ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Local Multi-trace Boundary Element Formulation for Diffusion Problems

Master Thesis

Simon Pintarelli

January 22, 2013

Advisors: Prof. Dr. Ralf Hiptmair, Dr. Lars Kielhorn

SAM, ETH Zürich

Abstract

The local multi-trace boundary element formulation, a domain decomposition method developed in [4], is implemented in C++ based on BETL, the generic Boundary Element Template Library, a C++ template library written by Dr. Lars Kielhorn [5]. Dirichlet boundary conditions are implemented in an efficient way. The convergence behaviour of the iterative solver is optimized by Caldéron preconditioning. Numerical results are discussed and the code is verified by numerical examples for which an analytical solution is known.

A list of notations can be found in the appendix A on page 39.

Contents

Contents **iii**

1 Introduction **1**
1.1 Geometry 1
1.2 Diffusion problem 2
1.3 Boundary integral equations 2

2 Theory **5**
2.1 Partial transmission conditions 5
2.2 Variational formulation 6
2.3 Boundary element Galerkin discretization 7
2.4 Caldéron preconditioning 7

3 Local Multi-trace Implementation in BETL **9**
3.1 Outline 9
3.2 System matrix 12
3.2.1 Transmission matrix 13
3.2.2 Construction 13
3.3 Dirichlet boundary conditions 16
3.3.1 Implementation in BETL 17
3.4 Caldéron preconditioning 19

4 Numerical Results **23**
4.1 Convergence tests 23
4.1.1 Diffusion 27
4.2 Caldéron preconditioning 30
4.2.1 Eigenvalue distributions 33

5 Conclusion **37**

A List of notations **39**

B Input file **41**

C ACA settings **43**

D lmt_config.hpp **45**

E Code **47**
E.1 System matrix 47
E.1.1 Neumann restriction operator 47
E.1.2 Transmission matrix 49
E.2 Caldéron preconditioner 51
E.2.1 Dual mesh 61
E.2.2 Coupling operators 65
E.3 Dirichlet boundary conditions 82
E.3.1 Subdomain handler 82
E.4 Load vector 97
E.4.1 Dirichlet boundary conditions 99
E.5 Mesh generation 112

Bibliography **121**

Chapter 1

Introduction

1.1 Geometry

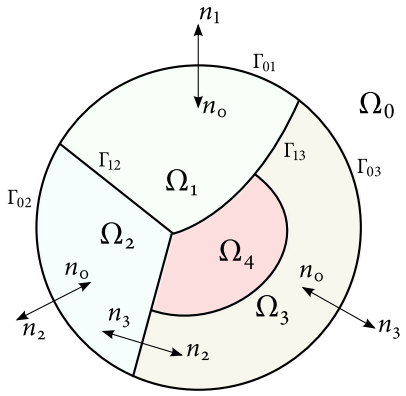


Figure 1.1: Typical geometry of a 2D "composite object", $N = 4$, induced orientations of some interfaces indicated by normal directions.

The computational domain (cf. figure 1.1) is a bounded domain $\Omega_* \subset \mathbb{R}^3$, and it is composed of so-called subdomains $\Omega_i \subset \mathbb{R}^3, i = 1, \ldots, N$. They do not intersect $(\Omega_i \cap \Omega_j = \varnothing)$ for $i \neq j$, and they form a partition of Ω_* that is

$$\overline{\Omega}_* = \bigcup_{i=1}^{N} \overline{\Omega}_i. \tag{1.1}$$

The unbounded complement of $\overline{\Omega}_*$ is connected and will provide another subdomain $\Omega_0 := \mathbb{R}^3 \setminus \cup_i \overline{\Omega}_i$. For each $i = 1, \ldots, N$, the boundary $\partial\Omega_i$ is orientable and features a unit normal vector field n_i, pointing into the exterior of Ω_i.

Γ_{ij} denotes the common interface of Ω_i and Ω_j, $\Gamma_{ij} := \overline{\Omega}_i \cap \overline{\Omega}_j$. Two subdomains Ω_i and Ω_j are adjacent, if Γ_{ij} is a 2-dimensional manifold (with boundary). The union of all genuine interfaces form the so-called *skeleton*

$$\Sigma := \bigcup_{ij} \Gamma_{ij} = \bigcup_{i=0}^{N} \partial\Omega_i. \tag{1.2}$$

1.2 Diffusion problem

$$
\begin{aligned}
&-\nabla\cdot(\alpha(x)\nabla u(x)) = 0 \quad \text{in } \Omega_i, \quad i = 0\dots N,\\
&+ \quad \text{transmission conditions across } \Gamma_{ij} \text{ for adjacent subdomains}\\
&+ \quad \text{Dirichlet boundary condtions on } \Gamma_{\text{Dir}}
\end{aligned}
\tag{1.3}
$$

Here, $\alpha(x) \in \mathbb{R}$ is considered to be constant on each subdomain Ω_i. Dirichlet boundary conditions can be imposed on $\Gamma_{\text{Dir}} = \partial\Omega_i$. Dirichlet boundary conditions on several subdomains $\partial\Omega_i$ are possible.

Sources are introduced into the model through a given function $\mathbf{u}_{inc}$, in Helmholtz terminology the incident wave, that satisfies

$$-\nabla\cdot(\alpha_0\nabla)u_{inc} = 0 \quad \text{everywhere in } \mathbb{R}^3. \tag{1.4}$$

The simplest form of u_{inc} is

$$u_{inc}(\mathbf{x}) = \mathbf{k}\cdot\mathbf{x} + \beta. \tag{1.5}$$

Where $\mathbf{k} \in \mathbb{R}^3$ and $\beta \in \mathbb{R}$. The solution to (1.3) is then sought for

$$u = u_{total} - u_{inc}.$$

1.3 Boundary integral equations

$\mathrm{T}_D, \mathrm{T}_N$ define the Dirichlet and Neumann trace operators

$$\mathrm{T}_D\, u(\mathbf{x}) := \lim_{\Omega\ni\tilde{\mathbf{x}}\to\mathbf{x}\in\Gamma} u(\tilde{\mathbf{x}}) \tag{1.6}$$

$$\mathrm{T}_N\, u(\mathbf{x}) := \lim_{\Omega\ni\tilde{\mathbf{x}}\to\mathbf{x}\in\Gamma} \alpha(\tilde{\mathbf{x}})n(\tilde{\mathbf{x}})\cdot\nabla u(\tilde{\mathbf{x}}) \tag{1.7}$$

They take a function on Ω and map it to the boundary Γ. A solution u to

$$-\nabla\cdot(\alpha u) = 0 \text{ in } \Omega \in \mathbb{R}^3, \tag{1.8}$$

is given by the representation formula

$$u = \mathrm{SL}(T_N u) - \mathrm{DL}(T_D u). \tag{1.9}$$

After application of $\mathrm{T}_D, \mathrm{T}_N$ to (1.9) and some lengthy calculations, not shown here, one arrives at the following system of equations

$$\left(\mathbb{A} + \tfrac{1}{2}\begin{pmatrix}\mathrm{Id} & \\ & \mathrm{Id}\end{pmatrix}\right)\begin{pmatrix}T_D u\\ T_N u\end{pmatrix} = \begin{pmatrix}T_D u\\ T_N u\end{pmatrix}. \tag{1.10}$$

Where the Caldéron matrix $\mathbb{A}$ is given by

$$\mathbb{A} = \begin{pmatrix} -K & V \\ W & K' \end{pmatrix}. \tag{1.11}$$

Id denotes the identity operator, V is the single layer boundary integral operator, K is the double layer boundary integral operator, and K' it's adjoint. W is called the hypersingular operator. The derivation can be found in [6, chapter 6].

The single- and double layer potential $\mathrm{SL}(\cdot), \mathrm{DL}(\cdot)$ are defined as

$$\mathrm{SL}(\varphi)(\mathbf{x}) := \int_{\partial\Omega} G(\mathbf{x}-\mathbf{y})\varphi(\mathbf{y})\, \mathrm{d}S_y \tag{1.12}$$

$$\mathrm{DL}(u)(\mathbf{x}) := \int_{\partial\Omega} T_{N,y} G(\mathbf{x}-\mathbf{y}) u(\mathbf{y})\, \mathrm{d}S_y. \tag{1.13}$$

The fundamental solution $G(\cdot)$ of

$$-\nabla_y \cdot (\alpha \nabla_y G(\mathbf{x}-\mathbf{y})) = \delta(\mathbf{x}-\mathbf{y}), \tag{1.14}$$

is

$$G(\mathbf{z}) = \frac{1}{4\pi\alpha\|z\|}. \tag{1.15}$$

In the following, we will denote the trace spaces associated to the Dirichlet and Neumann trace operators $\mathrm{T}_D, \mathrm{T}_N$ as $\mathcal{T}_D$ and $\mathcal{T}_N$. They can be merged into the Cauchy trace space

$$\mathcal{T}(\partial\Omega) := \mathcal{T}_D(\partial\Omega) \times \mathcal{T}_N(\partial\Omega). \tag{1.16}$$

The respective duality pairing is defined as

$$[\![\mathfrak{u}, \mathfrak{v}]\!]_{\mathcal{T}(\partial\Omega)} := [u, \varphi]_{\partial\Omega} + [v, \nu]_{\partial\Omega} \qquad \mathfrak{u} := \begin{pmatrix} u \\ v \end{pmatrix}, \mathfrak{v} := \begin{pmatrix} v \\ \varphi \end{pmatrix} \in \mathcal{T}(\partial\Omega) \tag{1.17}$$

and

$$[u, \varphi]_\Gamma := \int_\Gamma u\varphi \, \mathrm{d}S \tag{1.18}$$

Notation. By a subscript index we indiciate that a trace operator is applied to a particular subdomain, e.g. $\mathrm{T}_{D,i}, \mathrm{T}_{N,i}$. $\mathbb{A}_i$ is $\mathbb{A}$ assembled on $\partial\Omega_i$, with coefficient $\alpha = \alpha_i$ and all traces and potentials on $\partial\Omega_i$.

The skeleton multi-trace space is defined as

$$\mathcal{MT}(\Sigma) = \mathcal{MT}_D(\Sigma) \times \mathcal{MT}_N(\Sigma), \tag{1.19}$$

with

$$\begin{aligned} \mathcal{MT}_D(\Sigma) &= \mathcal{T}_D(\partial\Omega_0) \times \mathcal{T}_D(\partial\Omega_1) \times \cdots \times \mathcal{T}_D(\Omega_N) \\ \mathcal{MT}_N(\Sigma) &= \mathcal{T}_N(\partial\Omega_0) \times \mathcal{T}_N(\partial\Omega_1) \times \cdots \times \mathcal{T}_N(\Omega_N). \end{aligned} \tag{1.20}$$

To isolate the contributions of a single subdomain we rely on localization operators [1].

$$\mathbb{L}_i : \mathcal{MT}(\Sigma) \to \mathcal{T}(\partial\Omega_i),\ \mathbb{L}_i\vec{u} := \begin{pmatrix} u_i \\ v_i \end{pmatrix},\ \begin{array}{l} \vec{u} = (u_0, \ldots, u_N, v_0, \ldots, v_N), \\ u_i \in \mathcal{T}_D(\partial\Omega_i), v_i \in \mathcal{T}_N(\partial\Omega_i). \end{array} \tag{1.21}$$

For later usage we define also the Cauchy trace operator,

$$\mathbb{T} : \mathcal{H}(\Delta, \Omega) \to \mathcal{T}(\partial\Omega), \mathbb{T}u := \begin{pmatrix} \mathrm{T}_D\, u \\ \mathrm{T}_N\, u \end{pmatrix}. \tag{1.22}$$

which takes a function from the Sobolev space

$$\mathcal{H}(\Delta, \Omega) := \left\{ u \in L^2(\Omega) : \Delta u \in L^2(\Omega) \right\} \tag{1.23}$$

and maps it to the Cauchy trace space $\mathcal{T}(\partial\Omega)$.

[1] Functions in a multi-trace space will be distinguished by an overset arrow, e.g. $\vec{u}, \vec{v}$

Chapter 2

Theory

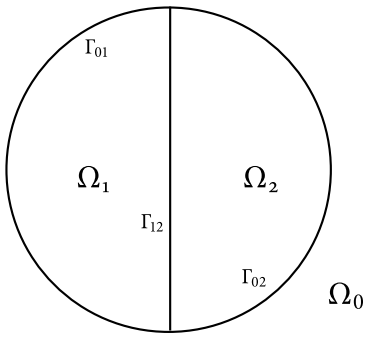


Figure 2.1: Generic geometric setting for $N = 2$

2.1 Partial transmission conditions

For the sake of simplicity, the local multi-trace formulation is first given for a generic domain with $N = 2$, as sketched in figure 2.1. The generalization to an arbitrary number of subdomains is then straight forward. Let $\vec{u} \in \mathcal{MT}(\Sigma)$.

$$(\tfrac{1}{2}\,\mathrm{Id} - \mathbb{A}_i)\mathbb{L}_i\vec{u} = 0 \quad \text{in } \mathcal{T}(\partial\Omega_i) \tag{2.1}$$

$$(\tfrac{1}{2}\,\mathrm{Id} - \mathbb{A}_0)(\mathbb{L}_0\vec{u} - \mathbb{T}_0\mathbf{u}_{\mathrm{inc}}) = 0 \quad \text{in } \mathcal{T}(\partial\Omega_0) \tag{2.2}$$

And thanks to Theorem 2.6 in [4] (2.2) can be simplified to

$$(\tfrac{1}{2}\,\mathrm{Id} - \mathbb{A}_0)\mathbb{L}_0\vec{u} = \mathbb{T}\mathbf{u}_{\mathrm{inc}}. \tag{2.3}$$

The next step is the replacement of $\frac{1}{2}\,\mathrm{Id}\,\mathbb{L}_i\vec{u}$ by the local transmission conditions

$$\mathrm{T}_{D,i}\,\mathbf{u} = \mathrm{T}_{D,j} \quad \text{and} \quad \mathrm{T}_{N,i}\,\mathbf{u} = -\,\mathrm{T}_{N,j}\,\mathbf{u} \quad \text{on } \Gamma_{ij}. \tag{2.4}$$

This is done on each interface Γ_{ij} and yields

$$\begin{array}{rcrcrcl} \mathbb{A}_0\mathbb{L}_0\vec{u} & - & \mathbb{X}_{1\to0}\mathbb{L}_1\vec{u} & - & \mathbb{X}_{2\to0}\mathbb{L}_2\vec{u} & = & -\mathbb{T}_0\mathbf{u}_{\mathrm{inc}} \\ -\mathbb{X}_{0\to1}\mathbb{L}_0\vec{u} & + & \mathbb{A}_1\mathbb{L}_1\vec{u} & - & \mathbb{X}_{2\to1}\mathbb{L}_2\vec{u} & = & 0 \\ -\mathbb{X}_{0\to2}\mathbb{L}_0\vec{u} & - & \mathbb{X}_{1\to2}\mathbb{L}_1\vec{u} & + & \mathbb{A}_2\mathbb{L}_2\vec{u} & = & 0 \end{array} \tag{2.5}$$

Where $\mathbb{X}_{i\to j}$ is called the *local transmission operator*, which, if Γ_{ij} is a genuine interface, is defined as follows

(i) take a pair of functions on $\partial\Omega_i$ as argument, corresponding to Dirichlet and Neumann traces,

(ii) restrict both functions to Γ_{ij},

(iii) flip the sign of the second function in order to take into account the transmission conditions (2.4),

(iv) and, finally, extend both functions by zero to functions on $\partial\Omega_j$.

Thus, formally this is

$$\mathbb{X}_{i\to j}: \quad \vec{u} := \begin{pmatrix} u \\ \varphi \end{pmatrix} \;\to\; (\mathbb{X}_{i\to j}\vec{u})(\mathbf{x}) := \begin{cases} \begin{pmatrix} u(\mathbf{x}) \\ -\varphi(\mathbf{x}) \end{pmatrix} & \text{for } x \in \Gamma_{ij} \\ 0 & \text{elsewhere on } \partial\Omega_j. \end{cases} \tag{2.6}$$

The generalization of (2.5) to an arbitrary number of subdomains is straight forward: find $\vec{u} \in \mathcal{MT}(\Sigma)$ that solves

$$\mathbb{A}_i\mathbb{L}_i\mathfrak{u} - \tfrac{1}{2}\sum_{\substack{j=0\\ j\neq i}}^{N} \mathbb{X}_{j\to i}\mathbb{L}_j\mathfrak{u} = \begin{cases} -\mathbb{T}_0\mathbf{u}_{\text{inc}} & \text{for } i = 0, \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

2.2 Variational formulation

The technical complications in the derivation of the variational formulation, arising from the fact that the local transmission operators fail to be continuous mappings, are completely omitted here, because they have finally no influence on the implementation aspects of the discretized version. Details can be found in [2, section 6.2].

The discrete variational formulation for $N = 2$ reads: Find $(\mathfrak{u}_0, \mathfrak{u}_1, \mathfrak{u}_2) \in \mathcal{MT}_h(\Sigma)$ such that for all $(\mathfrak{v}_0, \mathfrak{v}_1, \mathfrak{v}_2) \in \mathcal{MT}_h(\Sigma)$.

$$\begin{aligned}
\llbracket \mathbb{A}_0\mathfrak{u}_{0,h}, \mathfrak{v}_{0,h}\rrbracket_{\mathcal{T}_h(\Omega_0)} &- \tfrac{1}{2}\llbracket \mathbb{X}_{1\to 0}\mathfrak{u}_{1,h}, \mathfrak{v}_{0,h}\rrbracket_{\mathcal{T}_h(\Omega_0)} - \tfrac{1}{2}\llbracket \mathbb{X}_{2\to 0}\mathfrak{u}_{2,h}, \mathfrak{v}_{0,h}\rrbracket_{\mathcal{T}_h(\Omega_0)} &&= -\llbracket \mathbb{T}_0\mathbf{u}_{\text{inc}}, \mathfrak{v}_{0,h}\rrbracket_{\mathcal{T}_h(\Omega_0)} \\
-\tfrac{1}{2}\llbracket \mathbb{X}_{0\to 1}\mathfrak{u}_{0,h}, \mathfrak{v}_{1,h}\rrbracket_{\mathcal{T}_h(\Omega_1)} &+ \llbracket \mathbb{A}_1\mathfrak{u}_{1,h}, \mathfrak{v}_{1,h}\rrbracket_{\mathcal{T}_h(\Omega_1)} - \tfrac{1}{2}\llbracket \mathbb{X}_{2\to 1}\mathfrak{u}_{2,h}, \mathfrak{v}_{1,h}\rrbracket_{\mathcal{T}_h(\Omega_1)} &&= 0 \\
-\tfrac{1}{2}\llbracket \mathbb{X}_{0\to 2}\mathfrak{u}_{0,h}, \mathfrak{v}_{2,h}\rrbracket_{\mathcal{T}_h(\Omega_2)} &- \tfrac{1}{2}\llbracket \mathbb{X}_{1\to 2}\mathfrak{u}_{1,h}, \mathfrak{v}_{2,h}\rrbracket_{\mathcal{T}_h(\Omega_2)} + \llbracket \mathbb{A}_2\mathfrak{u}_{2,h}, \mathfrak{v}_{2,h}\rrbracket_{\mathcal{T}_h(\Omega_2)} &&= 0
\end{aligned} \tag{2.8}$$

And for an arbitrary number N of subdomains

$$\llbracket \mathbb{A}_i\mathfrak{u}_{i,h}, \mathfrak{v}_{i,h}\rrbracket_{\mathcal{T}_{h,i}} - \tfrac{1}{2}\sum_{\substack{j=0\\ j\neq i}}^{N} \llbracket \mathbb{X}_{j\to i}\mathfrak{u}_{h,j}, \mathfrak{v}_{h,i}\rrbracket_{\mathcal{T}_{h,i}} = \begin{cases} -\llbracket \mathbb{T}_0\mathbf{u}_{\text{inc}}, \mathfrak{v}_{i,h}\rrbracket_{\mathcal{T}_{h,i}} & i = 0 \\ 0 & i \neq 0 \end{cases}. \tag{2.9}$$

2.3 Boundary element Galerkin discretization

The equations are discretized using lowest order boundary elements, that is piecewise linear continuous functions for $\mathcal{T}_{D,h}$ and constant discontinuous functions for $\mathcal{T}_{N,h}$. The resulting system matrix is of size $\sum_{i=0}^{N} m_i$, $m_i := \dim \mathcal{T}_{D,h}(\partial\Omega_i) + \dim \mathcal{T}_{N,h}(\partial\Omega_i)$, and has the form

$$\begin{pmatrix} A_0 & X_{0,1} & \cdots & & \cdots & X_{0,N} \\ X_{1,0} & A_1 & X_{1,2} & \cdots & \cdots & X_{1,N} \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ & & & & & \\ \vdots & & & & & X_{N-1,N} \\ X_{N,1} & \cdots & & \cdots & X_{N,N-1} & A_N \end{pmatrix}, \tag{2.10}$$

the matrices $A_i \in \mathbb{C}^{m_i \times m_i}$, on the diagonal, arise from standard Galerkin BEM discretization of the compound boundary integral operators $\mathbb{A}_i$ as defined in (1.11). The off-diagonal matrices $X_{i,j} \in \mathbb{R}^{m_i \times m_j}$ are sparse and $X_{i,j} = 0$, if Ω_i and Ω_j have no common interface.

2.4 Caldéron preconditioning

The condition number of the system matrix (2.10) grows like $\mathcal{O}(h^{-2})$ [4, section 4], hence a preconditioner is mandatory for an efficient iterative solver. The operator preconditioning strategy presented in [3] will be employed. The preconditioner P is given by

$$P = D^{-T} B D^{-1}. \tag{2.11}$$

Where B is the block-diagonal of (2.7). The usage of a dual mesh, cf. figure 2.2 on the following page, ensures a stable duality pairing and allows matching dimensions $\dim \mathcal{T}_{h,N}(\Gamma) = \dim \mathcal{T}_{h,D}$. D is a sparse Galerkin matrix arising from the duality pairing

$$D : \mathcal{MT}_h(\Sigma) \times \widehat{\mathcal{MT}}_h(\Sigma) \mapsto \mathbb{R}, \tag{2.12}$$

where $\widehat{\mathcal{MT}}_h(\Sigma)$ denotes the lowest order basis functions on the dual mesh $\widehat{\mathcal{M}}_h$. h-independent convergence of the iterative solver is observed in practice.

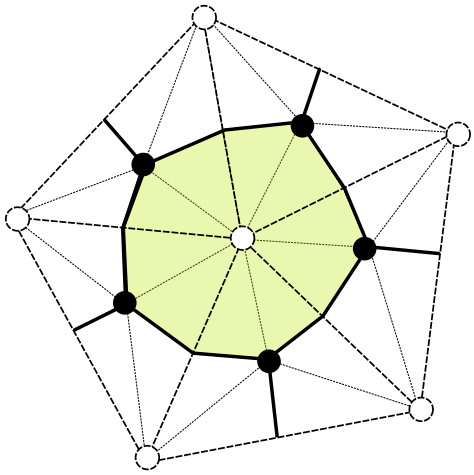


Figure 2.2: Parent and dual mesh $\mathcal{M}_h, \widehat{\mathcal{M}}_h$. Dual nodes (filled circles), dual edges (black lines), parent nodes (blank, dashed circles), parent edges (dashed lines), barycentric refinement (thin dashed lines). The dual cell around the center vertex is highlighted.

Chapter 3

Local Multi-trace Implementation in BETL

First the outline of the code is sketched by explaining the crucial steps in the main method. This is loading a mesh from an input file, enumerating the degrees of freedom, assembling the system matrix, the Caldéron preconditioner and the load vector, incorporating the Dirichlet boundary conditions, solving a linear system of equations and writing the results to an output file.

3.1 Outline

This section describes the fundamental steps of the code from the main routine. First we read the problem parameters from a file called "settings.conf", a detailed description can be found in appendix B on page 41.

```
typedef lmt::SettingsParser<std::ifstream> my_settings_parser_t;
my_settings_parser_t mySettingsParser;
mySettingsParser.read("settings.conf");
```

Next we define a mesh which can handle multiple domains.

```
typedef MultiElement< nodes_per_element > element_t;
typedef MultiMesh< element_t >            mesh_t;
mesh_t mesh(input);
```

Then the subdomains Ω_i are parsed, based on an additional input file 'mesh.sdm', which contains information about the faces forming the boundary of the subdomains and how they are connected.

```
inp::gmsh::SubdomainReader sd_reader( basename );
// create data structures in order to deal with multiple domains
mesh.extract_subdomains( input, sd_reader );
const bool reverseOrientation = true;
mesh.extract_hull(reverseOrientation);
```

The degree of freedom handler in the following denoted as Dofhandler, they are generated in a separate class called `SubdomainHandler`. Since there is one Dofhandler for each of the three different finite element spaces and on each subdomain. A

`DoFHandler` knows its finite element space and maps the basis functions from the reference element to the global basis functions, representing the finite element space.

```
typedef lmt::SubdomainHandler<acc_method, element_t> sd_handler_t;
sd_handler_t full_sd_handler( basename, NoOfDomains );
full_sd_handler.setup(mesh);
```

The class `BCSubdomainHandler` contains the Dofhandlers according to the Dirichlet boundary conditions. Details are explained in section (3.3).

```
typedef lmt::BCSubdomainHandler<acc_method, element_t>
bc_sd_handler_t;
bc_sd_handler_t bc_sd_handler( basename, NoOfDomains);
sd_handler_t& sd_handler = bc_sd_handler;
// sd handler on \Gamma_DIR
sd_handler_t nf_sd_handler( basename, NoOfDomains);
timer.restart();
// use the factory pattern to create nf_sd_handler, bc_sd_handler
lmt::BCSubdomainHandlerGenerator::Create(/* const */
                                         full_sd_handler,
                                         mesh,
                                         dirichlet_bc_set,
                                         /* will be filled */
                                         &nf_sd_handler,
                                         bc_sd_handler);
```

After having all the Dofhandlers ready, we can proceed to the next step, the creation of the boundary integral operators and some auxiliary sparse operators.

```
typedef typename lmt::config::qr_t quadrature_t;
typedef lmt::BemOperatorHandler< quadrature_t,
                                 GalerkinIntegrator,
                                 sd_handler_t,
                                 par,
                                 FS > bem_operator_handler_t;
bem_operator_handler_t bem_operator_handler( NoOfDomains,
                                             fastSettingsParser,
                                             coefficient_vec );
bem_operator_handler.setup( sd_handler, dirichlet_bc_set );
```

The template class `BemOperatorHandler` takes five template parameters, the first two describe the quadrature rule and the integrator routine. The third parameters passes the type of subdomain handler, `par` is the parallelization, usually `OpenMP`, for shared memory parallelism, `FS` stands for fundamental solution, this can be Laplace or Helmholtz. The `BemOperatorHandler` together with the auxiliary sparse operators

```
typedef lmt::SparseOperatorHandler<sd_handler_t> sparse_op_handler_t;
sparse_op_handler_t sparse_op_handler(NoOfDomains);
sparse_op_handler.setup(sd_handler);
```

are passed to the Caldéron handler, where the matrices V_i, K_i, W_i are inserted into the composite matrix A_i.

```
typedef lmt::CalderonHandler<bem_operator_handler_t,
                             sparse_op_handler_t> calderon_handler_t;
calderon_handler_t calderon_handler(NoOfDomains,
                                    coefficient_vec);
calderon_handler.setup( bem_operator_handler,
                        sparse_op_handler,
                        dirichlet_bc_set );
```

The creation of the local transmission operators, (cf. equation 2.6 on page 6), is done inside the class `LocalisationOperatorHandler`.

```
typedef lmt::LocalisationOperatorHandler<sd_handler_t>
  loc_op_handler_t;
loc_op_handler_t loc_op_handler(NoOfDomains+1);
loc_op_handler.compute(sd_handler);
```

The factory `SysteMatrixGenerator` takes the BEM matrices, the auxiliary sparse matrices, the local transmission operators and creates the system matrix.

```
typedef lmt::SystemMatrix<numeric_t> system_matrix_t;
system_matrix_t system_matrix( NoOfDomains+1 );
lmt::SystemMatrixGenerator::Create( /* constant */
                                    calderon_handler,
                                    bc_bem_operator_handler,
                                    sparse_op_handler,
                                    loc_op_handler,
                                    NoOfDomains+1,
                                    dirichlet_bc_set,
                                    /* will be filled */
                                    system_matrix );
```

Assembly of the load vector:

```
lmt::RhsGenerator<FS>::Create( /* const input */
                               sd_handler,
                               full_sd_handler,
                               dirichlet_bc_set,
                               mySettingsParser,
                               /* this data will be filled */
                               rhs);
```

Finally we set up the Caldéron preconditioner,

```
typedef lmt::PreconditionerHandler preconditioner_handler_t;
preconditioner_handler_t preconditioner_handler( basename,
                                                 NoOfDomains,
                                                 fastSettingsParser,
                                                 coefficient_vec );
preconditioner_handler.setup( mesh,
                              sd_handler,
                              exclusion_op_h,
                              dirichlet_bc_set);
```

and call the solver.

```
typedef solver::GMRes<numeric_t,true> solver_t;
const std::size_t max_iter = mySettingsParser.get_gmres_maxiter();
const double      tol      = mySettingsParser.get_gmres_tol();
solver_t solver(size, max_iter, tol);
vector_t sol(size, numeric_t(0.0));
solver.solve( system_matrix,
              rhs.give_data(),
              sol.give_data(),
              preconditioner_handler.giveMatrix() );
```

3.2 System matrix

The class `SystemMatrix` provides the MV-product with the system matrix, cf. (2.10). The presence of Dirichlet boundary conditions affords two different types of rows in the system matrix. The first type, for a regular domain

$$\left(-\tfrac{1}{2}X_{0\to i} \quad -\tfrac{1}{2}X_{1\to i} \quad \cdots \quad A_i \quad -\tfrac{1}{2}X_{i+1\to i} \quad \cdots \quad -\tfrac{1}{2}X_{N\to i}\right). \tag{3.1}$$

The second type, for a domain Ω_i, where $\partial\Omega_i \in \Gamma_{Dir}$, coincides with the system matrix of a Dirichlet boundary value problem

$$\left(\cdots \quad V_i \quad \cdots\right) \tag{3.2}$$

The implementation of the class `SystemMatrix` is given in listing 3.1.

