25     % Compute right-hand side vector
26     rhs = zeros(Ne, 1);
27     for j = 1 : Ne - 1
28         rhs(j) = pi * (X(j + 1, 1) ^ 2 - X(j, 1) ^ 2);
29     end
30     rhs(Ne) = pi * (X(1, 1) ^ 2 - X(end,1) ^ 2);
31     rhs(Ne / 2 + 1 : Ne) = -rhs(Ne / 2 + 1 : Ne);
32
33     % Compute V0 on the mesh
34     V = varV0_ms(X, elems, dofs, Nq);
35
36     % Solve the linear system
37     [sol, ~, ~, ~] = pcg(V, rhs, 1e-6, size(V, 1));
38
39     % Compute L2 error and estimated order
40     errv = [errv sqrt(abs(2 * pi ^ 2 - sol' * V * sol))];
41     if i > 1
42         p = log2(errv(end - 1) ./ errv(end));
43     else
44         p = 0;
45     end

```

```

46
47     out = strcat('|', 9, num2str(Ne), 9, '|', 9, num2str(errv(end), ...
48         '%e'), 9, 9, '|', '|', 9, num2str(p), 9, 9, '|');
    disp(out)

```

LISTING A.25: tests/segment_validation/segment_validation_W0_ST.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Segment limits [a,b]
3  a = -1; b = 1;
4  % Number of mesh vertices
5  Nl = [4, 8, 16, 32, 64, 128, 256, 512, 1024]+1;
6  % Auxiliary matrix for storing results
7  res = zeros(length(Nl), 2);
8  % Piecewise order of the Boundary Element Space (linear for linear)
9  order = 1;
10 % Number of quadrature points
11 Nq = 30;
12 % Mesh thickness
13 e = 0;
14 errv = [];
15
16 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17 for i = 1 : length(Nl)
18     % Compute the mesh
19     [ X, elems, sdofs] = segmentST_mesh(a, b, Nl(i), order, false);
20     [idofs] = segmentST_innerDofs(sdofs, Nl(i), order);
21     Ne = size(elems,1);
22
23     % Compute the right-hand side vector
24     rhs = zeros(Nl(i) - 2, 1);
25     eta = zeros(Nl(i) - 1, 1);
26     for j = 1 : Nl(i) - 1
27         eta(j) = (X(j, 1) + X(j + 1, 1)) / 2;
28     end
29     rhs = pi * diff(eta);
30
31     % Compute W0 on the mesh
32     W = varW0_ms(X, elems, idofs, Nq);
33
34     % Solve the linear system
35     [sol, ~, ~, ~] = pcg(W, rhs, 1e-6, 100);
36
37     % Compute L2 error and estimated order
38     errv = [errv sqrt(abs(pi ^ 2 / 2 - (sol)' * W * (sol)))];
39     if i > 1
40         p = log2(errv(end - 1) ./ errv(end));
41     else
42         p = 0;
43     end
44
45     out = strcat('|', 9, num2str(Ne), 9, '|', 9, num2str(errv(end), ...
46         '%e'), 9, 9, '|', '|', 9, num2str(p), 9, 9, '|');
    disp(out)

```

LISTING A.26: tests/segment_validation/segment_validation_W0_MT.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Segment limits [a,b]
3  a = -1; b = 1;

```

