Journal of Computational Mathematics
Vol.28, No.6, 2010, 767–789.

http://www.global-sci.org/jcm
doi:10.4208/jcm.1004-m3172

A COARSENING ALGORITHM ON ADAPTIVE GRIDS BY NEWEST VERTEX BISECTION AND ITS APPLICATIONS*

Long Chen
Department of Mathematics, University of California at Irvine, Irvine, CA 92697, USA
Email: chenlong@math.uci.edu
Chensong Zhang
Department of Mathematics, The Pennsylvania State University, University Park, PA, 16802, USA
Email: zhangcs@psu.edu

Abstract

In this paper, an efficient and easy-to-implement coarsening algorithm is proposed for adaptive grids obtained using the newest vertex bisection method in two dimemsions. The new coarsening algorithm does not require storing the binary refinement tree explicitly. Instead, the structure is implicitly contained in a special ordering of triangular elements. Numerical experiments demonstrate that the proposed coarsening algorithm is efficient when applied for multilevel preconditioners and mesh adaptivity for time-dependent problems.

Mathematics subject classification: 65M55, 65N55, 65N22, 65F10.
Key words: Adaptive finite element method, Coarsening, Newest vertex bisection, Multilevel preconditioning.

1. Introduction

Adaptive methods are now widely employed in the scientific computation to achieve better accuracy with minimum degree of freedom. A typical adaptive finite element method through local refinement can be written as the following loop:

$$\textbf{SOLVE} \rightarrow \textbf{ESTIMATE} \rightarrow \textbf{MARK} \rightarrow \textbf{REFINE/COARSEN}. \tag{1.1}$$

In this paper, we shall consider the modules **COARSEN** and **SOLVE**. More precisely, we propose a new efficient and easy-to-implement coarsening algorithm and apply to multilevel preconditioners for adaptive grids obtained by the newest vertex bisection in two spatial dimensions.

Classical recursive bisection and coarsening algorithms [21] are widely used in adaptive algorithms (see, for example, ALBERTA [31] and deal.II [3]). These algorithms make use of binary-tree related data structures and subroutines to store and access the bisection history.

We propose a new node-wise coarsening algorithm which does not require storing the bisection tree explicitly. We only store coordinates of vertices and connectivity of triangles which is the minimal information to represent a mesh for standard finite element computation. We can built a kind of tree structure into a special ordering of the triangles. By doing this way, we simplify the implementation of adaptive mesh refinement and coarsening and thus provide an easy-access interface for the usage of mesh adaptation without too much sacrifice in time.

* Received July 26, 2009 / Revised version received October 23, 2009 / Accepted December 7, 2009 / Published online August 9, 2010 /

The coarsening algorithm can be applied to construct efficient multilevel solvers for elliptic problems. Based on the special geometric relation between nested triangulations obtained by our coarsening algorithm, we develop a new multilevel preconditioner which numerically outperforms several classical multilevel preconditioners.

The proposed coarsening algorithm can also be employed for mesh adaptation, especially for time-dependent problems. For steady-state problems, quasi-optimal meshes can be obtained in practice using (1.1) without the **COARSEN** step [11]. However, it is not the case for time-dependent problems as the local features might change dramatically in time. We provide a numerical example for the application of our coarsening algorithm to time-dependent problems.

The rest of the paper is organized as follows. We review the classical coarsening algorithm and introduce our new algorithm in Section 2. We explain the data structures and implementation of our coarsening algorithm in Section 3. In order to demonstrate the performance of the proposed coarsening algorithm, then we show two applications of our coarsening algorithm: one is multilevel preconditioners for stationary problems in Section 4 and the other is mesh adaptation for time-dependent problems in Section 5.

2. Coarsening Algorithms

In this section, we present a new coarsening algorithm for triangular meshes obtained by the newest vertex bisection method. Unlike the classical recursive coarsening algorithm, the proposed algorithm is non-recursive and requires neither storing nor maintaining the bisection tree information such as the *parents*, *brothers*, *generation*, etc.