Listing 3.1: SystemMatrix

```
1   template< typename NUMERIC_T=double>
    class SystemMatrix
      : public linalg::MatrixExpression<SystemMatrix<NUMERIC_T> > {
    public:
      typedef NUMERIC_T numeric_type;
6     typedef numeric_type numeric_t;
    private:
      typedef SystemMatrixBlckRow<NUMERIC_T> Row_type;
      typedef boost::shared_ptr<Row_type>    R_ptr_t;
      typedef std::vector<R_ptr_t>           R_vec_t;
11  public:
      SystemMatrix(std::size_t blocksize);
      void SetRow(R_ptr_t& R, std::size_t id);
      void Finalize();
      void amux ( numeric_t alpha, numeric_t* x, numeric_t* y ) const;
16    void tamux( numeric_t alpha, numeric_t* x, numeric_t* y ) const;
      const std::size_t GiveRows( ) const { return size_; }
      const std::size_t GiveCols( ) const { return size_;}

      template<typename EXPORTER>
21    void Write( EXPORTER& exporter) const;
    private:
      std::size_t blocksize_;
      std::size_t size_;
```

```
    R_vec_t R_vec_;
26  bool finalize_;
};
```

Pointers to polymorphic objects implementing the MV-product of (3.1) and (3.2) are stored in the array `R_vec_` (line 25 in listing 3.1), they allow to have the boundary conditions known at run time. This is the only point in the code where a virtual function call is used. In general this should be avoided. But in this case the overhead can be neglected, since only N virtual function calls per MV-product are executed.

3.2.1 Transmission matrix

The class `TransmissionMatrix` provides the MV-product with the local transmission operator, cf. (2.6),

$$-0.5\begin{pmatrix} X_D & \\ & -X_N \end{pmatrix}. \tag{3.3}$$

The minus sign accounts for the Neumann transmission condition (2.4). The factor $-\frac{1}{2}$ originates from (2.9).

```
template<typename NUMERIC_T>
void TransmissionMatrix::
3   amux( NUMERIC_T alpha, NUMERIC_T* x, NUMERIC_T* y ) const
{
  const NUMERIC_T d2_alpha(d2_alpha_);
  // neumann transmission conditions
  const NUMERIC_T factor(-0.5); // -0.5 X_{ij}
8   D1_.amux(factor*alpha,x,y);
  std::size_t offsetX = D1_.GiveCols();
  std::size_t offsetY = D1_.GiveRows();
  D2_.amux(factor*d2_alpha*alpha, &x[offsetX], &y[offsetY]);
}
```

3.2.2 Construction

The system matrix is created by the factory `SystemMatrixGenerator`, cf. listing 3.2. The rows of the system matrix are created and the correct coupling of Dirichlet and Neumann degrees of freedom across adjacent subdomains is ensured. The transmission matrices are created in lines 74-81. The definition of the transmission operators was given in (2.9 on page 6)

$$-\tfrac{1}{2}[\![\mathbb{X}_{j\to i}\mathfrak{u}_{j,h},\mathfrak{v}_{i,h}]\!]_{\mathcal{T}_h(\Omega_i)}. \tag{3.4}$$

The discretization of (3.4) has the form

$$-\tfrac{1}{2}\begin{pmatrix} R\,Id\,L_D & \\ & -Id^T\,L_N \end{pmatrix}. \tag{3.5}$$

Where Id is the overlap mass matrix between $\mathcal{T}_{D,h}$ and $\mathcal{T}_{N,h}$ on subdomain i. L_D, L_N are the localization operators mapping the degrees of freedom from $\partial\Omega_j$ to $\partial\Omega_i$.

Special care has to be taken when mapping the Dirichlet degrees of freedom. Since the mass matrix Id is assembled on $\partial\Omega_i$ and by definition, cf. section 2.1 on page 5, the local transmission operator sets the function outside the common interface Γ_{ij} to zero, the overlaps of the linear continuous basis functions sitting on the edge between $\Omega_i \setminus \Gamma_{ij}$ and Γ_{ij} and the constant discontinuous basis functions on $\partial\Omega_i \setminus \Gamma_{ij}$ have to be removed. This is achieved by the sparse matrix R, cf. (3.5). The implementation can be found in appendix E.1.1 on page 47.

If $\partial\Omega_j \subset \Gamma_{DIR}$, there is no mapping of Dirichlet degrees of freedom between $\partial\Omega_j$ and $\partial\Omega_i$. In order to avoid a polymorphic inheritance of the `TransmissionMatrix`, X_D in (3.3) is replaced by a dummy matrix of dimension 0×0.

Listing 3.2: System matrix generator

```
};
//
    ------------------------------------------------------------------

template< typename CALDERON_HANDLER,
          typename BC_BEM_OP_HANDLER,
          typename SPARSE_OP_HANDLER,
          typename LOCALISATION_OP_HANDLER,
          typename SETTINGS,
          typename SYSTEM_MATRIX>
void SystemMatrixGenerator::
Create( const CALDERON_HANDLER&         calderon_h,
        const BC_BEM_OP_HANDLER&        bc_bem_op_h,
        const SPARSE_OP_HANDLER&        sp_op_h,
        const LOCALISATION_OP_HANDLER&  loc_op_h,
        const SETTINGS&                 settings,
        const std::size_t               blocksize,
        const index_set_t&              dirichlet_bc_id,
        SYSTEM_MATRIX&                  system_matrix )
{
  typedef typename SYSTEM_MATRIX::numeric_t numeric_t;
  typedef SystemMatrixBlckRow<numeric_t> row_t;
  typedef boost::shared_ptr<row_t> row_ptr_t;
  typedef TransmissionMatrix  X_t;
  typedef boost::shared_ptr<X_t>   X_ptr_t;
  typedef typename index_set_t::iterator iter;


  // helper empty sparse matrix
  double vals[0];
  int indices[0];
  int rows=0;
  int cols=0;
  int nnz=0;
  std::vector<int> indexPtr(1,0);
```

```
sparse_matrix_t empty_mat(rows,cols,nnz,vals,indices,&indexPtr
    [0]);
std::vector<double> coeffs = settings.get_kappa();

std::size_t nrows=0; // counter used to set offset in
    BCBlockRow
for (std::size_t rowID=0; rowID < blocksize; ++rowID) {
  iter it = dirichlet_bc_id.find(rowID);
  if( it != dirichlet_bc_id.end() ) {
    /* subdomain with Dirichlet BC  */
    typedef typename BC_BEM_OP_HANDLER::slp_bemoperator_t::
        const_reference
      slp_matrix_t;
    typedef DBVPBlockRow<slp_matrix_t,numeric_t>
        bc_block_row_t;

    row_ptr_t row_ptr(new bc_block_row_t( /* matrix V */
                                          bc_bem_op_h.giveV(
                                              rowID),
                                          /* offset */
                                          nrows,
                                          rowID) );
    system_matrix.SetRow(row_ptr, rowID);
    nrows += row_ptr->GiveRows();
  } else {
    /* regular subdomain */
    typedef TPBlockRow<CALDERON_HANDLER> block_row_t;
    block_row_t* block_row_ptr = new block_row_t( rowID,
                                                  blocksize,
                                                  calderon_h )
                                                      ;
    row_ptr_t row_ptr( block_row_ptr );
    system_matrix.SetRow(row_ptr, rowID);
    nrows += row_ptr->GiveRows();
    for(std::size_t colID=0; colID < blocksize; ++colID) {
      /* set X_ij */
      typedef typename LOCALISATION_OP_HANDLER::loc_pair_t
        loc_pair_t;
      typedef typename SPARSE_OP_HANDLER::identity_op_t::
          const_reference
        id_matrix_t;
      if(colID != rowID) {
        // are there dirichlet BC on |Omega(colID) ?
        // there is no coupling of Dirichlet and Neumann dofs
            with |Gamma_DIR
        bool is_dirichlet_col = (dirichlet_bc_id.find(colID)
            != dirichlet_bc_id.end());
        const loc_pair_t* loc_pair = loc_op_h.give_loc_op(
            rowID,colID);
        const sparse_matrix_t& LD = loc_pair->first->
            giveMatrix();
        const sparse_matrix_t& LN = loc_pair->second->
            giveMatrix();
        const sparse_matrix_t& R  = loc_op_h.
```

```
                    give_restriction_op(rowID,colID).giveMatrix();
                const id_matrix_t&      ID = sp_op_h.give_identity_op(
                    rowID).giveMatrix();
#ifdef __LAPLACE
78              double f = -1.0;
#endif
#ifdef __HELMHOLTZ
                double f = -1.0;
#endif
83              X_ptr_t X_ptr(new TransmissionMatrix(f) );
                block_row_ptr->set_X( X_ptr, colID);
                if(rowID < colID) {
                  X_ptr->giveD1() = (! is_dirichlet_col) ? (R*ID*LD) :
                       empty_mat;
                  X_ptr->giveD2() = sparse::trans(ID)*LN;
88              } else {
                  X_ptr->giveD1() = (! is_dirichlet_col) ? R*ID*sparse
                      ::trans(LD) : empty_mat;
                  X_ptr->giveD2() = sparse::trans(ID)*sparse::trans(LN
                      );
                }
              }
93          } // end iterator over cols (without BC)
          }
        } // end iterate over rows
        system_matrix.Finalize();
      }
98  } // end namespace lmt
} // end namespace bet1

#endif /* __SYSTEM_MATRIX_GENERATOR_H__ */
```

3.3 Dirichlet boundary conditions

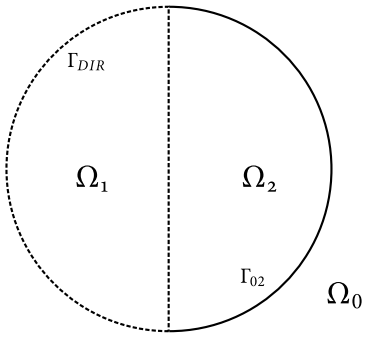


Figure 3.1: Generic domain for $N = 2$, Dirichlet boundary conditions are imposed on $\Gamma_{DIR} = \partial\Omega_1$.

For the sake of a simpler notation, the system matrix in presence of boundary conditions is shown for a generic domain with $N = 2$, Dirichlet boundary conditions are imposed on $\Gamma_{DIR} = \partial\Omega_1$, cf. figure 3.1 on the facing page, i.e. $\mathrm{T}_D\, u|_{\Gamma_{DIR}} = g$.

Insertion of the test functions $\varphi \in \tilde{S}^1_h \times S^0_h \subset \widetilde{H}^{\frac{1}{2}}(\partial\Omega_i \setminus \Gamma_{DIR}) \times H^{-\frac{1}{2}}(\partial\Omega_i)$ into (2.8 on page 6) leads to

$$\begin{bmatrix} A_0 & & -\frac{1}{2}X_{2,0} \\ & V_1 & \\ -\frac{1}{2}X_{0,2} & & A_2 \end{bmatrix} \begin{bmatrix} \begin{bmatrix} \widetilde{u}^D_0 \\ u^N_0 \end{bmatrix} \\ u^N_1 \\ \begin{bmatrix} \widetilde{u}^D_2 \\ u^N_2 \end{bmatrix} \end{bmatrix} = rhs. \tag{3.6}$$

Cancellation of $X^D_{1,\star}$ is immediate. $X^N_{1,\star}$ disappears also, since it is a local operator and tested with functions in $\tilde{S}^1_h$.

The contribution to the right hand side is

$$\cdots = \begin{bmatrix} \begin{bmatrix} (\frac{1}{2}Id_0 + K_0)g_0 \\ -W_0 g_0 \end{bmatrix} \\ (K_1 - \frac{1}{2}Id_1)g_1 \\ \begin{bmatrix} (\frac{1}{2}Id_2 + K_2)g_2 \\ -W_2 g_2 \end{bmatrix} \end{bmatrix}, \tag{3.7}$$

where the hypersingular matrix W_i is

$$W_i[k,l] = [\varphi_{Y,k}, \mathrm{W}\,\varphi_{X,l}]_{\partial\Omega_i}, \tag{3.8}$$

for $\varphi_{Y,k}, \varphi_{X,l}$

$$\varphi_{Y,k} \in S^1_{Y,h} \subset H^{\frac{1}{2}}(\partial\Omega_i \setminus \Gamma_{DIR}), \quad \varphi_{X,l} \in S^1_{X,h} \subset H^{\frac{1}{2}}(\Gamma_{DIR}),$$

and the double layer matrix K_i is

$$K_i[k,l] = [\nu_{Y,k}, \mathrm{K}\,\varphi_{X,l}]_{\partial\Omega_i},$$

for $\nu_{Y,k}, \varphi_{X,l}$

$$\nu_{Y,k} \in S^0_{Y,h} \subset H^{-\frac{1}{2}}(\partial\Omega_i), \quad \varphi_{X,l} \in S^1_{X,h} \subset H^{\frac{1}{2}}(\Gamma_{DIR}).$$

The test and trial spaces for the identity matrix Id_i are analogous to K_i.

3.3.1 Implementation in BETL

There is no concept of boundary conditions in BETL. The main obstacle is that the Dofhandler provided by BETL offers no possibility to remove single degrees of freedom from the enumeration.

The following example illustrates the difficulties in implementing Dirichlet boundary conditions. Let us consider an integral operator B, with test and trial spaces in S^1_h, in one

dimension on the domain $[0,1]$, with the elements $e_i = [ih, (i+1)h]$ $i \in 0, \ldots, N-1$ and with Dirichlet boundary conditions $u(x=0) = g(0)$ and $u(x=1) = g(1)$. Then it is necessary to remove the first node of the first element and the second node from the last element from the enumeration of the Dofhandler before it is passed to the assembly routine of B.

A simple workaround is to assemble B on the entire domain and then employ a sparse matrix EX with entries zero or one used to select the appropriate matrix entries. And write for the matrix on the effective degrees of freedom

$$B_{BC} = EX^T BEX, \tag{3.9}$$

if we assume that the test and trial spaces are identical. For the above example $EX \in [0,1]^N \times [0,1]^{N-2}$ would be given by

$$EX = \begin{pmatrix} 0 & & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 0 & \ddots & & \\ \vdots & & & 1 \\ 0 & & \cdots & 0 \end{pmatrix}$$

However, this method has the drawback that the matrix B is assembled on a too large set of degrees of freedom. And moreover, if the matrix product in (3.9) is not calculated explicitly, there is additional work for every MV-product with B_{BC}.

Since matrices arising from discretized boundary integral operators are dense, and in BEM code most of the processor time is used to assembly the matrices, neither the assembly of a matrix of which a substantial part of the entries is "thrown away" afterwards nor a MV-product of the form (3.9), where $\dim B \gg \mathrm{ncols}(EX)$, is a desirable task.

In order to avoid most of this unnecessary computational complexity, the implementation of the Dirichlet boundary conditions for the local multi-trace formulation is done in two steps, despite the that fact the implementation based on the left- and right-multiplication with a sparse matrix would lead to much simpler code.

In a first step all the elements intersecting with Γ_{DIR} are removed from the Dofhandler enumerating the Dirichlet degrees of freedom, and in a second step, the workaround with the left- and right-multiplication with a sparse matrix is used. Because the nodes on the edge connecting $\partial\Omega_i \setminus \Gamma_{DIR}$ and Γ_{DIR} have to be removed as well. Please note that there is actually no other alternative, since it is not possible to pick single entries from a boundary element matrix, when adaptive cross approximation (ACA) is turned on.

Finally, there is a lot of additional code involved for the implementation of the Dirichlet boundary conditions. A new subdomain handler is required, the code can be found in listing E.16 on page 89 and E.14 on page 82, a handler for the BEM matrices appearing in the right hand side, cf. (3.7 on the preceding page) and code to assemble the contributions to the load vector, cf. listing E.4.1 to E.27 on pages 99–117 .

3.4 Caldéron preconditioning

Handler

The construction of the Caldéron preconditioner is encapsulated in the class `PreconditionerHandler`, cf. listing E.3 on page 51, the steps are similar to the one of the system matrix and therefore not commented again.

Matrix

The class `CalderonPreconditioner` implements the abstract interface of `MatrixExpression`, this means it provides the member functions `amux, tamux, GiveRows, GiveCols`. Thus it behaves as an ordinary matrix, additionaly it provides the member function `template< class T> void operator()(T* x)` for compatibility with the preconditioner object expected by the GMRES solver.

According to section 2.4 on page 7, the Caldéron preconditioner reads

$$P = D^{-H} B D^{-1} \tag{3.10}$$

where

$$B = \begin{pmatrix} A_0 & & & & \\ & A_1 & & & \\ & & \ddots & & \\ & & & A_{N-1} & \\ & & & & A_N \end{pmatrix} \tag{3.11}$$

The sparse matrix D arising from the discretization of the duality pairing, is assembled inside the class. Since D is sparse, direct LU-factorization is used for the inversion. Treatment of Dirichlet boundary conditions is explicitly realized inside the class. Because of the product structure of P and the inversion of D, the sparse exclusion matrices have to be applied inside the class.

The geometric realization of the dual mesh is furnished by a barycentric refinement of the parent mesh. Direct assembly of the boundary integral operators on the dual mesh is not possible. Instead, they are constructed on the barycentric refinement, and then coupling operators are used to transfer them to the dual mesh. The coupling matrices $C_{\cdot,\cdot}$ take the basis functions on the barycentric refined mesh and couple them to basis functions on the dual mesh or the original mesh.

The Caldéron matrices on the dual mesh, the entries of the diagonal matrix B in 3.11, are given as

$$A_{\{\widehat{D},\widehat{N}\},\{\widehat{D},\widehat{N}\}} := \begin{pmatrix} C^T_{\widetilde{N},\widehat{N}} & \\ & C^T_{\widetilde{D},\widehat{D}} \end{pmatrix} \begin{pmatrix} -K_{\widetilde{N},\widetilde{D}} & V_{\widetilde{N},\widetilde{N}} \\ W_{\widetilde{D},\widetilde{D}} & K'_{\widetilde{D},\widetilde{N}} \end{pmatrix} \begin{pmatrix} C_{\widetilde{D},\widehat{D}} & \\ & C_{\widetilde{N},\widehat{N}} \end{pmatrix}. \tag{3.12}$$

The meaning of the subscripts is the following: $A_{\{\widetilde{N},\widetilde{D}\}}$ is an operator assembled with trial and test functions from the basis function space on the barycentric refined mesh. A $\hat{}$ on the subscript indicates that the coupling operator to the dual mesh was applied. The implementation of the coupling matrices is listed in appendix E.2.2 on page 65.

The matrix D takes the form

$$D = \begin{pmatrix} \mathrm{Id}_0 & & & & \\ & \mathrm{Id}_1 & & & \\ & & \ddots & & \\ & & & \mathrm{Id}_{N-1} & \\ & & & & \mathrm{Id}_N \end{pmatrix}, \tag{3.13}$$

where Id denotes the identity matrix

$$\mathrm{Id}_i := \mathrm{Id}_{\{D,N\},\{\widehat{N},\widehat{D}\}}\,.$$

A plain subscript, without $\hat{}$, means that the coupling matrix to the parent mesh has been applied. The implementation of (3.10 on the previous page) looks like:

Listing 3.3: Amux routine of the Caldéron preconditioner

```
template< typename SPARSE_OP_H_T,
          typename CALDERON_OP_H_T,
          typename HYPER_MATRIX_H_T,
          typename BC_DOF_H_T>
 template< class T >
 std::size_t CalderonPreconditioner< SPARSE_OP_H_T,
                                     CALDERON_OP_H_T,
                                     HYPER_MATRIX_H_T,
                                     BC_DOF_H_T>::
 amux_tp_(T alpha, T* x, T* y, std::size_t id, char op) const
 {
   const T one = T(1.0);
   const int nrhs = 1; // superlu
   typedef typename SPARSE_OP_H_T::pair_bary_t pair_bary_t;
   typedef typename SPARSE_OP_H_T::CBD_t CBD_t;
   typedef typename SPARSE_OP_H_T::CBN_t CBN_t;
   pair_bary_t* bary_coupling = sparse_op_h_.give_b_coupling_operator( id );
   const r_sparse_matrix_t CBD = bary_coupling->first->giveMatrix();
   const r_sparse_matrix_t CBN = bary_coupling->second->giveMatrix();
   slu_wrapper_t& invMD = *(lu_vec_[id].first);  // N, \tilde{D}
   slu_wrapper_t& invMN = *(lu_vec_[id].second); // D, \tilde{N}
   const std::size_t sizeD = invMD.GiveCols();
   const std::size_t sizeN = invMN.GiveCols();
   // matrix to remove nodes "aka dual cells" adjacent to
   // \Gamma_DIR
   const r_sparse_matrix_t EX = bc_dof_h_.giveMatrix(id);
   //
   //  | B_N^t          |    |  K      V  |   | B_D       |
   //  |                |  x |            | x |           |
   //  |        B_D^t   |    |  W     -K' |   |      B_N  |
```

```
    //
    if( op == 'N') {
      (diag_matrix(invMN,invMD)*
       diag_matrix(EX*(CBN^'T'), CBD^'T')*
       calderon_op_h_.giveMatrix(id)*
       diag_matrix(CBD,CBN*(EX^'T'))*
       diag_matrix(invMD,invMN))
        .amux(alpha, x, y);
    } else {
      (diag_matrix(invMN,invMD)*
       diag_matrix(EX*CBN^'T', CBD^'T')*
       calderon_op_h_.giveMatrix(id)*
       diag_matrix(CBD,CBN*(EX^'T'))*
       diag_matrix(invMD,invMN))
        .tamux(alpha, x, y);
    }
    // return offset
    return sizeD+sizeN;
  }
```

The above code is a good example for the usefulness of BETL's matrix expression template feature. It improves the readability of the code and helps to avoid programming errors caused by the manual handling of temporary arrays needed in matrix-matrix-products.

It is emphasized that the special geometric requirement of the dual mesh entails some additional computational complexity. First, the assembly of the discretized boundary integral operators require six times as many quadratures compared to the original mesh. And secondly, the matrix vector product, occurring in every step of the iterative solver, requires a matrix vector product on the space of the barycentric refinement and two times the application of the coupling operator, which is $\mathcal{O}(N)$. Where N is the size of the matrix on the barycentric refinement. In practice, this means that in order for the preconditioner to pay off in terms of processor time, the mesh width must be choosen sufficiently fine.

Preconditioning of V

The single layer potential, appearing in the rows of the system matrix corresponding to subdomains where Dirichlet boundary conditions are imposed, is preconditioned by

$$P_i = D_i^{-H} \widetilde{W}_i D_i^{-1}$$

where the stabilized hyper matrix $\widetilde{W}$ is

$$\widetilde{W} := W + \mathbf{a}\mathbf{a}^T$$

and $\mathbf{a}$ is defined as

$$\mathbf{a} := M(1, \ldots, 1)^T, \quad M = [\varphi_i^1, \varphi_i^1]_\Gamma,$$

where $\varphi_i^1 \in S_h^1(\Gamma)$ are piecewise linear continuous basis functions. The code can be found in listing E.6 on page 60.

Chapter 4

Numerical Results

Unless specified, it is assumed that all diffusion constants $\alpha_i = 1$ and that $u_{inc} \equiv 0$. Adaptive cross approximation, provided by the AHMED-Library, was enabled for all computions. The ACA parameters can be found in appendix C. Parallelization of the code is achieved through AHMED and by linking to the multi threaded version of Intels math kernel library MKL. Moreover, the code relies on the SuperLU and the CXSparse library.

4.1 Convergence tests

A sphere, composed of two half spheres, with radius one is placed at the origin, the diffusion constants α_i are all set to one and u_{inc} is choosen as $4\sqrt{\frac{\pi}{5}} r^2 Y_2^0 = r^2(3\cos^2\theta - 1)$. Since all diffusion constants are equal, the solution u is $u = u_{inc}$. Solutions are computed iteratively, by the generalized minimal residual method (GMRES), on flat triangulations with uniform mesh width h until a relative residual of 10^{-7} is reached. $\|u_h - u\|_{L_2}$ converges with $\mathcal{O}(h^2)$ and $\|\frac{\partial u_h}{\partial n} - \frac{\partial u}{\partial n}\|_{L_2}$ converges with $\mathcal{O}(h)$, see figure 4.1 on the next page. The same parameters are used for a box, also composed of two subdomains. For the box the error in u is $\mathcal{O}(h^2)$, and $\mathcal{O}(h^{1.5})$ for $\frac{\partial u_h}{\partial n}$, cf. figure 4.2 on the following page. There is a difference in the convergence rate for $\frac{\partial u_h}{\partial n}$, because flat triangles were used for the approximation of the sphere.

In order to verify the implementation of the Dirichlet boundary conditions, the following test problem is considered.

$$\begin{aligned} -\Delta u(\mathbf{x}) &= 0 \quad \mathbf{x} \in \mathbb{R}^3 \\ u_{inc} &\equiv 0 \\ g(\mathbf{x}) &= \frac{1}{\|\mathbf{x}\|} \quad \mathbf{x} \text{ on } \Gamma_{DIR} \end{aligned} \tag{4.1}$$

Where Dirichlet boundary conditions are imposed on a cube centered at the origin. The exact solution u is then given by $u = \frac{1}{\|x\|}$, since we have choosen $u|_{\Gamma_{DIR}} = G(\mathbf{x})$. Again quadratic convergence rates in u are obtained for the unit cube (8 subdomains) and the box (2 subdomains), cf. figures 4.5 and 4.6 on page 26.

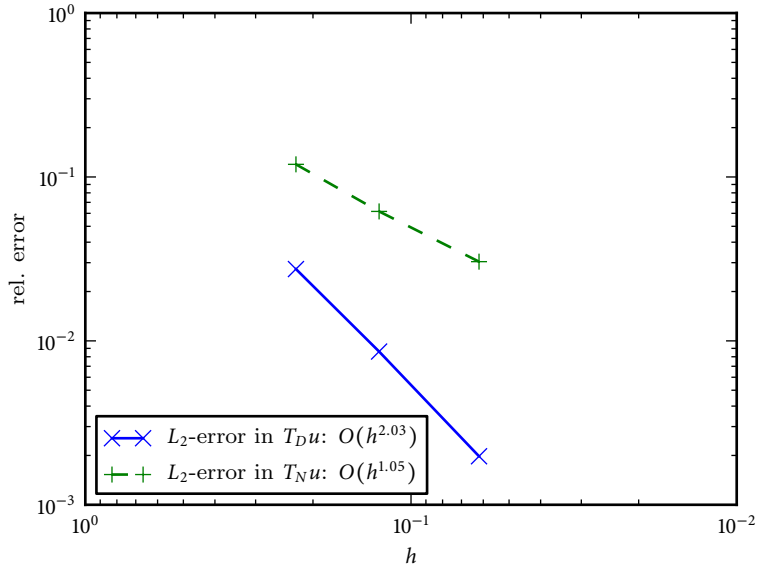


Figure 4.1: Convergence rates for the unit sphere, two subdomains, $u_{inc} = 4\sqrt{\frac{\pi}{5}} r^2 Y_2^0$

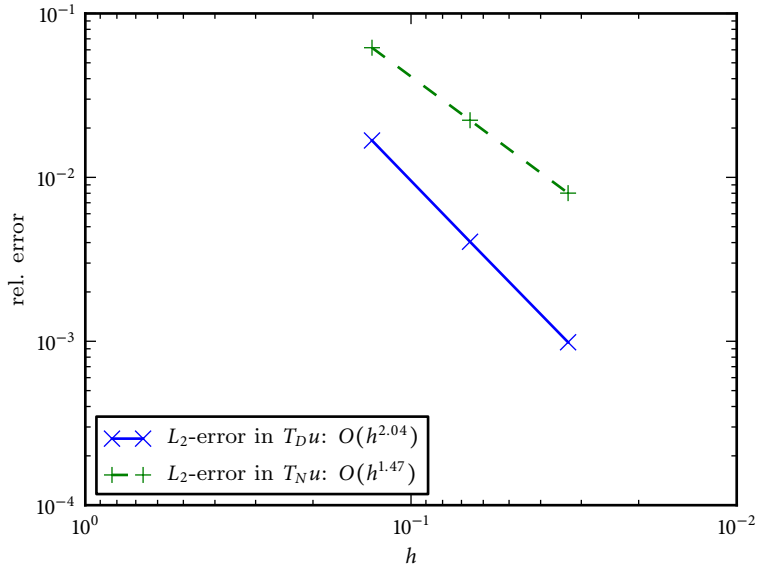


Figure 4.2: Convergence rates for the box, two subdomains, $u_{inc} = 4\sqrt{\frac{\pi}{5}} r^2 Y_2^0$.

