Tino Eibner

A fast and efficient algorithm to compute
BPX- and overlapping preconditioner for
adaptive 3D-FEM

CSC/08-02
A fast and efficient algorithm to compute
BPX- and overlapping preconditioner for
adaptive 3D-FEM

Abstract
In this paper we consider the well-known BPX-preconditioner in con-
junction with adaptive FEM. We present an algorithm which enables
us to compute the preconditioner with optimal complexity by a total of
only $O(\text{DoF})$ additional memory. Furthermore, we show how to com-
bine the BPX-preconditioner with an overlapping Additive-Schwarz-
preconditioner to obtain a preconditioner for finite element spaces with
arbitrary polynomial degree distributions. Numerical examples illus-
trate the efficiency of the algorithms.


1 Introduction

Adaptive FEM is one of the most powerful tools to solve boundary value problems. Whether we consider low order adaptive h-FEM [AO00, BS01, BR03, Ver96] or adaptive hp-FEM [AS97, AS98, AS99, Eib06, HS05, MW01], in both cases we obtain approximate solutions of high accuracy for a minimum of degrees of freedom. However, in order to solve the arising system of linear equations efficiently we have to apply good preconditioners. In the last twenty years a lot of preconditioners leading to optimal condition number $O(1)$ or almost optimal condition numbers have been developed, e.g. see [BPX91, EM07, HKK05, KL04, Nep86, TW05].

In this paper we will not develop a new preconditioner. Instead we consider the well-known BPX-preconditioner [BPX90, Bramble91, Bramble90] and present a fast and efficient algorithm to compute this preconditioner in the case of adaptive FEM. Moreover, since the applicability of the BPX-preconditioner is restricted to finite element spaces based on piecewise linear functions, we consider the combination of BPX- with an overlapping Additive-Schwarz-preconditioner (see [Osw94, EM07, Eib06, SMPZ08]) to overcome this restriction and obtain a preconditioner for adaptive FEM with arbitrary polynomial degree distribution.

For simplicity of exposition, we will restrict our attention to scalar reaction-diffusion equations in 3D and adaptive h-FEM on tetrahedral meshes without hanging nodes.

The key of our implementation is a properly chosen data-structure which enables us to compute the BPX-preconditioner with optimal complexity and a total of only $O(|\text{DoF}|)$ additional memory. To our knowledge, this is a new approach to compute the BPX-preconditioner for an adaptive FEM scenario and we want to emphasize that our implementation can easily be adapted to adaptive hp-FEM and other types of equations such as, for example, elasticity equations. Furthermore, our algorithm does not depend on the dimension of the domain, that is it can be applied to adaptive 2D-FEM on triangular meshes without changes. We also expect that our algorithm can be extended to hexahedral meshes as well as to meshes containing hanging nodes.

2 Model problem and FE-discretization

Let $\Omega \subset \mathbb{R}^3$ be a Lipschitz domain and assume that its boundary $\Gamma = \partial \Omega$ consists of two disjoint portions,

$$\Gamma = \Gamma_D \cup \Gamma_N,$$

where $\Gamma_D$ and $\Gamma_N$ consist of a finite number of disjoint parts of positive measure, or they are empty, respectively. Let $A = [a_{ij}]_{i,j=1}^3$ be a coefficient matrix and
a_0 a scalar valued function. Denote by \( f_r \) the right hand side function and by \( g_N \) the boundary conditions on \( \Gamma_N \). Then the classical formulation of our model problem reads as follows:

**Problem 2.1** (model problem - classical formulation). Find a solution \( u \) of

\[
-\nabla \cdot (A \nabla u) + a_0 u = f_r \quad \text{in } \Omega,
\]

\[
u = 0 \quad \text{on } \Gamma_D,
\]

\[
(A \nabla u, \eta) = g_N \quad \text{on } \Gamma_N.
\]

Since the more appropriated formulation of this problem is the so called weak formulation, we write

\[
H^1_D(\Omega) := \{ u \in H^1(\Omega) \mid u|_{\Gamma_D} = 0 \}
\]

and consider henceforward the Galerkin formulation of Problem 2.1.

**Problem 2.2** (model problem - weak formulation). Find \( u \in H^1_D(\Omega) \) such that

\[
\int_{\Omega} \nabla u \cdot A \nabla v + a_0 uv d\Omega = \int_{\Omega} f_r v d\Omega + \int_{\Gamma_N} g_N v d\Gamma \quad \forall v \in H^1_D(\Omega).
\]

Throughout the paper we demand \( A = A(x) \) to be uniformly symmetric positive definite for \( x \in \bar{\Omega} \). We demand \( a_{ij}, a_0 \in L^\infty(\Omega) \) with \( a_0 \geq a_0 > 0 \) in the case of \( \Gamma_D = \emptyset \) and \( a_0 \geq 0 \) otherwise. Finally we claim \( g_N \in L^2(\Gamma_N) \) and \( f_r \in L^2(\Omega) \).

Due to these assumptions the bilinear form \( a : H^1_D(\Omega) \times H^1_D(\Omega) \rightarrow \mathbb{R} \), given by

\[
a(u, v) := \int_{\Omega} \nabla u \cdot A \nabla v + a_0 uv d\Omega,
\]

is symmetric, continuous and coercive and the functional \( f : H^1_D(\Omega) \rightarrow \mathbb{R} \) with

\[
f(v) := \int_{\Omega} f_r v d\Omega + \int_{\Gamma_N} g_N v d\Gamma
\]

belongs to the dual space \([H^1_D(\Omega)]^*\). Thus the assumptions of Lax-Milgram [Bra07, Sch08] are satisfied which gives us the existence and uniqueness of a solution to Problem 2.2.

However, in order to solve Problem 2.2 numerically, we have to discretize it. To that end we cover the domain \( \Omega \) with an admissible shape-regular FE-mesh \( T \) consisting of tetrahedra (see, e.g. [Cha70]) and choose a polynomial degree \( p \). On the reference tetrahedron \( \hat{K} \) we denote with \( P_p(\hat{K}) \) the space of polynomials of

**Lemma 10.2.** Let \( \rho(E) \) be given by (5). Then for all \( u \in V \)

\[
a(Pu, u) \leq (1 + \rho(E))\|u\|^2_A.
\]

**Proof.** We observe that for all \( u \in V \) and \( i \in \{0, \ldots, K\} \)

\[
\|P_i u\|^2_A = a(P_i u, Pu) = a(u, Pu).
\]

Hence we have \( \|P_i u\|^2_A \leq \|u\|_A \|Pu\|_A \) which is

\[
\|Pu\|_A \leq \|u\|_A \quad \forall u \in V.
\]

Next we consider

\[
\|(P - P_h)u\|^2_A = \sum_{i,j=1}^K a(P_i u, P_j u) \leq \sum_{i,j=1}^K \kappa_{ij} \|P_i u\|_A \|P_j u\|_A.
\]

