Term Project

Dual Mesh Calderon Preconditioning for Single Layer Boundary Integral Operator

Yulia Smirnova

31.07.2012

1. THEORETICAL BACKGROUND

The construction of the preconditioner is based on the following theorem [1]:

THEOREM: Let V_h and W_h be finite-dimensional subspaces of reflexive Banach spaces V, W such that

$$dim V_h = dim W_h = N. \tag{1}$$

Let $a \in L(V \times V, \mathbb{C})$, $b \in L(W \times W, \mathbb{C})$ be continuous sesquilinear forms that fulfill the inf-sup conditions

$$\sup_{v_h \in V_h} \frac{\mid a(u_h, v_h) \mid}{\parallel v_h \parallel_V} \geq c_A \parallel u_h \parallel_V, \qquad \forall u_h \in V_h, \tag{2}$$

$$\sup_{w_h \in W_h} \frac{\mid b(q_h, w_h) \mid}{\parallel w_h \parallel_W} \geq c_B \parallel q_h \parallel_W, \qquad \forall q_h \in W_h. \tag{3}$$

Let $d \in L(V \times W, \mathbb{C})$ be a continuous sesquilinear form that satisfies another inf-sup condition

$$\sup_{w_h \in W_h} \frac{\mid d(v_h, w_h) \mid}{\parallel w_h \parallel_W} \geq c_D \parallel v_h \parallel_V, \qquad \forall v_h \in V_h. \tag{4}$$

Then picking up bases $b_1, ..., b_N$ of V_h and $q_1, ..., q_N$ of W_h and introducing Galerkin-matrices

$$\mathbf{A} := (a(b_i, b_j))_{i,j=1}^N, \qquad \mathbf{D} := (d(b_i, q_j))_{i,j=1}^N, \qquad \mathbf{B} := (b(q_i, q_j))_{i,j=1}^N \tag{5}$$

we get

$$\kappa(\ \mathbf{D}^{-1}\mathbf{B}\mathbf{D}^{-T}\mathbf{A}) \leq \frac{\|\ a\ \|\|\ b\ \|\|\ d\ \|^2}{c_A c_B c_D^2} \tag{6}$$

where $\kappa(\cdot)$ stands for the spectral condition number of a square matrix.

In BEM we want to use this theorem in the following way: we chose the bilinear forms corresponding to single-layer and hyper-singular potentials as $a(u,v)$ and $b(q,w)$ correspondingly and standard L_2 scalar product as $d(v,w)$:

$$a(u,v) := \int_\Gamma \int_\Gamma \frac{1}{4\pi \mid \boldsymbol{x} - \boldsymbol{y} \mid} u(\boldsymbol{x}) v(\boldsymbol{y}) dS(\boldsymbol{x}, \boldsymbol{y}), \qquad u, v \in H^{-\frac{1}{2}}\Gamma \tag{7}$$

$$b(u,v) := \int_\Gamma \int_\Gamma \frac{1}{4\pi \mid \boldsymbol{x} - \boldsymbol{y} \mid} \mathbf{curl}_\Gamma u(\boldsymbol{x}) \mathbf{curl}_\Gamma v(\boldsymbol{y}) dS(\boldsymbol{x}, \boldsymbol{y}),\ u, v \in H^{-\frac{1}{2}}(curl_\Gamma, \Gamma) \tag{8}$$

$$d(u,v) := (u,v)_{L^2(\Gamma)} \tag{9}$$

In order to satisfy the condition (1) we use a dual mesh (based on the barycentric refinement). Then choosing as V_h piecewise constant functions on the original surface mesh and as W_h linear continuous functions on the dual mesh we make sure that this condition holds.

The preconditioner constructed in this way is called Calderon preconditioner.

2. IMPLEMENTATION

The construction of the preconditioner was implemented in the BETL library [2], that already has modules to compute single-layer and hyper-singular potentials, assemble mass-matrices and perform many other necessary BEM operations.