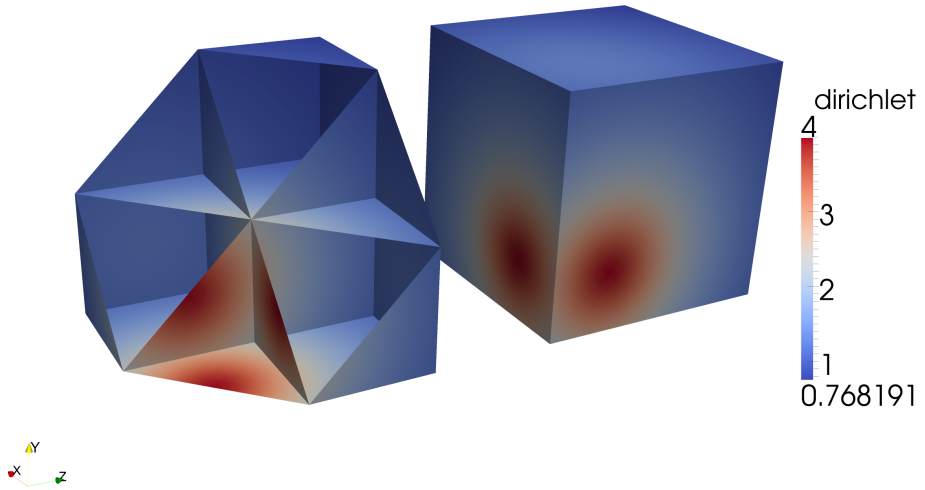


Figure 4.3: Solution u, unit cube, 8 subdomains, Dirichlet boundary conditions on $\partial\Omega_1$, $g = \frac{1}{r}$. The mesh contains 9216 elements. Left: $\partial\Omega_1 \dots \partial\Omega_8$, Right: Hull, i.e. $\partial\Omega_0$.

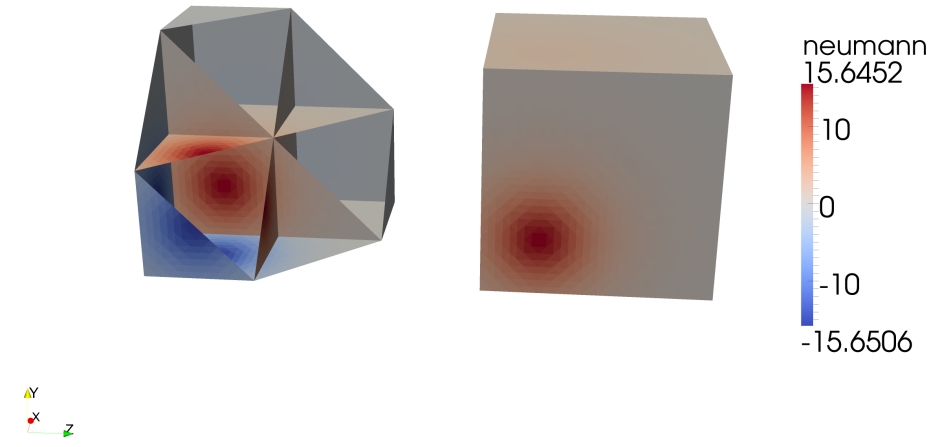


Figure 4.4: Solution $\frac{\partial u}{\partial n}$, unit cube, 8 subdomains, Dirichlet boundary conditions on $\partial\Omega_1$,$g = \frac{1}{r}$. The mesh contains 9216 elements. Left: $\partial\Omega_1 \dots \partial\Omega_8$, Right: Hull, i.e. $\partial\Omega_0$.

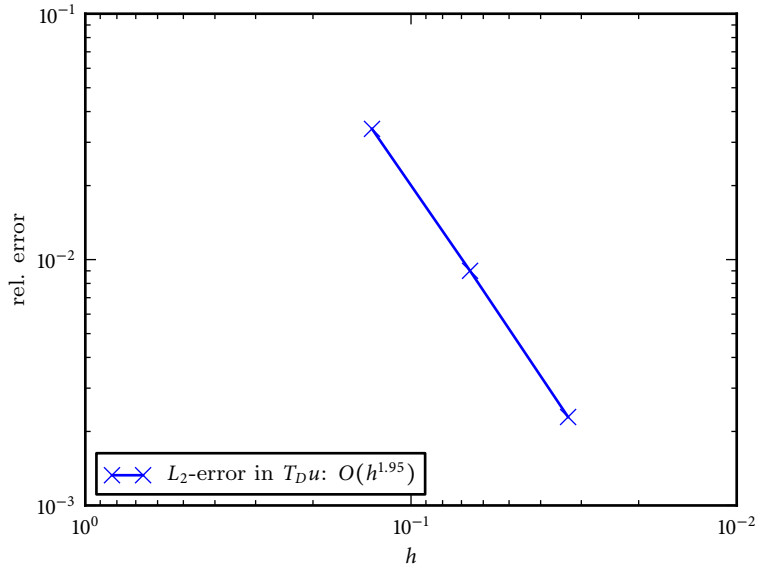


Figure 4.5: Unit cube, 8 subdomains, $g = \frac{1}{r}$

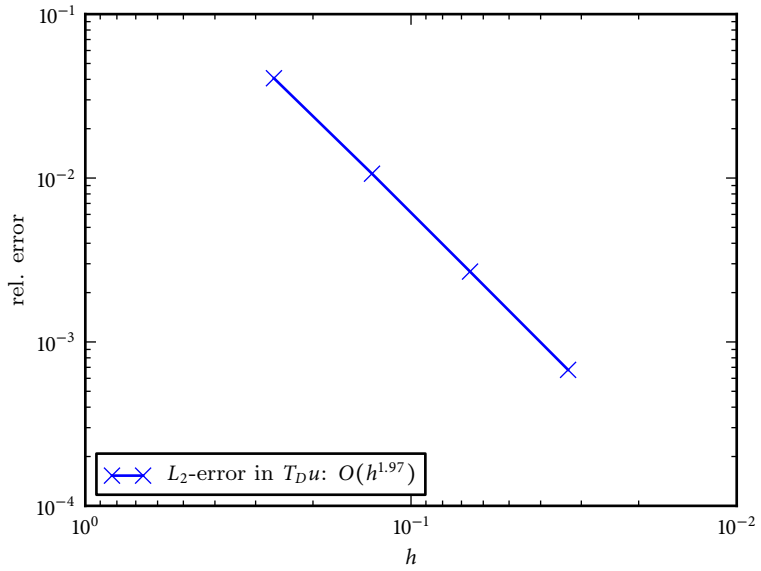


Figure 4.6: Box, 2 subdomains, $g = \frac{1}{r}$

4.1.1 Diffusion

An analytical solution for diffusion constants $\alpha_0 \neq \alpha_1 \equiv \alpha_2 \equiv \cdots \equiv \alpha_N$ can be found for a sphere. The source is modelled by u_{inc} of the following form,

$$U_{inc} = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} c_{l,m}^{inc} r^l Y_l^m, \tag{4.2}$$

where Y_l^m are the spherical harmonics. In order to fullfil the radiation conditions, i.e. $\lim_{r->\infty} u = \mathcal{O}(r^{-1})$, the follwing ansatz for U is made,

$$\begin{aligned} U_0 &= U_{inc} + U_0^s \\ U_0^s &= \sum_{l=0}^{\infty} \sum_{m=-l}^{l} c_{l,m}^0 r^{-(l+1)} Y_l^m \\ U_1 &= \sum_{l=0}^{\infty} \sum_{m=-l}^{l} c_{l,m}^1 r^l Y_l^m. \end{aligned} \tag{4.3}$$

The transmission conditions read

$$\begin{aligned} \alpha_0 U_0 - \alpha_1 U_1 &= 0 \\ \alpha_0 \frac{\partial U_0}{\partial n} - \alpha_1 \frac{\partial U_1}{\partial n} &= 0. \end{aligned} \tag{4.4}$$

Inserting the ansatz for U_0 and U_1 into (4.3) and integration with $\int \cdot Y_l^{m\,*}\, \mathrm{d}\Omega$ yields the coefficients $c_{l,m}^0, c_{l,m}^1$,

$$\begin{aligned} c_{l,m}^0 &= \frac{l(\alpha_0 - \alpha_1)}{l\alpha_1 + (l+1)\alpha_0} \\ c_{l,m}^1 &= c_{l,m}^{inc} + c_{l,m}^0. \end{aligned} \tag{4.5}$$

Inspection of the eigenvalue distributions, cf. figures (4.9) and (4.10 on page 29), reveals that the system matrix becomes more ill-conditioned when the jump in the diffusion constants is increased.

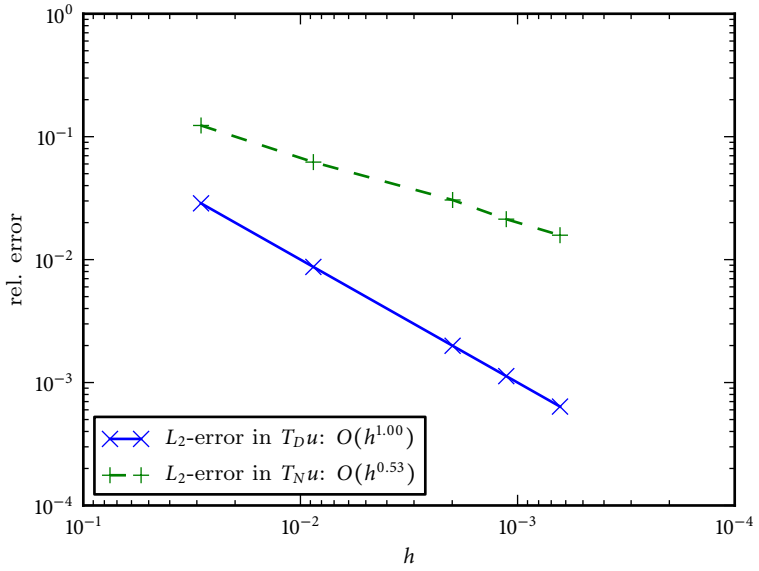


Figure 4.7: L_2-errors for the unit sphere (2 subdomains), diffusion constants $\alpha_0 = 1.5$, $\alpha_1 = \alpha_2 = 1$.

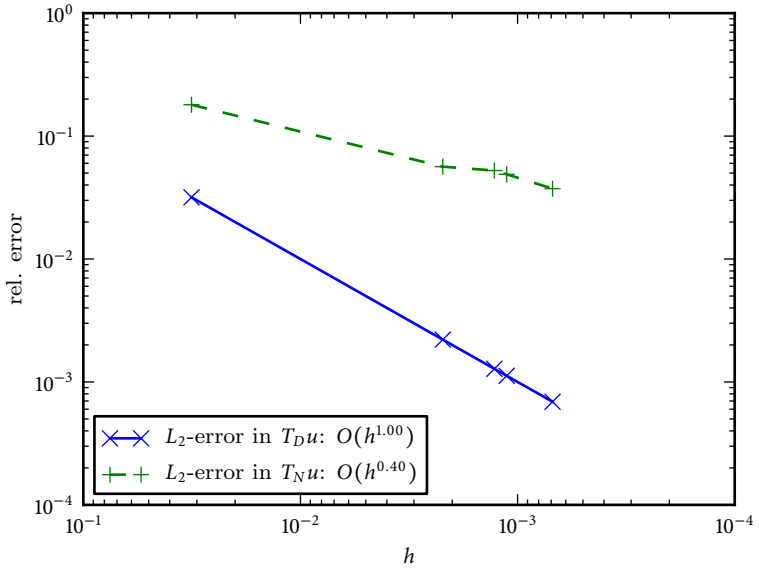


Figure 4.8: L_2-errors fot the unit sphere (2 subdomains), diffusion constants $\alpha_0 = 10$, $\alpha_1 = \alpha_2 = 1$.

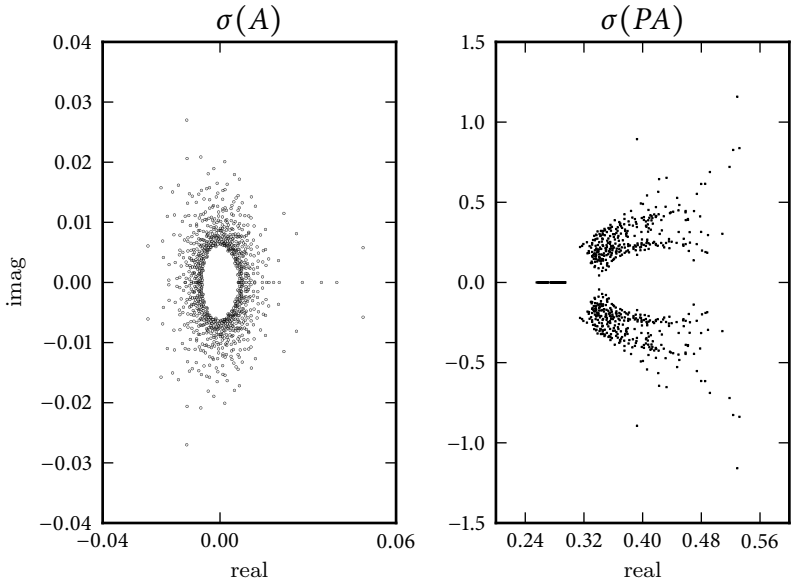


Figure 4.9: Eigenvalue distributions for the unit sphere (2 subdomains), diffusion constants $\alpha_0 = 1.5$, $\alpha_1 = \alpha_2 = 1$.

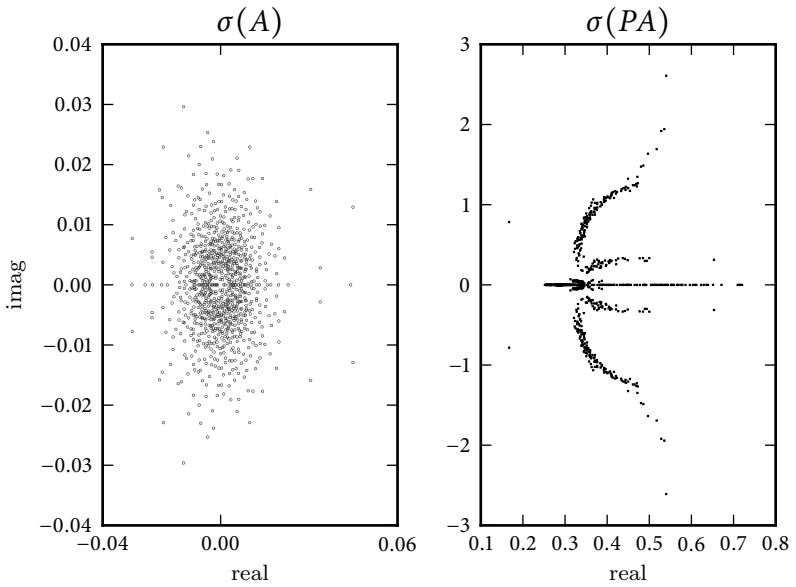
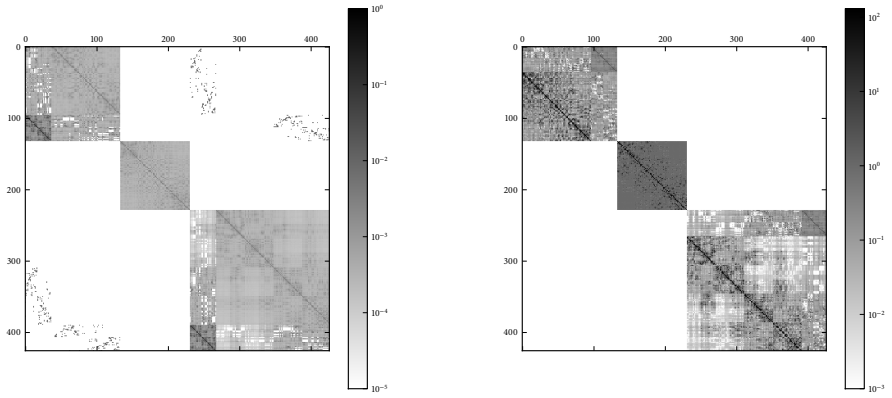


Figure 4.10: Eigenvalue distributions for the unit sphere (2 subdomains), diffusion constants $\alpha_0 = 10$, $\alpha_1 = \alpha_2 = 1$.

4.2 Caldéron preconditioning

In order to assess the performance of the Caldéron preconditioner, the relative residual of the GMRES solver versus the iteration number is plotted for various problem parameters. For the unit cube, composed of 8 subdomains, almost optimal convergence rates are obtained, i.e. $\mathcal{O}(h^{-0.1})$, where N is the number of elements in the mesh, iterations are required to obtain the desired accuracy, cf. figure 4.12 on the facing page. On the other hand, the iteration numbers required by unpreconditioned system matrix increase rapidly, from a mesh with 576 elements to one with 2304 elements the iteration numbers grow from 1745 to 6896, which is $\mathcal{O}(h^{-2})$. Similar results were obtained for the same system with strong diffusion, $\alpha_i = 1$ except $\alpha_1 = \alpha_2 = 10$, the preconditioned system matrix required $\mathcal{O}(h^{-0.13})$ iterations until convergence.

With Dirichlet boundary conditions on one subdomain, the number of iterations required on the box (2 subdomains) was $\mathcal{O}(h^{-0.2})$, while on the unit cube (8 subdomains) $\mathcal{O}(h^{-0.3})$ was measured. See figures 4.13 and 4.14 on page 32.



(a) System matrix A (b) Caldéron preconditioner P

Figure 4.11: Sparsity patterns for the box, 2 subdomains, Dirichlet boundary conditions on $\partial\Omega_2$.

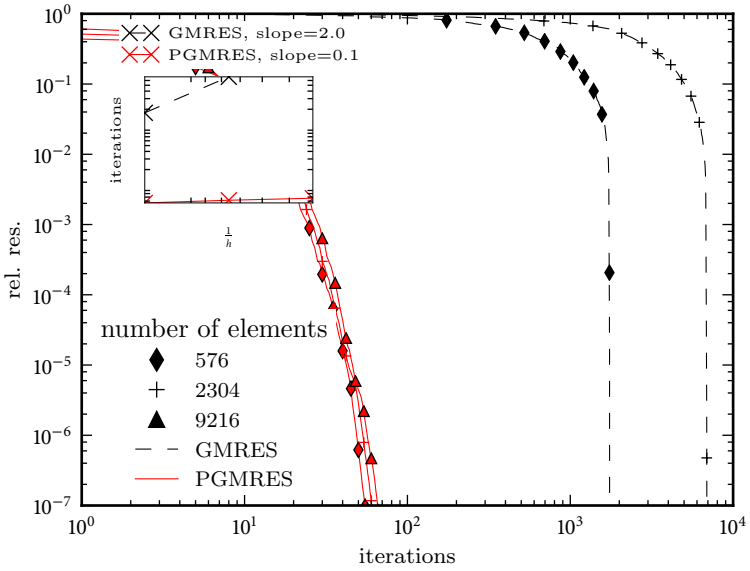


Figure 4.12: GMRES convergence, unit cube (8 subdomains).

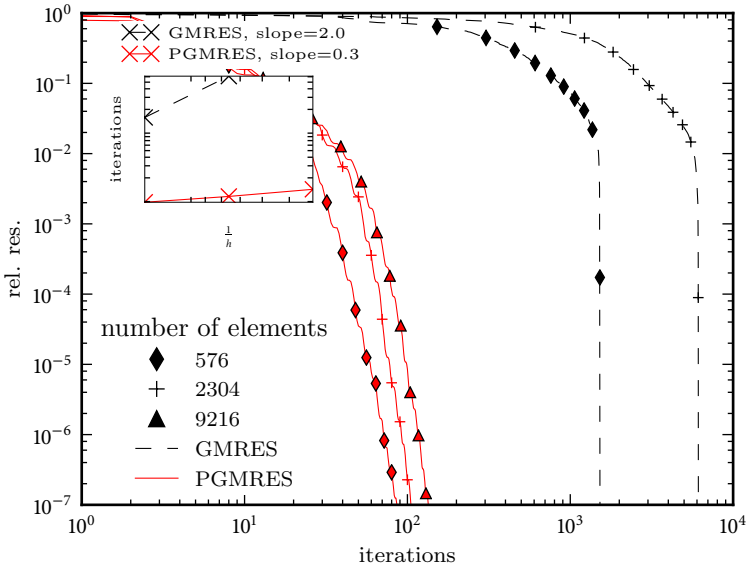


Figure 4.13: GMRES convergence, unit cube (8 subdomains), Dirichlet boundary conditions on $\partial\Omega_1$.

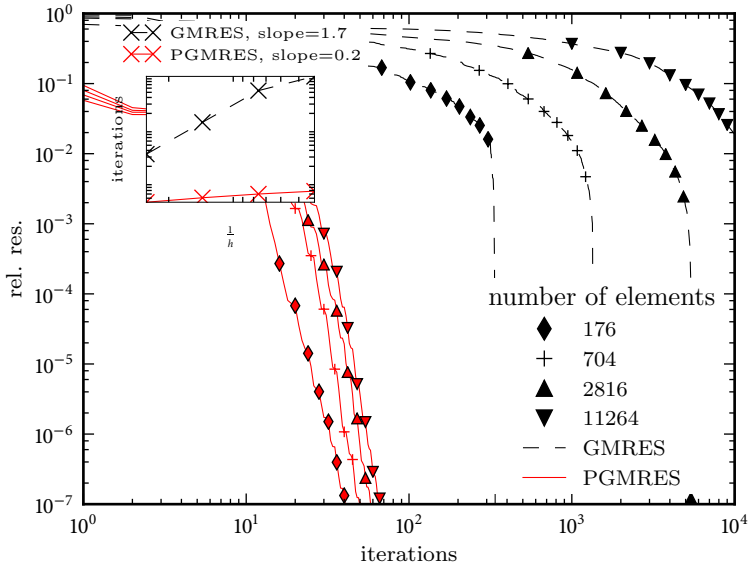


Figure 4.14: GMRES convergence, box (2 subdomains), Dirichlet boundary conditions on $\partial\Omega_1$.

4.2.1 Eigenvalue distributions

Eigenvalue distributions are presented in figures 4.15 to 4.18 on pages 33–35, one observes that after Caldéron preconditioning the eigenvalues spread strongly, while the eigenvalues of the unpreconditioned system matrix A are clustered around zero.

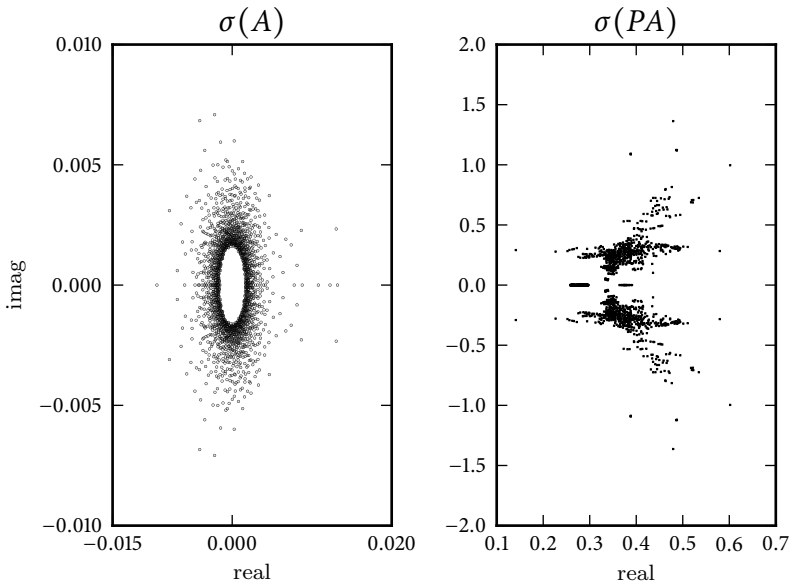


Figure 4.15: Unit cube (8 subdomains), 576 elements, $\alpha_i = 1$

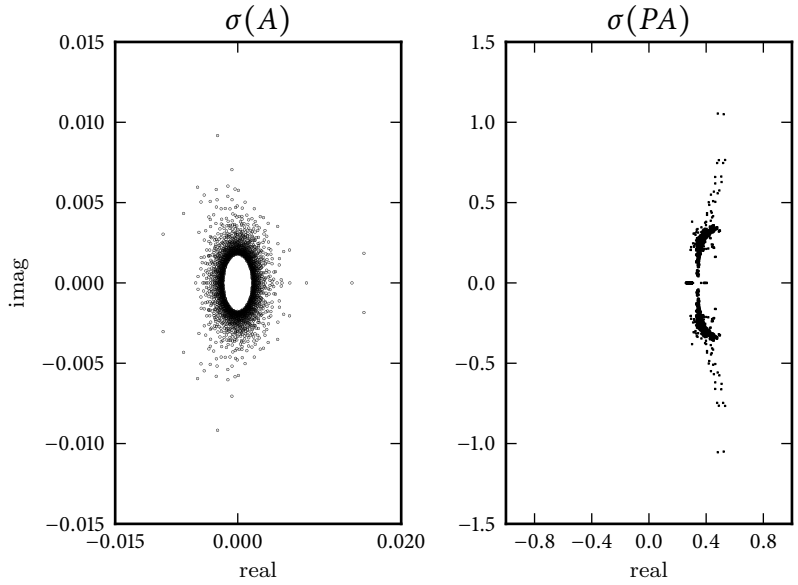


Figure 4.16: Box, 2 subdomains, 704 elements, $\alpha_i = 1$

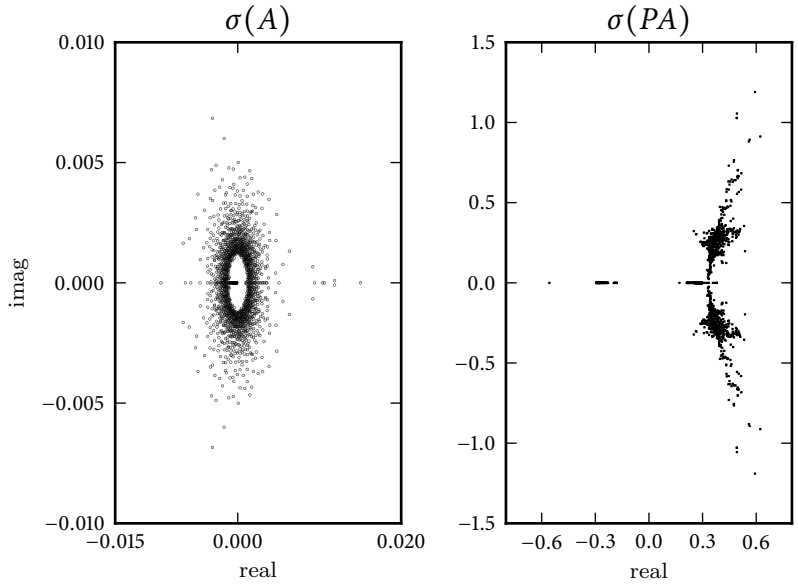


Figure 4.17: Unit cube (8 subdomains), 576 elements, $\alpha_i = 1$, Dirichlet boundary conditions on Ω_1.

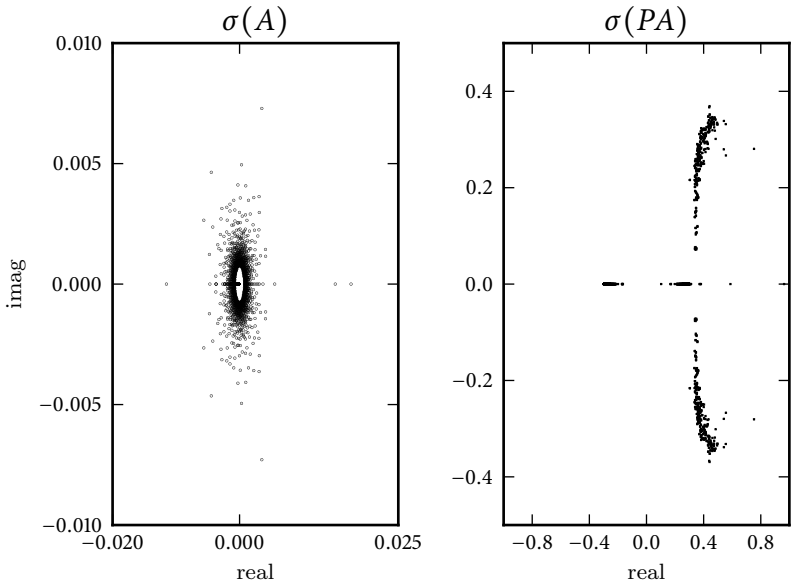


Figure 4.18: Box (2 subdomains), 704 elements, $\alpha_i = 1$, Dirichlet boundary conditions on Ω_1.

Chapter 5

Conclusion

Convergence rates of order $\mathcal{O}(h^2)$ for the Dirichlet data and $\mathcal{O}(h^{1.5})$ or $\mathcal{O}(h)$ for the Neumann data, depending on the geometry, were observed for systems with and without Dirichlet boundary conditions. Thus it is assumed that the code is correct. The Caldéron preconditioner yields h-independent convergence behaviour for systems without Dirichlet boundary conditions, as described in the literature. Eigenvalue distributions of the preconditioned system matrices show a substantial improvement compared to the unpreconditioned system matrices, which is important for an efficient solution by iterative solvers.