2.1. Conformity and shape-regularity of triangulations

Let $\Omega \subset \mathbb{R}^2$ be a polygonal domain. A triangulation $\mathcal{T}$ (also known as mesh or grid) of Ω is a set of triangles (also indicated by elements) which is a partition of Ω. The set of nodes (also indicated by vertices or points) of the triangulation $\mathcal{T}$ is denoted by $\mathcal{N}(\mathcal{T})$ and the set of all edges by $\mathcal{E}(\mathcal{T})$. As a convention, all triangles $t \in \mathcal{T}$ and edges $e \in \mathcal{E}(\mathcal{T})$ are closed sets.

We define the *first ring* of a point $p \in \Omega$ or an edge $e \in \mathcal{E}(\mathcal{T})$ as

$$\mathcal{R}_p := \{t \in \mathcal{T} \,|\, p \in t\} \qquad \text{and} \qquad \mathcal{R}_e := \{t \in \mathcal{T} \,|\, e \subset t\},$$

respectively; and define the *local patch* of p or e as

$$\omega_p := \bigcup_{t \in \mathcal{R}_p} t \qquad \text{and} \qquad \omega_e := \bigcup_{t \in \mathcal{R}_e} t,$$

respectively. Note that ω_p and ω_e are subdomains of $\Omega \subset \mathbb{R}^2$, while $\mathcal{R}_p$ and $\mathcal{R}_e$ are sets of triangles which can be viewed as triangulations of ω_p and ω_e, respectively. The cardinality of a set S is denoted by $\#S$. For each vertex $p \in \mathcal{N}(\mathcal{T})$, the *valence* of p is defined as the number of triangles in $\mathcal{R}_p$, i.e., $\#\mathcal{R}_p$.

For finite element discretizations, there are two standard conditions imposed on triangulations. The first condition is the conformity. A triangulation $\mathcal{T}$ is *conforming* if the intersection of any two triangles t_1 and t_2 in $\mathcal{T}$ either consists of a common vertex, a common edge, or empty. The second condition is the shape-regularity. A set of triangulations $\mathscr{F}$ is called *shape-regular* if there exists a constant σ such that

$$\max_{t \in \mathcal{T}} \frac{\operatorname{diam}(t)}{|t|^{\frac{1}{2}}} \leq \sigma, \quad \text{for all } \mathcal{T} \in \mathscr{F}, \tag{2.1}$$

where $\mathrm{diam}(t)$ is the diameter of t and $|t|$ is the area of t.

2.2. Newest vertex bisection

We review the newest vertex bisection method studied in detail by Mitchell [24, 25]. More recent study on newest vertex bisection can be found in Binev, DeVore and Dahmen [8]. A short implementation of such bisection method in MATLAB can be found in [12]; see also [13].

For each triangle $t \in \mathcal{T}$, we label one vertex of t as the *newest vertex* and call it $V(t)$. The opposite edge of $V(t)$ is called the *refinement edge* and denoted by $E(t)$. This process is called a *labeling* of $\mathcal{T}$. Starting with a labeled initial grid $\mathcal{T}_0$, newest vertex bisection follows two rules:

1. a triangle (*father*) is bisected to two new triangles (*children*) by connecting its newest vertex with the midpoint of its refinement edge;
2. the new vertex created at the midpoint of the refinement edge is labeled as the newest vertex of each child.

Once the labeling is done for an initial grid, the decent grids inherit labels according to the second rule and the bisection process can thus proceed.