Since \( E = [\kappa_{ij}]_{i,j=1}^K \) is a symmetric matrix we obtain

\[
\|(P - P_h)u\|^2_A \leq \rho(E) \sum_{i=1}^K \|P_i u\|^2_A = \rho(E) \sum_{i=1}^K a(P_i u, P_h u) \leq \rho(E)\|u\|_A \|(P - P_h)u\|_A.
\]

Thus, we finally arrive at

\[
a(Pu, u) = a(P_h u, u) + a((P - P_h) u, u) \leq a(P_h u, Pu) + \|(P - P_h)u\|_A \|u\|_A \leq (1 + \rho(E))\|u\|^2_A.
\]

**Lemma 10.1** and **Lemma 10.2** together with (9) give he desired bound for the condition number of \( P \)

**Theorem 10.3.**

\[
\kappa(P) = \kappa(C^{-1}A) \leq C_0^2(1 + \rho(E)).
\]

**References**

required to reduce the initial residual by $C_{\text{res}}$. Finally the column “H1-Err” shows the error $\|u_{\text{FEM}} - u_{\text{exact}}\|_{H^1(\Omega)}$ of our finite element solution.

Since the exact solution $u = \cos(x) \cos(y) \cos(z)$ of Example 9.1 is a very smooth function, the advantage of using $S^4(\Omega, T_H)$ instead of $S^1(\Omega, T_H)$ is a much smaller error at equal number of degrees of freedom. However, due to the smaller error, we have to increase $C_{\text{res}}$ for $p = 3$ which in turn yields to higher iteration numbers and, moreover, solving all $S_p$-subproblems takes a lot of computing time even if we apply the runtime optimal strategy. On the other hand we have a good approach for a possible parallelization. These $S_p$-subproblems are independent local problems, thus, if we compute them in parallel the computing time for higher polynomial degrees will decrease significantly.

10 Proof of Theorem 5.2

In this section we want to give a proof for Theorem 5.2. We make use of the notation of Section 5 and define operators $P_i : \mathcal{V} \rightarrow \mathcal{V}$ and $P : \mathcal{V} \rightarrow \mathcal{V}$ via

$$a(P_i u, v) = a(u, v) \quad \forall v \in \mathcal{V}_i \quad \text{and} \quad P := \sum_{i=0}^K P_i.$$  \hfill (8)

Thus $P = \mathcal{C}^{-1}A$ and we have to derive a bound for

$$\kappa(P) = \frac{\lambda_{\max}(P)}{\lambda_{\min}(P)}$$  \hfill (9)

where $\lambda_{\min}(P) = \inf_{u \neq 0} \frac{a(Pu, u)}{a(u, u)}$ and $\lambda_{\max}(P) = \sup_{u \neq 0} \frac{a(Pu, u)}{a(u, u)}$.

**Lemma 10.1 (Lemma of Lions).** Let $C_0$ be given by (2). Then for all $u \in \mathcal{V}$

$$\|u\|^2_{\mathcal{V}} \leq C_0^2 a(Pu, u).$$

**Proof.** Let $u \in \mathcal{V}$ and choose a representation $u = \sum_{i=0}^K u_i$ such that

$$\|u\|^2_{\mathcal{V}} \leq C_0^2 \|u\|^2_{\mathcal{V}}$$  \hfill (10)

Then, applying (8) and Cauchy-Schwarz-inequality we obtain

$$\|u\|^2_{\mathcal{V}} = \sum_{i=0}^K a(u, u_i) = \sum_{i=0}^K a(Pu, u_i) \leq \left( \sum_{i=0}^K \|Pu\|^2_{\mathcal{V}} \right)^{1/2} \left( \sum_{i=0}^K \|u_i\|^2_{\mathcal{V}} \right)^{1/2}$$

and the claim follows in combination with (10).

Now, if we define an Operator $\mathcal{A} : \mathcal{V} \rightarrow \mathcal{V}^*$ via

$$[\mathcal{A}u](\cdot) := a(u, \cdot)$$

Problem 2.3 becomes equivalent to a finite dimensional operator equation in $\mathcal{V}^*$.

**Problem 2.4 (Operator equation).** Find $u \in \mathcal{V}$ such that

$$\mathcal{A} u = f.$$  \hfill (11)

3 PCG-algorithm

Since the operator $\mathcal{A}$ with $[\mathcal{A}u](v) := a(u, v)$ is related to a symmetric positive definite bilinear form $a(\cdot, \cdot)$ we can solve Problem (2.4) by the well-known PCG-algorithm. At this point one usually introduces a basis for the FE-space $\mathcal{V}$ (see Remark (3.2) below). Thus, $\mathcal{V}$ and the dual space $\mathcal{V}^*$ equipped with the corresponding dual basis become equivalent to $\mathbb{R}^N$ and the PCG-algorithm can be considered in $\mathbb{R}^N$ using matrix-vector operations. However, for this paper it is more convenient to postpone the definition of a basis and to consider the PCG-algorithm in a slightly more abstract way, namely as an algorithm which operates in the FE-space $\mathcal{V}$ and its dual space $\mathcal{V}^*$. Before we formulate the algorithm, we define for given $u \in \mathcal{V}$ a corresponding residuum functional $r_u \in \mathcal{V}^*$ via

$$r_u := \mathcal{A} u - f.$$

and omit the index $u$ whenever it is clear from the context.

**Algorithm 3.1 (PCG).**

- Choose an initial guess $u \in \mathcal{V}$ and compute its residuum functional $r \in \mathcal{V}^*$.
- Apply a preconditioner $\mathcal{C}^{-1} : \mathcal{V}^* \rightarrow \mathcal{V}$:

$$q = w = \mathcal{C}^{-1} r \in \mathcal{V}.$$
• Compute $\gamma = r(w)$.

• While not converged do
  1. $v = Aq \in V^*$
  2. $\alpha = -\gamma/v(q) \in \mathbb{R}$
  3. $u = u + \alpha q \in V$
  4. $r = r + \alpha v \in V^*$
  5. $\hat{w} = r(w) \in \mathbb{R}$
  6. $\hat{q} = \hat{r}(w) \in \mathbb{R}$
  7. $q = w + (\hat{q}/\gamma)q \in V$
  8. $\gamma = \hat{\gamma} \in \mathbb{R}$

An important issue concerning an iterative solver is its convergence rate. In the case of the PCG-Algorithm (3.1) this convergence rate is strongly related to the preconidioner $C^{-1}$. If we define the energy norm $\| \cdot \|_A$ and the condition number $\kappa$ of $C^{-1}A$ via

$$
\|u\|^2_A := a(u, u)
$$

then it is a well-known fact that with the initial guess $u^0$ and the exact solution $u'$ of $Au = f$ the $i$-th iterate $u'^{(i)}$ of the PCG-algorithm (3.1) satisfies

$$
\|u' - u'^{(i)}\| \leq 2 \left( \frac{\sqrt{\lambda} - 1}{\sqrt{\lambda} + 1} \right)^i \|u' - u^0\|_A.
$$