Appendix A

List of notations

Ω_i	material subdomains $\subset \mathbb{R}^d$
N	number of subdomains
Ω_*	union of closure of bounded subdomains
Ω_0	unbounded complement of Ω_*
n, n_i	exterior unit normal vector fields
Γ_{ij}	interface $\partial\Gamma_i \cap \partial\Gamma_j$, often assumed to have non-vanishing $d-1$-dimensional measure
$\mathbf{u}_{inc}$	in a scattering problem "incoming wave", in general: $\mathbf{u}_{inc} \in \{u(\mathbf{x}) : \mathrm{L}\, u(\mathbf{x}) = 0\}$
L	2nd-order partial differential operator
$[\cdot,\cdot], [\![\cdot,\cdot]\!]$	bi-linear pairings of L^2-type
T_N	Neumann trace operator
T_D	Dirichlet trace operator
$\mathcal{T}(\partial\Omega)$	Cauchy trace space, product of Dirichlet and Neumann trace space
$\mathfrak{u}, \mathfrak{v}$	elements of Cauchy trace spaces
$\mathbb{A}, \mathbb{B}$	linear operators on Cauchy trace spaces
$\mathbb{X}$	exterior-to-interior trace transfer operator
SL, DL	single and double layer potentials
$\mathbb{P}$	Caldéron projector
$\mathbb{L}_i$	localization operators, $\mathbb{L}_i : \mathcal{MT}(\Sigma) \to \mathcal{T}(\partial\Omega_i)$
$\Sigma_h, \Gamma_{j,h}$	boundary meshes
S_h^0	the finite element space of piecewise constant discontinuous basis functions
S_h^1	the finite element space of piecewise linear continuous basis functions

Appendix B

Input file

The input file must have the filename "settings.conf" and it must reside in the directory where the binary is executed. The template can be found in:
`local_multi_trace/settings/settings.conf`

Listing B.1: Sample input file

```
# geometry / parameters
NO_OF_DOMAINS 2
# UINC TYPE: 0 => plane wave, 1 => spherical harmonic
UINC_TYPE 0
# KAPPA: diffusion constants in Laplace setting, size= NO_OF_DOMAINS+1
KAPPA          1.0          1.0          1.0
# BETA: only used in LAPLACE, active iff UINC_TYPE=0
BETA           0.0
# WAVE_DIR:  wave direction, active iff UINC_TYPE=0
WAVE_DIR       0.0          1.0          0.0
# solver
GMRES_MAX_ITER 10000
GMRES_TOL      1e-7
# preconditioner
PRECOND        1
# indices of subdomains where dirichlet bc (g = GTYPE) is present
DIRICHLET_BC 1
USE_MAT_EXPORTER 0
# GTYPE: 0 => g ==0, 1 => g == 1, 2 => g == 1/r (boundary condition)
GTYPE 0
```

Note: Some entries have different meanings depending on whether the code was compiled in Helmholtz or Laplace mode.

PRECOND Caldéron preconditioner on/off.

USE_MAT_EXPORTER Write ".mat" file on/off. Write the system matrix and the preconditioner to disk. Use with caution, can lead to excessive memory requirements on fine grids. This feature relies on the availability of the MATIO library [1].

GMRES_MAX_ITER Abort GMRES solver after N iterations.

GMRES_TOL Terminate GMRES if desired tolerance achieved.

DIRICHLET_BC A list of subdomain indices where Dirichlet boundary conditions are present. `DIRICHLET_BC` = $1, \ldots, N$. Leave empty in case of a pure transmission problem

GTYPE
`GTYPE=0`, Zero Dirichlet boundary conditions $g \equiv 0$.

`GTYPE=1`, Constant Dirichlet boundary conditions $g \equiv 1$.

`GTYPE=2`, $g = \frac{1}{\|x\|}$.

Note: only has an effect if `DIRICHLET_BC` is non-empty.

Laplace

UINC_TYPE=0

$$u_{inc} = k \cdot x + \beta$$

where $k =$ `WAVE_DIR` and $\beta =$`BETA`

UINC_TYPE=1

$$u_{inc} = u(r, \theta) = r^2(3\cos^2\theta - 1)$$

`WAVE_DIR` and `BETA` are ignored.

KAPPA
Diffusion constants α_i. The following ordering of subdomains is assumed

$$\Omega_1, \Omega_2, \ldots, \Omega_N, \Omega_0.$$

Helmholtz

UINC_TYPE=0

$$u_{inc} = e^{-\imath\kappa k \cdot x}$$

where κ is the last entry in the line `KAPPA` and $k =$ `WAVE_DIR`, the direction of the incoming plane wave.

KAPPA The wavenumbers, Ω_0 comes last.

Appendix C

ACA settings

Listing C.1: aca.conf

```
# accuracy of the ACA approximation of the matrix
EPS_ACA 1.e-4
# accuracy of the ILU preconditioner
EPS_ILU 1.e-3
# solver accuracy
EPS_SOL 1.e-9
# preconditioning (0-no, 1-yes)
PRECOND 1
# solver (0-GMRES, 1-FGMRES, 2-BICGSTAB, 3-CG, 4-MINRES)
SOLVER 0
# recompression (0-no, 1-yes)
RECOMPRESSION 0
# eta
ETA .5
# maximum number of iterations
MAX_ITER 2500
# block min-size
BMIN 30
# maximum rank
MAX_RANK 200
# using omp
USE_OMP 0
```

Appendix D

lmt_config.hpp

This file contains some global constants. The fundamental solution in line 17, can either be set to `LAPLACE` or `HELMHOLTZ`. The code is written such that the change of this line is sufficient to switch to Helmholtz scattering problems. The quadrature rule is set in line 24. Adaptive cross approximation is used if the AHMED Library is activated in the CMake configuration.

```
  #ifndef __LMT_CONFIG_H__
#define __LMT_CONFIG_H__

// betl includes ------------------------------------------------------------
#include <enumerators/enumerators.hpp>
#include <element/multi_element.hpp>
#include <integration/galerkinintegrator.hpp>
#include <integration/galerkinquadrature.hpp>
#include <traits/complextraits.hpp>


namespace betl {
  namespace lmt {
    class config {
    public:
      typedef MultiElement<3>        element_t;
      static const enum FUNDSOL      FS=LAPLACE;
#ifdef __AHMED_ENABLED
      static const enum ACCELERATION ACC=ACA;
#else
      static const enum ACCELERATION ACC=NO_ACCELERATION;
#endif
      static const enum PARALLEL     PAR=OMP;
      typedef GalerkinQuadrature< element_t, 7, 36, 25, 16> qr_t;
      /** @name numeric types */
      //@{
      template< enum FUNDSOL FS, typename DUMMY>
      struct numeric_traits{};
      template <typename DUMMY>
      struct numeric_traits<LAPLACE,DUMMY>
      {
        typedef double numeric_t;
      };
      template< typename DUMMY>
      struct numeric_traits<HELMHOLTZ,DUMMY>
```

```
          {
            typedef typename ComplexTraits<double>::complex_type
38          numeric_t;
          };
          typedef typename numeric_traits<FS,void>::numeric_t numeric_t;
          //@}
        };
43    }
  }
  #endif /* __LMT_CONFIG_H__ */
```

Appendix E

Code

E.1 System matrix

E.1.1 Neumann restriction operator

Listing E.1: *R*

```
#ifndef __NEUMANN_RESTRICTION_OPERATOR_H__
#define __NEUMANN_RESTRICTION_OPERATOR_H__

// betl includes ---------------------------------------------------------
#include <sparse/sparse_matrix.hpp>
#include <sparse_operators/scalar_matrix_policy.hpp>
#include <sparse_operators/interlace.hpp>

// -----------------------------------------------------------------------
namespace betl {
  namespace lmt {
    template< typename DOFH_T,
              typename ALLOCATOR_POLICY = scalar_matrix_policy< double >
              >
    class neumann_restriction_operator : public ALLOCATOR_POLICY {
    private:
      typedef typename ALLOCATOR_POLICY::base_type base_t;

    public:
      neumann_restriction_operator( const DOFH_T& dhni,
                                    const DOFH_T& dhnj );

      void compute();

      template< typename INTERLACE_T>
      void compute( INTERLACE_T& interlace );

    private:
      template<typename INTERLACE_T>
      void compute_(INTERLACE_T& interlace);

      const DOFH_T& dhni_;
      const DOFH_T& dhnj_;
    };

```

```
template< typename DOFH_T,
          typename ALLOCATOR_POLICY>
neumann_restriction_operator<DOFH_T, ALLOCATOR_POLICY>::
neumann_restriction_operator( const DOFH_T& dhni,
                              const DOFH_T& dhnj ) :
  ALLOCATOR_POLICY( dhni.GiveNoOfDofs(), dhni.GiveNoOfDofs()),
  dhni_(dhni), dhnj_(dhnj)
{
  /* empty */
}

template< typename DOFH_T,
          typename ALLOCATOR_POLICY>
void neumann_restriction_operator<DOFH_T, ALLOCATOR_POLICY>::
compute() {
  helper::DefaultInterlace< DOFH_T > di;
  this -> compute_(di);
}


template< typename DOFH_T,
          typename ALLOCATOR_POLICY>
template< typename INTERLACE_T>
void neumann_restriction_operator<DOFH_T, ALLOCATOR_POLICY>::
compute(INTERLACE_T& interlace)
{
  compute_(interlace);
}

template< typename DOFH_T,
          typename ALLOCATOR_POLICY>
template< typename INTERLACE_T>
void neumann_restriction_operator<DOFH_T, ALLOCATOR_POLICY>::
compute_( INTERLACE_T& interlace)
{
  typedef typename DOFH_T::const_elem_iterator
    const_elem_iterator_t;

  typedef typename DOFH_T::const_elem_dof_iterator
    const_elem_dof_iterator_t;

  for( const_elem_iterator_t iter_i = dhni_.begin( );
       iter_i != dhni_.end(); ++iter_i ) {
    const_elem_iterator_t iter_j = interlace(iter_i, dhnj_ );

    const bool doesExist = dhnj_.assertElemIterator( iter_j );

    if(doesExist) {
      const_elem_dof_iterator_t d_begin =
        interlace.begin(iter_i, dhni_ );
      const_elem_dof_iterator_t d_end =
        interlace.end(iter_i, dhni_ );
      const_elem_dof_iterator_t d_iter;
      for( d_iter = d_begin; d_iter != d_end; ++d_iter ) {
```

```
            const std::size_t global_idx = (*d_iter) -> GivePermutatedDofId();

            const base_t val = 1.0;
            this->assemble_(val, global_idx,global_idx);
          }
        }
      }
      this->finalize_( );
    }
  } // end namespace lmt
} // end namespace betl

#endif /* __NEUMANN_RESTRICTION_OPERATOR_H__ */
```

E.1.2 Transmission matrix

Listing E.2: Transmission matrix

```
#ifndef __TRANSMISSION_MATRIX_H__
#define __TRANSMISSION_MATRIX_H__
// system includes ------------------------------------------------------
#include <vector>
#include <cassert>
#include <iostream>
#include <string>
// betl includes --------------------------------------------------------
#include <linalg/linalg.hpp>

namespace betl {
  namespace lmt {
    /** @brief off-diagonal blocks of local multi trace formulation
              | D1          |
     -0.5  *  |             |
              |        -D2  |
     */
    class TransmissionMatrix :
      public linalg::MatrixExpression<TransmissionMatrix> {
    private:
      typedef scalar_matrix_policy<double> smp;
    public:
      typedef typename smp::matrix_type sparse_matrix_t;
      typedef double numeric_type;
    public:
      TransmissionMatrix() : d2_alpha_(-1.0) {}
      TransmissionMatrix(double d2_alpha) : d2_alpha_(d2_alpha) {}

      /** @name Methods needed by MatrixExpression */
      template< typename NUMERIC_T>
      void amux( NUMERIC_T alpha, NUMERIC_T* x, NUMERIC_T* y) const;
      template< typename NUMERIC_T>
      void tamux(NUMERIC_T alpha, NUMERIC_T* x, NUMERIC_T* y) const;
      std::size_t GiveRows() const;
      std::size_t GiveCols() const;
```

```
    sparse_matrix_t& giveD1() { return D1_; }
    sparse_matrix_t& giveD2() { return D2_; }

    const sparse_matrix_t& giveD1() const { return D1_; }
    const sparse_matrix_t& giveD2() const { return D2_; }

    template< typename EXPORTER>
    void Write(EXPORTER& exporter, std::string idstr);
  private:
    double d2_alpha_;
    sparse_matrix_t D1_;
    sparse_matrix_t D2_;
  };

  // ------------------------------------------------------------------
  std::size_t TransmissionMatrix::GiveRows() const
  {
    return D1_.GiveRows() + D2_.GiveRows();
  }

  // ------------------------------------------------------------------
  std::size_t TransmissionMatrix::GiveCols() const
  {
    return D1_.GiveCols() + D2_.GiveCols();
  }

  // ------------------------------------------------------------------
  template<typename NUMERIC_T>
  void TransmissionMatrix::
  amux( NUMERIC_T alpha, NUMERIC_T* x, NUMERIC_T* y ) const
  {
    const NUMERIC_T d2_alpha(d2_alpha_); // neumann transmission conditions
    const NUMERIC_T factor(-0.5); // -0.5 X_{ij}
    D1_.amux(factor*alpha,x,y);
    std::size_t offsetX = D1_.GiveCols();
    std::size_t offsetY = D1_.GiveRows();
    D2_.amux(factor*d2_alpha*alpha, &x[offsetX], &y[offsetY]);
  }

  // ------------------------------------------------------------------
  template<typename NUMERIC_T>
  void TransmissionMatrix::
  tamux( NUMERIC_T alpha, NUMERIC_T* x, NUMERIC_T* y ) const
  {
    const NUMERIC_T d2_alpha(d2_alpha_); // neumann transmission conditions
    const NUMERIC_T factor(-0.5);
    D1_.tamux(factor*alpha, x, y);
    std::size_t offsetX = D1_.GiveRows();
    std::size_t offsetY = D1_.GiveCols();
    D2_.tamux(factor*d2_alpha*alpha, &x[offsetX], &y[offsetY]);
  }

  // ------------------------------------------------------------------
  template< typename EXPORTER>
```

```
    void TransmissionMatrix::
    Write(EXPORTER& exporter, std::string idstr)
93  {
      exporter("XD"+idstr, D1_);
      exporter("XN"+idstr, D1_);
    }

98

  } // end namespace lmt
} // end namespace betl


103 #endif /* __TRANSMISSION_MATRIX_H__ */
```

E.2 Caldéron preconditioner

Listing E.3: Caldéron preconditioner header file handler

```
#ifndef __PRECONDITIONER_HANDLER_H__
2   #define __PRECONDITIONER_HANDLER_H__
    // system includes ---------------------------------------------------------
    #include <boost/shared_ptr.hpp>
    #include <set>
    // betl includes -----------------------------------------------------------
7   #include <enumerators/enumerators.hpp>
    #include <integration/galerkinintegrator.hpp>
    #include <geometry/multimesh.hpp>

    // own includes ------------------------------------------------------------
12  #include "../bem_operator_handler.hpp"
    #include "../sparse_operator_handler.hpp"
    #include "../calderon_handler.hpp"
    #include "dual_mesh.hpp"
    #include "dual_mesh_generator.hpp"
17  #include "multi_bary_element.hpp"
    #include "preconditioner_sparse_operator_handler.hpp"
    #include "calderon_preconditioner.hpp"
    #include "hyper_matrix_handler.hpp"
    #include "../bc/bc_subdomain_handler.hpp"
22  #include "../bc/bc_subdomain_handler_generator.hpp"
    #include "../lmt_config.hpp"
    #include "../bc/exclusion_operator_handler.hpp"

    namespace betl {
27    namespace lmt {
        class PreconditionerHandler {
        private:
          static const enum ACCELERATION ACC=config::ACC;
          static const enum PARALLEL     PAR=config::PAR;
32        static const enum FUNDSOL      FS =config::FS;
          typedef typename config::qr_t QUADRATURE_T;
          typedef MultiBaryElement<3> multi_bary_element_t;
          typedef SubdomainHandler< ACC,
```

```
                                multi_bary_element_t >
37      sd_h_t;
        typedef MultiElement<3> multi_element_t;
        typedef SubdomainHandler< ACC,
                                multi_element_t>
        PARENT_SD_H_T;
42      typedef ExclusionOperatorHandler<PARENT_SD_H_T>
        exclusion_op_h_t;
        typedef BemOperatorHandler< QUADRATURE_T,
                                    GalerkinIntegrator,
                                    sd_h_t,
47                                  PAR,FS> bem_op_h_t;
        typedef SparseOperatorHandler<sd_h_t> sparse_op_h_t;
        typedef DualMeshGenerator::dual_mesh_vec_t mesh_vec_t;
        typedef CalderonHandler< bem_op_h_t,
                                sparse_op_h_t>
52      calderon_h_t;
        typedef MultiMesh<multi_element_t> PARENT_MESH_T;
        typedef HyperMatrixHandler<QUADRATURE_T,
                                    GalerkinIntegrator,
                                    sd_h_t,
57                                  FS,PAR>
        hyper_matrix_h_t;
        typedef dual::SparseOperatorHandler< PARENT_SD_H_T,
                                             sd_h_t>
        precond_sparse_op_h_t;
62      typedef CalderonPreconditioner< calderon_h_t,
                                        hyper_matrix_h_t>
        calderon_preconditioner_t;
        typedef calderon_preconditioner_t matrix_type;
        typedef calderon_preconditioner_t& reference;
67      typedef boost::shared_ptr<calderon_preconditioner_t> matrix_ptr_t;
        typedef std::set<std::size_t> index_set_t;
        typedef std::vector<double>   d_vec_t;
      public:
        template< typename SETTINGS_PARSER_T>
72      PreconditionerHandler( const std::string        basename,
                               const std::size_t        NoOfDomains,
                               const SETTINGS_PARSER_T& settings_parser,
                               const d_vec_t&           coeff_vec );
        void setup( const PARENT_MESH_T& parent_mesh,
77                  const PARENT_SD_H_T& parent_sd_handler,
                    const exclusion_op_h_t& exclusion_op_h,
                    const index_set_t&   dirichlet_bc_id);
        const matrix_type& giveMatrix() const { return *precond_ptr_; }
      private:
82      mesh_vec_t       mesh_vec_;
        /// sd handler incorporating dirichlet bc
        sd_h_t        sd_h_;
        /// subdomain handler on entire mesh
        sd_h_t        entire_sd_h_;
87      sparse_op_h_t    sparse_op_h_;
        bem_op_h_t       bem_op_h_;
        calderon_h_t     calderon_h_;
```

```
      hyper_matrix_h_t hyper_matrix_h_;
      precond_sparse_op_h_t precond_sparse_op_h_;
92    matrix_ptr_t          precond_ptr_;
    };
    // ------------------------------------------------------------------
    template< typename SETTINGS_PARSER_T>
    PreconditionerHandler::
97  PreconditionerHandler( const std::string        basename,
                           const std::size_t        NoOfDomains,
                           const SETTINGS_PARSER_T& settings_parser,
                           const d_vec_t&           coeff_vec ) :
      sd_h_(basename, NoOfDomains),
102   entire_sd_h_(basename, NoOfDomains),
      sparse_op_h_(NoOfDomains),
      bem_op_h_( NoOfDomains, settings_parser, coeff_vec ),
      calderon_h_( NoOfDomains, coeff_vec ),
      hyper_matrix_h_(NoOfDomains, settings_parser, coeff_vec),
107   precond_sparse_op_h_(NoOfDomains+1)
    {
      /* empty */
    }
  } // end namespace lmt
112 }  // end namespace betl

#endif /* __PRECONDITIONER_HANDLER_H__ */
```

Listing E.4: Caldéron preconditioner cpp file handler

```
1   #include "preconditioner_handler.hpp"
    #include "generator_subdomain_handler.hpp"
    #include "generator_calderon_preconditioner.hpp"


6   namespace betl {
      namespace lmt {
        // ------------------------------------------------------------------
      void PreconditionerHandler::
      setup( const PARENT_MESH_T& parent_mesh,
11           const PARENT_SD_H_T& parent_sd_handler,
             const exclusion_op_h_t& exclusion_op_h,
             const index_set_t&   dirichlet_bc_id)
      {
        // setting up barycentric refined mesh
16      DualMeshGenerator::Create(parent_mesh, mesh_vec_);
        // Subdomain Handler
        entire_sd_h_.setup(mesh_vec_);
        // setting up subdomain handler (Dirichlet BC)
        // SubdomainHandlerGenerator::Create( /* const data */
21      //                                    parent_sd_handler,
        //                                    entire_sd_h_,
        //                                    dirichlet_bc_id,
        //                                    mesh_vec_,
        //                                    /* non-const data */
26      //                                    sd_h_);
```

```
      // setting up sparse operator handler
      sparse_op_h_.setup(entire_sd_h_);
      // setting up bem operator handler
      bem_op_h_.setup(entire_sd_h_, dirichlet_bc_id);
      // setting up calderon operator handler
      calderon_h_.setup(bem_op_h_, sparse_op_h_, dirichlet_bc_id);
      // setting up hyper matrix handler (preconditioning matrix for V on
      // dual mesh)
      hyper_matrix_h_.setup( entire_sd_h_, sparse_op_h_, dirichlet_bc_id );
      // setting up coupling operators
      precond_sparse_op_h_.compute( parent_sd_handler,
                                    entire_sd_h_ );
      const std::size_t blocksize = sd_h_.give_blocksize();
      matrix_ptr_t ptr(new calderon_preconditioner_t( blocksize,
                                                      calderon_h_,
                                                      hyper_matrix_h_,
                                                      dirichlet_bc_id));

      CalderonPreconditionerGenerator::Create( entire_sd_h_,
                                               parent_sd_handler,
                                               precond_sparse_op_h_,
                                               exclusion_op_h,
                                               dirichlet_bc_id,
                                               *ptr );
      precond_ptr_ = ptr;
    }
  } // end namespace lmt
} // end namespace betl
```

Listing E.5: Caldéron preconditioner

```
#ifndef __CALDERON_PRECONDITIONER_H__
#define __CALDERON_PRECONDITIONER_H__

// system includes ---------------------------------------------------------
#include <vector>
#include <set>
#include <boost/shared_ptr.hpp>
#include <iostream>

// betl includes -----------------------------------------------------------
#include <sparse/sparse_matrix.hpp>
#include <sparse/sparse_transpose.hpp>
#include <sparse/sparse_utilities.hpp>
#include <linalg/linalg.hpp>
#include <linalg/matrix_expression.hpp>
#include <linalg/block_diagonal_AB_matrix.hpp>
#include <sparse_operators/scalar_matrix_policy.hpp>

// own includes ------------------------------------------------------------
#include "helper/my_umfpack_wrapper.hpp"
#include "helper/sort_sparse_matrix.hpp"
#include "lmt_config.hpp"
```

```
// debug
#include <boost/lexical_cast.hpp>
#include <mat_exporter/mat_exporter.hpp>

namespace betl {
  namespace lmt {
    /** @brief ...
        @tparam SPARSE_OP_H_T  lmt::dual::SparseOperatorHandler
        in preconditioner_sparse_operator_handler.hpp
        @tparam CALDERON_OP_H_T  calderon operator handler on dual mesh
        @tparam HYPER_MATRIX_H_T hyper_matrix on dual mesh (preconditioning of V)
        @tparam BC_DOF_H_T       exclusion operator handler (contains sparse
        matrices to restrict dofs)
    */
    template< typename CALDERON_OP_H_T,
              typename HYPER_MATRIX_H_T>
    class CalderonPreconditioner :
      public linalg::MatrixExpression<CalderonPreconditioner<CALDERON_OP_H_T,
                                                              HYPER_MATRIX_H_T> >{
    private:
      typedef typename scalar_matrix_policy<double>::matrix_type r_sparse_matrix_t;
    public:
      typedef typename HYPER_MATRIX_H_T::numeric_t numeric_t;
      typedef numeric_t numeric_type;
    private:

      static const ::betl::sparse::MATRIXSTORAGE MS = ::betl::sparse::CRS;
      // sparse matrix storage type for cxsparse op's (M_N_Dt, M_D_Nt)
      typedef sparse::Matrix<double, sparse::GENERAL, sparse::CRS >
      sparse_matrix_t;
      /** @name direct solver storage D^-1 */
      //@{
    public:
      typedef MyUmfPackWrapper<sparse_matrix_t> direct_solver_t;
    private:
      typedef boost::shared_ptr<direct_solver_t> ptr_t;
      typedef std::pair<ptr_t,ptr_t> lu_pair_t;
      typedef std::vector<lu_pair_t> lu_vec_t;
      //@}
      /** @name coupling operator storage*/
      //@{
      typedef std::pair< sparse_matrix_t, sparse_matrix_t > sparse_pair_t;
      typedef std::vector< sparse_pair_t> coupling_vec_t;
      //@}
      typedef std::set<std::size_t> index_set_t;
      typedef typename index_set_t::const_iterator set_iter_t;

    public:
      CalderonPreconditioner( const std::size_t       blocksize,
                              const CALDERON_OP_H_T&  calderon_op_h,
                              const HYPER_MATRIX_H_T& hyper_matrix_h,
                              const index_set_t&      dirichlet_bc_id );
      void Finalize();
```

```
      void amux(  numeric_t alpha, numeric_t* x, numeric_t* y ) const;
      void tamux( numeric_t alpha, numeric_t* x, numeric_t* y ) const;

      template<class T>
      void operator()(T* x) const;

      std::size_t GiveRows() const { return size_; }
      std::size_t GiveCols() const { return size_; }

      /** @name methods needed by the initializer */
      //@{
      lu_vec_t& give_invD_vec() { return lu_vec_; }
      coupling_vec_t& give_coupling_vec() { return coupling_vec_; }
      //@}
    private:
      const CALDERON_OP_H_T&  calderon_op_h_;
      const HYPER_MATRIX_H_T& hyper_matrix_h_;
      const index_set_t&      dirichlet_bc_id_;
      lu_vec_t      lu_vec_;
      coupling_vec_t coupling_vec_;
      std::size_t   blocksize_;
      std::size_t   size_;
      bool finalized_;

      /// returns offset
      template<class T>
      std::size_t amux_dir_(T alpha, T* x, T* y, std::size_t id, char op='N') const;
      /// returns offset
      template<class T>
      std::size_t amux_tp_(T alpha, T* x, T* y, std::size_t id, char op='N')
const;
    };


    // ------------------------------------------------------------------------
    template< typename CALDERON_OP_H_T,
              typename HYPER_MATRIX_H_T>
    CalderonPreconditioner< CALDERON_OP_H_T,
                            HYPER_MATRIX_H_T>::
    CalderonPreconditioner( const std::size_t       blocksize,
                            const CALDERON_OP_H_T&  calderon_op_h,
                            const HYPER_MATRIX_H_T& hyper_matrix_h,
                            const index_set_t&      dirichlet_bc_id )
      :
      blocksize_( blocksize ),
      calderon_op_h_( calderon_op_h ),
      hyper_matrix_h_( hyper_matrix_h ),
      dirichlet_bc_id_( dirichlet_bc_id ),
      lu_vec_( blocksize ),
      coupling_vec_( blocksize ),
      finalized_(false)
    {
      /* empty */
    }
```

```
130
    // ------------------------------------------------------------------
    template< typename CALDERON_OP_H_T,
              typename HYPER_MATRIX_H_T>
    void CalderonPreconditioner< CALDERON_OP_H_T,
135                              HYPER_MATRIX_H_T>::
    Finalize( )
    {
      size_ =0;
      if (! finalized_ ) {
140     for (std::size_t i=0; i < blocksize_; ++i) {
          if ( dirichlet_bc_id_.find(i) != dirichlet_bc_id_.end() )
{
            /* subdomain with Dirichlet BC */
            size_ += lu_vec_[i].first->GiveCols();
          } else {
145         /* regular subdomain */
            size_ += lu_vec_[i].first->GiveCols() + lu_vec_[i].second->GiveCols();
          }
        }
        finalized_ = true;
150   }
    }

    // ------------------------------------------------------------------
    template< typename CALDERON_OP_H_T,
155           typename HYPER_MATRIX_H_T>
    template< class T >
    void CalderonPreconditioner<CALDERON_OP_H_T,
                                HYPER_MATRIX_H_T>::
    operator()(T* x) const
160 {
      std::vector<T> t( size_, T(0.0) );

      const T one (1.0);
      const T zero(0.0);
165
      std::copy(x, x+size_, t.begin() );
      std::fill(x, x+size_, zero);

      this->amux( one, &t[0] , x );
170 }

    // ------------------------------------------------------------------
    template< typename CALDERON_OP_H_T,
              typename HYPER_MATRIX_H_T>
175 void CalderonPreconditioner<CALDERON_OP_H_T,
                              HYPER_MATRIX_H_T>::
    amux( numeric_t alpha, numeric_t* x, numeric_t* y ) const
    {
      if (!finalized_ ) {
180     std::cerr << "CalderonPreconditioner not finalized\n exit!\n";
        exit(-1);
      }
```

```
    std::size_t offset=0;
    for ( std::size_t id = 0; id < blocksize_; ++id ) {
      set_iter_t it = dirichlet_bc_id_.find(id);
      if ( it != dirichlet_bc_id_.end() ) {
        /* dirichlet bc on this subdomain */
        offset += amux_dir_( alpha, &x[offset], &y[offset], id );
      } else {
        /* no bc on this subdomain */
        offset += amux_tp_ ( alpha, &x[offset], &y[offset], id );
      }
    }
  }

  // ------------------------------------------------------------------------
  template< typename CALDERON_OP_H_T,
            typename HYPER_MATRIX_H_T>
  void CalderonPreconditioner< CALDERON_OP_H_T,
                               HYPER_MATRIX_H_T>::
  tamux( numeric_t alpha, numeric_t* x, numeric_t* y ) const
  {
    if (!finalized_ ) {
      std::cerr << "CalderonPreconditioner not finalized\n exit!\n";
      exit(-1);
    }
    std::size_t offset=0;
    for ( std::size_t id = 0; id < blocksize_; ++id ) {
      set_iter_t it = dirichlet_bc_id_.find(id);
      if ( it != dirichlet_bc_id_.end() ) {
        /* dirichlet bc on this subdomain */
        offset += amux_dir_( alpha, &x[offset], &y[offset], id, 'T' );
      } else {
        /* no bc on this subdomain */
        offset += amux_tp_ ( alpha, &x[offset], &y[offset], id, 'T' );
      }
    }
  }


  // ------------------------------------------------------------------------
  template< typename CALDERON_OP_H_T,
            typename HYPER_MATRIX_H_T>
  template< class T >
  std::size_t CalderonPreconditioner< CALDERON_OP_H_T,
                                      HYPER_MATRIX_H_T>::
  amux_dir_(T alpha, T* x, T* y, std::size_t id, char op) const
  {
    auto& CD = coupling_vec_[id].first;
    auto& CN = coupling_vec_[id].second;
    const std::size_t sizeD = CN.GiveCols();
    const std::size_t sizeN = CD.GiveCols();
    typedef typename HYPER_MATRIX_H_T::matrix_t W_t;
    const W_t& W = hyper_matrix_h_.giveMatrix(id);
    direct_solver_t& invM = *(lu_vec_[id].first);
```

```
      if(op=='N') {
        ((invM^'T')*(CD^'T')*W*CD*invM).
          amux(alpha, x, y );
      } else {
        ((invM^'T')*(CD^'T')*W*CD*invM).
          tamux(alpha,x,y);
      }
      // return offset!
      return sizeN;
    }

    // ------------------------------------------------------------------------
    template< typename CALDERON_OP_H_T,
              typename HYPER_MATRIX_H_T>
    template< class T >
    std::size_t CalderonPreconditioner< CALDERON_OP_H_T,
                                        HYPER_MATRIX_H_T>::
    amux_tp_(T alpha, T* x, T* y, std::size_t id, char op) const
    {
      const T one = T(1.0);
      auto& CBD = coupling_vec_[id].first;
      auto& CBN = coupling_vec_[id].second;
      auto& invMD = *(lu_vec_[id].first);  // N, \tilde{D}
      auto& invMN = *(lu_vec_[id].second); // D, \tilde{N}
      const std::size_t sizeD = invMD.GiveCols();
      const std::size_t sizeN = invMN.GiveCols();
      // matrix to remove nodes "aka dual cells" adjacent to
      // \Gamma_DIR
      auto& C = calderon_op_h_.giveMatrix(id);
      //
      //  | B_N^t          |    |  K      V  |   | B_D       |
      //  |                |  x |            | x |           |
      //  |        B_D^t   |    |  W     -K' |   |      B_N  |
      //

      if( op == 'N') {
        (diag_matrix(invMN^'T',invMD^'T')*
         diag_matrix(CBN^'T', CBD^'T')*
         C*
         diag_matrix(CBD,CBN)*
         diag_matrix(invMD,invMN))
          .amux(alpha, x, y);
      } else {
        (diag_matrix(invMN^'T',invMD^'T')*
         diag_matrix(CBN^'T', CBD^'T')*
         C*
         diag_matrix(CBD,CBN)*
         diag_matrix(invMD,invMN))
          .tamux(alpha, x, y);
      }
      // return offset
      return sizeD+sizeN;
    }
  } // end namespace lmt
```

```
} // end namespace betl

#endif /* __CALDERON_PRECONDITIONER_H__ */
```

This class provides the stabilize hyper matrix $\widetilde{W}$.