Our aim was to write a module that constructs Galerkin matrices $\mathbf{D}$ and $\mathbf{B}$ and then to create a preconditioner in the form, given by the theorem, i.e.

$$P^{-1} = \mathbf{D}^{-1}\mathbf{B}\mathbf{D}^{-T}, \tag{10}$$

where both matrices $\mathbf{D}$ and $\mathbf{B}$ are of the size $n \times n$, where n is a number of elements in the original mesh.

The strategy of the construction is the following:

1. construct a barycentric refinement;
2. assemble embedding matrices that establish weighted mappings between original and dual meshes and between barycentric refinement and dual mesh (see definitions in **2.1**);
3. assemble Galerkin matrix $\mathbf{B}$ for the hyper-singular potential;
4. assemble mass matrix $\mathbf{D}$;
5. arrange all the constructed matrices into the final preconditioner.

2.1 Construction of the barycentric refinement

We limit ourselves with 3-nodal triangular meshes.

The construction of the barycentric refinement is a simple geometrical task: given some original mesh we create new nodes at the center of mass of each element and at the centers of the edges of these elements. Afterwards using these new nodes original triangles are split into six new ones (see figure 1).

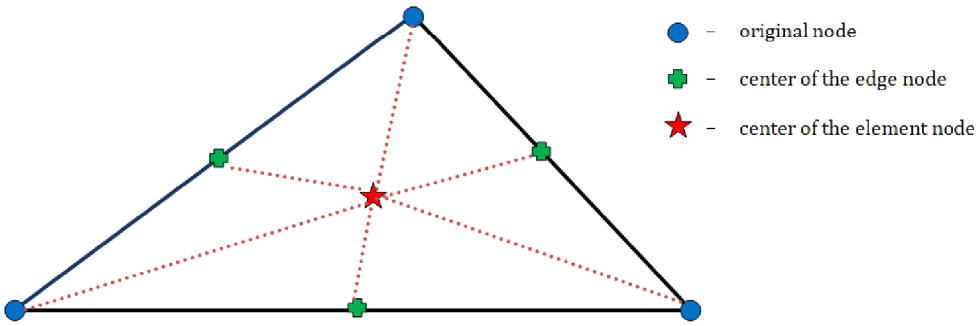


Figure 1: Scheme of the barycentric refinement

The advantage of the implemented barycentric refinement (in comparison with the circumscribed constructions) is that it can be created for any mesh and does not depend on the quality of the original elements.

The barycentric refinement is not a dual mesh. Dual mesh is a mesh with polygonal elements, whose nodes are the centers of the original triangles. (On the figure 2 an element of the barycentric refinement is depicted in blue and an element of the dual mesh is depicted in pink.)

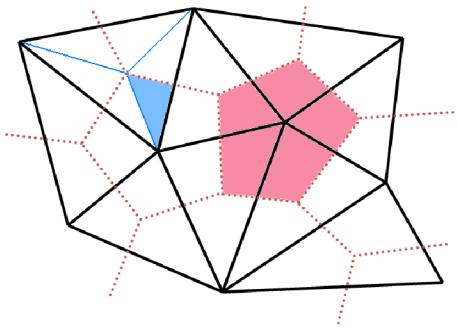


Figure 2: Barycentric refinement and dual mesh

The number of elements in the original triangular mesh equals the number of nodes in the dual mesh, thus if we chose as b_i – piecewise constant basis functions and as q_i – linear continuous basis functions the condition (1) will be satisfied.

However BEM cannot work with polygonal elements. This is the reason why we create the barycentric refinement and introduce embedding matrices that relate these two meshes.

2.2 Embedding matrices

Since we have three different meshes – original, barycentric refinement and dual – and we have different BE bases on these meshes we also need to establish some correspondence between these meshes/bases. This is done via embedding matrices.

The first embedding matrix – B_d – tells us how we want to combine the linear basis functions of the barycentric refinement $\{\psi_i\}_{i=1}^m$ (where m is a number of nodes in the barycentric refinement) to get linear basis functions of the dual mesh $\{\Phi_i\}_{i=1}^n$. This is our choice of basis for this particular problem.