```

4 % Number of mesh vertices
5 Nl = [4, 8, 16, 32, 64, 128, 256, 512, 1024]+1;
6 % Auxiliary matrix for storing results
7 res = zeros(length(Nl), 2);
8 % Piecewise order of the Boundary Element Space (linear for linear)
9 order = 1;
10 % Number of quadrature points
11 Nq = 30;
12 % Mesh thickness
13 e = 0;
14 errv = [];
15
16 out = strcat('|', 9, 'N', 9, '|', 9, 'L2-error', 9, 9, '|', '|', 9, ...
    'order', 9, 9, '|');
17 disp(out)
18
19 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
20 for i = 1 : length(Nl)
21     % Compute the mesh
22     [ X, elems, mdofs] = segmentMT_mesh(a, b, Nl(i), order, 0, false);
23     inner_dofs = segmentST_innerDofs(mdofs, Nl(i), order);
24     Ne = size(elems, 1);
25
26     % Compute the right-hand side
27     rhs = zeros(Nl(i) - 2, 1);
28     eta = zeros(Nl(i) - 1, 1);
29     for j = 1 : Nl(i) - 1
30         eta(j) = (X(j,1) + X(j + 1, 1)) / 2;
31     end
32     rhs = 2 * [pi * diff(eta); - pi * diff(eta)];
33
34     % Compute W0 on the mesh
35     W = varW0_ms(X, elems, inner_dofs, Nq);
36
37     % Solve the linear system
38     [sol, ~, ~, ~] = pcg(W, rhs, 1e-6, size(W, 1));
39
40     % Compute L2 error and estimated order
41     errv = [errv sqrt(abs(2 * pi ^ 2 - sol' * W * sol))];
42     if i > 1
43         p = log2(errv(end - 1) ./ errv(end));
44     else
45         p = 0;
46     end
47
48     out = strcat('|', 9, num2str(Ne), 9, '|', 9, num2str(errv(end), ...
    '%e'), 9, 9, '|', '|', 9, num2str(p), 9, 9, '|');
49     disp(out)

```

Elimination of the Kernels

The scripts in which we implemented the elimination of the kernels of V and W from section 2.4 are

LISTING A.27: tests/eliminate_dofs/V0_segment.m

```

1 % Segment limits [a,b]
2 a = -1; b = 1;

```

```

3 % Number of mesh elements per side of the mesh
4 Nl = [32 64 128 256 512 1024];
5 % Number of quadrature points
6 Nq = 4;
7 % Mesh thickness
8 e = 0;
9 % Piecewise order of the Boundary Element Space (constant for V)
10 order = 0;
11
12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13 for i = 1 : length(Nl)
14     % Compute the mesh
15     [ X, elems, dofs] = segmentMT_mesh( a, b, Nl(i), order, 0, false);
16     Ne = size(elems, 1);
17
18     % Compute V0 on the mesh
19     V = varV0_ms(X, elems, dofs, Nq);
20
21     % Compute random right-hand side
22     rhs = V * rand(size(V, 1), 1);
23
24     % Compute the T matrix
25     T = zeros(Ne, Ne / 2);
26     T(1 : Ne / 2, 1 : Ne / 2) = eye(Ne / 2, Ne / 2);
27     T(Ne / 2 + 1 : 2 * Ne / 2, 1 : Ne / 2) = flipud(eye(Ne / 2, Ne ...
28         / 2));
29
30     % Compute reduced system by transforming the DoFs
31     Vred = T' * V * T;
32     rhsRed = T' * rhs;
33
34     % Solve the full linear system
35     out = strcat('Full system with N = ', num2str(Nl(i)));

```

LISTING A.28: tests/eliminate_dofs/V0_triple.m

```

1 % Tips coordinates
2 tips = [-sqrt(2)/2 -sqrt(2)/2; sqrt(2)/2 -sqrt(2)/2; 0 1];
3 % Number of mesh elements per side of the mesh
4 Nl = [32, 64, 128, 256];
5 % Number of quadrature points
6 Nq = 4;
7 % Mesh thickness
8 e = 0;
9 % Piecewise order of the Boundary Element Space (constant for V)
10 order = 0;
11
12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13 for i = 1 : length(Nl)
14     % Compute the mesh
15     [X, elems, dofs] = tripleJMT_mesh(tips, Nl(i), order, 0, false);
16
17     % Compute V on the mesh
18     V = varV0_ms(X, elems, dofs, Nq);
19     % Compute random rhs
20     rhs = V * rand(size(V, 1), 1);
21
22     % Compute the T matrix
23     Tblock = zeros(2 * (Nl(i) - 1), Nl(i) - 1);
24     Tblock(1 : Nl(i) - 1, :) = eye(Nl(i) - 1, Nl(i) - 1);
25     Tblock(Nl(i) : end, :) = flipud(eye(Nl(i) - 1, Nl(i) - 1));

```