Listing E.6: Stabilized Hyper Matrix

```
#ifndef __LAPLACE_STABIL_HYPER_MATRIX_H__
#define __LAPLACE_STABIL_HYPER_MATRIX_H__

#include <linalg/linalg.hpp>

namespace betl {
  namespace lmt {
    namespace laplace {
      template< typename BEM_MATRIX>
      class HyperMatrix :
        public linalg::MatrixExpression<HyperMatrix<BEM_MATRIX> >{
      public:
        typedef double numeric_type;
      private:
        typedef linalg::Vector<double> vector_t;
        typedef vector_matrix_policy<double> vmp;
        typedef typename vmp::array_t        sparse_matrix_array_t;
      public:
        HyperMatrix( const BEM_MATRIX& V,
                     const sparse_matrix_array_t& C,
                     const vector_t& stabil,
                     const double factor) :
          V_(V),
          C_(C),
          stabil_(stabil),
          factor_(factor)
        {
          /* empty */
        }

        template< class NUMERIC>
        void amux(NUMERIC alpha, NUMERIC* x, NUMERIC* y) const
        {
          typedef linalg::VectorWrapper<NUMERIC> vector_wrapper_t;
          vector_wrapper_t xw(this->GiveCols(), x);
          vector_wrapper_t yw(this->GiveRows(), y);

          for ( int i=0; i<3; ++i)
            yw += alpha*factor_*factor_*C_[i] * V_ * (C_[i]^'T')*xw;

          // int n = xw.Size();
          // double dot=0;
          // for(int i=0;i<n;++i)
          //   dot+=xw[i]*stabil_[i];
          //      for(int i=0;i<n;++i)
          //yw[i] += alpha*dot*stabil_[i];
```

```
47            double dot = xw*stabil_;
              yw.add(alpha*dot, stabil_);
            }

            template <class NUMERIC>
52          void tamux(NUMERIC alpha, NUMERIC* x, NUMERIC* y) const
            {
              this->amux(alpha,x,y);
            }

57          std::size_t GiveRows() const
            {
              return C_[0].GiveRows();
            }

62          std::size_t GiveCols() const
            {
              return this -> GiveRows();
            }

67        private:
            const BEM_MATRIX&   V_;
            const sparse_matrix_array_t& C_;
            const vector_t      stabil_;
            const double        factor_;
72        };
      } // end namespace laplace
    } // end namespace lmt
  } // end namespace betl

77 #endif /* __LAPLACE_STABIL_HYPER_MATRIX_H__ */
```

E.2.1 Dual mesh

Listing E.7: Multi Bary Element

```
#ifndef BETL_LOCAL_MULTI_TRACE__MULTI_BARY_ELEMENT_H__
#define BETL_LOCAL_MULTI_TRACE__MULTI_BARY_ELEMENT_H__
3


#include <element/multi_element.hpp>
#include <iostream>

8


namespace betl {
  /**
   * @brief MultiBaryElement needed by preconditioner, this class
   * is just the composition of MultiElement and MultiBaryElement
13 * the code is repeated in order to avoid virtual inheritance
   */
  template< std::size_t N >
  class MultiBaryElement : public Element< N >
  {
18  template< std::size_t M >
```

```
    friend
    std::ostream& operator<<( std::ostream& out, const MultiBaryElement<M>& e );
  private:
    typedef short value_t;

  public:
    typedef Element< N > base_t;
    typedef typename base_t::index_t index_t;
    typedef MultiElement< N > parent_t;

    typedef const parent_t* ptr_type;

    using base_t::vertices_;

  public:
    /** Construct with a given index
     * \param[in] idx the element's index
     * \parem[in] parent pointer to parent element
     * \param[in] pos_in_parent position in parent element [0..5]
     */
    MultiBaryElement( const index_t idx,
                      ptr_type parent,
                      const int pos_in_parent );
    /** Construct with a given index and orientation
     * \param[in] idx the element's index
     * \param[in] orientation the element's orientation
     * \parem[in] parent pointer to parent element
     * \param[in] pos_in_parent position in parent element [0..5]
     */
    //! Destructor
    ~MultiBaryElement( );

    //! return the orientation
    value_t giveOrientation( ) const;


    ptr_type giveParent( ) const;
    int givePositionInParent( ) const;
    const Vertex* giveDualCenter( ) const;

  private:
    ptr_type parent_;
    const int pos_in_parent_;


    std::ostream& write_( std::ostream& out ) const;
  };

  //------------------------------------------------------------------------
  template< std::size_t N >
  MultiBaryElement< N >::MultiBaryElement( const index_t idx,
                                           ptr_type parent,
                                           const int pos_in_parent )
    : base_t( idx ),  parent_(parent),
```

```
73      pos_in_parent_(pos_in_parent)
    {
      /* empty */
    }

78
    //-----------------------------------------------------------------------
    template< std::size_t N >
    MultiBaryElement< N >::~MultiBaryElement( )
    {
83    parent_ = NULL;
    }


    // =======================================================================
88  // Multi element stuff
    // =======================================================================

    //-----------------------------------------------------------------------
    template< std::size_t N >
93  typename MultiBaryElement< N >::value_t
    MultiBaryElement< N >::giveOrientation( ) const
    {
      return parent_->giveOrientation();
    }
98




    //-----------------------------------------------------------------------
103 template< std::size_t N >
    std::ostream& MultiBaryElement< N >::write_( std::ostream& out ) const
    {
      // write out the base class
      out << static_cast< const base_t& >( *this ) << std::endl
108       << "  Additional information for MultiElement" << std::endl
          << "    Orientation: " << parent_->giveOrientation();
      return out;
    }

113

    //-----------------------------------------------------------------------
    template< std::size_t M >
    std::ostream& operator<<( std::ostream& out, const MultiBaryElement<M>& e )
    {
118   return e.write_( out );
    }

    // =======================================================================
    // Bary element stuff
123 // =======================================================================

    //-----------------------------------------------------------------------
    template< std::size_t N >
```

```
  typename MultiBaryElement< N >::ptr_type MultiBaryElement< N >::
  giveParent( ) const
  {
    return parent_;
  }



  //-------------------------------------------------------------------------
  template< std::size_t N >
  int MultiBaryElement< N >::givePositionInParent( ) const
  {
    return pos_in_parent_;
  }

  //-------------------------------------------------------------------------
  template< std::size_t N >
  const Vertex* MultiBaryElement< N >::giveDualCenter( ) const
  {
    // if 'pos_in_parent_' is even it is the 1st node which represents
    // the dual cell's center. in case of an odd number it is the
    // 2nd node number which is the dual cell's center
    const std::size_t node_position = ( pos_in_parent_%2 == 0 ? 1 : 2 );
    return ( *vertices_ )[ node_position ];
  }
} // end namespace betl



#endif /* __MULTI_BARY_ELEMENT_H__ */
```

This code extracts the subdomains from the `MultiMesh` class, creates barycentric refined meshes and stores them in a `std::vector`.

Listing E.8: Multi Bary Element

```
#ifndef __DUAL_MESH_GENERATOR_H__
#define __DUAL_MESH_GENERATOR_H__
// system includes ------------------------------------------------------
#include <vector>
// betl includes --------------------------------------------------------
#include <geometry/mesh.hpp>
// own includes ---------------------------------------------------------
#include "dual_mesh.hpp"
#include "multi_bary_element.hpp"
// ----------------------------------------------------------------------
namespace betl {
  namespace lmt {
    class DualMeshGenerator {
    public:
```

```
16      typedef MultiBaryElement<3> element_t;
        typedef Mesh<element_t> mesh_t;
        typedef std::vector<mesh_t*> dual_mesh_vec_t;

    public:
21
        template<typename PARENT_MESH>
        static void Create( const PARENT_MESH&   parent_mesh,
                            dual_mesh_vec_t&     dual_mesh_vec);
    };
26  // ------------------------------------------------------------------------
    template<typename PARENT_MESH>
    void DualMeshGenerator::Create( const PARENT_MESH&  parent_mesh,
                                    dual_mesh_vec_t&    dual_mesh_vec)
    {
31
        typedef DualMesh<PARENT_MESH> dual_mesh_t;
        std::size_t NoOfSubdomains = parent_mesh.giveNoOfSubdomains();

        dual_mesh_vec.resize(NoOfSubdomains+1);
36      for (std::size_t i = 0; i < NoOfSubdomains; ++i ) {
          dual_mesh_vec[i] = new dual_mesh_t( parent_mesh.e_begin(i),
                                              parent_mesh.e_end  (i) );
        }
        dual_mesh_vec[NoOfSubdomains] = new dual_mesh_t( parent_mesh.e_hull_begin(),
41                                                       parent_mesh.e_hull_end()
);
    }
  } // end namespace lmt
} // end namespace betl

46 #endif /* __DUAL_MESH_GENERATOR_H__ */
```

E.2.2 Coupling operators

Handler class for the coupling operators required by the Caldéron preconditioner.

Listing E.9: PreconditionerSparseOperatorHandler

```
#ifndef __PRECONDITIONER_SPARSE_OPERATOR_HANDLER_H__
#define __PRECONDITIONER_SPARSE_OPERATOR_HANDLER_H__

4   // own includes ------------------------------------------------------------
#include "coupling_operator.hpp"
// betl includes ----------------------------------------------------------
#include <sparse_operators/identity_operator.hpp>
// system includes --------------------------------------------------------
9   #include <vector>
#include <utility>
#include <boost/lexical_cast.hpp>
#include <string>
#include <boost/shared_ptr.hpp>
14
namespace betl {
```

```
namespace lmt {
  namespace dual {
    /**
       @brief
       @tparam PARENT_SD_H_T parent subdomain handler
       @tparam DUAL_SD_H_T   dual subdomain handler
    */
    template< typename PARENT_SD_H_T,
              typename DUAL_SD_H_T>
    class SparseOperatorHandler {
    private:
      typedef typename PARENT_SD_H_T::dirichlet_dofhandler_t
      parent_dirichlet_dh_t;
      typedef typename PARENT_SD_H_T::neumann_dofhandler_t
      parent_neumann_dh_t;
      typedef typename DUAL_SD_H_T::dirichlet_dofhandler_t
      dual_dirichlet_dh_t;
      typedef typename DUAL_SD_H_T::neumann_dofhandler_t
      dual_neumann_dh_t;

    public:
      //* @name coupling operators (PARENT) */
      //@{
      typedef dual::coupling_operator< dual_dirichlet_dh_t,
                                       parent_dirichlet_dh_t>
      CPD_t;

      typedef dual::coupling_operator< dual_neumann_dh_t,
                                       parent_neumann_dh_t>
      CPN_t;

      typedef boost::shared_ptr<CPD_t> CPD_ptr_t;
      typedef boost::shared_ptr<CPN_t> CPN_ptr_t;
      typedef std::pair<CPD_ptr_t,CPN_ptr_t> pair_parent_t;
      typedef boost::shared_ptr< pair_parent_t> pair_parent_ptr_t;
      //@}

      //* @name coupling operators (DUAL) */
      //@{
      typedef dual::coupling_operator< dual_dirichlet_dh_t,
                                       parent_neumann_dh_t>
      CBD_t;
      typedef dual::coupling_operator< dual_neumann_dh_t,
                                       parent_dirichlet_dh_t>
      CBN_t;
      typedef boost::shared_ptr<CBD_t> CBD_ptr_t;
      typedef boost::shared_ptr<CBN_t> CBN_ptr_t;
      typedef std::pair<CBD_ptr_t, CBN_ptr_t> pair_bary_t;
      typedef boost::shared_ptr<pair_bary_t> pair_bary_ptr_t;
      //@}

      typedef IdentityOperator< dual_neumann_dh_t,
                                dual_dirichlet_dh_t >
      id_op_t;
```

```
    typedef boost::shared_ptr< id_op_t> id_op_ptr_t;

    typedef std::vector<pair_parent_ptr_t>
    coupling_parent_op_vec_t;
    typedef std::vector<pair_bary_ptr_t >
    coupling_bary_op_vec_t;
    typedef std::vector< id_op_ptr_t >
    id_op_vec_t;

  public:
    SparseOperatorHandler(std::size_t blocksize);
    ~SparseOperatorHandler();
    void compute( const PARENT_SD_H_T& parent_sd_h,
                  const DUAL_SD_H_T&   dual_sd_h );

    /// parent coupling operator
    pair_parent_ptr_t  give_p_coupling_operator ( std::size_t domain_id )
    { return coupling_p_op_vec_[domain_id]; }
    /// barycentric coupling operator
    pair_bary_ptr_t give_b_coupling_operator( std::size_t domain_id)
    { return coupling_b_op_vec_[domain_id]; }

    std::size_t   give_blocksize() const { return blocksize_; }

    template< typename EXPORTER_T >
    void Write(EXPORTER_T& exporter);

  private:
    void compute_( const dual_dirichlet_dh_t&   dual_dirichlet_dh,
                   const dual_neumann_dh_t&     dual_neumann_dh,
                   const parent_dirichlet_dh_t& parent_dirichlet_dh,
                   const parent_neumann_dh_t&   parent_neumann_dh,
                   std::size_t            domain_id );
  private:
    /// NoOfSubdomains+1
    std::size_t       blocksize_;
    /// coupling for dofs on barycentric refined mesh
    coupling_bary_op_vec_t coupling_b_op_vec_;
    /// coupling parent dofs
    coupling_parent_op_vec_t coupling_p_op_vec_;
    /// mass matrices on barycentric refined mesh
    id_op_vec_t       id_op_vec_;


  };

  // ------------------------------------------------------------------
  template< typename PARENT_SD_H_T,
            typename DUAL_SD_H_T>
  SparseOperatorHandler< PARENT_SD_H_T,
                         DUAL_SD_H_T>::
  SparseOperatorHandler(std::size_t blocksize)
    :
    blocksize_(blocksize),
```

```
124           coupling_p_op_vec_(blocksize),
              coupling_b_op_vec_(blocksize),
              id_op_vec_(blocksize)
            {
              /* empty */
129         }
            // -------------------------------------------------------------------
            template< typename PARENT_SD_H_T,
                      typename DUAL_SD_H_T>
            void SparseOperatorHandler< PARENT_SD_H_T,
134                                     DUAL_SD_H_T>::
            compute( const PARENT_SD_H_T& parent_sd_h,
                     const DUAL_SD_H_T&   dual_sd_h )
            {
              for (std::size_t i = 0; i < blocksize_; ++i ) {
139             compute_( dual_sd_h.giveDirichletDofHandler(i),
                          dual_sd_h.giveNeumannDofHandler(i),
                          parent_sd_h.giveDirichletDofHandler(i),
                          parent_sd_h.giveNeumannDofHandler(i),
                          i);
144           }
            }

            // -------------------------------------------------------------------
            template< typename PARENT_SD_H_T,
149                     typename DUAL_SD_H_T>

            void SparseOperatorHandler< PARENT_SD_H_T,
                                        DUAL_SD_H_T>::
            compute_( const dual_dirichlet_dh_t&   dual_dirichlet_dh,
154                   const dual_neumann_dh_t&     dual_neumann_dh,
                      const parent_dirichlet_dh_t& parent_dirichlet_dh,
                      const parent_neumann_dh_t&   parent_neumann_dh,
                      std::size_t domain_id )
            {
159           // mass matrix on barycentric refined mesh
              id_op_ptr_t id_op( new id_op_t(dual_neumann_dh, dual_dirichlet_dh));
              id_op->compute();
              id_op_vec_[domain_id] = id_op;

164           // coupling operators (dual)
              CBD_ptr_t CBD (new CBD_t(dual_dirichlet_dh, parent_neumann_dh) );
              CBD->compute();
              CBN_ptr_t CBN (new CBN_t(dual_neumann_dh, parent_dirichlet_dh) );
              CBN->compute();
169           pair_bary_ptr_t pair_bary(new pair_bary_t( CBD, CBN ));
              coupling_b_op_vec_[domain_id] =pair_bary;

              // coupling operators (parent)
              CPD_ptr_t CPD( new CPD_t(dual_dirichlet_dh, parent_dirichlet_dh));
174           CPD->compute();
              CPN_ptr_t CPN( new CPN_t(dual_neumann_dh, parent_neumann_dh));
              CPN->compute();
              pair_parent_ptr_t pair_parent (new pair_parent_t( CPD, CPN ));
```

```
        coupling_p_op_vec_[domain_id] = pair_parent;
      }

      // ------------------------------------------------------------------
      template< typename PARENT_SD_H_T,
                typename DUAL_SD_H_T>
      SparseOperatorHandler< PARENT_SD_H_T,
                             DUAL_SD_H_T>::
      ~SparseOperatorHandler()
      {
        /* empty */
      }

      // ------------------------------------------------------------------
      template< typename PARENT_SD_H_T,
                typename DUAL_SD_H_T>
      template< typename EXPORTER_T >
      void SparseOperatorHandler< PARENT_SD_H_T,
                                  DUAL_SD_H_T>::
      Write(EXPORTER_T& exporter)
      {

        for (size_t i = 0; i < blocksize_; ++i ) {
          std::string stri = boost::lexical_cast<std::string>(i);

          // export coupling
          exporter( "CBD"+stri,
                    coupling_b_op_vec_[i]->first->giveMatrix() );
          exporter( "CBN"+stri,
                    coupling_b_op_vec_[i]->second->giveMatrix() );
          exporter( "CPD"+stri,
                    coupling_p_op_vec_[i]->first->giveMatrix() );
          exporter( "CPN"+stri,
                    coupling_p_op_vec_[i]->second->giveMatrix() );

          exporter( "ID"+stri,
                    id_op_vec_[i]->giveMatrix() );
        }
      }
    } // end namespace dual
  } // end namespace lmt
} // end namespace betl


#endif /* __PRECONDITIONER_SPARSE_OPERATOR_HANDLER_H__ */
```

Listing E.10: Dual coupling (Dirichlet)

```
#ifndef __DUAL_COUPLING_DIRICHLET_H__
#define __DUAL_COUPLING_DIRICHLET_H__


// ------------------------------------------------------------------
// Dual Dirichlet -> Dirichlet on barycentric refined mesh
```

```
// ! equivalent to averaging_operator
// ! in preconditioner/lagrange/averaging_operator.hpp
//
namespace betl{
  namespace lmt{
    namespace dual {
      template< enum ACCELERATION ACC>
      class coupling_operator<
        DoFHandler< FEBasis< MultiBaryElement<3>,
                             LINEAR,
                             Continuous,
                             LagrangeTraits>, ACC >,
        DoFHandler< FEBasis< MultiElement<3>,
                             CONSTANT,
                             Discontinuous,
                             LagrangeTraits>, ACC >
        >
        : public
      RootSparseOperator<
        DoFHandler< FEBasis< MultiBaryElement<3>,
                             LINEAR,
                             Continuous,
                             LagrangeTraits>, ACC >,
        DoFHandler< FEBasis< MultiElement<3>,
                             CONSTANT,
                             Discontinuous,
                             LagrangeTraits>, ACC >,
        scalar_matrix_policy< double, sparse::Assign > >


      {
      private:
        typedef FEBasis< MultiBaryElement<3>,
                         LINEAR,
                         Continuous,
                         LagrangeTraits > test_febasis_t;
        typedef FEBasis< MultiElement<3>,
                         CONSTANT,
                         Discontinuous,
                         LagrangeTraits> trial_febasis_t;

        // Test dofhandler
        typedef DoFHandler< test_febasis_t,ACC>  test_dh_t;
        typedef test_dh_t dual_dh_t;
        // Trial dofhandler
        typedef DoFHandler< trial_febasis_t,ACC> trial_dh_t;
        typedef trial_dh_t parent_dh_t;

        typedef RootSparseOperator< dual_dh_t, parent_dh_t,
                                    scalar_matrix_policy< double,
                                                          sparse::Assign >
                                    > rso_t;
        typedef typename dual_dh_t::egd_type egd_t;
```

```
    using rso_t::testdof_;
    using rso_t::trialdof_;

    typedef ElementTraits<MultiBaryElement<3> > ET;
    typedef typename ET::Index::VarType index_t;

    typedef std::vector< double > vector_t;
    typedef std::map< index_t, vector_t> map_t;


  public:
    coupling_operator ( const dual_dh_t&   dual_dh,
                        const parent_dh_t& parent_dh )
      : rso_t ( dual_dh, parent_dh )
    {
      /* empty */
    }

    void compute( )
    {
      map_t weights;
      this -> initialize_( testdof_, trialdof_, weights );

      averaging_functor_   af( weights );
      averaging_interlace_ ai;
      this -> rso_t::compute( af, ai );
    }

  private:

    //-----------------------------------------------------------------------
    void initialize_( const dual_dh_t& dual_dh,
                      const parent_dh_t& parent_dh,
                      map_t& weights ) const {
      typedef typename dual_dh_t  ::const_elem_iterator
        const_dual_e_iterator;

      typedef std::set< std::size_t > set_t;
      typedef std::map< std::size_t, set_t > set_map_t;
      set_map_t set_map;

      // go through all dual dofs
      const_dual_e_iterator dual_begin = dual_dh.begin( );
      const_dual_e_iterator dual_end   = dual_dh.end( );
      const_dual_e_iterator dual_iter  = dual_begin;
      for( ; dual_iter != dual_end ; ++dual_iter ) {
        // get the iterator attached to its parent
        typedef typename parent_dh_t::const_elem_iterator
          const_parent_e_iterator;
        const_parent_e_iterator parent_iter =
          averaging_interlace_()( dual_iter, parent_dh );
        // get the parents dof index
        const std::size_t parent_idx =
          (*parent_dh.elem_dofs_begin(*parent_iter))->GivePermutatedDofId();
```

```
            // go through all dual dofs
            typedef typename dual_dh_t::const_elem_dof_iterator
              const_d_iterator;
            const_d_iterator dof_begin = dual_dh.elem_dofs_begin( *dual_iter );
            const_d_iterator dof_end   = dual_dh.elem_dofs_end  ( *dual_iter );
            const_d_iterator dof_iter  = dof_begin;
            for( ; dof_iter != dof_end; ++dof_iter ) {
              const std::size_t dof_idx = (*dof_iter) -> GivePermutatedDofId();
              set_map_t::iterator dof_found = set_map.find( dof_idx );
              if( dof_found != set_map.end() )
                ( dof_found -> second ).insert( parent_idx );
              else {
                set_t init_set;
                init_set.insert( parent_idx );
                set_map.insert( std::make_pair(dof_idx, init_set) );
              }
            }
          }
          // go through all dual dofs again
          dual_begin = dual_dh.begin( );
          dual_end   = dual_dh.end( );
          dual_iter  = dual_begin;
          for( ; dual_iter != dual_end ; ++dual_iter ) {
            // get the element's index
            const index_t elem_idx =
              typename
              ET::Index()((dual_iter->second ).second -> giveElemPtr());
            // go through all dual dofs
            typedef typename dual_dh_t::const_elem_dof_iterator
              const_d_iterator;
            const_d_iterator dof_begin = dual_dh.elem_dofs_begin( *dual_iter );
            const_d_iterator dof_end   = dual_dh.elem_dofs_end  ( *dual_iter );
            const_d_iterator dof_iter  = dof_begin;
            vector_t vec( dof_end - dof_begin );
            for( ; dof_iter != dof_end; ++dof_iter ) {
              const std::size_t idx = (*dof_iter) -> GivePermutatedDofId();
              // lookup idx in set_map
              set_map_t::const_iterator found = set_map.find( idx );
              const double value =
                1.0 / static_cast< double >( (found -> second).size() );
              vec[ dof_iter - dof_begin ] = value;
            }
            weights.insert( std::make_pair( elem_idx, vec ) );
          }
        }


        //-------------------------------------------------------------------
        struct averaging_interlace_ {
          typedef typename parent_dh_t::const_elem_iterator
          const_parent_iterator;
          typedef typename dual_dh_t  ::const_elem_iterator
          const_dual_iterator;
```

```
    typedef const_parent_iterator const_elem_iterator;
    typedef typename parent_dh_t::const_elem_dof_iterator
    const_elem_dof_iterator;

    const_parent_iterator operator()( const_dual_iterator& iter,
                                      const parent_dh_t& dofhandler )
      const
    {
      typedef ElementTraits< Element<3> > PET;
      typedef typename PET::Index::VarType index_t;
      const Element<3>* elem =
        ( (*iter).second ).second -> giveElemPtr() -> giveParent( );
      const index_t idx = typename PET::Index()( elem );
      return dofhandler.giveElemIterator( idx );
    }

    const_elem_dof_iterator begin( const_elem_iterator iter,
                                   const parent_dh_t& dofhandler) const
    {
      return dofhandler.elem_dofs_begin(*iter);
    }

    const_elem_dof_iterator end( const_elem_iterator iter,
                                 const parent_dh_t& dofhandler) const
    {
      return dofhandler.elem_dofs_end(*iter);
    }
  };

  //-------------------------------------------------------------------------
  class averaging_functor_ {
  public:
    averaging_functor_( const map_t& weights )
      : weights_( weights ) { /* empty */ }

    void operator()( const egd_t* tau ) {
      const index_t elem_idx =
        typename ET::Index()( tau -> giveElemPtr() );
      elem_iter_ = weights_.find( elem_idx );
      if( elem_iter_ == weights_.end( ) ) {
        std::cerr << "dual_averaging::averaging_functor_::operator()"
                  << "Cannot find element index '" << elem_idx
                  << "'. Abort!" << std::endl;
        exit( -1 );
      }
    }

    double operator()( const std::size_t row_idx,
                       const std::size_t col_idx ) const
    {
      return ( elem_iter_ -> second ).at( row_idx );
    }

  private:
```

```
223           const map_t&          weights_;
              map_t::const_iterator elem_iter_;

          };
        };
228   } // end namespace dual
    } // end namespace lmt
} // end namespace betl

#endif /* __DUAL_COUPLING_DIRICHLET_H__ */
```