Remark 3.2. Later we introduce a basis $\Phi = [\phi_1, \ldots, \phi_N]$ for our FE-space $V$ and the dual basis $\Psi = [\psi_1, \ldots, \psi_N]$ for $V^*$. With such bases each $u \in V$ becomes equivalent to a vector $u \in \mathbb{R}^N$,

$$
u = \sum_{i=1}^N u_i \phi_i \in V \iff u = (u_1, \ldots, u_N)^T \in \mathbb{R}^N
$$

and, moreover, each $f \in V^*$ becomes equivalent to a vector $f \in \mathbb{R}^N$

$$
f = \sum_{i=1}^N f_i \psi_i \in V^* \iff f = (f_1, \ldots, f_N)^T = (f(\phi_1), \ldots, f(\phi_N))^T \in \mathbb{R}^N.
$$

Hence, since the matrix representation of the operator $A$ is a matrix whose $k$-th column is the vector representations of the operator $A\phi_k \in V^*$ with respect to $\Psi$, we obtain

$$A : V \mapsto V^* \iff A = \begin{bmatrix} a(\phi_1, \phi_1) & \cdots & a(\phi_1, \phi_N) \\ \vdots & \ddots & \vdots \\ a(\phi_N, \phi_1) & \cdots & a(\phi_N, \phi_N) \end{bmatrix} \in \mathbb{R}^{N \times N}.
$$

With these matrix and vector representations we obtain the equivalences

$$v = Av \iff v = Aq, \quad \gamma = r(w) \iff \gamma = w^T r \quad \cdots \quad \text{and we end up with the well-known } \mathbb{R}^N\text{-version of the PCG-algorithm.}
$$

---

### Table 4: Results for Example 9.1

<table>
<thead>
<tr>
<th>Elements</th>
<th>DoF</th>
<th>Iterations</th>
<th>H1-Err</th>
<th>H1-Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8</td>
<td>2.0e-05</td>
<td>0</td>
<td>5.5e-01</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>4.1e-05</td>
<td>1</td>
<td>4.1e-01</td>
</tr>
<tr>
<td>30</td>
<td>17</td>
<td>4.0e-05</td>
<td>1</td>
<td>3.3e-01</td>
</tr>
<tr>
<td>70</td>
<td>31</td>
<td>8.1e-05</td>
<td>2</td>
<td>2.7e-01</td>
</tr>
<tr>
<td>166</td>
<td>58</td>
<td>2.8e-04</td>
<td>5</td>
<td>2.0e-02</td>
</tr>
<tr>
<td>414</td>
<td>120</td>
<td>6.4e-04</td>
<td>5</td>
<td>1.5e-01</td>
</tr>
<tr>
<td>999</td>
<td>256</td>
<td>2.7e-03</td>
<td>8</td>
<td>1.5e-01</td>
</tr>
<tr>
<td>2,429</td>
<td>551</td>
<td>2.2e-02</td>
<td>9</td>
<td>8.6e-02</td>
</tr>
<tr>
<td>5,718</td>
<td>1,251</td>
<td>5.2e-03</td>
<td>9</td>
<td>6.6e-02</td>
</tr>
<tr>
<td>13,460</td>
<td>2,768</td>
<td>1.5e-02</td>
<td>10</td>
<td>5.0e-02</td>
</tr>
<tr>
<td>30,960</td>
<td>6,172</td>
<td>3.9e-02</td>
<td>11</td>
<td>3.8e-02</td>
</tr>
<tr>
<td>70,932</td>
<td>13,617</td>
<td>9.3e-02</td>
<td>12</td>
<td>2.9e-02</td>
</tr>
<tr>
<td>158,904</td>
<td>30,809</td>
<td>2.2e-01</td>
<td>12</td>
<td>2.3e-02</td>
</tr>
<tr>
<td>355,674</td>
<td>65,614</td>
<td>4.9e-01</td>
<td>12</td>
<td>1.7e-02</td>
</tr>
<tr>
<td>782,288</td>
<td>146,532</td>
<td>1.2e+00</td>
<td>13</td>
<td>1.4e-02</td>
</tr>
<tr>
<td>1,717,898</td>
<td>317,675</td>
<td>2.5e+00</td>
<td>12</td>
<td>1.1e-02</td>
</tr>
<tr>
<td>3,723,934</td>
<td>681,607</td>
<td>6.1e+00</td>
<td>13</td>
<td>8.6e-03</td>
</tr>
<tr>
<td>8,067,178</td>
<td>1,466,444</td>
<td>1.3e+01</td>
<td>13</td>
<td>6.8e-03</td>
</tr>
<tr>
<td>17,284,898</td>
<td>3,191,462</td>
<td>3.5e+01</td>
<td>13</td>
<td>5.5e-03</td>
</tr>
<tr>
<td>37,051,776</td>
<td>6,646,901</td>
<td>8.0e+01</td>
<td>13</td>
<td>4.2e-03</td>
</tr>
</tbody>
</table>

Let us consider the following example.

**Example 9.1.** Let $\Omega = (0, 1)^3, \Gamma_0 = \partial \Omega$. Let $(T_\iota)_{\iota > 0}$ be a sequence of nested meshes on $\Omega$. For $p = 1, 2, 3$ find $u \in S^p(\Omega, M)$ with $u_{\mid \Gamma_0} = \cos(x) \cos(y) \cos(z)$ such that

$$
\int_\Omega \nabla u \cdot \nabla v + uv d\Omega = \int_\Omega v d\Omega \quad \forall v \in S^p_0(\Omega, M).
$$

We start with $u = 0$ on a coarse grid and apply a total mesh refinement strategy which gives us a sequence of nested meshes. We use the FE-solution of the current mesh as initial guess for the FE-solution on the refined mesh and iterate until the initial residual $r_1$ is reduced by a given factor $C_{res}$. That is we iterate until $r_1^2 < C_{res}^2 a_0$. In the first run we consider the finite element space $S^p(\Omega, M)$ for $p = 1$ and thereafter we compute the runtime optimal strategy for $p = 3$.