```

26     zblock = zeros(2 * (Nl(i) - 1), Nl(i) - 1);
27     T = [Tblock, zblock, zblock;
28         zblock, Tblock, zblock;
29         zblock, zblock, Tblock];
30
31     % Compute reduced system by transforming the DoFs
32     Vred = T' * V * T;
33     rhsRed = T' * rhs;

```

LISTING A.29: tests/eliminate_dofs/V0_quadruple.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Length of each fin
3  l = 1;
4  % Number of mesh elements per side of the mesh
5  Nl = [32, 64, 128, 256];
6  % Number of quadrature points
7  Nq = 4;
8  % Mesh thickness
9  e = 0;
10 % Piecewise order of the Boundary Element Space (constant for V)
11 order = 0;
12
13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
14 for i = 1 : length(Nl)
15     [X, elems, dofs] = quadrupleJMT_mesh(l, Nl(i), order, 0, false);
16     % compute W in the two approaches
17     V = varV0_ms(X, elems, dofs, Nq);
18     % compute dummy rhs
19     rhs = V * rand(size(V, 1), 1);
20     % compute T matrix for dofs reduction
21     Tblock = zeros(2 * (Nl(i) - 1), Nl(i) - 1);
22     Tblock(1 : Nl(i) - 1, :) = eye(Nl(i) - 1, Nl(i) - 1);
23     Tblock(Nl(i) : end, :) = flipud(eye(Nl(i) - 1, Nl(i) - 1));
24     zblock = zeros(2 * (Nl(i) - 1), Nl(i) - 1);
25     T = [Tblock, zblock, zblock, zblock;
26         zblock, Tblock, zblock, zblock;
27         zblock, zblock, Tblock, zblock;
28         zblock, zblock, zblock, Tblock];
29
30     % Compute reduced system by transforming the DoFs
31     Vred = T' * V * T;
32     rhsRed = T' * rhs;
33
34     % Solve the non-reduced linear system

```

LISTING A.30: tests/eliminate_dofs/W0_segment.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Segment limits [a,b]
3  a = -1; b = 1;
4  % Number of mesh elements per side of the mesh
5  Nl = [32, 64, 128, 256];
6  % Number of quadrature points
7  Nq = 4;
8  % Mesh thickness
9  e = 0;
10 % Piecewise order of the Boundary Element Space (linear for W)
11 order = 1;
12

```

```

13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
14 for i = 1 : length(Nl)
15     % Compute the mesh
16     [ X, elems, dofs] = segmentMT_mesh(a, b, Nl(i), order, 0, false);
17     % Extract the interior DoFs
18     [idofs] = segmentMT_innerDofs(dofs, Nl(i), order);
19     Ndofs = size(idofs, 1);
20
21     % Compute W0 on the mesh
22     W = varW0_ms(X, elems, idofs, Nq);
23
24     % Compute random rhs
25     rhs = W * rand(size(W, 1), 1);
26
27     % Compute the T matrix
28     T = zeros(Ndofs, Ndofs / 2);
29     T(1 : Ndofs / 2, :) = eye(Ndofs / 2, Ndofs / 2);
30     T(Ndofs / 2 + 1 : 2 * Ndofs / 2, :) = ...
        -flipud(eye(Ndofs/2, Ndofs/2));
31
32     % Compute reduced system by transforming the DoFs
33     Wred = T' * W * T;
34     rhsRed = T' * rhs;

```

LISTING A.31: tests/eliminate_dofs/W0_triple.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Tips coordinates
3 tips = [-sqrt(2)/2 -sqrt(2)/2; sqrt(2)/2 -sqrt(2)/2; 0 1];
4 % Number of mesh elements per side of the mesh
5 Nl = [32 64 128 256];
6 % Number of quadrature points
7 Nq = 4;
8 % Mesh thickness
9 e = 0;
10 % Piecewise order of the Boundary Element Space (linear for W)
11 order = 1;
12
13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
14 for i = 1 : length(Nl)
15     % Compute the mesh
16     [X, elems, dofs] = tripleJMT_mesh(tips, Nl(i), order, 0, false);
17     % Extract the interior DoFs
18     idofs = tripleJMT_innerDofs(dofs, Nl(i), order);
19     Ndofs = size(idofs, 1);
20
21     % Compute W0 on the mesh
22     W = varW0_ms(X, elems, idofs, Nq);
23
24     % Reorder the DoffFs for the sake of simplicity
25     N = Nl(i);
26     mask = [2 : 2 * N - 3, 2 * N - 1 : 4 * N - 6, 4 * N - 4 : 6 * N ...
        - 9, 1, 2 * N - 2, 4 * N - 5];
27     W_masked = W(mask, mask);
28
29     % Compute random rhs
30     rhs = W_masked * rand(size(W_masked, 1), 1);
31
32     % Compute the T matrix
33     Tblock = zeros(2 * (Nl(i) - 2), Nl(i) - 2);
34     Tblock(1 : Nl(i) - 2, :) = eye(Nl(i) - 2, Nl(i) - 2);