The matrix is sparse and has the following non zero entries:

- one entry $B_d^{ij} = 1$ in the columns that correspond to the nodes of the original mesh,

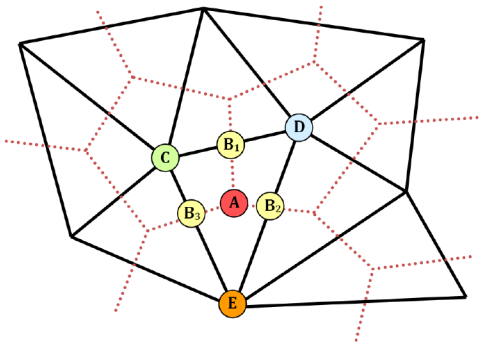


Figure 3: Linear continuous basis function of the dual mesh

- two entries $B_d^{ij} = \frac{1}{2}$ in the columns that correspond to the nodes in the centers of the edges of the original mesh,
- k entries $B_d^{ij} = \frac{1}{k}$ in the columns corresponding to the centers of mass of the elements, where k is the number of triangles in the original mesh that have a vertex at the given node.

$$B_d^T = \begin{pmatrix} \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \frac{1}{2} & \dots & \frac{1}{2} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \dots & \frac{1}{4} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & 1 & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix} \begin{matrix} \\ \leftarrow \textit{node at the center of the edge} \\ \\ \leftarrow \textit{node at the center of the element} \\ \\ \leftarrow \textit{node of the original mesh} \\ \end{matrix} \tag{11}$$

Thus in the case depicted on the figure 3 we have:

$$\Phi_A = \psi_A + \frac{1}{2}(\psi_{B_1} + \psi_{B_2} + \psi_{B_3}) + \frac{1}{k_C}\psi_C + \frac{1}{k_D}\psi_D + \frac{1}{k_E}\psi_E, \tag{12}$$

where in this particular case $k_C = k_D = 5$ and $k_E \geq 6$.

The second embedding matrix – B_v – transforms piecewise constant basis on the barycentric refinement $\{\varphi_i\}_{i=1}^{6n}$ into the the piecewise constant basis on the original mesh $\{\phi_i\}_{i=1}^{n}$. I.e. matrix $B_v \in \mathbb{R}^{6n \times n}$ and has six entries $B_v^{ij} = 1$ in each column.

$$B_v = \begin{pmatrix} \dots & \dots & \dots & \dots \\ \dots & 1 & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & 1 & \dots & \dots \\ \dots & 1 & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & 1 & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & 1 & \dots & \dots \\ \dots & 1 & \dots & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix} \begin{matrix} \\ \leftarrow \\ \\ \leftarrow \\ \leftarrow \\ \\ \leftarrow \\ \\ \leftarrow \\ \leftarrow \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \textit{rows with '1' correspond to} \\ \textit{six small triangles into which} \\ \textit{the original element was cut} \\ \\ \\ \\ \\ \end{matrix} \tag{13}$$

2.3 Galerkin matrix for the hyper-singular potential

Using the build-in BETL function we calculate the Galerkin matrix for the hyper-singular potential D_{hs} on the barycentric refinement using linear continuous functions $\{\psi_i\}_{i=1}^m$, where m is a number of nodes in the barycentric refinement. However the desired **B**–matrix is the Galerkin matrix for the hyper-singular potential on the dual mesh, evaluated using linear continuous functions $\{\Phi_i\}_{i=1}^n$.

The obtained $D_{hs} \in \mathbb{R}^{m\times m}$, hence we need to adjust this matrix to the desired size – $n \times n$. We do this using an embedding matrix $B_d \in \mathbb{R}^{n\times m}$, i.e. the desired **B** has the following form

$$\mathbf{B} = B_d D_{hs} B_d^T. \tag{14}$$

2.4 Mass matrix

The mass matrix **D** in our case should be the matrix of scalar products between piecewise constant functions of the original mesh $\{\phi_i\}_{i=1}^n$ – this is the basis on which we calculate the matrix V_{sl} for which we are building a preconditioner, and between linear continuous functions on the dual mesh $\{\Phi_i\}_{i=1}^n$, which are combinations of $\{\psi_i\}_{i=1}^m$ that are formed using the matrix B_d.