Listing E.11: Dual coupling (Neumann)

```
#ifndef __DUAL_COUPLING_NEUMANN_H__
#define __DUAL_COUPLING_NEUMANN_H__
3
namespace betl {
  namespace lmt {
    namespace dual{
      // ------------------------------------------------------------------------
8     // this is B_N
      // Coupling of the barycentric piecewise constant discontinuous fcts,
      // forms piecewise constant discontinous fcts on the dual cells
      template< enum ACCELERATION ACC>
      class coupling_operator<
13      DoFHandler< FEBasis< MultiBaryElement<3>,
                             CONSTANT,
                             Discontinuous,
                             LagrangeTraits >, ACC >,
        DoFHandler< FEBasis< MultiElement<3>,
18                           LINEAR,
                             Continuous,
                             LagrangeTraits >, ACC >
        > : public
      RootSparseOperator< DoFHandler< FEBasis< MultiBaryElement<3>,
23                                               CONSTANT,
                                                 Discontinuous,
                                                 LagrangeTraits >, ACC >,
                          DoFHandler< FEBasis< MultiElement<3>,
                                               LINEAR,
28                                             Continuous,
                                               LagrangeTraits >, ACC >,
                          scalar_matrix_policy< double,
                                                sparse::Assign >
                          >
33    {
      private:
        typedef FEBasis< MultiBaryElement<3>,
                         CONSTANT,
                         Discontinuous,
38                       LagrangeTraits > dual_febasis_t;
        typedef FEBasis< MultiElement<3>,
                         LINEAR,
                         Continuous,
```

```
                    LagrangeTraits > parent_febasis_t;

  typedef DoFHandler< dual_febasis_t, ACC  > dual_dh_t;
  typedef DoFHandler< parent_febasis_t, ACC> parent_dh_t;

  typedef RootSparseOperator< dual_dh_t, parent_dh_t,
                              scalar_matrix_policy< double,
                                                    sparse::Assign >
                              > rso_t;

  typedef typename dual_dh_t::egd_type egd_t;


public:
  coupling_operator( const dual_dh_t&   dual_dh,
                     const parent_dh_t& parent_dh )
    : rso_t( dual_dh, parent_dh )
  {
    /* empty */
  }

  void compute( )
  {
    coupling_functor_ cf;
    coupling_interlace_ ci;

    this -> rso_t::compute( cf, ci );
  }

private:

  // Functor
  // ------------------------------------------------------------------
  struct coupling_functor_ {

    typedef typename egd_t::ElementType ElementType;

    void operator()( const egd_t* tau)
    {
      const ElementType* E = tau->giveElemPtr();
      std::size_t pos_in_parent = E->givePositionInParent();
      parent_vertex_ = pos_to_parent_vertex_id_[pos_in_parent];
    }

    double operator()( std::size_t row_idx, std::size_t col_idx )
    {
      return parent_vertex_ == col_idx ? 1.0 : 0.0;
    }


  private:
    short parent_vertex_;
    const boost::array<short,6> pos_to_parent_vertex_id_ =
      {{0,1,1,2,2,0}};
```

```
        };

98      // Interlace
        // ------------------------------------------------------------------
        struct coupling_interlace_ {
          typedef typename parent_dh_t::const_elem_iterator
          const_parent_iterator;
103       typedef typename dual_dh_t  ::const_elem_iterator
          const_dual_iterator;

          typedef const_parent_iterator const_elem_iterator;
          typedef typename parent_dh_t::const_elem_dof_iterator
108       const_elem_dof_iterator;

          const_parent_iterator operator() ( const_dual_iterator& iter,
                                             const parent_dh_t& dofhandler )
            const
113       {
            typedef ElementTraits< Element<3> > PET;
            typedef typename PET::Index::VarType index_t;
            const Element<3>* elem =
              ( (*iter).second ).second -> giveElemPtr() -> giveParent( );
118         const index_t idx = typename PET::Index()( elem );
            return dofhandler.giveElemIterator( idx );
          }

          const_elem_dof_iterator begin( const_elem_iterator iter,
123                                      const parent_dh_t& dofhandler) const
          {
            return dofhandler.elem_dofs_begin(*iter);
          }

128       const_elem_dof_iterator end( const_elem_iterator iter,
                                     const parent_dh_t& dofhandler) const
          {
            return dofhandler.elem_dofs_end(*iter);
          }
133     };
      };
    } // end namespace dual
  } // end namespace lmt
} // end namespace betl
138
#endif /* __DUAL_COUPLING_NEUMANN_H__ */
```

Listing E.12: Parent coupling (Dirichlet)

```
#ifndef __PARENT_COUPLING_DIRICHLET_H__
#define __PARENT_COUPLING_DIRICHLET_H__


5   namespace betl {
      namespace lmt {
        namespace dual {
```

```
// ------------------------------------------------------------------
// parent_mesh -> barycentric refined mesh
// coupling of barycentric lin. cont. fcts to lin. cont. fcts.
template< enum ACCELERATION ACC>
class coupling_operator<
  DoFHandler< FEBasis< MultiBaryElement<3>,
                       LINEAR,
                       Continuous,
                       LagrangeTraits >, ACC >,
  DoFHandler< FEBasis< MultiElement<3>,
                       LINEAR,
                       Continuous,
                       LagrangeTraits >, ACC >
  > : public
RootSparseOperator< DoFHandler< FEBasis< MultiBaryElement<3>,
                                         LINEAR,
                                         Continuous,
                                         LagrangeTraits >, ACC >,
                    DoFHandler< FEBasis< MultiElement<3>,
                                         LINEAR,
                                         Continuous,
                                         LagrangeTraits >, ACC >,
                    scalar_matrix_policy< double,
                                          sparse::Assign >
                    >
{
private:
  typedef FEBasis< MultiBaryElement<3>,
                   LINEAR,
                   Continuous,
                   LagrangeTraits > dual_febasis_t;
  typedef FEBasis< MultiElement<3>,
                   LINEAR,
                   Continuous,
                   LagrangeTraits > parent_febasis_t;

  typedef DoFHandler< dual_febasis_t, ACC  > dual_dh_t;
  typedef DoFHandler< parent_febasis_t, ACC> parent_dh_t;

  typedef RootSparseOperator< dual_dh_t, parent_dh_t,
                              scalar_matrix_policy< double,
                                                    sparse::Assign >
                              > rso_t;

  typedef typename dual_dh_t::egd_type egd_t;

  typedef boost::array< double,  3 > array_t;
  typedef boost::array< array_t, 2 > coupling_coeffs_t;

public:
  coupling_operator( const dual_dh_t&   dual_dh,
                     const parent_dh_t& parent_dh )
    : rso_t( dual_dh, parent_dh )
  {
```

```
      /* empty */
    }

    void compute( )
    {
      coupling_functor_ cf;
      coupling_interlace_ ci;

      this -> rso_t::compute( cf, ci );
    }


  private:
    coupling_coeffs_t coupling_coeffs_;

    struct coupling_functor_ {

    public:
      coupling_functor_( )
      {
        // position in parent is even
        coupling_coeffs_[0][0] = 1/3.;
        coupling_coeffs_[0][1] = 1.;
        coupling_coeffs_[0][2] = 0.5;
        // position in parent is odd
        coupling_coeffs_[1][0] = 1/3.;
        coupling_coeffs_[1][1] = 0.5;
        coupling_coeffs_[1][2] = 1.;

      }

      void operator()( const egd_t* tau )
      {
        const ElementType* E = tau->giveElemPtr();
        std::size_t pos_in_parent = E->givePositionInParent();
        parent_vertex_id_ = pos_to_parent_vertex_id_[pos_in_parent];
        coeffs_row_idx_ = pos_in_parent % 2;
      }

      double operator()( std::size_t row_idx, std::size_t col_idx)
      {
        return col_idx == parent_vertex_id_ ?
          coupling_coeffs_[coeffs_row_idx_][row_idx] :
          0.0;
      }

    private:
      typedef typename egd_t::ElementType ElementType;

      // maps pos_in_parent  (barycentric element) to parent vertex id
      const boost::array<short,6> pos_to_parent_vertex_id_ =
        {{0,1,1,2,2,0}};

      std::size_t parent_vertex_id_;
```

```
          std::size_t coeffs_row_idx_;
          coupling_coeffs_t coupling_coeffs_;
        };

        struct coupling_interlace_ {
          typedef typename parent_dh_t::const_elem_iterator
          const_parent_iterator;
          typedef typename dual_dh_t  ::const_elem_iterator
          const_dual_iterator;

          typedef const_parent_iterator const_elem_iterator;
          typedef typename parent_dh_t::const_elem_dof_iterator
          const_elem_dof_iterator;

          const_parent_iterator operator() ( const_dual_iterator& iter,
                                             const parent_dh_t& dofhandler )
            const
          {
            typedef ElementTraits< Element<3> > PET;
            typedef typename PET::Index::VarType index_t;
            const Element<3>* elem =
              ( (*iter).second ).second -> giveElemPtr() -> giveParent( );
            const index_t idx = typename PET::Index()( elem );
            return dofhandler.giveElemIterator( idx );
          }

          const_elem_dof_iterator begin( const_elem_iterator iter,
                                         const parent_dh_t& dofhandler) const
          {
            return dofhandler.elem_dofs_begin(*iter);
          }

          const_elem_dof_iterator end( const_elem_iterator iter,
                                       const parent_dh_t& dofhandler) const
          {
            return dofhandler.elem_dofs_end(*iter);
          }
        };
      };
    } // end namespace dual
  } // end namespace lmt
} // end namespace betl


#endif /* __PARENT_COUPLING_DIRICHLET_H__ */
```

Listing E.13: Parent coupling (Neumann)

```
#ifndef __PARENT_COUPLING_NEUMANN_H__
#define __PARENT_COUPLING_NEUMANN_H__

namespace betl {
  namespace lmt {
    namespace dual {
```

```
// ------------------------------------------------------------------
// coupling of barycentric refined constant functions to parent mesh
//
template< enum ACCELERATION ACC>
class coupling_operator<
  DoFHandler< FEBasis< MultiBaryElement<3>,
                       CONSTANT,
                       Discontinuous,
                       LagrangeTraits >, ACC >,
  DoFHandler< FEBasis< MultiElement<3>,
                       CONSTANT,
                       Discontinuous,
                       LagrangeTraits >, ACC >
  > : public
RootSparseOperator<
  DoFHandler< FEBasis< MultiBaryElement<3>,
                       CONSTANT,
                       Discontinuous,
                       LagrangeTraits >, ACC >,
  DoFHandler< FEBasis< MultiElement<3>,
                       CONSTANT,
                       Discontinuous,
                       LagrangeTraits >, ACC >,
  scalar_matrix_policy< double,
                        sparse::Assign >
  >
{
private:
  typedef FEBasis< MultiBaryElement<3>,
                   CONSTANT,
                   Discontinuous,
                   LagrangeTraits >
  test_febasis_t;
  typedef  FEBasis< MultiElement<3>,
                    CONSTANT,
                    Discontinuous,
                    LagrangeTraits >
  trial_febasis_t;
  typedef DoFHandler<test_febasis_t,ACC> test_dh_t;
  typedef test_dh_t dual_dh_t;
  typedef DoFHandler<trial_febasis_t,ACC> trial_dh_t;
  typedef trial_dh_t parent_dh_t;
  typedef RootSparseOperator< dual_dh_t, parent_dh_t,
                              scalar_matrix_policy< double,
                                                    sparse::Assign >
                              > rso_t;
  typedef typename dual_dh_t::egd_type egd_t;

public:
  coupling_operator( const dual_dh_t& dual_dh, const parent_dh_t& parent_dh )
    : rso_t(dual_dh, parent_dh) { /* empty */ }

  void compute()
  {
```

```
        coupling_functor_ cf;
        coupling_interlace_ ci;

64      this->rso_t::compute( cf, ci );
      }

    private:

69      struct coupling_functor_ {
        void operator()( const egd_t* tau )
        { /* empty */ }

        double operator()(std::size_t row_idx, std::size_t col_idx)
74      {
          return 1.0;
        }
      };

79      struct coupling_interlace_ {
        typedef typename parent_dh_t::const_elem_iterator
        const_parent_iterator;
        typedef typename dual_dh_t  ::const_elem_iterator
        const_dual_iterator;
84
        typedef const_parent_iterator const_elem_iterator;
        typedef typename parent_dh_t::const_elem_dof_iterator
        const_elem_dof_iterator;

89      const_parent_iterator operator() ( const_dual_iterator& iter,
                                           const parent_dh_t& dofhandler )
        const
        {
          typedef ElementTraits< Element<3> > PET;
94        typedef typename PET::Index::VarType index_t;
          const Element<3>* elem =
            ( (*iter).second ).second -> giveElemPtr() -> giveParent( );
          const index_t idx = typename PET::Index()( elem );
          return dofhandler.giveElemIterator( idx );
99      }

        const_elem_dof_iterator begin( const_elem_iterator iter,
                                       const parent_dh_t& dofhandler) const
        {
104       return dofhandler.elem_dofs_begin(*iter);
        }

        const_elem_dof_iterator end( const_elem_iterator iter,
                                     const parent_dh_t& dofhandler) const
109     {
          return dofhandler.elem_dofs_end(*iter);
        }
      };
    };
114 }  // end namespace dual
```

```
  } // end namespace lmt
} // end namespace betl


#endif /* __PARENT_COUPLING_NEUMANN_H__ */
```

E.3 Dirichlet boundary conditions

E.3.1 Subdomain handler

This is a factory for the class `BCSubdomainHandler`, described in listing E.16 on page 89.

Listing E.14: BCSubdomainHandler

```
#ifndef __BC_SUBDOMAIN_HANDLER_GENERATOR_H__
#define __BC_SUBDOMAIN_HANDLER_GENERATOR_H__

// system includes ------------------------------------------------------------
#include <set>
#include <utility>
#include <vector>
#include <boost/foreach.hpp>

// own includes ---------------------------------------------------------------
#include "interlace.hpp"

namespace betl {
  namespace lmt {
    class BCSubdomainHandlerGenerator {
    public:
      typedef std::set<std::size_t> index_set_t;
      typedef typename index_set_t::const_iterator const_iterator;

    public:
      /**
       *
       * @param sd_handler       subdomain handler on entire domain
       * @param mesh
       * @param dirichlet_bc_id  indices of subdomains in \Gamma_DIR
       * @param nf_sd_handler    all dofhandlers live on |Gamma_{DIR}
       * @param bc_sd_handler    subdomain handler taking care of dirichlet bc
       */
      template< typename SD_HANDLER,
                typename MESH,
                typename BC_SD_HANDLER >
      static void Create( const SD_HANDLER&  sd_handler,
                          const MESH&        mesh,
                          const index_set_t& dirichlet_bc_id,
                          SD_HANDLER*        nf_sd_handler,
                          BC_SD_HANDLER&     bc_sd_handler);
```

```
  template< typename SD_HANDLER,
            typename MESH,
            typename BC_SD_HANDLER >
  static void Create( const SD_HANDLER&        sd_handler,
                      const std::vector<MESH*> mesh,
                      const index_set_t&       dirichlet_bc_id,
                      SD_HANDLER*        nf_sd_handler,
                      BC_SD_HANDLER&     bc_sd_handler);


private:
  template< typename ELEMENT_ITERATOR,
            typename SD_HANDLER,
            typename BC_SD_HANDLER>
  static void init_( const ELEMENT_ITERATOR  begin,
                     const ELEMENT_ITERATOR  end,
                     const SD_HANDLER&       sd_handler,
                     const index_set_t&      dirichlet_bc_id,
                     const size_t            domain_id,
                     SD_HANDLER*             nf_sd_handler,
                     BC_SD_HANDLER&          bc_sd_handler);
};

// ------------------------------------------------------------------
template< typename SD_HANDLER,
          typename MESH,
          typename BC_SD_HANDLER>
void BCSubdomainHandlerGenerator::
Create( const SD_HANDLER&  sd_handler,
        const MESH&        mesh,
        const index_set_t& dirichlet_bc_id,
        SD_HANDLER*        nf_sd_handler,
        BC_SD_HANDLER&     bc_sd_handler)
{
  const std::size_t blocksize = sd_handler.give_blocksize();
  for (std::size_t id = 0; id < blocksize; ++id) {
    //        std::cout << "Creation on SD=" << id << std::endl;
    const_iterator it = dirichlet_bc_id.find(id);
    if( it != dirichlet_bc_id.end() ) {
      /* subdomain with dirichlet bc */
      // set const dofhandler and lin disc
      bc_sd_handler.setup(sd_handler.give_const_ptr(id),
                          sd_handler.give_lin_disc_ptr(id),
                          id);
      // set lin dofhandler
      bc_sd_handler.setup(
                           sd_handler.give_lin_cont_ptr(id),
                           id);
    } else {
      /* regular subdomain */
      // copy Neumann dofhandler and lin disc dofhandler
      bc_sd_handler.setup(sd_handler.give_const_ptr(id),
                          sd_handler.give_lin_disc_ptr(id),
```

```
                                id);
          // create `lin disc` and `lin cont` dofhandlers on NON-Dirichlet
          // BC elements
          if (id < blocksize-1) {
            init_( mesh.e_begin(id),
                   mesh.e_end(id),
                   sd_handler,
                   dirichlet_bc_id,
                   id,
                   nf_sd_handler,
                   bc_sd_handler);
          } else {
            //            std::cout << "creation hull" << std::endl;
            init_( mesh.e_hull_begin(),
                   mesh.e_hull_end(),
                   sd_handler,
                   dirichlet_bc_id,
                   id,
                   nf_sd_handler,
                   bc_sd_handler);
          }
        }
      }
    }

    // ------------------------------------------------------------------------
    template< typename SD_HANDLER,
              typename MESH,
              typename BC_SD_HANDLER >
    void BCSubdomainHandlerGenerator::
    Create( const SD_HANDLER&        sd_handler,
            const std::vector<MESH*> mesh,
            const index_set_t&       dirichlet_bc_id,
            SD_HANDLER*        nf_sd_handler,
            BC_SD_HANDLER&     bc_sd_handler)
    {
      const std::size_t blocksize = sd_handler.give_blocksize();
      for (std::size_t id = 0; id < blocksize; ++id) {
        //        std::cout << "Creation on SD=" << id << std::endl;
        if( dirichlet_bc_id.find(id) != dirichlet_bc_id.end() ) {
          /* subdomain with dirichlet bc */
          // set const dofhandler
          bc_sd_handler.setup(sd_handler.give_const_ptr(id),
                              sd_handler.give_lin_disc_ptr(id),
                              id);
          // set lin and lin disc. dofhandlers
          bc_sd_handler.setup( sd_handler.give_lin_cont_ptr(id),
                               id);
        } else {
          /* regular subdomain */
          // copy Neumann dofhandler and lin. disc. dofhandler
          bc_sd_handler.setup(sd_handler.give_const_ptr(id),
                              sd_handler.give_lin_disc_ptr(id),
                              id);
```

```
145           // create `lin disc` and `lin cont` dofhandlers on NON-Dirichlet
              // and lin cont on |Omega_i | \Gamma_DIR
              // BC elements
              init_( mesh[id]->e_begin(),
                     mesh[id]->e_end(),
150                  sd_handler,
                     dirichlet_bc_id,
                     id,
                     nf_sd_handler,
                     bc_sd_handler);
155         }
          }
        }

        // ------------------------------------------------------------------
160     template< typename ELEMENT_ITERATOR,
                  typename SD_HANDLER,
                  typename BC_SD_HANDLER>
        void BCSubdomainHandlerGenerator::
        init_( const ELEMENT_ITERATOR  begin,
165            const ELEMENT_ITERATOR  end,
               const SD_HANDLER&       sd_handler,
               const index_set_t&      dirichlet_bc_id,
               const size_t            domain_id,
               SD_HANDLER*             nf_sd_handler,
170            BC_SD_HANDLER&          bc_sd_handler)
        {
          typedef typename SD_HANDLER::element_t element_t;
          typedef typename SD_HANDLER::dirichlet_dofhandler_t dh_t;
          typedef local_::Interlace<element_t,dh_t> interlace_t;
175       interlace_t interlace;
          std::vector<element_t*> elem_vec;
          std::vector<element_t*> non_dof_vec;
          // iterate over the elements of subdomain i
          for ( ELEMENT_ITERATOR it = begin; it != end; ++it ) {
180         bool elem_found = false;
            // check if this element belongs to \Gamma_DIR
            BOOST_FOREACH( typename index_set_t::value_type id, dirichlet_bc_id) {
              // get dofhandler on \Gamma_DIR
              const dh_t& dh = sd_handler.giveDirichletDofHandler(id);
185           typedef typename dh_t::const_elem_iterator dh_elem_iter_t;
              const dh_elem_iter_t& dh_elem_iter = interlace(*it, dh);
              if (dh_elem_iter != dh.end()) {
                // there are boundary conditions imposed on this element
                elem_found = true;
190             break;
              }
            }
            if (!elem_found) elem_vec.push_back(*it);
            else non_dof_vec.push_back(*it);
195       } // end iterate over elements

          bc_sd_handler.setup( elem_vec.begin(),
                               elem_vec.end(),
```

```
                              domain_id );
      if ( non_dof_vec.begin() != non_dof_vec.end() ) {
        // do not initialize dofhandlers without elements (-> ahmed will
        // crash)
        if ( nf_sd_handler != NULL)
          nf_sd_handler->setup( non_dof_vec.begin(),
                                non_dof_vec.end(),
                                domain_id);
      }

    }
  } // end namespace lmt
} // end namespace betl

#endif /* __BC_SUBDOMAIN_HANDLER_GENERATOR_H__ */
```

Listing E.15: BCSubdomainHandler

```
#ifndef __BC_SUBDOMAIN_HANDLER_H__
#define __BC_SUBDOMAIN_HANDLER_H__

// own includes ------------------------------------------------------------
#include "../subdomain_handler.hpp"

namespace betl {
  namespace lmt {

    /**
     * This class contains the same dofhandler
     * as in SubdomainHandler, but with
     * Dirichlet dofhandlers on \pd \Omega_i \setminus \Gamma_DIR
     *
     * @param basename
     * @param NoOfSubdomains
     *
     * @return
     */
    template< enum ACCELERATION ACC,
              typename ELEMENT_T >
    class BCSubdomainHandler :
      public SubdomainHandler< ACC,
                               ELEMENT_T>
    {
    public:
      typedef SubdomainHandler<ACC,ELEMENT_T> SDH;
    public:
      using typename SDH::dofhandler_lin_cont_t;
      using typename SDH::dofhandler_const_t;
    private:
      using typename SDH::vec_dh_lin_cont_t;
      using typename SDH::vec_dh_const_t;
      /** @name pointer types */
      //@{
      using typename SDH::dh_const_ptr_t;
```

```
37      using typename SDH::dh_lin_disc_ptr_t;
        using typename SDH::dh_lin_cont_ptr_t;
        //@}
        /** @name factory types */
        //@{
42      using typename SDH::FF_lin_disc_t;
        using typename SDH::FF_lin_cont_t;
        using typename SDH::FF_const_t;
        //@}
      public:
47      BCSubdomainHandler( const std::string basename,
                            const std::size_t NoOfSubdomains);
        template< typename ELEMENT_ITERATOR_T>
        void setup( const ELEMENT_ITERATOR_T& begin,
                    const ELEMENT_ITERATOR_T& end,
52                  size_t domain_id);
        void setup( dh_const_ptr_t dh_const_ptr,
                    dh_lin_disc_ptr_t dh_lin_disc_ptr,
                    size_t domain_id );
        void setup( dh_lin_cont_ptr_t dh_lin_cont_ptr,
57                  size_t domain_id);


      private:
        template< typename ELEMENT_ITERATOR_T>
62      void distribute_( const ELEMENT_ITERATOR_T& begin,
                          const ELEMENT_ITERATOR_T& end,
                          size_t domain_id);
        void clusterize_(size_t domain_id);

67    private:
        using SDH::vec_dh_const_;
        using SDH::vec_dh_lin_disc_;
        using SDH::vec_dh_lin_cont_;
        using SDH::fast_settings_parser_;
72    };

      // ------------------------------------------------------------------
      template< enum ACCELERATION ACC,
                typename ELEMENT_T >
77    BCSubdomainHandler<ACC,ELEMENT_T>::
      BCSubdomainHandler( const std::string basename,
                          const std::size_t NoOfSubdomains) :
        SDH(basename, NoOfSubdomains)
      {
82      /* empty */
      }

      // ------------------------------------------------------------------
      template< enum ACCELERATION ACC,
87              typename ELEMENT_T >
      template< typename ELEMENT_ITERATOR_T>
      void BCSubdomainHandler<ACC,ELEMENT_T>::
      setup( const ELEMENT_ITERATOR_T& begin,
```

```
            const ELEMENT_ITERATOR_T& end,
            size_t domain_id)
    {
      this->distribute_( begin, end, domain_id);
      this->clusterize_( domain_id );
    }

    // ------------------------------------------------------------------------
    template< enum ACCELERATION ACC,
              typename ELEMENT_T >
    void BCSubdomainHandler<ACC,ELEMENT_T>::
    setup( dh_const_ptr_t    dh_const_ptr,
           dh_lin_disc_ptr_t dh_lin_disc_ptr,
           size_t domain_id )
    {
      vec_dh_const_[domain_id] = dh_const_ptr;
      vec_dh_lin_disc_[domain_id] = dh_lin_disc_ptr;
    }
    // ------------------------------------------------------------------------
    template< enum ACCELERATION ACC,
              typename ELEMENT_T >
    void BCSubdomainHandler<ACC,ELEMENT_T>::
    setup( dh_lin_cont_ptr_t dh_lin_cont_ptr,
           size_t domain_id)
    {
      //vec_dh_lin_disc_[domain_id] = dh_lin_disc_ptr;
      vec_dh_lin_cont_[domain_id] = dh_lin_cont_ptr;
    }

    // ------------------------------------------------------------------------
    template< enum ACCELERATION ACC,
              typename ELEMENT_T >
    template< typename ELEMENT_ITERATOR_T>
    void BCSubdomainHandler<ACC,ELEMENT_T>::
    distribute_( const ELEMENT_ITERATOR_T& begin,
                 const ELEMENT_ITERATOR_T& end,
                 size_t domain_id)
    {
      //vec_dh_lin_disc_[domain_id] -> distributeDoFs( begin, end);
      vec_dh_lin_cont_[domain_id] -> distributeDoFs( begin, end);
    }

    // ------------------------------------------------------------------------
    template< enum ACCELERATION ACC,
              typename ELEMENT_T >
    void BCSubdomainHandler<ACC,ELEMENT_T>::
    clusterize_(size_t domain_id)
    {
      SDH::FF_lin_cont_t::Clusterize( *(vec_dh_lin_cont_[domain_id]), fast_settings_pa
    }
  } // end namespace lmt
} // end namespace betl

#endif /* __BC_SUBDOMAIN_HANDLER_H__ */
```

The following code checks for intersection in Dofhandlers for the element types `MultiElement` and `MultiBaryElement`.