```



```
35     Tblock(Nl(i) - 1 : end, : ) = -flipud(eye(Nl(i) - 2, Nl(i) - 2));
```

LISTING A.32: tests/eliminate_dofs/W0_quadruple.m

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Define parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Length of each fin
3  l = 1;
4  % Number of mesh elements per side of the mesh
5  Nl = [32, 64, 128, 256];
6  % Number of quadrature points
7  Nq = 4;
8  % Mesh thickness
9  e = 0;
10 % Piecewise order of the Boundary Element Space (linear for W)
11 order = 1;
12
13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Computations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
14 for i = 1 : length(Nl)
15     % Compute the mesh
16     [X, elems, dofs] = quadrupleJMT_mesh(l, Nl(i), order, 0, false);
17     % Extract the interior DoFs
18     idofs = quadrupleJMT_innerDofs(dofs, Nl(i), order);
19     Ndofs = size(idofs, 1);
20
21     % Compute W0 on the mesh
22     W = varW0_ms(X, elems, idofs, Nq);
23
24     % Reorder the DofFs for the sake of simplicity
25     N = Nl(i);
26     mask = [2 : 2 * N - 3, 2 * N - 1 : 4 * N - 6, 4 * N - 4 : 6 * N ...
27             - 9, 6 * N - 7 : 8 * N - 12, 1, 2 * N - 2, 4 * N - 5, 6 * N ...
28             - 8];
29     W_masked = W(mask, mask);
30
31     % Compute random rhs
32     rhs = W_masked * rand(size(W_masked, 1), 1);
33
34     % Compute the T matrix
35     Tblock = zeros(2 * (N - 2), N - 2);
36     Tblock(1 : N - 2, : ) = eye(N - 2, N - 2);
37     Tblock(N - 1 : end, : ) = -flipud(eye(N - 2, N - 2));
```


Appendix B

The C++ Codes

In this appendix we briefly describe the location of our codes in the GitLab repository of BETL2 (<https://gitlab.math.ethz.ch/bet1/Bet12.git>). The codes have been included in an application called `bem_multiscreen` whose structure is depicted in figure B.1. This directory contains two sub-directories called `laplace_multi-screen` and `efie_multi-screen`: the former contains the codes which solve the weakly singular BIE (`mainV0.cpp`) and the hypersingular BIE (`mainW0.cpp`), the latter contains the main drivers used to solve the static EFIE (`mainS0.cpp`) and the indefinite EFIE (`mainSk.cpp`). In addition, we have a folder which contains the codes used to correct the BEM matrices to retrieve the non-empty kernels and a folder which contains the grids used in the numerical experiments. The grids are of two types: the ordered meshes have been used for the weakly singular BIE while the oriented meshes have been used for all the other experiments. More detailed explanations about how to run the codes can be found in the `README.md` file of the application.