The results of the experiments are collected in Table 4.
vertices. Moreover we define for each \( v \in \mathcal{V}_M \) a subspace
\[
S_v = \{ u \in S^p(\Omega, T_M) \mid \text{supp } u \subset w_M^v \}.
\]
Then, in the context of ASM, the splitting
\[
S^p(\Omega, T_M) = S^1(\Omega, T_M) + \sum_{v \in \mathcal{V}_M} S_v
\]
defines an overlapping preconditioner with \( \kappa(C^{-1}A) = O(1) \). If we decompose
\( S^1(\Omega, T_M) \) further, such that
\[
S^p(\Omega, T) = \sum_{m=0}^M \sum_{v \in \mathcal{V}_m} V_m^v + \sum_{v \in \mathcal{V}_M} S_v,
\]
we obtain a preconditioner for \( S^p(\Omega, T_M) \) which can be considered as a combination of BPX- and overlapping preconditioner. Thus, due to Definition 5.1 we get for a given residuum \( r \in V^* \) the preconditioned residuum \( C^{-1}r \) by using the BPX-routine from the previous sections to compute
\[
\sum_{m=0}^M \sum_{v \in \mathcal{V}_m} \left( \frac{r(\phi_v^m)}{a(\phi_v^m, \phi_v^m)} \right) \phi_v^m
\]
and adding \( w_v \in S_v \) for all \( v \in \mathcal{V}_M \), given by
\[
a(w_v, u) = r(u) \ \forall u \in S_v.
\]
That is, after applying the BPX-routine we have to solve a linear system of equations for each subspace \( S_v \). However, these linear systems of equations are low-dimensional and for a fixed subspace \( S_v \), the corresponding system matrix is s.p.d. and always the same. Only right hand sides change.

There are two possible strategies to implement the preconditioner

1. Runtime optimal strategy: We compute a Choleky decomposition once for each system matrix of the \( S_v \)-subproblems. We store these Cholesky decompositions and reduce solving the linear systems of equations to forward and backward substitutions. Since the polynomial degree \( p \) is fixed and the number of tetrahedra sharing a vertex is bounded by a constant, the dimension of \( S_v \) is also bounded by a constant and hence we have:
   \[
   \text{additional memory} = O(N_M).
   \]

2. Memory optimal strategy: We compute the Cholesky decompositions anew whenever we have to solve a \( S_v \)-subproblem. Since the dimension of \( S_v \) is bounded by a constant we have
   \[
   \text{computing time for all Cholesky decompositions} = O(N_M).
   \]

### 4 Adaptive FEM
Considering Problem 2.2 it is not common practice to start with a highly refined mesh to compute an approximate solution. Instead we start on a coarse grid with as few nodes as possible. We compute the corresponding FE-solution and, thereafter, in order to improve the accuracy of the approximation, we employ an a-posteriori error estimator which gives us information about the error distribution and flags all elements with large local error. Next we refine all flagged elements and obtain a finer mesh. We compute a new approximation of higher accuracy, estimate the error for this new approximation and in case the approximation is not sufficiently accurate a new iteration of the adaptive loop is begun. Thus, we obtain a sequence of refined meshes together with FE-solutions of increasing accuracy. That is we have

<table>
<thead>
<tr>
<th>coarse grid</th>
<th>stiff. matrix</th>
<th>FE-solution</th>
<th>error est.</th>
<th>refined mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0)</td>
<td>(1)</td>
<td>(2)</td>
<td>(3)</td>
<td>(4)</td>
</tr>
</tbody>
</table>

In this paper we focus on tetrahedral meshes without hanging nodes. A suited algorithm for local mesh refinement which avoids hanging nodes can be found in [AMP00].

### 5 Additive-Schwarz-Method (ASM)
As we have already seen in Section 3, the key to solve the arising system of linear equations with only a few PCG-iterations is the definition of a good preconditioner \( C \). The preconditioners which we will consider are the BPX-preconditioner for the piecewise linear functions of \( S^1(\Omega, T) \) and an overlapping preconditioner for the higher polynomial degrees of \( S^p(\Omega, T) \). Both preconditioners can be considered as members of the class of Additive-Schwarz-preconditioner which means they are completely determined by a not necessarily direct decomposition
\[
V := S^p_M(\Omega, T) = \sum_{i=0}^K V_i
\]
of our finite element space \( V \) into subspaces \( V_i \). The action of such a preconditioner is best described by the realization of \( C^{-1}r \), where \( r \in V^* \) denotes the residuum.

**Definition 5.1** (ASM-preconditioner). For each subspace \( V_i \) find \( w_i \in V_i \) such that
\[
a(w_i, v) = r(v) \ \forall v \in V_i.
\]
Then
\[ C^{-1}r := \sum_{i=0}^{K} w_i. \]

We see, the computation of \( C^{-1}r \) is equivalent to solve local versions of the original problem for all subspaces \( V_i \). However, these subspaces are ordinarily low-dimensional in contrast to the FE-space \( V \).

In order to analyze ASM-preconditioners we define \( C^2 \) such that
\[
\min \left\{ \sum_{i=0}^{K} o(u_i, u_i) \left| u = \sum_{i=0}^{K} u_i, u_i \in V_i \right. \right\} \leq C^2 o(u, u) \quad \forall u \in V
\]
and denote with \( \rho(E) \) the spectral radius of the matrix \( E = [e_{ij}]^{K}_{i,j=1} \) with entries
\[
e_{ij} = \sup_{u \in V_i, v \in V_j} \frac{|a(u, v)|}{\sqrt{a(u, u)} \sqrt{a(v, v)}} \in [0, 1],
\]
which are the angles between the subspaces \( V_1, \ldots, V_K \). With these settings we are able to derive a bound for the condition number of \( C^{-1}A \).

**Theorem 5.2.**
\[
\kappa(C^{-1}A) \leq C^2 (1 + \rho(E)).
\]

**Proof.** See Section 10.

\[ \square \]

## 6 MDS-BPX

In the following two sections we will introduce the Multilevel-Diagonal-Scaling-BPX (MDS-BPX) as an additive Schwarz preconditioner for \( V = S^1(Ω, T) \) and describe its implementation in detail.

To that end we assume that we are within an adaptive FE-process (see Section 4) which started on a coarse grid \( T_0 \) and produced a sequence of nested tetrahedral meshes
\[ T_0 \subset T_1 \subset \ldots \subset T_M = T. \]

We define
\[
V^m := \text{Set of all free vertices on } T_m
\]
and
\[
\varphi^m := \text{Set of all vertices of } T_m \text{ which are not located on } \Gamma_D,
\]
and we denote with \( \varphi^m_v \) the hat function on \( T_m \) with respect to the vertex \( v \).