Listing E.16: BCSubdomainHandler

```
#ifndef __BETL_LMT_INTERLACE_H__
#define __BETL_LMT_INTERLACE_H__


// own includes ----------------------------------------------------------
#include "../preconditioner/multi_bary_element.hpp"
// system includes -------------------------------------------------------
#include <iostream>

namespace betl {
  namespace local_ {
    template< typename ELEMENT,
              typename DOFHANDLER>
    struct Interlace {

      typedef typename DOFHANDLER::const_elem_iterator elem_iterator;

      template<typename ELEMENT_PTR>
      elem_iterator operator() (ELEMENT_PTR elem_ptr,
                                const DOFHANDLER& dofhandler) const
      {
        return dofhandler.giveElemIterator(elem_ptr);
      }
    };

    template< typename DOFHANDLER >
    struct Interlace<MultiBaryElement<3>,DOFHANDLER > {

      typedef typename DOFHANDLER::const_elem_iterator elem_iterator;
      typedef typename DOFHANDLER::egd_type egd_type;
      typedef MultiBaryElement<3> element_t;

      template< typename ELEMENT_PTR>
      elem_iterator operator() ( ELEMENT_PTR elem_ptr,
                                 const DOFHANDLER& dofhandler ) const
      {
        typedef typename element_t::ptr_type parent_ptr_t;
        parent_ptr_t pptr = elem_ptr->giveParent();
        const int pos_in_parent = elem_ptr->givePositionInParent();
        const std::size_t global_parent_idx = pptr->giveIdx();

        std::size_t idx;
        bool found = false;
        for (elem_iterator it = dofhandler.begin(); it != dofhandler.end(); ++it) {
          const egd_type* egd = dofhandler.giveElemGeometry(it);
          parent_ptr_t tmp = egd->giveElemPtr()->giveParent();
          // global parent idx
```

```
          const std::size_t gpi = tmp->giveIdx();
          // pos_in_parent
          const int pip = egd->giveElemPtr()->givePositionInParent();
          if(gpi == global_parent_idx) {
            // the element 'elem_ptr' was found in dofhandler, save idx and abort
            idx = it->first;
            found = true;
            break;
          } else {
            continue;
          }
        }

        if (found) return dofhandler.giveElemIterator(idx);
        else return dofhandler.end();
      }
    };
  }
} // end namespace betl


#endif /* __INTERLACE_H__ */
```

Sparse matrix EX

Listing E.17: ExclusionOperator

```
#ifndef __EXCLUSION_OPERATOR_HANDLER_H__
#define __EXCLUSION_OPERATOR_HANDLER_H__

// own includes ------------------------------------------------------------
#include "exclusion_operator.hpp"
#include "exclusion_matrix.hpp"
#include "../helper/dof_intersection.hpp"

// system includes ---------------------------------------------------------
#include <boost/foreach.hpp>
#include <map>
#include <set>
#include <iostream>
#include <cstddef>

namespace betl {
  namespace lmt {

    template< typename SD_HANDLER >
    class ExclusionOperatorHandler {
    private:
      typedef ExclusionOperatorHandler<SD_HANDLER>
      this_type;
      typedef std::set<size_t> index_set_t;
      typedef exclusion_operator operator_t;
      typedef std::map<std::size_t, operator_t*>  op_map_t;
```

```
    typedef std::map<std::size_t, index_set_t*> exclusion_map_t;
  public:
    typedef scalar_matrix_policy<double>::matrix_type matrix_type;
    typedef ExclusionMatrix<SD_HANDLER, this_type> exclusion_matrix_t;
  public:
    ExclusionOperatorHandler(std::size_t NoOfDomains);
    void compute( const SD_HANDLER& sd_handler,
                  index_set_t& dirichlet_bc_id );
    const index_set_t&        give_bc_nodes(std::size_t i) const;
    const matrix_type&        giveMatrix(std::size_t i)    const;
    const exclusion_matrix_t& giveMatrix()                 const;

  private:
    std::size_t NoOfDomains_;
    op_map_t exclude_map_;
    exclusion_map_t ex_index_map_;
    const index_set_t empty_set_;
    exclusion_matrix_t* matrix_;
  };

  // ------------------------------------------------------------------------
  template< typename SD_HANDLER>
  ExclusionOperatorHandler<SD_HANDLER>::
  ExclusionOperatorHandler( std::size_t NoOfDomains )
    : NoOfDomains_(NoOfDomains)
  {
    /* empty */
  }

  // ------------------------------------------------------------------------
  template< typename SD_HANDLER>
  void ExclusionOperatorHandler<SD_HANDLER>::
  compute( const SD_HANDLER& sd_handler,
           index_set_t& dirichlet_bc_id )
  {
    typedef typename SD_HANDLER::dirichlet_dofhandler_t
      dirichlet_dofhandler_t;
    for ( std::size_t i = 0; i < NoOfDomains_+1; ++i ) {
      if ( dirichlet_bc_id.find(i) == dirichlet_bc_id.end()) {
        /* this is a non dirichlet bc subdomain  */
        const dirichlet_dofhandler_t& dhi =
          sd_handler.giveDirichletDofHandler(i);

        index_set_t* exclude_i = new index_set_t();
        ex_index_map_[i] = exclude_i;
        // find non-free dofs in dhi
        BOOST_FOREACH( std::size_t v, dirichlet_bc_id) {
          const dirichlet_dofhandler_t& dhv =
            sd_handler.giveDirichletDofHandler(v);
          dof_intersection::get( dhi, dhv, *exclude_i );
        }
        // create exclusion operator from indices in exclude_i
        exclusion_operator* op = new exclusion_operator( dhi.GiveNoOfDofs(),
                                                         *exclude_i );
```

```
81            op->compute();
              exclude_map_[i] = op;
            }
          }

86        matrix_ = new exclusion_matrix_t(*this, sd_handler, dirichlet_bc_id);
        }

        // ------------------------------------------------------------------
        template< typename SD_HANDLER>
91      const std::set<std::size_t>& ExclusionOperatorHandler<SD_HANDLER>::
        give_bc_nodes(std::size_t i) const
        {
          typedef typename exclusion_map_t::const_iterator iter;
          iter it = ex_index_map_.find(i);
96        if ( it != ex_index_map_.end() )
            return *(it->second);
          else /* this subdomain id is a dirichlet domain */
            return empty_set_;
        }
101
        // ------------------------------------------------------------------
        template< typename SD_HANDLER>
        const typename ExclusionOperatorHandler<SD_HANDLER>::matrix_type&
        ExclusionOperatorHandler<SD_HANDLER>::
106     giveMatrix( std::size_t i) const
        {
          typedef typename op_map_t::const_iterator iterator;
          iterator it = exclude_map_.find(i);
          if (it != exclude_map_.end() ) {
111         return it->second->giveMatrix();
          } else {
            std::cerr << " ***Error: ExclusionOperatorHandler.giveMatrix(i=" << i << ")
    \n"
                      << "           this matrix does not exist. Exit!\n";
            exit(-1);
116       }
        }

        // ------------------------------------------------------------------
        template< typename SD_HANDLER>
121     const typename ExclusionOperatorHandler<SD_HANDLER>::exclusion_matrix_t&
        ExclusionOperatorHandler<SD_HANDLER>::
        giveMatrix() const
        {
          return *matrix_;
126     }

      } // end namespace lmt
    } // end namespace bet1

131 #endif /* __EXCLUSION_OPERATOR_HANDLER_H__ */
```

Listing E.18: ExclusionOperator

```
#ifndef __BETL_LMT_EXCLUSION_OPERATOR_H__
#define __BETL_LMT_EXCLUSION_OPERATOR_H__

// betl includes ------------------------------------------------------------
#include <sparse/sparse_matrix.hpp>
#include <sparse_operators/scalar_matrix_policy.hpp>
// system includes ----------------------------------------------------------
#include <set>

namespace betl {
  namespace lmt {
    class exclusion_operator : public  scalar_matrix_policy<double> {
    private:
      typedef scalar_matrix_policy<double> policy;
      typedef std::set<std::size_t> index_set_t;
    public:
      exclusion_operator( std::size_t size, const index_set_t& exclude_idx )
        : policy( size-exclude_idx.size(), size),
          size_(size),
          exclude_idx_(exclude_idx)
      {
        /* empty */
      }

      void compute(  ){
      std::size_t row_c = 0;
      for ( std::size_t i = 0; i < size_; ++i ) {
        if ( exclude_idx_.find(i) != exclude_idx_.end() ) {
          // this index is excluded
        } else {
          this -> assemble_ ( 1.0, row_c++, i );
        }
      }
      this->finalize_( );
    }

    private:
      std::size_t size_;
      const index_set_t& exclude_idx_;
    };
  } // end namespace lmt
} // end namespace betl

#endif /* __EXCLUSION_OPERATOR_H__ */
```

Listing E.19: Exclusion matrix

```
#ifndef __EXCLUSION_MATRIX_H__
#define __EXCLUSION_MATRIX_H__

#include "../lmt_config.hpp"
```

```
6  // betl includes ------------------------------------------------------------
   #include <linalg/linalg.hpp>
   #include <linalg/matrix_expression.hpp>

   // system includes ----------------------------------------------------------
11 #include <set>
   #include <iostream>

   namespace betl {
     namespace lmt {
16     template< typename SD_HANDLER,
                  typename EXCLUSION_OP_H>
       class ExclusionMatrix
         : public linalg::MatrixExpression< ExclusionMatrix< SD_HANDLER,
                                                              EXCLUSION_OP_H> > {
21     private:
         typedef SD_HANDLER sd_h_t;
         typedef EXCLUSION_OP_H ex_op_h_t;
         typedef std::set<std::size_t> index_set_t;
         typedef typename SD_HANDLER::dirichlet_dofhandler_t
26       dirichlet_dofhandler_t;
         typedef typename SD_HANDLER::neumann_dofhandler_t
         neumann_dofhandler_t;
       public:
         typedef config::numeric_t numeric_type;
31     public:
         ExclusionMatrix( const ex_op_h_t&   ex_op_h,
                          const sd_h_t&      sd_h,
                          const index_set_t& dirichlet_bc_id);

36       template< typename NUMERIC_T>
         void amux(NUMERIC_T alpha, NUMERIC_T* x, NUMERIC_T* y) const;
         template< typename NUMERIC_T>
         void tamux(NUMERIC_T alpha, NUMERIC_T* x, NUMERIC_T* y) const;
         std::size_t GiveRows() const { return rows_; }
41       std::size_t GiveCols() const { return cols_; }

       private:
         const ex_op_h_t&   ex_op_h_;
         const sd_h_t&      sd_h_;
46       const index_set_t& dirichlet_bc_id_;
         std::size_t        rows_;
         std::size_t        cols_;
         std::size_t        blocksize_;
       };
51
       // ------------------------------------------------------------------------
       template< typename SD_HANDLER,
                 typename EXCLUSION_OP_H>
       ExclusionMatrix< SD_HANDLER,
56                      EXCLUSION_OP_H>::
       ExclusionMatrix( const ex_op_h_t&   ex_op_h,
                        const sd_h_t&      sd_h,
                        const index_set_t& dirichlet_bc_id)
```

```
    : ex_op_h_(ex_op_h),
      sd_h_(sd_h),
      dirichlet_bc_id_(dirichlet_bc_id),
      rows_(0),
      cols_(0),
      blocksize_(sd_h.give_blocksize())
{
  for ( std::size_t i = 0; i < blocksize_; ++i ) {
    const dirichlet_dofhandler_t& ddofh = sd_h_.giveDirichletDofHandler(i);
    const neumann_dofhandler_t& ndofh = sd_h_.giveNeumannDofHandler(i);

    if ( dirichlet_bc_id_.find(i) != dirichlet_bc_id_.end()  ) {
      /* dirichlet bc domain */
      // no contribution
    } else {
      /* regular domain */
      rows_ += ex_op_h.giveMatrix(i).GiveCols();
      cols_ += ex_op_h.giveMatrix(i).GiveRows();
    }
    rows_ += ndofh.GiveNoOfDofs();
    cols_ += ndofh.GiveNoOfDofs();
  }
}

// ------------------------------------------------------------------
template< typename SD_HANDLER,
          typename EXCLUSION_OP_H>
template< typename NUMERIC_T>
void ExclusionMatrix< SD_HANDLER,
                      EXCLUSION_OP_H>::
amux(NUMERIC_T alpha, NUMERIC_T* x, NUMERIC_T* y) const
{
  typedef linalg::VectorWrapper<NUMERIC_T> vector_wrapper_t;

  std::size_t offsetX = 0;
  std::size_t offsetY = 0;
  for ( std::size_t i = 0; i < blocksize_; ++i ) {
    const dirichlet_dofhandler_t& ddofh = sd_h_.giveDirichletDofHandler(i);
    const neumann_dofhandler_t& ndofh = sd_h_.giveNeumannDofHandler(i);
    const std::size_t sizeD = ddofh.GiveNoOfDofs();
    const std::size_t sizeN = ndofh.GiveNoOfDofs();
    if ( dirichlet_bc_id_.find(i) != dirichlet_bc_id_.end() ) {
      // copy
      vector_wrapper_t vx(sizeN, x+offsetX);
      vector_wrapper_t vy(sizeN, y+offsetY);
      vy.add(alpha, vx);
      offsetX += sizeN;
      offsetY += sizeN;
    } else {
      // apply exclusion operator
      typedef typename EXCLUSION_OP_H::matrix_type matrix_type;
      const matrix_type& EX = ex_op_h_.giveMatrix(i);
      const std::size_t sizeDx = EX.GiveRows();
      vector_wrapper_t vdx(sizeDx, x+offsetX);
```

```
            vector_wrapper_t vdy(sizeD , y+offsetY);
            vdy += alpha*(EX^'T')*vdx;
            offsetX += sizeDx;
            offsetY += sizeD;
            // copy neumann
            vector_wrapper_t vnx(sizeN, x+offsetX);
            vector_wrapper_t vny(sizeN, y+offsetY);
            vny.add(alpha,vnx);
            offsetX += sizeN;
            offsetY += sizeN;
          }
        }
      }

      // ------------------------------------------------------------------------
      template< typename SD_HANDLER,
                typename EXCLUSION_OP_H>
      template< typename NUMERIC_T>
      void ExclusionMatrix< SD_HANDLER,
                            EXCLUSION_OP_H>::
      tamux(NUMERIC_T alpha, NUMERIC_T* x, NUMERIC_T* y) const
      {
        typedef linalg::VectorWrapper<NUMERIC_T> vector_wrapper_t;

        std::size_t offsetX = 0;
        std::size_t offsetY = 0;
        for ( std::size_t i = 0; i < blocksize_; ++i ) {
          const dirichlet_dofhandler_t& ddofh = sd_h_.giveDirichletDofHandler(i);
          const neumann_dofhandler_t& ndofh = sd_h_.giveNeumannDofHandler(i);
          const std::size_t sizeD = ddofh.GiveNoOfDofs();
          const std::size_t sizeN = ndofh.GiveNoOfDofs();
          if ( dirichlet_bc_id_.find(i) != dirichlet_bc_id_.end() ) {
            // copy
            vector_wrapper_t vx(sizeN, x+offsetX);
            vector_wrapper_t vy(sizeN, y+offsetY);
            vy.add(alpha,vx);
            offsetX += sizeN;
            offsetY += sizeN;
          } else {
            // copy neumann
            vector_wrapper_t vnx(sizeN, x+offsetX);
            vector_wrapper_t vny(sizeN, y+offsetY);
            vny.add(alpha,vnx);
            offsetX += sizeN;
            offsetY += sizeN;
            // map dirichlet
            typedef typename EXCLUSION_OP_H::matrix_type matrix_type;
            const matrix_type& EX = ex_op_h_.giveMatrix(i);
            const std::size_t sizeDy = EX.GiveRows();
            vector_wrapper_t vdx(sizeD, x+offsetX);
            vector_wrapper_t vdy(sizeDy,y+offsetY);
            vdy += alpha*EX*vdx;
            offsetX += sizeD;
            offsetY += sizeDy;
```

```
        }
      }
    }
  } // end namespace lmt
} // end namespace betl

#endif /* __EXCLUSION_MATRIX_H__ */
```

E.4 Load vector

This class assembles the right hand side of the local multi-trace formulation without boundary conditions.

Listing E.20: RHS generator

```
#ifndef BETL_LOCAL_MULTI_TRACE__RHS_GENERATOR_H__
#define BETL_LOCAL_MULTI_TRACE__RHS_GENERATOR_H__
// own includes ------------------------------------------------------------

// system includes ---------------------------------------------------------
#include <vector>
#include <algorithm>
#include <map>
#include <boost/foreach.hpp>
#include <set>
// betl includes -----------------------------------------------------------
#include <traits/geometrytraits.hpp>
#include <physics/plane_wave_helmholtz.hpp>
#include "physics/planewave_laplace.hpp"
#include "physics/uinc_wrapper.hpp"
#include <enumerators/enumerators.hpp>
#include <interpolation/point_evaluator.hpp>
#include <sparse_operators/identity_operator.hpp>
#include <linalg/vector.hpp>
#include "traits/lmt_traits.hpp"

namespace betl {
  namespace lmt {
    template< enum FUNDSOL FS=LAPLACE>
    class RhsGenerator {
    private:
      typedef typename FundSolOutTraits<FS>::FundSolVarType  numeric_t;
      typedef linalg::Vector< numeric_t > vector_t;
      typedef GeometryTraits::Point3d point3d_t;
      typedef std::set<std::size_t> index_set_t;
      typedef index_set_t::const_iterator iter_t;

    public:
      template< typename SD_HANDLER_T,
                typename BC_SD_HANDLER_T,
                typename LMT_SETTINGS_T>
      static void Create( const BC_SD_HANDLER_T& sd_handler,
                          const SD_HANDLER_T&    full_sd_handler,
```

```
                          const index_set_t&     dirichlet_bc_id,
                          const LMT_SETTINGS_T&  settings,
                          vector_t&  rhs,
                          vector_t&  dirichlet_data,
                          vector_t&  neumann_data );
};

// -T_0 u_inc
template< enum FUNDSOL FS>
template< typename SD_HANDLER_T,
          typename BC_SD_HANDLER_T,
          typename LMT_SETTINGS_T>
void RhsGenerator<FS>::Create( const BC_SD_HANDLER_T& sd_handler,
                               const SD_HANDLER_T&    full_sd_handler,
                               const index_set_t&     dirichlet_bc_id,
                               const LMT_SETTINGS_T&  settings,
                               vector_t&  rhs,
                               vector_t&  dirichlet_data,
                               vector_t&  neumann_data )
{
  typedef typename SD_HANDLER_T::dirichlet_dofhandler_t
      dirichlet_dofhandler_t;
    typedef typename SD_HANDLER_T::neumann_dofhandler_t
      neumann_dofhandler_t;

  const std::size_t blocksize = sd_handler.give_blocksize();
    std::size_t offset = 0;
  for (std::size_t i = 0; i < blocksize ; ++i) {
    const dirichlet_dofhandler_t& d_dofh =
      sd_handler.giveDirichletDofHandler(i);
    const neumann_dofhandler_t& n_dofh =
      sd_handler.giveNeumannDofHandler(i);
    const std::size_t sizeD = d_dofh.GiveNoOfDofs();
    const std::size_t sizeN = n_dofh.GiveNoOfDofs();
    if ( i == blocksize-1 ) { // create plane wave on \partial Omega_0
      const dirichlet_dofhandler_t& d_dofh_full =
      full_sd_handler.giveDirichletDofHandler(i);
      const std::size_t sizeDf = d_dofh_full.GiveNoOfDofs();

      typedef typename dirichlet_dofhandler_t::egd_type d_egd_t;
      typedef typename neumann_dofhandler_t::egd_type   n_egd_t;
      // uinc function wrappers (read function type from settings parser)
      typedef physics::UincWrapper<DIRICHLET, d_egd_t, FS> d_func_t;
      typedef physics::UincWrapper<NEUMANN, n_egd_t, FS>   n_func_t;
      d_func_t d_func;
      d_func.init(settings);
      n_func_t n_func;
      n_func.init(settings);
      // create two instances of plane waves
      typedef point_evaluator< d_func_t > d_df_t;
      typedef point_evaluator< n_func_t > n_df_t;
      // create functors to spread plane waves over the hull
      d_df_t d_df;
      n_df_t n_df;
```

```
          // compute the coefficients
          d_df( d_func, d_dofh_full );
          n_df( n_func, n_dofh );
          // get pointers to coefficients
          typename d_df_t::ptr_t uD = d_df.give_data();
          typename n_df_t::ptr_t uN = n_df.give_data();

          dirichlet_data = vector_t(sizeDf,0.0);
          neumann_data   = vector_t(sizeN,0.0);

          std::copy(uD,uD + sizeDf, dirichlet_data.give_data() );
          std::copy(uN,uN + sizeN , neumann_data  .give_data() );

          typedef IdentityOperator< neumann_dofhandler_t, dirichlet_dofhandler_t >
            id_op_nd_t;
          typedef IdentityOperator< dirichlet_dofhandler_t, neumann_dofhandler_t >
            id_op_dn_t;

          id_op_nd_t id_op_nd(n_dofh, d_dofh_full);
          id_op_nd.compute();
          id_op_dn_t id_op_dn(d_dofh, n_dofh);
          id_op_dn.compute();

          typename id_op_nd_t::const_reference M_ND = id_op_nd.giveMatrix( );
          typename id_op_dn_t::const_reference M_DN = id_op_dn.giveMatrix( );

          const numeric_t mone(-1.0 );

          M_ND.amux( mone, uD, rhs.give_data() + offset       );
          M_DN.amux( mone, uN, rhs.give_data() + offset+sizeN );
        } else {
          iter_t it = dirichlet_bc_id.find(i);
          if(it != dirichlet_bc_id.end()) {
            // dirichlet boundary conditions on this subdomain
            offset += sizeN;
          } else {
            offset += sizeD+sizeN;
          }
        }
      }
    }
  } // end namespace lmt
} // end namespace betl

#endif /* __RHS_GENERATOR_H__ */
```

E.4.1 Dirichlet boundary conditions

```
#ifndef BETL_LOCAL_MULTI_TRACE__BC_RHS_GENERATOR_H__
#define BETL_LOCAL_MULTI_TRACE__BC_RHS_GENERATOR_H__

// system includes ------------------------------------------------------
```

```
#include <vector>
#include <algorithm>
#include <set>
#include <map>
#include <boost/foreach.hpp>
// betl includes ------------------------------------------------------------
#include <traits/geometrytraits.hpp>
#include <interpolation/point_evaluator.hpp>
#include <sparse_operators/identity_operator.hpp>
#include <linalg/vector.hpp>
#include <linalg/linalg.hpp>
#include <enumerators/enumerators.hpp>
// own includes -------------------------------------------------------------
#include "subdomain_handler.hpp"
#include "lmt_config.hpp"
#include "rhs_bem_operator_handler.hpp"

namespace betl {
  namespace lmt {
    class BCRhsGenerator {
    public:
      typedef typename config::numeric_t numeric_t;
    private:
      typedef linalg::Vector< numeric_t >           vector_t;
      typedef GeometryTraits::Point3d               point3d_t;
      typedef sparse::Matrix<double, sparse::GENERAL, sparse::CRS, sparse::Assign>
      sparse_matrix_t;
      typedef std::set<std::size_t> index_set_t;

      static const enum ACCELERATION ACC = config::ACC;
      static const enum FUNDSOL      FS  = config::FS;
      typedef typename config::element_t element_t;
      typedef SubdomainHandler<ACC,element_t>  sd_handler_t;

    public:
      template< typename FUNCTION_T,
                typename BC_BEM_HANDLER_T,
                typename SPARSE_OP_H_T,
                typename SETTINGS_PARSER_T,
                typename LMT_SETTINGS>
      static void Create( sd_handler_t&      sd_handler,
                          sd_handler_t&      nf_sd_handler,
                          SPARSE_OP_H_T&           sparse_op_h,
                          const SETTINGS_PARSER_T& settings_parser,
                          const LMT_SETTINGS&      lmt_settings,
                          FUNCTION_T&              g,
                          const BC_BEM_HANDLER_T&  bc_bem_handler,
                          const index_set_t&       dirichlet_domains,
                          vector_t&                rhs );
    };
    //          | 0.5 Id + K |
    //  rhs +=  |            | * g
    //          | -W         |
    template< typename FUNCTION_T,
```

```
              typename BC_BEM_HANDLER_T,
              typename SPARSE_OP_H_T,
              typename SETTINGS_PARSER_T,
              typename LMT_SETTINGS>
void BCRhsGenerator::Create( sd_handler_t&       sd_handler,
                             sd_handler_t&       nf_sd_handler,
                             SPARSE_OP_H_T&            sparse_op_h,
                             const SETTINGS_PARSER_T& settings_parser,
                             const LMT_SETTINGS&       lmt_settings,
                             FUNCTION_T&               g,
                             const BC_BEM_HANDLER_T&  bc_bem_handler,
                             const index_set_t&        dirichlet_domains,
                             vector_t&                 rhs)
{
  typedef typename sd_handler_t::dirichlet_dofhandler_t
    dirichlet_dofhandler_t;
  typedef typename sd_handler_t::neumann_dofhandler_t
    neumann_dofhandler_t;

  const std::size_t blocksize = sd_handler.give_blocksize();
  typedef point_evaluator<FUNCTION_T> g_df_t;
  numeric_t one(1.0);
  std::size_t offset = 0;
  for (std::size_t i = 0; i < blocksize ; ++i) {
    // get dofhandlers
    const neumann_dofhandler_t& n_dofh =
      sd_handler.giveNeumannDofHandler(i);
    const dirichlet_dofhandler_t& d_dofh =
      sd_handler.giveDirichletDofHandler(i);
    const std::size_t sizeD = d_dofh.GiveNoOfDofs();
    const std::size_t sizeN = n_dofh.GiveNoOfDofs();
    // identity operator on subdomain i
    typedef IdentityOperator<neumann_dofhandler_t,dirichlet_dofhandler_t>
      id_op_t;
    typedef typename id_op_t::reference ID_t;
    // boundary conditions?
    typedef typename index_set_t::const_iterator iter;
    iter it = dirichlet_domains.find(i);
    typedef typename BC_BEM_HANDLER_T::K_t K_t;
    if ( it != dirichlet_domains.end() ) {
      /* this subdomain has dirichlet boundary conditions */
      g_df_t g_df;
      g_df(g, d_dofh);
      // mass matrix
      id_op_t id_op(n_dofh,d_dofh);
      id_op.compute();
      ID_t ID = id_op.giveMatrix();
      // matrix K
      const K_t K = bc_bem_handler.giveK(i);
      (0.5*ID - K).amux(one, g_df.give_data(), rhs.give_data() + offset);
      // update offset
      offset += sizeN;
    } else {
      /* no boundary condition on this subdomain */
```

```
            // "degrees of freedom" of \Gamma_DIR
            const dirichlet_dofhandler_t& d_dofh_nonfree =
              nf_sd_handler.give_lin_cont_dh(i);
            const std::size_t sizeBCi = d_dofh_nonfree.GiveNoOfDofs();
117         if (sizeBCi > 0 ) {            // g enters RHS only if there is
                                           // an adjacent subdomain with
                                           // Dirichlet BC
              // create K, W
              std::vector<double> mu = lmt_settings.get_kappa();
122           RHSBemOperatorHandler rhs_bem_handler( sd_handler, nf_sd_handler, i,
                                                     mu[i] );
              rhs_bem_handler.compute( sparse_op_h, settings_parser );
              const typename RHSBemOperatorHandler::K_t& K = rhs_bem_handler.giveK();
              const typename RHSBemOperatorHandler::W_t& W = rhs_bem_handler.giveW();
127
              g_df_t g_df;
              // compute coefficients
              g_df(g, d_dofh_nonfree);
              // mass matrix
132           id_op_t id_op(n_dofh,d_dofh_nonfree);
              id_op.compute();
              ID_t ID = id_op.giveMatrix();
              const numeric_t mone(-1.0);
              (0.5*ID+K).amux(one, g_df.give_data(), rhs.give_data() + offset);
137           W.amux(mone, g_df.give_data(), rhs.give_data() + offset + sizeN);
            }
            // update offset
            offset += sizeD+sizeN;
          }
142     }
    }
  } // end namespace lmt
} // end namespace betl

147

#endif /* BETL_LOCAL_MULTI_TRACE__BC_RHS_GENERATOR_H__ */
```

This class imposes contributions from single nodes on the boundary of Γ_{DIR} to the load vector.

Listing E.21: exclusion_rhs

```
#ifndef __UPDATE_RHS_H__
2 #define __UPDATE_RHS_H__

#include "../lmt_config.hpp"
// betl includes ------------------------------------------------------------
#include <linalg/linalg.hpp>
7 #include <interpolation/point_evaluator.hpp>
#include <boost/foreach.hpp>
// system includes ----------------------------------------------------------
#include <set>

12 namespace betl {
```

```
namespace lmt {
  class exclusion_rhs {
  private:
    typedef typename config::numeric_t numeric_t;
    typedef linalg::Vector<numeric_t>  vector_t;
    typedef linalg::VectorWrapper<numeric_t> vector_wrapper_t;
    typedef std::set<std::size_t>      index_set_t;

  public:
    template< typename SYSTEM_MATRIX,
              typename EXCLUSION_OP_H,
              typename SD_HANDLER,
              typename FUNCTION>
    static void Create( const SYSTEM_MATRIX&  A,
                        const EXCLUSION_OP_H& ex_op_h,
                        const SD_HANDLER&     sd_handler,
                        const SD_HANDLER&     nf_sd_handler,
                        FUNCTION&             g,
                        const index_set_t&    dirichlet_bc_id,
                        vector_t&             rhs,
                        vector_t&             rhs_out )
    {
      const std::size_t blocksize = sd_handler.give_blocksize();

      typedef typename SD_HANDLER::dirichlet_dofhandler_t dirichlet_dh_t;
      typedef typename SD_HANDLER::neumann_dofhandler_t   neumann_dh_t;
      vector_t g_vec(0, 0.0);
      for (size_t i = 0; i < blocksize; ++i) {
        const dirichlet_dh_t& ddofh = sd_handler.giveDirichletDofHandler(i);
        const neumann_dh_t&   ndofh = sd_handler.giveNeumannDofHandler(i);
        const std::size_t sizeD = ddofh.GiveNoOfDofs();
        const std::size_t sizeN = ndofh.GiveNoOfDofs();
        if ( dirichlet_bc_id.find(i) == dirichlet_bc_id.end() ) {
          // here we have additional contributions to rhs_out
          point_evaluator< FUNCTION > g_df;
          g_df( g, ddofh );
          const index_set_t& non_free_dofs = ex_op_h.give_bc_nodes(i);
          vector_t vd(sizeD, numeric_t(0.0));
          vector_t vn(sizeN, numeric_t(0.0));
          const dirichlet_dh_t& nf_ddofh = nf_sd_handler.giveDirichletDofHandler(i);
          BOOST_FOREACH( size_t v, non_free_dofs) {
            vd[v] = *(g_df.give_data()+v);
          }
          // concatenate vectors
          g_vec =(g_vec.Size()==0) ? (vd|vn) : (g_vec | vd | vn);
        } else {
          // no dirichlet dofs on this domain at all
          vector_t vn(sizeN, numeric_t(0.0));
          g_vec = (g_vec.Size()==0)  ? vn : (g_vec | vn);
        }
      }
      typedef typename EXCLUSION_OP_H::exclusion_matrix_t EX_t;
      const EX_t& EX = ex_op_h.giveMatrix();
      const std::size_t size_out = (EX^'T').GiveRows();
```

```
        rhs_out.resize( size_out, numeric_t(0.0) );
        // rhs_out = EX^T (-A*g_vec + rhs)
        const numeric_t mone(-1.0);
        rhs_out += mone*(EX^'T')*A*g_vec;
        rhs_out += (EX^'T')*rhs;
      }
    };
  } // end namespace lmt
} // end namespace betl

#endif /* __UPDATE_RHS_H__ */
```

BEM operators

Assembling of BEM operators for the load vector.