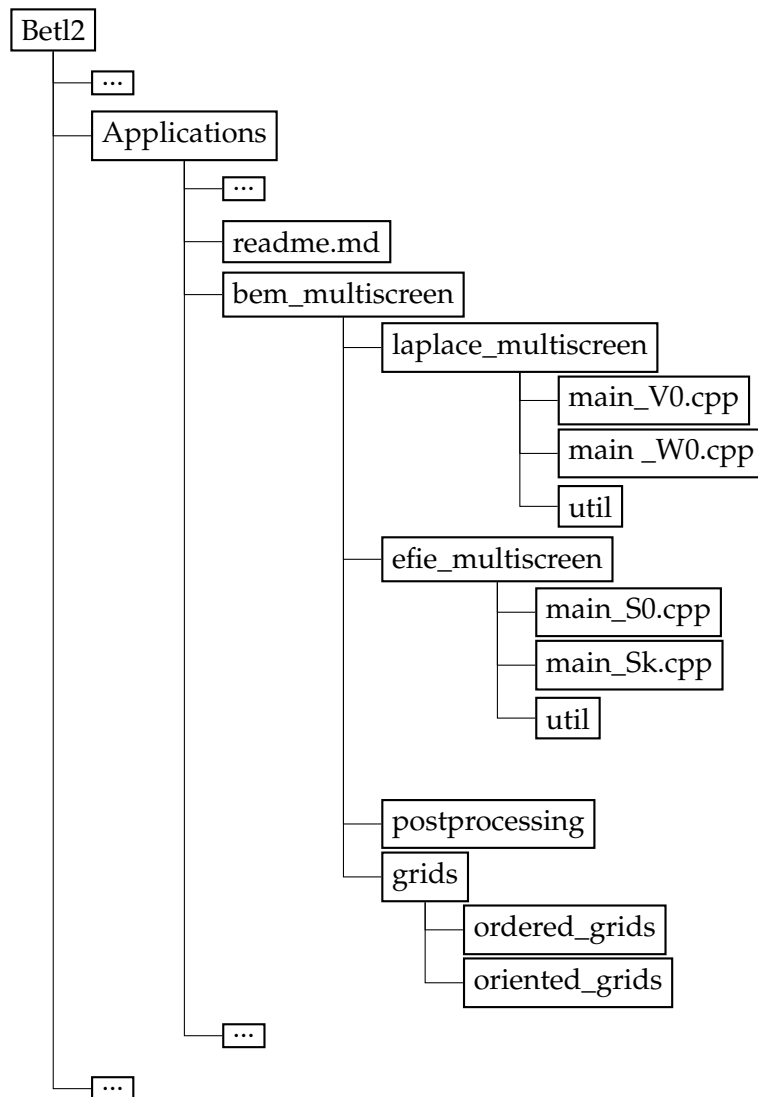


FIGURE B.1: Structure of the BETL2 Gitab repository

Bibliography

- [1] Peter N. Brown and Homer F. Walker. “GMRES on (nearly) singular systems”. In: *SIAM J. Matrix Anal. Appl.* 18.1 (1997), pp. 37–51. ISSN: 0895-4798. DOI: [10.1137/S0895479894262339](https://doi.org/10.1137/S0895479894262339). URL: <https://doi.org/10.1137/S0895479894262339>.
- [2] A. Buffa and S. H. Christiansen. “The electric field integral equation on Lipschitz screens: definitions and numerical approximation”. In: *Numer. Math.* 94.2 (2003), pp. 229–267. ISSN: 0029-599X. DOI: [10.1007/s00211-002-0422-0](https://doi.org/10.1007/s00211-002-0422-0). URL: <https://doi.org/10.1007/s00211-002-0422-0>.
- [3] A. Buffa, M. Costabel, and D. Sheen. “On traces for $\mathbf{H}(\mathbf{curl}, \Omega)$ in Lipschitz domains”. In: *J. Math. Anal. Appl.* 276.2 (2002), pp. 845–867. ISSN: 0022-247X. DOI: [10.1016/S0022-247X\(02\)00455-9](https://doi.org/10.1016/S0022-247X(02)00455-9). URL: [https://doi.org/10.1016/S0022-247X\(02\)00455-9](https://doi.org/10.1016/S0022-247X(02)00455-9).
- [4] Annalisa Buffa and Ralf Hiptmair. “Galerkin boundary element methods for electromagnetic scattering”. In: *Topics in computational wave propagation*. Vol. 31. Lect. Notes Comput. Sci. Eng. Springer, Berlin, 2003, pp. 83–124. DOI: [10.1007/978-3-642-55483-4_3](https://doi.org/10.1007/978-3-642-55483-4_3). URL: https://doi.org/10.1007/978-3-642-55483-4_3.
- [5] X. Claeys and R. Hiptmair. “Integral equations for electromagnetic scattering at multi-screens”. In: *Integral Equations Operator Theory* 84.1 (2016), pp. 33–68. ISSN: 0378-620X. DOI: [10.1007/s00020-015-2242-5](https://doi.org/10.1007/s00020-015-2242-5). URL: <https://doi.org/10.1007/s00020-015-2242-5>.
- [6] X. Claeys and R. Hiptmair. “Integral equations on multi-screens”. In: *Integral Equations Operator Theory* 77.2 (2013), pp. 167–197. ISSN: 0378-620X. DOI: [10.1007/s00020-013-2085-x](https://doi.org/10.1007/s00020-013-2085-x). URL: <https://doi.org/10.1007/s00020-013-2085-x>.
- [7] R. Hiptmair and L. Kielhorn. *BETL – A generic boundary element template library*. Tech. rep. 2012-36. Switzerland: Seminar for Applied Mathematics, ETH Zürich, 2012. URL: https://www.sam.math.ethz.ch/sam_reports/reports_final/reports2012/2012-36.pdf.