Of course we could have calculated this matrix straightforwardly, but due to the specifics of the BETL library it is easier to use another approach.

BETL allows to assemble mass matrix between any two bases on the same mesh and due to specific construction of the matrix D_{hs} we already have in place linear continuous and piecewise constant bases on the barycentric refinement, $\{\psi_i\}_{i=1}^m$ and $\{\varphi_i\}_{i=1}^{6n}$ correspondingly. Thus for these two bases we get a matrix M of the size $m \times 6n$ with $M_{ij} = \int_\Omega \psi_i \cdot \varphi_j dS$. And afterwards we transform it into the desired matrix $\mathbf{D}$ using embedding matrices B_d and B_v.

$$\begin{array}{ccccc} M & (& \psi_i & , & \varphi_j &) \\ & & \updownarrow B_d & & \updownarrow B_v & \\ \mathbf{D} & (& \Phi_i & , & \phi_j &) \end{array} \tag{15}$$

I.e. the desired $\mathbf{D}$ has the following form

$$\mathbf{D} = B_d M B_v. \tag{16}$$

2.5 Assembling matrices

Finally gathering the formulae (10), (14), (16) for matrices $\mathbf{D}$ and $\mathbf{B}$ together we obtain the following formula for the preconditioner

$$P^{-1} = (B_d M B_v)^{-1} (B_d D_{hs} B_d^T)(B_d M B_v)^{-T} \tag{17}$$

where

$M \in \mathbb{R}^{6n \times m}$ – is an auxiliary mass matrix between bases $\{\varphi_i\}_{i=1}^{6n}$ and $\{\psi_i\}_{i=1}^m$,

$B_d \in \mathbb{R}^{n \times m} : \{\Phi_i\}_{i=1}^n \to \{\psi_i\}_{i=1}^m$ – embedding matrix that specifies correspondence between linear bases on the barycentric refinement and dual mesh,

$B_v \in \mathbb{R}^{6n \times n} : \{\varphi_i\}_{i=1}^{6n} \to \{\phi_i\}_{i=1}^n$ – embedding matrix between elements of the original mesh and the barycentric refinement,

$D_{hs} \in \mathbb{R}^{m \times m}$ – hyper-singular potential matrix on the barycentric refinement.

2.6 Algorithmic concerns

2.6.1 Assembling matrices

Matrices D_{hs} and M are assembled using existing BETL functions.

Embedding matrices B_d, B_v are assembled using maps between elements and nodes of the three existing meshes. These matrices are sparse, as is the matrix M. This property is widely utilized to speed up the algorithm and to decrease memory usage. A special matrix product module for sparse matrices was implemented, allowing to perform matrix-matrix multiplication just once during the construction of the preconditioner.

2.6.2 Matrix Inversion

BETL architecture does not require explicit matrix storage and all matrix operations are carried out using a matrix-vector product routine ($amux$) that is implemented for all matrix-types.

That is why we did not explicitly compute inverse matrices using additional existing packages, but used an iterative solver to implement an $amux$ method for an inverse matrix. I.e. for each given vector $\boldsymbol{x}$ our routine calculates $\boldsymbol{y} = A^{-1}\boldsymbol{x}$ using GMRes algorithm. Hence we have created an implicit inverse matrix.

The termination criterion for the GMRes is:

- either tolerance is less then the given one, i.e. $\epsilon < 10^{-8}$;
- or the maximum number of iterations is achieved $N_{max} = 10,000$.

Using GMRes for matrix inversion proved to be very efficient in our case. The matrix that we are inverting – the mass matrix – is a regular sparse matrix. Moreover it is diagonally dominant, and thus Jacobi preconditioner improves the convergence of the method. As a result in all the models, that we have tested, the number of iterations needed for inversion was always $\approx 10 - 20$. And since this approach proved to be so efficient for our matrices, we decided to stay with it.