### Table 3: Adaptive mesh refinement, \( C^2_{\text{res}} = 10^{-8} \)

<table>
<thead>
<tr>
<th>M</th>
<th>Elements</th>
<th>DoF</th>
<th>It</th>
<th>Total[s]</th>
<th>Precond[s]</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>108</td>
<td>21.227</td>
<td>26</td>
<td>3.3e-01</td>
<td>4.4e-02</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>16</td>
<td>9.2e-05</td>
<td>9.2e-05</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>23</td>
<td>1.7e-04</td>
<td>3.2e-05</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>32</td>
<td>1.4e-04</td>
<td>4.4e-05</td>
<td>0.31</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>46</td>
<td>9.1e-05</td>
<td>9.2e-05</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>50</td>
<td>8.1e-04</td>
<td>3.0e-05</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>40</td>
<td>80</td>
<td>1.7e-02</td>
<td>2.3e-03</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>108</td>
<td>1.209.941</td>
<td>227.229</td>
<td>4.1e+00</td>
<td>6.9e-01</td>
</tr>
<tr>
<td>9</td>
<td>80</td>
<td>118</td>
<td>8.040</td>
<td>1.694</td>
<td>22</td>
<td>1.7e-02</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>182</td>
<td>32</td>
<td>1.82</td>
<td>22</td>
<td>1.7e-02</td>
</tr>
<tr>
<td>11</td>
<td>120</td>
<td>298</td>
<td>46</td>
<td>2.98</td>
<td>11</td>
<td>1.82</td>
</tr>
<tr>
<td>12</td>
<td>140</td>
<td>508</td>
<td>6.209.941</td>
<td>227.229</td>
<td>4.1e+00</td>
<td>6.9e-01</td>
</tr>
<tr>
<td>13</td>
<td>160</td>
<td>8.040</td>
<td>1.694</td>
<td>22</td>
<td>1.7e-02</td>
<td>2.3e-03</td>
</tr>
<tr>
<td>14</td>
<td>180</td>
<td>108</td>
<td>1.209.941</td>
<td>227.229</td>
<td>4.1e+00</td>
<td>6.9e-01</td>
</tr>
<tr>
<td>15</td>
<td>200</td>
<td>118</td>
<td>8.040</td>
<td>1.694</td>
<td>22</td>
<td>1.7e-02</td>
</tr>
<tr>
<td>16</td>
<td>220</td>
<td>298</td>
<td>46</td>
<td>2.98</td>
<td>11</td>
<td>1.82</td>
</tr>
<tr>
<td>17</td>
<td>240</td>
<td>508</td>
<td>6.209.941</td>
<td>227.229</td>
<td>4.1e+00</td>
<td>6.9e-01</td>
</tr>
<tr>
<td>18</td>
<td>260</td>
<td>8.040</td>
<td>1.694</td>
<td>22</td>
<td>1.7e-02</td>
<td>2.3e-03</td>
</tr>
<tr>
<td>19</td>
<td>280</td>
<td>108</td>
<td>1.209.941</td>
<td>227.229</td>
<td>4.1e+00</td>
<td>6.9e-01</td>
</tr>
<tr>
<td>20</td>
<td>300</td>
<td>118</td>
<td>8.040</td>
<td>1.694</td>
<td>22</td>
<td>1.7e-02</td>
</tr>
<tr>
<td>21</td>
<td>320</td>
<td>298</td>
<td>46</td>
<td>2.98</td>
<td>11</td>
<td>1.82</td>
</tr>
<tr>
<td>22</td>
<td>340</td>
<td>508</td>
<td>6.209.941</td>
<td>227.229</td>
<td>4.1e+00</td>
<td>6.9e-01</td>
</tr>
<tr>
<td>23</td>
<td>360</td>
<td>8.040</td>
<td>1.694</td>
<td>22</td>
<td>1.7e-02</td>
<td>2.3e-03</td>
</tr>
<tr>
<td>24</td>
<td>380</td>
<td>108</td>
<td>1.209.941</td>
<td>227.229</td>
<td>4.1e+00</td>
<td>6.9e-01</td>
</tr>
<tr>
<td>25</td>
<td>400</td>
<td>118</td>
<td>8.040</td>
<td>1.694</td>
<td>22</td>
<td>1.7e-02</td>
</tr>
<tr>
<td>26</td>
<td>420</td>
<td>298</td>
<td>46</td>
<td>2.98</td>
<td>11</td>
<td>1.82</td>
</tr>
<tr>
<td>27</td>
<td>440</td>
<td>508</td>
<td>6.209.941</td>
<td>227.229</td>
<td>4.1e+00</td>
<td>6.9e-01</td>
</tr>
<tr>
<td>28</td>
<td>460</td>
<td>8.040</td>
<td>1.694</td>
<td>22</td>
<td>1.7e-02</td>
<td>2.3e-03</td>
</tr>
<tr>
<td>29</td>
<td>480</td>
<td>108</td>
<td>1.209.941</td>
<td>227.229</td>
<td>4.1e+00</td>
<td>6.9e-01</td>
</tr>
<tr>
<td>30</td>
<td>500</td>
<td>118</td>
<td>8.040</td>
<td>1.694</td>
<td>22</td>
<td>1.7e-02</td>
</tr>
</tbody>
</table>

### 9 Higher polynomial degrees

Up to now we considered finite element spaces \( V = S^1(Ω, T) \). However, often it is useful to apply \( S^p(Ω, T) \) with \( p > 1 \) or even a \( kp \)-version of FEM, where different elements may have different polynomial degrees. Thus the question arises how to construct a preconditioner for such a setting where higher polynomial degrees appear.

As in the previous sections, we assume that we have a sequence of nested meshes \( T_0 \subset T_1 \subset \ldots \subset T_M \) and we define for \( v \in V^m \) the patch \( \omega^m = \{ K \in T_M | v \in K \} \) which is the closure of the union of all elements \( K \in T_M \) with \( v \) as one of its
We define one-dimensional subspaces
\[ V^m_v := \text{span} \phi^m_v. \]
and equip the finite element space \( V \) with the basis \( \{ \phi^m_v | v \in V^M \} \).

Then, in the context of ASM, the MDS-BPX-preconditioner is defined by the decomposition
\[ V = \sum_{m=0}^{M} \sum_{v \in V^m_v} \phi^m_v, \]
where the "\( \sum \)" indicates that the summation is only over pairwise distinct spaces \( V^m_v \). For \( (T_0)^M_{m=0} \), arising from a total or a boundary concentrated mesh refinement it is shown that the BPX-preconditioner yields
\[ \kappa(C^{-1}A) = O(1) \quad \text{for } M \rightarrow \infty, \]
independent on the dimension of \( \Omega \) (see [Eib06, Xu90, Xu92, Zha92]) and for adaptive mesh refinements all numerical examples indicate that (5) is valid too.

Due to Definition 5.1 of general ASM-preconditioners, the finite element space decomposition (4) and the fact that all subspaces \( V^m_v \) are one-dimensional spaces, we compute for a given residuum \( r \in V^p \) the BPX-preconditioner as
\[ C^{-1}r = \sum_{m=0}^{M} \sum_{v \in V^m_v} \left( \frac{r(\phi^m_v)}{a(\phi^m_v, \phi^m_v)} \right) \phi^m_v. \]

To illustrate Splitting (4) we want to close this section with a one-dimensional example.

**Example 6.1.** For \( \Omega = (0,1), \Gamma_D = \emptyset \), given bilinear form \( a(\cdot, \cdot) \), and given right hand side functional \( f \) we want to solve Problem 2.2. Assume we start with a coarse grid \( T_0 \) and that our adaptive FE-algorithm produced a sequence of meshes \( T_0, \ldots, T_3 \) as in shown in Figure 1 so far.