- [8] E. F. Kaasschieter. “Preconditioned conjugate gradients for solving singular systems”. In: *J. Comput. Appl. Math.* 24.1-2 (1988). Iterative methods for the solution of linear systems, pp. 265–275. ISSN: 0377-0427. DOI: [10.1016/0377-0427\(88\)90358-5](https://doi.org/10.1016/0377-0427(88)90358-5). URL: [https://doi.org/10.1016/0377-0427\(88\)90358-5](https://doi.org/10.1016/0377-0427(88)90358-5).
- [9] J.-C. Nédélec. “A new family of mixed finite elements in \mathbf{R}^3 ”. In: *Numer. Math.* 50.1 (1986), pp. 57–81. ISSN: 0029-599X. DOI: [10.1007/BF01389668](https://doi.org/10.1007/BF01389668). URL: <https://doi.org/10.1007/BF01389668>.
- [10] P.-A. Raviart and J. M. Thomas. “A mixed finite element method for 2nd order elliptic problems”. In: (1977), 292–315. Lecture Notes in Math., Vol. 606.

- [11] Stefan A. Sauter and Christoph Schwab. *Boundary element methods*. Vol. 39. Springer Series in Computational Mathematics. Translated and expanded from the 2004 German original. Springer-Verlag, Berlin, 2011, pp. xviii+561. ISBN: 978-3-540-68092-5. DOI: [10.1007/978-3-540-68093-2](https://doi.org/10.1007/978-3-540-68093-2). URL: <https://doi.org/10.1007/978-3-540-68093-2>.
- [12] Olaf Steinbach. *Numerical approximation methods for elliptic boundary value problems*. Finite and boundary elements, Translated from the 2003 German original. Springer, New York, 2008, pp. xii+386. ISBN: 978-0-387-31312-2. DOI: [10.1007/978-0-387-68805-3](https://doi.org/10.1007/978-0-387-68805-3). URL: <https://doi.org/10.1007/978-0-387-68805-3>.
- [13] Ernst P. Stephan. "Boundary integral equations for screen problems in \mathbf{R}^3 ". In: *Integral Equations Operator Theory* 10.2 (1987), pp. 236–257. ISSN: 0378-620X. DOI: [10.1007/BF01199079](https://doi.org/10.1007/BF01199079). URL: <https://doi.org/10.1007/BF01199079>.
- [14] Carolina Urzua-Torres. "Optimal Preconditioners for Solving Two-Dimensional Fractures and Screen Using Boundary Elements". Pontificia Universidad Catolica de Chile, Jan. 2014. URL: <https://repositorio.uc.cl/handle/11534/4935>.



Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):

First name(s):

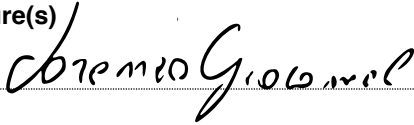
With my signature I confirm that

- I have committed none of the forms of plagiarism described in the ['Citation etiquette'](#) information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.