2.6.3 Problems with the matrix D_{hs}

The matrix D_{hs} is singular (it allows rigid body movement). Since the preconditioner should be non-singular, we have to correct this matrix. This is done via special gauger (already implemented in BETL). In short $rank D_{hs} = dim D_{hs} - k$, where k – is a number of connected components in the mesh. In order to make the rank of the matrix D_{hs} full, we sequentially form k

gauger vectors [2] and add k matrices in the form $v_i \cdot v_i^T$. Thus we make the following substitution:

$$D_{hs} \to D_{hs} + \sum_{i=1}^{k} v_i \cdot v_i^T \tag{18}$$

Finally we get a preconditioner that is symmetric (by construction), non-singular and positive-definite.

But assembling of the D_{hs} matrix is still the bottleneck of the calculation. Since $m \approx 3n$, for reasonable original mesh sizes $n \approx 300.000$ the size of the D_{hs} matrix becomes $m \approx 900.000$. Even with accelerators that are implemented in BETL this part of the code takes most of the time.

3. RESULTS

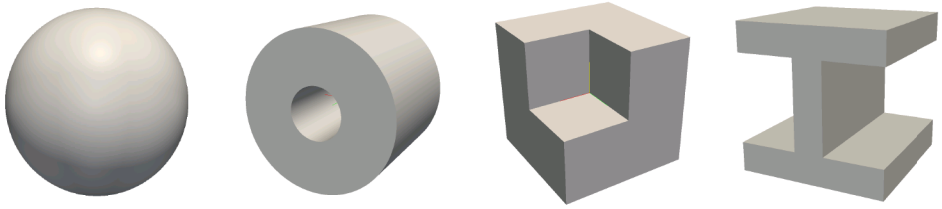
Originally the following system of equations is solved:

$$V_{sl}\boldsymbol{x} = \boldsymbol{b}. \tag{19}$$

After implementing the preconditioner the system changes to:

$$P^{-1}V_{sl}\boldsymbol{x} = \tilde{\boldsymbol{b}} = P^{-1}\boldsymbol{b}. \tag{20}$$

Our preconditioner should increase the convergence speed, i.e the number of iterations in iterative methods should significantly decrease. Moreover this number should stabilize and remain unchanged after reaching some mesh size. In order to check this we performed simulations for several basic geometries: a sphere, a hollow cylinder, 3D L-shape and an h-beam (see figure 4).



em (a) sphere (b) hollow cylinder (c) 3D L-shape (d) H-beam

Figure 4: Test geometry

Since we are not interested in physical meaning of the problem that we solve, we just let $\boldsymbol{b}$ be a vector of all ones, i.e. we are solving the following system of equations:

$$V_{sl}\boldsymbol{x} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}. \tag{21}$$

And we solve this system with two methods – conjugate gradient and GMRes. For both methods we use the same termination criterion as we did in matrix inversion, i.e.

- either tolerance is less then the given one, i.e. $\epsilon < 10^{-8}$;
- or the maximum number of iterations is achieved $N_{max} = 10,000$.

The results are presented in the tables 1–4 and in the figure 5.

# of elements	GMRes w/o PC	GMRes with PC	CG w/o PC	CG with PC
128	28	9	30	10
512	39	10	44	11
2048	51	10	58	11
8192	65	10	80	11
32768	80	10	106	11
131072	100	10	142	11
524288	125	10	191	11

Table 1: Results for the sphere

# of elements	GMRes w/o PC	GMRes with PC	CG w/o PC	CG with PC
1144	58	15	73	24
4576	76	15	99	22
18304	100	14	140	21
73216	127	13	189	19
292864	165	14	263	20

Table 2: Results for the cylinder

# of elements	GMRes w/o PC	GMRes with PC	CG w/o PC	CG with PC
662	51	24	60	32
2648	86	30	115	39
10592	146	36	238	48
42368	239	44	474	58
169472	602	64	1422	145
677888	951	73	2650	149