According to (4) Example 6.1 induces the splitting
\[ V = S^I(\Omega, T_3) = (V_1^0 + V_3^0) + (V_1^1 + V_3^1 + V_4^1) + (V_1^2 + V_2^2 + V_4^2) + (V_1^3 + V_2^3 + V_4^3). \]

That is, we start with adding up all subspaces of the coarse grid and thereafter, since no subspace of \( T_1 \) is equivalent to \( V_1^0 \) or \( V_2^0 \), we have to add all subspaces of \( T_2 \). When we come to the subspaces of \( T_3 \) we only add \( V_1^1, V_2^1, V_3^1 \) and \( V_4^1 \) since this subspace is equivalent to \( V_1^2 \) and hence already a part of our splitting. Finally we consider the subspaces of \( T_3 \) and it remains to add \( V_1^2, V_2^2, V_3^2 \).

---

We consider two different scenarios. At first we consider the case where the mesh \( T_{r+1} \) is derived from \( T_r \) by a total mesh refinement. Thereafter, in a second run, we employ a standard residuum estimator to refine only those elements with high local error and hence obtain a sequence of locally refined meshes.

All iterations start with an initial guess \( u = 0 \) and we use the PCG-algorithm in conjunction with our implementation of the BPX-preconditioner to reduce the initial residual (measured in the Euclidean norm) by a given factor \( C_{\text{rel}} \). That is we iterate until \( r^2 < C_{\text{rel}}^2 r_0^2 \).

The results of the experiments are collected in Table 2 and Table 3.

The column "Elements" shows the number of tetrahedra and "DoF" denotes the dimension of the finite element space \( S^I(\Omega, T_M) \). The column "It" gives the number of PCG iterations required to reduce the residual by \( C_{\text{rel}} \). "Total" is the computing time of the PCG algorithm including the application of the preconditioner and the column "Precond" contains the time to execute the preconditioner. Finally the column "Ratio" gives the quotient of "Total" and "Precond".

As we can see, the preconditioner leads to small iteration numbers and its eval-

<table>
<thead>
<tr>
<th>M</th>
<th>Elements</th>
<th>DoF</th>
<th>It</th>
<th>Total[s]</th>
<th>Precond[s]</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>6.1e-05</td>
<td>1.5e-05</td>
<td>0.29</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>10</td>
<td>8</td>
<td>8.4e-05</td>
<td>2.4e-05</td>
<td>0.28</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>17</td>
<td>8</td>
<td>1.2e-04</td>
<td>3.3e-05</td>
<td>0.28</td>
</tr>
<tr>
<td>3</td>
<td>70</td>
<td>31</td>
<td>9</td>
<td>2.2e-04</td>
<td>5.4e-05</td>
<td>0.24</td>
</tr>
<tr>
<td>4</td>
<td>166</td>
<td>58</td>
<td>11</td>
<td>5.0e-04</td>
<td>1.0e-04</td>
<td>0.20</td>
</tr>
<tr>
<td>5</td>
<td>414</td>
<td>120</td>
<td>12</td>
<td>5.1e-04</td>
<td>9.1e-05</td>
<td>0.17</td>
</tr>
<tr>
<td>6</td>
<td>990</td>
<td>256</td>
<td>15</td>
<td>1.4e-03</td>
<td>2.1e-04</td>
<td>0.14</td>
</tr>
<tr>
<td>7</td>
<td>2.420</td>
<td>554</td>
<td>16</td>
<td>3.5e-03</td>
<td>4.3e-04</td>
<td>0.12</td>
</tr>
<tr>
<td>8</td>
<td>5.718</td>
<td>1.251</td>
<td>17</td>
<td>8.8e-03</td>
<td>1.1e-03</td>
<td>0.12</td>
</tr>
<tr>
<td>9</td>
<td>13.460</td>
<td>2.768</td>
<td>19</td>
<td>2.6e-02</td>
<td>3.4e-03</td>
<td>0.13</td>
</tr>
<tr>
<td>10</td>
<td>30.960</td>
<td>6.172</td>
<td>19</td>
<td>6.1e-02</td>
<td>7.7e-03</td>
<td>0.12</td>
</tr>
<tr>
<td>11</td>
<td>70.932</td>
<td>13.617</td>
<td>20</td>
<td>1.5e-01</td>
<td>1.9e-02</td>
<td>0.12</td>
</tr>
<tr>
<td>12</td>
<td>158.904</td>
<td>30.809</td>
<td>20</td>
<td>3.6e-01</td>
<td>5.1e-02</td>
<td>0.14</td>
</tr>
<tr>
<td>13</td>
<td>355.674</td>
<td>65.614</td>
<td>21</td>
<td>8.5e-01</td>
<td>1.3e-01</td>
<td>0.15</td>
</tr>
<tr>
<td>14</td>
<td>782.288</td>
<td>146.532</td>
<td>21</td>
<td>1.9e+00</td>
<td>3.1e-01</td>
<td>0.15</td>
</tr>
<tr>
<td>15</td>
<td>1.717.898</td>
<td>317.675</td>
<td>21</td>
<td>4.4e+00</td>
<td>7.0e-01</td>
<td>0.16</td>
</tr>
<tr>
<td>16</td>
<td>3.723.834</td>
<td>681.607</td>
<td>22</td>
<td>1.0e+01</td>
<td>1.7e+00</td>
<td>0.16</td>
</tr>
<tr>
<td>17</td>
<td>8.967.178</td>
<td>1.466.444</td>
<td>22</td>
<td>2.3e+01</td>
<td>3.8e+00</td>
<td>0.16</td>
</tr>
<tr>
<td>18</td>
<td>17.284.808</td>
<td>3.191.462</td>
<td>22</td>
<td>5.4e+01</td>
<td>8.5e+00</td>
<td>0.15</td>
</tr>
<tr>
<td>19</td>
<td>37.051.776</td>
<td>6.646.901</td>
<td>22</td>
<td>1.4e+02</td>
<td>2.9e+01</td>
<td>0.21</td>
</tr>
</tbody>
</table>

Table 2: Total mesh refinement, \( C_{\text{rel}} = 10^{-6} \)
Figure 1: 1D-example of a sequence of adaptively refined meshes

Remark 6.2. We number the nodes of $T_M$ consecutively from 1 to $N_M$ such that if $N_m$ denotes the number of nodes of $T_m$, the nodes of $T_m$ correspond to the numbers $1, \ldots, N_m$ for $m = 0, \ldots, M$. Thus, due to this isomorphism between the nodes of $T_m$ and the numbers $\{1, \ldots, N_m\}$ we also write $\phi^m_v$ instead of $\phi^m_i$ for a vertex $v$ with number $i$. Similarly we may write $\mathcal{V}^m_i$ for $\mathcal{V}^m_v$.

7 Implementation of the MDS-BPX

Now, after we have seen the definition of the BPX-preconditioner, the remaining question is, how to evaluate the double sum (6) fast and efficiently, which means with at most $O(N_M)$ work and at most $O(N_M)$ additional memory, independent on the number of mesh refinements.