Listing E.22: Bem operator handler header

```
#ifndef __RHS1_BEM_OPERATOR_H__
#define __RHS1_BEM_OPERATOR_H__

// own includes ------------------------------------------------------------
#include "../lmt_config.hpp"
#include "../subdomain_handler.hpp"

#include "hyper_matrix_wrapper.hpp"

// betl includes -----------------------------------------------------------
#include <bem_operator/bem_operator.hpp>
#include <integration/galerkinintegrator.hpp>
#include <kernel/galerkinkernel.hpp>

#include <fundsol/fundsol.hpp>
#include <fast_bem/fast_factory.hpp>

// system includes ---------------------------------------------------------
namespace betl {
  namespace lmt {
    /** @brief creates matrices K,W
     *         required for assembling the RHS
     *         in presence of Dirichlet BC
     */
    class RHSBemOperatorHandler {
    private:
      /** @name config */
      //@{
      static const enum FUNDSOL FS = config::FS;
      static const enum ACCELERATION ACC = config::ACC;
      static const enum PARALLEL PAR = config::PAR;
      typedef typename config::qr_t qr_t;
      //@}

      typedef SubdomainHandler<ACC, typename config::element_t>
      sd_handler_t;
```

```
    typedef SparseOperatorHandler<sd_handler_t> sp_operator_h_t;
  private:
    typedef fastbem::FastFactory< ACC > fast_factory_t;
    typedef typename fast_factory_t::settings_parser_t settings_parser_t;

    //@{
    typedef typename sd_handler_t::dofhandler_lin_cont_t dh_lin_cont_t;
    typedef typename sd_handler_t::dofhandler_const_t dh_const_t;
    typedef typename dh_const_t::febasis_t const_febasis_t;
    typedef typename dh_lin_cont_t::febasis_t lin_cont_febasis_t;
    typedef FundSol< FS, DLP> dlp_t;
    typedef GalerkinKernel< dlp_t,
                            const_febasis_t,
                            lin_cont_febasis_t> kernel_t;
    typedef GalerkinIntegrator< kernel_t, qr_t> integrator_t;
    // double layer matrix `K`
    typedef BemOperator<integrator_t, dh_const_t, dh_lin_cont_t, PAR, NON_SYMMETRIC>
    bemoperator_t;
    //@}
    // create `W` inside a separate class
    typedef HyperMatrixWrapper<sd_handler_t,FS> hyper_matrix_wrapper_t;
  public:
    typedef typename hyper_matrix_wrapper_t::matrix_type W_t;
    typedef typename bemoperator_t::matrix_type          K_t;
  public:
    RHSBemOperatorHandler( sd_handler_t& sd_h_Y,
                           sd_handler_t& sd_h_X,
                           size_t        domain_id,
                           double        alpha );

    ~RHSBemOperatorHandler()
    {
      if (K_op_ptr_ != NULL) delete K_op_ptr_;
    }

    void compute( sp_operator_h_t&         sp_op_h_Y,
                  const settings_parser_t& settings_parser);

    const W_t& giveW() const;
    const K_t& giveK() const;
  private:
    sd_handler_t& sd_h_Y_;
    sd_handler_t& sd_h_X_;
    // alpha in diffusion or kappa in Helmholtz case
    double       alpha_;
    size_t       domain_id_;
    hyper_matrix_wrapper_t hyper_matrix_wrapper_;
    // double layer matrix
    dlp_t dlp_;
    kernel_t kernel_;
    integrator_t integrator_;
    bemoperator_t* K_op_ptr_;
  };
```

```
  } // end namespace lmt
} // end namespace betl

#endif /* __RHS_BEM_OPERATOR_H__ */
```

Listing E.23: Bem operator handler code

```
#include "rhs_bem_operator_handler.hpp"
#include <integration/galerkinquadrature.hpp>
//#include <misc/functionals.hpp>
#include <misc/app.hpp>

namespace betl {
  namespace lmt {
    // ------------------------------------------------------------------
    RHSBemOperatorHandler::
    RHSBemOperatorHandler( sd_handler_t& sd_h_Y,
                           sd_handler_t& sd_h_X,
                           size_t      domain_id,
                           double      alpha) :
      sd_h_Y_(sd_h_Y),
      sd_h_X_(sd_h_X),
      alpha_(alpha),
      domain_id_(domain_id),
      hyper_matrix_wrapper_(alpha),
      /* in laplace mode alpha drops out in dlp! */
      dlp_( (FS==HELMHOLTZ) ? alpha : 1.0),
      kernel_(dlp_),
      integrator_(kernel_),
      K_op_ptr_(NULL)
    {
      /* empty */
    }

    // ------------------------------------------------------------------
    void RHSBemOperatorHandler::
    compute( sp_operator_h_t&         sp_op_h_Y,
             const settings_parser_t& settings_parser )
    {
      hyper_matrix_wrapper_.compute( sd_h_Y_,
                                     sd_h_X_,
                                     sp_op_h_Y,
                                     settings_parser,
                                     domain_id_);

      K_op_ptr_ = new bemoperator_t( integrator_,
                                     sd_h_Y_.give_const_dh(domain_id_),
                                     sd_h_X_.give_lin_cont_dh(domain_id_));
      K_op_ptr_->Setup( settings_parser );
      K_op_ptr_->compute( );
    }

    const RHSBemOperatorHandler::W_t&
    RHSBemOperatorHandler::giveW() const
```

```
    {
      return hyper_matrix_wrapper_.giveMatrix();
    }

    const RHSBemOperatorHandler::K_t&
    RHSBemOperatorHandler::giveK() const
    {
      return K_op_ptr_->giveMatrix();
    }
  } // end namespace lmt
} // end namespace betl
```

Listing E.24: HyperMatrixWrapper

```
#ifndef __HYPER_MATRIX_WRAPPER_H__
#define __HYPER_MATRIX_WRAPPER_H__

// betl includes ----------------------------------------------------------
#include <enumerators/enumerators.hpp>
#include <linalg/linalg.hpp>
#include <bem_operator/bem_operator.hpp>
#include <fundsol/fundsol.hpp>
#include <fast_bem/fast_factory.hpp>
#include <sparse_operators/vector_matrix_policy.hpp>
#include <sparse_operators/scalar_matrix_policy.hpp>
#include <sparse_operators/curl_operator.hpp>
#include <sparse_operators/embedding_operator.hpp>
#include <sparse_operators/normal_operator.hpp>
#include <sparse_operators/identity_operator.hpp>
#include <composite_matrices/laplace_hyper_matrix.hpp>
#include <composite_matrices/helmholtz_hyper_matrix.hpp>
#include <kernel/galerkinkernel.hpp>

// own includes -----------------------------------------------------------
#include "../lmt_config.hpp"
#include "../sparse_operator_handler.hpp"

namespace betl {
  namespace lmt {
    /**
     *
     *   @brief HyperMatrix 'Y' x 'X'
     *
     *   Sparse operators on Y are give as input, sparse operators on
     *   X are created inside class
     *
     */
    // ------------------------------------------------------------------------
    template < typename SUBDOMAIN_HANDLER,
               enum FUNDSOL FS>
    class HyperMatrixWrapper { };

    // ------------------------------------------------------------------------
    // LAPLACE
```

```
template< typename SUBDOMAIN_HANDLER>
class HyperMatrixWrapper<SUBDOMAIN_HANDLER, LAPLACE> {
private:
  /** @name matrix types */
  //@{
  typedef scalar_matrix_policy<double> smp;
  typedef typename smp::const_reference      sparse_matrix_t;
  typedef vector_matrix_policy<double> vmp;
  typedef typename vmp::array_t        sparse_matrix_array_t;
  //@}
  /** @name lmt config */
  //@{
  typedef typename config::qr_t    qr_t;
  static const enum ACCELERATION ACC = config::ACC;
  static const enum PARALLEL     PAR = config::PAR;
  //@}

  typedef fastbem::FastFactory<ACC> fast_factory_t;
  typedef SparseOperatorHandler<SUBDOMAIN_HANDLER> sparse_op_handler_t;
  typedef typename fast_factory_t::settings_parser_t settings_parser_t;

  /** @name dofhandler types */
  //@{
  typedef typename SUBDOMAIN_HANDLER::dofhandler_lin_cont_t
  dh_lin_cont_t;
  typedef typename SUBDOMAIN_HANDLER::dofhandler_lin_disc_t
  dh_lin_disc_t;
  typedef typename SUBDOMAIN_HANDLER::dofhandler_const_t
  dh_const_t;
  //@}
  /** @name types for single layer matrix */
  //@{
  typedef typename dh_const_t   ::febasis_t const_febasis_t;
  typedef FundSol< LAPLACE, SLP> slp_t;
  typedef GalerkinKernel< slp_t,
                          const_febasis_t,
                          const_febasis_t>
  kernel_t;
  typedef GalerkinIntegrator<kernel_t,qr_t>
  integrator_t;

  typedef BemOperator< integrator_t,
                       dh_const_t, dh_const_t,
                       PAR, NON_SYMMETRIC> slp_bemoperator_t;
  //@}
  typedef CurlOperator     < dh_lin_cont_t, dh_const_t    > curl_op_t;
  typedef laplace::hyper_matrix<typename slp_bemoperator_t::matrix_type>
  hyper_matrix_type;
public:
  typedef hyper_matrix_type matrix_type;
public:
  HyperMatrixWrapper( double alpha )
    : slp_(alpha), kernel_(slp_), integrator_(kernel_), alpha_(alpha)
  {/* empty */ }
```

```
    void compute( SUBDOMAIN_HANDLER&   sd_h_Y,
                  SUBDOMAIN_HANDLER&   sd_h_X,
                  sparse_op_handler_t& sp_op_h_Y,
                  const settings_parser_t&   settings_parser,
                  size_t domain_id );
    const hyper_matrix_type& giveMatrix() const  { return *W_ptr_; }
    ~HyperMatrixWrapper() { delete V_ptr_; delete CX_ptr_; delete W_ptr_; }
  private:
    // V
    slp_bemoperator_t* V_ptr_;
    kernel_t          kernel_;
    slp_t             slp_;
    integrator_t      integrator_;
    // W
    curl_op_t*         CX_ptr_;
    hyper_matrix_type* W_ptr_;
    // conductivity, permeability, permittivity
    double alpha_;
  };

  // ---------------------------------------------------------------------
  template< typename SUBDOMAIN_HANDLER>
  void HyperMatrixWrapper<SUBDOMAIN_HANDLER, LAPLACE>::
  compute( SUBDOMAIN_HANDLER&   sd_h_Y,
           SUBDOMAIN_HANDLER&   sd_h_X,
           sparse_op_handler_t& sp_op_h_Y,
           const settings_parser_t&   settings_parser,
           size_t domain_id)
  {
    dh_const_t&    dh_const_X = sd_h_X.give_const_dh(domain_id);
    dh_lin_cont_t& dh_lin_cont_X = sd_h_X.give_lin_cont_dh(domain_id);
    dh_const_t&    dh_const_Y = sd_h_Y.give_const_dh(domain_id);
    V_ptr_ = new slp_bemoperator_t( integrator_, dh_const_Y, dh_const_X );
    V_ptr_->Setup(settings_parser);
    V_ptr_->compute();

    CX_ptr_ = new curl_op_t(dh_lin_cont_X, dh_const_X);
    CX_ptr_->compute();

    const sparse_matrix_array_t& CX = CX_ptr_->giveMatrixArray();
    const sparse_matrix_array_t& CY = sp_op_h_Y.give_curl_op(domain_id).giveMatrixArray();

    W_ptr_ = new hyper_matrix_type( V_ptr_->giveMatrix(),
                                    CY, CX, alpha_);
  }

  // ---------------------------------------------------------------------
  // HELMHOLTZ
  template< typename SUBDOMAIN_HANDLER>
  class HyperMatrixWrapper<SUBDOMAIN_HANDLER, HELMHOLTZ> {
    HyperMatrixWrapper();
  private:
    typedef typename ComplexTraits<double>::complex_type complex_t;
    /** @name matrix types */
```

```
      //@{
      typedef scalar_matrix_policy<double> smp;
      typedef typename smp::const_reference      sparse_matrix_t;
152   typedef vector_matrix_policy<double> vmp;
      typedef typename vmp::array_t        sparse_matrix_array_t;
      //@}
      /** @name lmt config */
      //@{
157   typedef typename config::qr_t    qr_t;
      static const enum ACCELERATION ACC = config::ACC;
      static const enum PARALLEL     PAR = config::PAR;
      //@}

162   typedef fastbem::FastFactory< ACC > fast_factory_t;
      typedef SparseOperatorHandler<SUBDOMAIN_HANDLER> sparse_op_handler_t;
      typedef typename fast_factory_t::settings_parser_t settings_parser_t;

      /** @name dofhandler types */
167   //@{
      typedef typename SUBDOMAIN_HANDLER::dofhandler_lin_cont_t
      dh_lin_cont_t;
      typedef typename SUBDOMAIN_HANDLER::dofhandler_lin_disc_t
      dh_lin_disc_t;
172   typedef typename SUBDOMAIN_HANDLER::dofhandler_const_t
      dh_const_t;
      //@}
      /** @name types for single layer matrix */
      //@{
177   typedef typename dh_lin_disc_t   ::febasis_t lin_disc_febasis_t;
      typedef FundSol< HELMHOLTZ, SLP> slp_t;
      typedef GalerkinKernel< slp_t,
                              lin_disc_febasis_t,
                              lin_disc_febasis_t>
182   kernel_t;
      typedef GalerkinIntegrator<kernel_t,qr_t>
      integrator_t;

      typedef BemOperator< integrator_t,
187                        dh_lin_disc_t, dh_lin_disc_t,
                           PAR, NON_SYMMETRIC> slp_bemoperator_t;
      //@}
      typedef CurlOperator     < dh_lin_cont_t, dh_const_t    > curl_op_t;
      typedef EmbeddingOperator< dh_const_t   , dh_lin_disc_t > lin2const_op_t;
192   typedef EmbeddingOperator< dh_lin_cont_t, dh_lin_disc_t > lin2lin_op_t;
      typedef NormalOperator   < dh_lin_disc_t, dh_lin_disc_t > normal_op_t;

      typedef helmholtz::hyper_matrix<typename slp_bemoperator_t::matrix_type>
      hyper_matrix_type;
197 public:
      typedef  hyper_matrix_type matrix_type;
    public:
      HyperMatrixWrapper(double kappa)
        : kappa_(kappa), slp_(kappa), kernel_(slp_), integrator_(kernel_) { }
202   void compute( SUBDOMAIN_HANDLER&   sd_h_Y,
```

```
                    SUBDOMAIN_HANDLER&   sd_h_X,
                    sparse_op_handler_t& sp_op_h_Y,
                    const settings_parser_t&   settings_parser,
                    size_t domain_id);
207   const matrix_type& giveMatrix() const {return *W_ptr_; }
      ~HyperMatrixWrapper() {
        delete CX_ptr_;
        delete L2C_X_ptr_;
        delete L2L_X_ptr_;
212     delete N_X_ptr_;
        delete W_ptr_;}
    private:
      // V
      slp_bemoperator_t* V_ptr_;
217   kernel_t           kernel_;
      slp_t              slp_;
      integrator_t       integrator_;
      // W
      curl_op_t*      CX_ptr_;
222   lin2const_op_t* L2C_X_ptr_;
      lin2lin_op_t*   L2L_X_ptr_;
      normal_op_t*    N_X_ptr_;
      // wavenumber
      double          kappa_;
227   hyper_matrix_type* W_ptr_;
    };

    // ------------------------------------------------------------------------
    template< typename SUBDOMAIN_HANDLER>
232 void HyperMatrixWrapper<SUBDOMAIN_HANDLER, HELMHOLTZ>::
    compute( SUBDOMAIN_HANDLER&   sd_h_Y,
             SUBDOMAIN_HANDLER&   sd_h_X,
             sparse_op_handler_t& sp_op_h_Y,
             const settings_parser_t&   settings_parser,
237          size_t domain_id)
    {
      dh_const_t&    dh_const_X = sd_h_X.give_const_dh(domain_id);
      dh_lin_cont_t& dh_lin_cont_X = sd_h_X.give_lin_cont_dh(domain_id);
      dh_lin_disc_t& dh_lin_disc_X = sd_h_X.give_lin_disc_dh(domain_id);
242   dh_lin_disc_t& dh_lin_disc_Y = sd_h_Y.give_lin_disc_dh(domain_id);
      // 'V'
      V_ptr_ = new slp_bemoperator_t( integrator_, dh_lin_disc_Y, dh_lin_disc_X);
      V_ptr_->Setup( settings_parser );
      V_ptr_->compute();
247   // 'W'
      // prepare sparse operators
      // curl
      CX_ptr_ = new curl_op_t(dh_lin_cont_X, dh_const_X);
      CX_ptr_->compute();
252   // dh const , dh_lin_disc
      L2C_X_ptr_ = new lin2const_op_t(dh_const_X, dh_lin_disc_X);
      L2C_X_ptr_->compute();
      // lin 2 lin
      L2L_X_ptr_ = new lin2lin_op_t(dh_lin_cont_X, dh_lin_disc_X);
```

```
257         L2L_X_ptr_->compute();
            // normal
            N_X_ptr_ = new normal_op_t( dh_lin_disc_X, dh_lin_disc_X);
            N_X_ptr_->compute();
            // sparse matrices living on 'X'
262         const sparse_matrix_array_t& CX = CX_ptr_->giveMatrixArray();
            const sparse_matrix_array_t& NX = N_X_ptr_->giveMatrixArray();
            const sparse_matrix_t        L2C_X = L2C_X_ptr_->giveMatrix();
            const sparse_matrix_t        L2L_X = L2L_X_ptr_->giveMatrix();
            // sparse matrices living on 'Y'
267         const sparse_matrix_array_t& CY = sp_op_h_Y.give_curl_op(domain_id).giveMatrixAr
            const sparse_matrix_array_t& NY = sp_op_h_Y.give_normal_op(domain_id).giveMatrix
            const sparse_matrix_t       L2C_Y = sp_op_h_Y.give_lin2const_op(domain_id).giveM
            const sparse_matrix_t       L2L_Y = sp_op_h_Y.give_lin2lin_op(domain_id).giveMat
            sp_op_h_Y.give_lin2lin_op(domain_id).giveMatrix();
272         const complex_t k(0.0, kappa_);
            // create hyper matrix
            W_ptr_ = new hyper_matrix_type( V_ptr_->giveMatrix(),
                                            CY,
                                            CX,
277                                         NY,
                                            NX,
                                            L2C_Y,
                                            L2C_X,
                                            L2L_Y,
282                                         L2L_X,
                                            k);
        }
      } // end namespace lmt
    } // end namespace betl
287
    #endif /* __HYPER_MATRIX_WRAPPER_H__ */
```

E.5 Mesh generation

The following python code generates the input file for Gmsh and creates the subdomain file needed by BETL for the unit cube containing 8 subdomains.

Listing E.25: Cube8.py

```
    from pylab import *
2   # custom modules

    import utility
    import entities

7   dreload(utility)
    dreload(entities)

    points=[]
    center_offset=array([0.5,0.5,0.5])
12
```

```
for ix in range(3):
    for iy in range(3):
        for iz in range(3):
            points.append(
                array([ix*0.5,iy*0.5,iz*0.5])-center_offset
                )
get_index = utility.get_index
# creation of lines
lines = entities.Lines()
for ix in range(3):
    for iy in range(3):
        for iz in range(3):
            if (ix < 2):
                p1 = utility.get_index(ix  ,iy, iz)
                p2 = utility.get_index(ix+1,iy, iz)
                lines.insert((p1,p2))
            if (iy < 2):
                p1 = utility.get_index(ix, iy,   iz)
                p2 = utility.get_index(ix, iy+1, iz)
                lines.insert((p1,p2))
            if (iz < 2):
                p1 = utility.get_index(ix, iy, iz  )
                p2 = utility.get_index(ix, iy, iz+1)
                lines.insert((p1,p2))

# create cubes
cubes={}
for ix in range(2):
    for iy in range(2):
        for iz in range(2):
            cubes[(ix,iy,iz)] = entities.Cube((ix,iy,iz))

## initialize neighbors
for c in cubes:
    ix,iy,iz=c
    neighbors={}
    neighbors['top'  ] = cubes.get( (ix,iy,iz+1) )
    neighbors['down' ] = cubes.get( (ix,iy,iz-1) )
    neighbors['east' ] = cubes.get( (ix+1,iy,iz) )
    neighbors['west' ] = cubes.get( (ix-1,iy,iz) )
    neighbors['north'] = cubes.get( (ix,iy+1,iz) )
    neighbors['south'] = cubes.get( (ix,iy-1,iz) )
    cubes[c].set_neighbors(neighbors)

# create surfaces
## xy-plane
surfaces = []
for iz in range(3):
    for xo in range(2):
        for yo in range(2):
            p1 = get_index(0+xo,0+yo, iz)
            p2 = get_index(1+xo,0+yo, iz)
            seg1 = lines.get( (p1,p2) )
            p1 = p2
```

```
            p2 = get_index(1+xo,1+yo, iz)
            seg2 = lines.get( (p1,p2) )
            p1 = p2
            p2 = get_index(xo, 1+yo, iz)
            seg3 = lines.get( (p1,p2) )
            p1 = p2
            p2 = get_index(xo,yo,iz)
            seg4 = lines.get( (p1,p2) )

            surf=entities.Surface([seg1,seg2,seg3,seg4], len(surfaces))
            neighbors={}
            neighbors['left' ] = cubes.get( (xo,yo,iz-1) )
            neighbors['right'] = cubes.get( (xo,yo,iz) )
            surf.set_neighbors(neighbors)
            surfaces.append(surf )

## zx-plane
for iy in range(3):
    for zo in range(2):
        for xo in range(2):
            p1 = get_index(0+xo,iy, zo)
            p2 = get_index(0+xo,iy, zo+1)
            seg1 = lines.get( (p1,p2) )
            p1 = p2
            p2 = get_index(1+xo, iy, zo+1)
            seg2 = lines.get( (p1,p2) )
            p1 = p2
            p2 = get_index(1+xo, iy, zo)
            seg3 = lines.get( (p1,p2) )
            p1 = p2
            p2 = get_index(xo,iy,zo)
            seg4 = lines.get( (p1,p2) )

            surf=entities.Surface([seg1,seg2,seg3,seg4], len(surfaces))
            neighbors={}
            neighbors['left' ] = cubes.get( (xo,iy-1,zo) )
            neighbors['right'] = cubes.get( (xo,iy  ,zo) )
            surf.set_neighbors(neighbors)
            surfaces.append(surf)

## yz-plane
for ix in range(3):
    for yo in range(2):
        for zo in range(2):
            p1 = get_index(ix,yo, zo)
            p2 = get_index(ix,yo+1, zo)
            seg1 = lines.get( (p1,p2) )
            p1 = p2
            p2 = get_index(ix, yo+1, zo+1)
            seg2 = lines.get( (p1,p2) )
            p1 = p2
            p2 = get_index(ix, yo, zo+1)
            seg3 = lines.get( (p1,p2) )
            p1 = p2
```

```
            p2 = get_index(ix,yo,zo)
            seg4 = lines.get( (p1,p2) )

            surf=entities.Surface([seg1,seg2,seg3,seg4], len(surfaces))
            neighbors={}
            neighbors['left' ] = cubes.get( (ix-1,yo,zo) )
            neighbors['right'] = cubes.get( (ix,yo,zo) )
            surf.set_neighbors(neighbors)
            surfaces.append(surf)

utility.set_orientation(surfaces)
utility.set_faces(surfaces)

print 'cl1=1;'
# PRINT POINTS
for i,p in enumerate(points):
    print 'Point(%d) = ' % (i+1),
    print '{', ', '.join(map(str,p)),',1};'
# PRINT LINES
utility.print_lines(lines)
# Print LINELOOPS
for i,s in enumerate(surfaces):
    print 'Line Loop(%d) = ' % (i+1),
    print '{', ','.join(map(str,s.lines)), '};'

for s in surfaces:
    print s


# print cubes and faces
print '*** cubes and faces (sdm file) *** '
print len(cubes)
for c in cubes.values():
    print c.id, ' ',
    for s in c.faces:
        print '  ', s.surface_id+1, ' ',
    print ''
```

Listing E.26: entities.py

```
from utility import *

class Lines(object):
    """
    """
    counter = 0
    def __init__(self, ):
        """
        """
        self.dict  = {}
        self.lines = []

    def get(self,t):
        p1,p2 = t
```

```
        if p1 < p2:
            return self.dict.get((p1,p2))
        else:
            val=(self.dict.get((p2,p1)))
            if val is None:
                return None
            else:
                return -1*val

    def get_endpoints(self,idx):
        return self.lines[idx]

    def insert(self,t):
        """
        """
        p1,p2 = t
        counter=len(self.dict)+1
        if p1 < p2:
            self.dict[(p1,p2)] = counter
            self.lines.append((p1,p2))
            Lines.counter += 1
        else:
            self.dict[(p2,p1)] = counter
            self.lines.append((p1,p2))
            Lines.counter += 1

    def __iter__(self):
        return self.dict.iteritems()

    def __len__(self):
        """
        """
        return len(self.dict)

class Cube(object):
    """
    """
    def __init__(self, origin):
        """
        Arguments:
        - `origin`: (ix,iy,iz)
        """
        self.__origin = origin
        ix,iy,iz=origin
        self.id     = 4*ix+2*iy+iz
        self.__neighbors_set = False

        self.neighbors = {} # 'top', 'down', 'east', 'west', ...
        self.faces = []
    def set_neighbors(self,neighbors):
        if not self.__neighbors_set:
            self.neighbors = neighbors
            self.__neighbors_set = True
        else:
```

```
            print 'neighbors are already set'

    def __str__(self):
        return 'Cube (%d,%d,%d)' % self.__origin


class Surface(object):
    """
    """
    counter = 0
    def __init__(self, lines, surface_id ):
        """
        """
        self.lines       = lines
        self.surface_id  = surface_id
        self.orientation = 1
        # keys: 'left','right', values: Cube objects
        self.neighbors={}
        Surface.counter += 1

    def set_neighbors(self, neighbors):
        self.neighbors=neighbors

    def __str__(self):
        """
        Arguments:
        - `self`:
        """
        surf_id = (self.surface_id+1)
        tmp= 'Plane Surface(%d) = {' % surf_id
        tmp += str((surf_id)*self.orientation)
        tmp+= '};\n'
        tmp += 'Physical Surface(%d) = {%d};' % (surf_id,surf_id)
        return tmp



if __name__ == '__main__':
    print 'to some testing here'
```

Listing E.27: utility.py

```
def get_index(i,j,k):
    return i*9+j*3+k

# def get_index(tuple):
#     return get_index(tuple[0],tuple[1],tuple[2])

def create_line_loops(lines,ix,iy,iz):
    """
    Arguments:
    - `lines`: dictionary (p1,p2): line_id
    - `ix`: origin indices
    - `iy`:
```

```
    - 'iz':
    """
    loops=[]
    #xz plane
    for cy in range(2):
        p1 = get_index(ix , iy+cy , iz )
        p2 = get_index(ix+1,iy+cy , iz )
        l1 = lines.get((p1,p2))
        p1 = p2
        p2 = get_index(ix+1,iy+cy ,iz+1)
        l2 = lines.get((p1,p2))
        p1 = p2
        p2 = get_index(ix , iy+cy ,iz+1)
        l3 = lines.get((p1,p2))
        p1 = p2
        p2 = get_index(ix , iy+cy ,iz  )
        l4 = lines.get((p1,p2))
        loops.append((l1,l2,l3,l4))

    #xy plane
    for cz in range(2):
        p1 = get_index(ix  , iy  , iz+cz )
        p2 = get_index(ix+1, iy  , iz+cz )
        l1 = lines.get((p1,p2))
        p1 = p2
        p2 = get_index(ix+1, iy+1, iz+cz )
        l2 = lines.get((p1,p2))
        p1 = p2
        p2 = get_index(ix  , iy+1, iz+cz )
        l3 = lines.get((p1,p2))
        p1 = p2
        p2 = get_index(ix  , iy  , iz+cz )
        l4 = lines.get((p1,p2))
        loops.append((l1,l2,l3,l4))

    #zy plane
    for cx in range(2):
        p1 = get_index(ix+cx, iy  , iz  )
        p2 = get_index(ix+cx, iy  , iz+1)
        l1 = lines.get((p1,p2))
        p1 = p2
        p2 = get_index(ix+cx, iy+1, iz+1)
        l2 = lines.get((p1,p2))
        p1 = p2
        p2 = get_index(ix+cx, iy+1, iz  )
        l3 = lines.get((p1,p2))
        p1 = p2
        p2 = get_index(ix+cx, iy  , iz  )
        l4=lines.get((p1,p2))
        loops.append((l1,l2,l3,l4))

    return loops
```

```
def set_orientation(surfaces):
    for s in surfaces:
        if s.neighbors['left'] is None:
            s.orientation = -1
        elif s.neighbors['right'] is None:
            pass
        else:
            if s.neighbors['left'].id < s.neighbors['right'].id:
                s.orientation = -1
            else:
                pass

def set_faces(surfaces):
    """
    """
    for s in surfaces:
        try:
            s.neighbors['left'].faces.append(s)
        except:
            pass
        try:
            s.neighbors['right'].faces.append(s)
        except:
            pass



def print_lines(lines):
    """
    """
    for it in lines.dict.items():
        points,index=it
        p1,p2=points
        print 'Line(%d) = {' % (index),
        print '%d,%d' % (p1+1,p2+1),
        print '};'


if __name__ == '__main__':
    print 'do some testing here'
```

Bibliography

[1] MAT file I/O library. http://sourceforge.net/projects/matio/.

[2] X. Claeys, R. Hiptmair, and C. Jerez-Hanckes. Multi-trace boundary integral equations. Technical Report 2012-20, Seminar for Applied Mathematics, ETH Zürich, Switzerland, 2012.

[3] R. Hiptmair. Operator preconditioning. 52(5):699–706, September 2006.

[4] R. Hiptmair and C. Jerez-Hanckes. Multiple traces boundary integral formulation for helmholtz transmission problems. 37(1):39–91, 2012.

[5] R. Hiptmair and L. Kielhorn. BETL - a generic boundary element template library. Technical Report 2012-36, Seminar for Applied Mathematics, ETH Zürich, Switzerland, 2012.

[6] Olaf Steinbach. *Numerical Approximation Methods for Elliptic Boundary Value Problems: Finite and Boundary Elements*. Springer, 2008.