Table 3: Results for the L-shape

# of elements	GMRes w/o PC	GMRes with PC	CG w/o PC	CG with PC
3376	72	52	95	67
13504	117	62	178	80
54016	193	75	352	97
216064	384	86	885	118
864256	498	115	3218	132

Table 4: Results for the H-beam

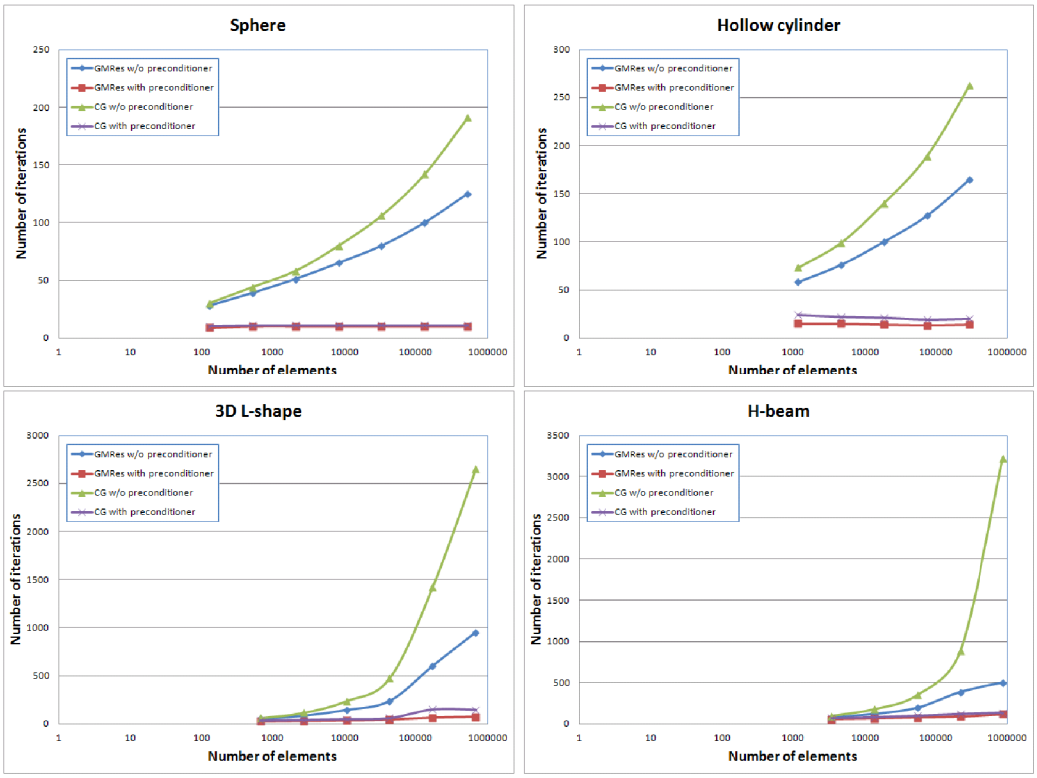


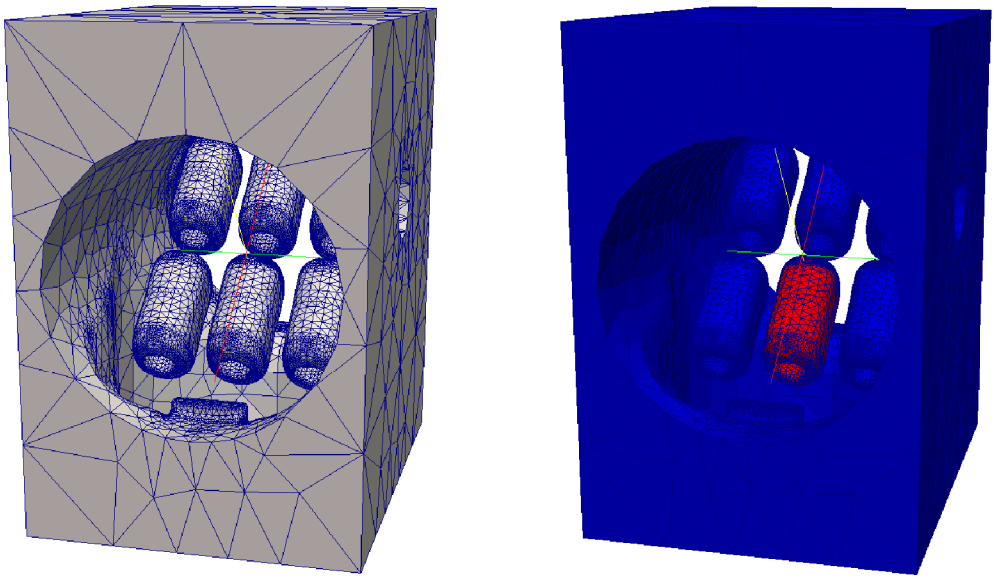
Figure 5: Test results

We can clearly see that for a sphere and for a cylinder the number of iterations for the case with preconditioner stabilizes almost immediatly and stays constant, when we increase the size of the mesh. For the other two geometries we don't see definite stabilization (judging from the plots we can safely assume that they will, however the computational resources are not available to make these tests). But the number of iterations with preconditioner is drastically small compared with the case with no preconditioner.

REMARK. Of course we controlled that the solutions in both cases are identical (up to a given precision).

Thus our preconditioner significantly speeds up the solution of the system.

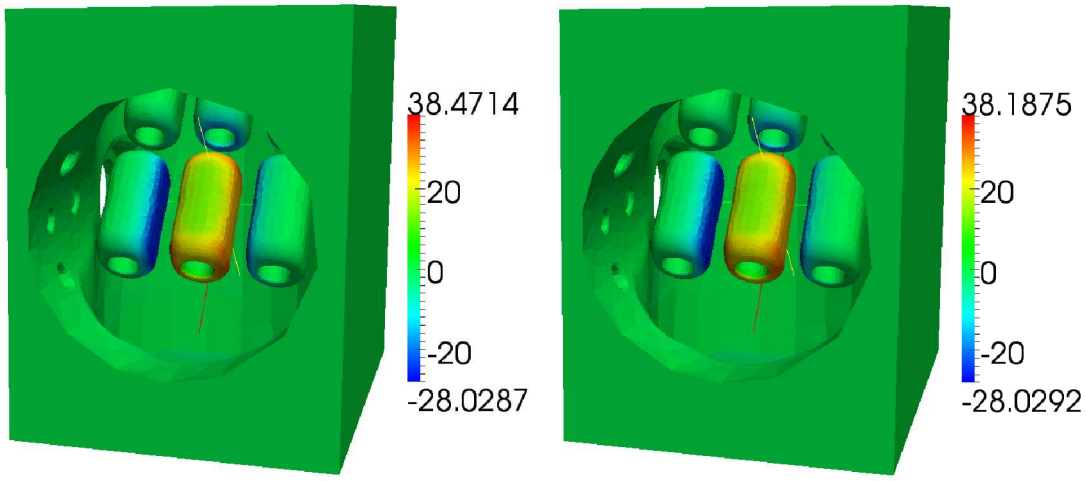
To make sure that our preconditioner does not somehow 'spoils' the solution, we ran one test on the realistic physical model. The model consists of six electrodes in a casing. One of the electrodes has non-zero potential, i.e. $V = 1$, the rest of the structure has free boundary (see figure 6). And we are interested in the distribution of the charge density.



(a) Mesh (b) BC: Potential

Figure 6: Test model "six electrodes"

Solutions of the problem with and without preconditioner are represented on the figure 7.



(a) Without preconditioner (b) With preconditioner

Figure 7: Test model "six electrodes": charge density distribution

The results correspond well: the qualitative pictures are identical, the difference in values is of the order $\approx 1e-6$.

But this is a rather specific model:

- the mesh is very uneven (with $A_{max}/A_{min} \approx 1000$);
- it has several separate components.

Thus the original number of iterations using CG method is very high. But with preconditioner it decreased enormously for both methods (see table 5).