The key of our implementation is a properly chosen data-structure the so called extended vertex tree table or short EVT-table, which will be introduced in the following subsection.

7.1 Preliminary work

In order to compute $C^{-1}r$ for a given residuum $r \in \mathcal{V}$ we need $r(\phi^m_v)$ and $a(\phi^m_v, \phi^m_v)$ for several but not all basis functions $\phi^m_v$.

Since the summation (4) is only over pairwise distinct spaces $\mathcal{V}_v^m$ we observe, that starting with all subspaces $\mathcal{V}_v^0$ for $v \in \mathcal{V}^0$ we have to add for $m = 1, \ldots, M$ exactly all these subspaces $\mathcal{V}_v^m$ which correspond to a vertex $v$, newly created on $T_m$, or one of its parents. That is, for fixed $T_m$ we need $r(\phi^m_v)$ and $a(\phi^m_v, \phi^m_v)$ for all the basis functions $\phi^m_v$, where $v$ is a newly created vertex or a parent of such a vertex.

Remark 7.4. The extension of $w$ to length $N_{m+1}$ is mere formality. In general we start with a vector $w$ of length $N_M$ from the very beginning and an extension as above is simply a change of the upper array index.

Remark 7.5. Consider Example 6.1 and assume that we obtain a mesh $T_i$ by subdividing the elements $(1, 5)$ and $(4, 5)$ of $T_0$. That is we get

and our EVT-table looks like

<table>
<thead>
<tr>
<th>$N$</th>
<th>Father 1</th>
<th>Father 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1 (r(\phi^1_1), a(\phi^1_1, \phi^1_1))</td>
<td>1 (r(\phi^1_2), a(\phi^1_2, \phi^1_2))</td>
</tr>
<tr>
<td>3</td>
<td>1 (r(\phi^1_1), a(\phi^1_1, \phi^1_1))</td>
<td>1 (r(\phi^1_3), a(\phi^1_3, \phi^1_3))</td>
</tr>
</tbody>
</table>

Now, adding all $w_i$ contributions of (6) means to perform

$$w = \sum_{i = 1}^7 \left( \frac{r(\phi^1_i)}{a(\phi^1_i, \phi^1_i)} \right) \phi^1_i$$

with $i \in \{1, 4, 5, 6, 7\}$.

However, if we went through the rows of the $T_4$ section of our EVT-table the vertex with number 5 appears twice as a parent and consequently we would add

$$\left( \frac{r(\phi^1_5)}{a(\phi^1_5, \phi^1_5)} \right) \phi^1_5$$

twice. The easiest way to avoid such an undesired doubling is to use a flag for each node.

8 Numerical example

In this section we present some numerical examples to demonstrate the efficiency of our BPX-implementation. All examples start with a coarse grid $T_0$ of the given domain and then we create a sequence of nested meshes by applying the mesh refinement algorithm of [AMP00]. This algorithm is based on the bisection of tetrahedra and allows us to construct locally refined meshes without hanging nodes.

Example 8.1. Let $\Omega = (0, 1)^3$, $\Gamma_D = (0, 1) \times (0, 1) \times \{0, 1\}$ and $\Gamma_N = \partial \Omega \setminus \Gamma_D$. Let $\{T_i\}_{i = 0}^M$ be a sequence of nested meshes on $\Omega$. Find $u \in \mathcal{V} = S^3_0(\Omega, T_M)$ such that

$$\int_\Omega \nabla u \cdot \nabla v + uw \mathrm{d} \Omega = \int_\Omega f \mathrm{d} \Omega \quad \forall v \in \mathcal{V},$$

where the right hand side is given by $f_r = 1 + \sum_{i = 1}^3 x_i^2$. 

8
Algorithm 7.3.  
for m = 0, . . . , M  
(0) Clear all flags.  
(1) Add $\sum_{v \in V^m} w^m_v$ to the current w given as $w = [w^m_1, \ldots, w^m_{N_m}]$ with  
$$w = \sum_{i=1}^{N_m} w^{m_i} \phi^m_i.$$  
(2) If m < M  
Decompose w after the $S^1(\Omega, T_{m+1})$ basis such that  
$$w = \sum_{i=1}^{N_{m+1}} w^{m+1} \phi^{m+1}_i$$  
and set $w = [w^{m+1}_1, \ldots, w^{m+1}_{N_{m+1}}].$  

Let us consider parts (1) and (2) of Algorithm 7.3 in detail.  
In part (1), all we have to do is to go through the rows $i = N_{m-1} + 1, \ldots, N_m$ of our EVT-table. From each row we read the numbers $p_1$ and $p_2$ of the parents of vertex $i$, compute  
$$w^{m+1}_i = r(\phi^m_i)/a(\phi^m_i, \phi^m_i) \quad \text{if vertex } i \text{ belongs to } V^m,$$
$$w^{m+1}_i = r(\phi^m_i)/a(\phi^{p_1}_i, \phi^{p_2}_i) \quad \text{if vertex } p_1 \text{ is not flagged and belongs to } V^m,$$
$$w^{m+1}_i = r(\phi^m_i)/a(\phi^{p_2}_i, \phi^{p_2}_i) \quad \text{if vertex } p_2 \text{ is not flagged and belongs to } V^m,$$
and finally flag the vertices $p_1$ and $p_2$. All required data can be found in the $i$-th row of our EVT-table, the flags are necessary to ensure that we add the $r(\phi^m_i)/a(\phi^m_i, \phi^m_i)$ terms only once, see Remark 7.5.

In part (2) we convert the current vector representation of $w$ with respect to the $S^1(\Omega, T_{m+1})$ basis into a vector representation of $w$ with respect to the $S^1(\Omega, T_{m+1})$ basis. To do so we have to extent the vector representation $[w^{m}_1, \ldots, w^{m}_{N_m}]$ to $[w^{m+1}_1, \ldots, w^{m+1}_{N_{m+1}}]$ with  
$$w^{m+1}_i := \begin{cases}  
\frac{1}{2} w^{m}_i & \text{for } i \leq N_m \\
\frac{1}{2} (w^{m+1}_i + w^{m+1}_i) & \text{for } i \in \{N_m + 1, \ldots, N_{m+1}\} 
\end{cases}$$
where $p_1$, $p_2$ are the parents of vertex $i$. Now we have the desired vector representation of $w$ with respect to the $S^1(\Omega, T_{m+1})$ basis and once again, all the required data could be obtained directly from the $i = N_{m-1} + 1, \ldots, N_{m+1}$ rows of our EVT-table.

Remark 7.1. Newly created vertex with respect to $T_m$ means that the vertex is created within the refinement process $T_{m-1} \rightarrow T_m$. In the case $m = 0$ all vertices of $T_0$ are called newly created.