And thus the calculation time also decreased, making the total time of creating the preconditioner and solving the preconditioned system less than the time of solving the system straightforwardly (see table 6). The difference in time is not so considerable, if we only make one calculation, i.e. have only one load-case. However if we would want to calculate several load-cases for one model, using a preconditioner would have been much more advantageous.

# of elements	GMRes w/o PC	GMRes with PC	CG w/o PC	CG with PC
50648	2708	63	9719	47

Table 5: Test model "six electrodes": number of iterations

# of elements	GMRes w/o PC	GMRes with PC	CG w/o PC	CG with PC
50648	1651	1591	2633	1631

Table 6: Test model "six electrodes": calculation time (in seconds)

APPENDIX: Description of C++ files

File: dual_mesh.hpp

Implements a new class *DualMesh*, which is inherited from the BETL *Mesh* class.

It is used to create a barycentric refinement of a given mesh.

Has no methods. The creation of the barycentric refinement is done in the class constructor:

```
DualMesh ( const PARENT_MESH_T& parent_mesh )
```

Given a mesh in the BETL mesh format the constructor uses internal methods to create all the necessary geometrical entities and to make a barycentric refinement in the same format, i.e. in the ordinary BETL mesh-format.

File: dual_preconditioner.hpp

Implements a new class *DualPreconditioner*.

It is used to assemble a preconditioner for a given mesh.

The class constructor has two arguments: original mesh and dofhandler of the piecewise constant basis functions on the original mesh.

```
DualPreconditioner( const parent_mesh_t& parent_mesh,
                    const dh_parent_const_t& v_orig).
```

Using these input variables the class creates all the necessary matrices and assembles them.

The class has an operator `()`:

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

For any given vector $\mathbf{x}$ it returns a vector $P^{-1}\mathbf{x}$, i.e. implements multiplication by the matrix P^{-1}.

And as preconditioner can be considered as a particular matrix type, it supports the standard BETL matrix functions, i.e.

- ```
  template <class T>
  void amux ( T* x, T* y, T alpha, T beta, char op) const
  ```

  Calculates standard BETL matrix-vector product:

  $P^{-1}.amux(x, y, \alpha, \beta, N) \rightarrow \alpha(P^{-1})^N x + \beta y,$
  ```

where N indicates, whether we want to use a transpose matrix;

- `size_t GiveCols( ) const`

  returns number of columns in P^{-1};

- `size_t GiveRows( ) const`

  returns number of rows in P^{-1};

File: dual_gauger.hpp

Implements a new class *HyperMatrixGauger*, which is inherited from the BETL *LaplaceGauger* class.

It is used to perform the correction of the D_{hs} matrix (see §2.6.3).

It supports the standard BETL matrix functions, i.e.

- ```
  template <class T>
  void amux ( T* x, T* y, T alpha, T beta, char op) const
  ```
- `size_t GiveCols( ) const`
- `size_t GiveRows( ) const`

File: inverse_matrix.hpp

Implements a new class *InverseMatrixWithPreconditioner*. This class inverts a given matrix, using a given preconditioner.

It is used to invert a matrix by means of the GMRes algorithm (see §2.6.2).

The class constructor has the following structure.

```
InverseMatrixWithPreconditioner(const MATRIX_T& A,
                                const PRECONDITIONER_T* P,
                                int GMRes_max_iter,
                                double GMRes_tolerance)
```

Since the result of the inversion is again a matrix, it supports the standard BETL matrix functions, i.e.

- ```
  template <class T>
  void amux ( T* x, T* y, T alpha, T beta, char op) const
  ```
- `size_t GiveCols( ) const`
- `size_t GiveRows( ) const`

The file also includes class specializations for the existing Jacobi preconditioner and for the case, when no preconditioner is used.

References

[1] R.Hiptmair, *Operator Preconditioning*. Computers and mathematics with Applications 52 (2006) 699-706.

[2] L.Kielhorn, *BETL Documentation*. SAM - Seminar for Applied Mathematics, ETH Zurich.