Thus, in order to provide all the required $r(\phi^m_i)$ and $a(\phi^m_i, \phi^m_i)$, we generate an EVT-table synchronously to the adaptive finite element process. Our EVT-table contains one row for each node and each of these rows stores two integers and six real numbers. To be more precisely let us consider the node with number i and assume that this node appears first on mesh $T_m$. Furthermore we assume that its parents have the numbers $p_1$ and $p_2$. Then the i-th row of our table provides storage space for  
$$\begin{array}{cccccc}
\phi^m_1 & \phi^m_2 & r(\phi^m_1) & r(\phi^m_2) & a(\phi^m_1, \phi^m_1) & a(\phi^m_2, \phi^m_2) \\
p_1 & p_2 & r(\phi^{p_1}_1) & r(\phi^{p_2}_2) & a(\phi^{p_1}_1, \phi^{p_1}_1) & a(\phi^{p_2}_2, \phi^{p_2}_2) \\
\end{array}$$
Since the nodes of $T_0$ have no parents the last columns of the first $N_0$ rows remain empty.

All but the $r(\phi^m_i)$ cells have to be initialized once and thereafter its contents will never change again. Thus we call the set of cells which do not contain an $r(\phi^m_i)$ entry static cells and the preliminary work is to initialize the static cells of our EVT-table.

The static cells of the first $N_0$ rows are initialized directly after setting up the stiffness matrix for the initial mesh $T_0$. For $v \in V^0$ we only have to copy $a(\phi^0_v, \phi^0_v)$ from the diagonal of the stiffness matrix into our EVT-table. For $v \notin V^0$ we set $a(\phi^0_v, \phi^0_v) := 1$. Thereafter the mesh refinement process has to record all newly created nodes together with the numbers of its parents and after setting up the stiffness matrix for the refined mesh we run through the rows of our EVT-table associated with the newly created nodes and copy the required entries $a(\cdot, \cdot)$ from the diagonal of the stiffness matrix into our table or set $a(\phi^m_v, \phi^m_v) := 1$ for $v \notin V^m$, respectively.

Considering Example 6.1 this yields to:

<table>
<thead>
<tr>
<th>Table 1: EVT-table for Example 6.1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_0$</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>$T_1$</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

(Note that only the cells with a white background color are part of our data structure.)
7.2 Computation of $C^{-1}r$ - Step I

To compute $C^{-1}r$ for a given residuum $r \in V^*$ we assume that the preliminary work is completed, that is we have a fully initialized EVT-table, such as Table 7.1, at hand. Then the first step to compute $C^{-1}r$ is to fill all the $r(\phi^m_i)$ cells of our EVT-table. Since we use the standard nodal basis for $V$ and its dual basis for $V^*$ the residuum $r$ is given via a vector $[r(\phi^m_i)]_{i \in V^*}$ (see Remark 3.2). We also need $r(\phi^m_i)$ for $m \neq M$. These values are not given directly but since we know the refinement history, we can compute them recursively from $L^M$. At this point and in order to simplify notation we define the extended version of the residual vector

$$L^m = [L^m_1, \ldots, L^m_m]$$

with $L^m_i := \{r(\phi^m_i) : \text{if vertex } i \text{ belongs to } V^m, 0 : \text{otherwise}\}$

and we assume that our finite element code provides the extended $L^M$ rather than the compact version which omits the entries $r(\phi^m_i)$ for $r \notin V^m$.

Considering Example 6.1 the $T_5$ section of the corresponding EVT-table 7.1 tells us:

$$\phi_i^1 = \phi_i^1, \quad \phi_i^2 = \phi_i^2, \quad \text{and } \phi_i^3 = \phi_i^3 \text{ for } i \notin \{1,4\}.$$ 

Thus we have

$$r(\phi_i^1) = r(\phi_i^1) + \frac{1}{2} r(\phi_i^2), \quad r(\phi_i^2) = r(\phi_i^1) + \frac{1}{2} r(\phi_i^2),$$

and

$$r(\phi_i^3) = r(\phi_i^3) \quad \text{for } i \notin \{1,4\}.$$ 

Generally spoken, in order to obtain $L^{m-1}$ from $L^m$ we go through the rows $i = N_{m-1} + 1, \ldots, N_m$ of our EVT-table and compute for the parents $p_1$ and $p_2$ of vertex $i$

$$L_{ip1} = \frac{1}{2} L_i \quad \text{if the vertex with number } p_1 \text{ belongs to } V^{m-1} \quad \text{(7)}$$

and

$$L_{ip2} = \frac{1}{2} L_i \quad \text{if the vertex with number } p_2 \text{ belongs to } V^{m-1}.$$ 

These operations are performed in place, directly on $L^m$ without any auxiliary vector and finally the first $N_{m-1}$ positions of the input vector $L^m$ contain the entries of $L^{m-1}$.

That is, given an initialized EVT-table and the extended residuum vector $L^M$, we fill all $r(\phi^m_i)$ cells of our EVT-table by the following algorithm:

Algorithm 7.2.

\begin{align*}
\text{for } m = M, \ldots, 0 \{
& \text{for } i = N_m + 1, \ldots, N_m \{
& \text{copy } r(\phi^m_i), r(\phi^m_{p1}), r(\phi^m_{p2}) \text{ from } L^m \text{ into the } i\text{-th row of our EVT-table if } m > 0
& \text{compute } L^{m-1} \text{ from } L^m
\}
\}
\end{align*}

Considering Example 6.1 once again, we arrive at:

$$\begin{array}{cccc}
\text{N} & \text{Father1} & \text{Father2} \\
T_6 & 1 & r(\phi^1_1) & a(\phi^1_1, \phi^1_0) \\
T_5 & 2 & r(\phi^2_1) & a(\phi^2_1, \phi^2_0) \\
T_4 & 3 & r(\phi^3_1) & a(\phi^3_1, \phi^3_0) \\
T_3 & 4 & r(\phi^4_1) & a(\phi^4_1, \phi^4_0) \\
T_2 & 5 & r(\phi^5_1) & a(\phi^5_1, \phi^5_0) \\
\end{array}$$

7.3 Computation of $C^{-1}r$ - Step II

Now with a complete EVT-table at hand we want to add up all

$$\hat{w}^m := \left( \frac{r(\phi^m_i)}{a(\phi^m_i, \phi^m_p)} \right) \phi^m_i$$

of sum (6). To that end we start with $w = 0$. The variable $m$ goes step-by-step from $0$ to $M$ and in each step we add all the required contributions $\hat{w}^m$ of level $m$ to $w$. Obviously, the intermediate result

$$w^k := \sum_{m=0}^k \sum_{v \in V^m} \hat{w}^m_v$$

is an element of $S^1(\Omega, T_0)$ and thus its most appropriate representation is the vector $w^k = [w^k_1, \ldots, w^k_N]$ with

$$w^k = \sum_{i=1}^{N} w^k_i \phi_i.$$ 

That is, we start with $w = [0,0,\ldots,0]$, the vector representation of $w = 0$ with respect to the basis $\{\phi_1^0, \ldots, \phi_N^0\}$ of $S^1(\Omega, T_0)$ and apply the following algorithm