For a given labeled initial grid $\mathcal{T}_0$, we define

$$\mathbb{F}(\mathcal{T}_0) := \Big\{ \mathcal{T} \,|\, \mathcal{T} \text{ is obtained from } \mathcal{T}_0 \text{ by newest vertex bisection(s)} \Big\}. \tag{2.2}$$

Sewell [32] showed that all the descendants of a triangle in $\mathcal{T}_0$ fall into four similarity classes and hence any triangulation $\mathcal{T} \in \mathbb{F}(\mathcal{T}_0)$ is shape-regular. It is worth to note that $\mathcal{T} \in \mathbb{F}(\mathcal{T}_0)$ is not necessarily conforming. Therefore we define a subset of $\mathbb{F}(\mathcal{T}_0)$:

$$\mathbb{C}(\mathcal{T}_0) := \Big\{ \mathcal{T} \in \mathbb{F}(\mathcal{T}_0) \,|\, \mathcal{T} \text{ is conforming} \Big\}. \tag{2.3}$$

We now give an example to illustrate the bisection procedure above and address the conformity issue. To begin with, we introduce some notation. The *generation* of each triangle in the initial grid is defined to be 0; once a triangle is bisected, the generations of both children are defined as one plus the generation of their father. The generation of a triangle $t \in \mathcal{T}$ will be denoted by $g(t)$. Children with the same father are called *brothers* to each other.

In Figure 2.1, we start from an initial grid $\mathcal{T}_0$ with only one triangle $t_{0,1}$. In the notation $t_{i,j}$, the first subscript i is the generation of the triangle and the second subscript j is the index of the triangle in the generation i. A vertex with a 'dot' next to it is the newest vertex of that triangle. Adaptive methods usually mark some triangles for refinement according some local error indicator. Those marked triangles are indicated by drawing in light gray and are bisected. The interesting case is that after $t_{2,1}$ is bisected, to keep conformity, we need to bisect $t_{1,2}$ and $t_{2,4}$ using newest vertex bisection. The dashed lines in the tree as well as in the grid $\mathcal{T}_3$ in Figure 2.1 means they are generated due to the conformity requirement.

2.3. A classical coarsening algorithm

The bisection procedure in Section 2.2 is fully revertible using a recursive coarsening algorithm developed by Kossaczký [21] and implemented in ALBERTA [31].

Let us still use the same example in Figure 2.1 to illustrate the classical coarsening algorithm. In the final grid $\mathcal{T}_3$, suppose we want to coarsen the triangle $t_{2,3}$, the algorithm will first find

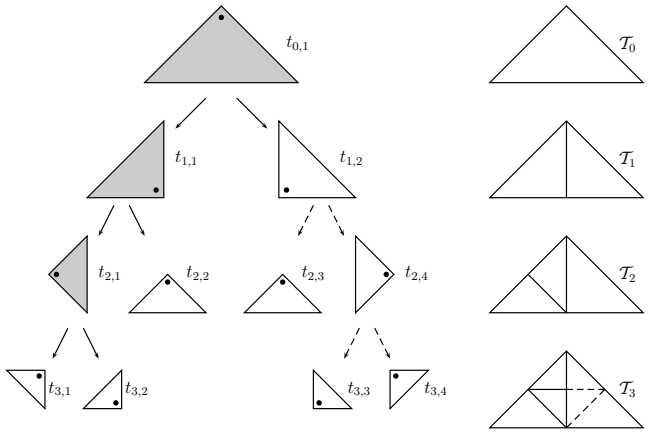


Fig. 2.1. Bisection tree (left) and its corresponding grids (right).

its neighboring triangle $t_{3,4}$ and it should be intelligent enough to tell that these two triangles are not brothers and thus cannot be glued together. The algorithm will then try to coarsen $t_{3,4}$ first. This can be done in a recursive manner. The triangle $t_{3,3}$ is found to be the brother of $t_{3,4}$. Once the algorithm glue $t_{3,3}$ and $t_{3,4}$ together to get $t_{2,4}$ back again, the grid becomes non-conforming. To keep conformity, $t_{3,1}$ and its brother should be glued together (if there is a problem with this step as before, do the same recursive step for $t_{3,1}$ first.) Once this conformity step has been completed, the algorithm returns to $t_{2,3}$ and glue it with its brother $t_{2,4}$ to obtain $\mathcal{T}_2$. To allow the algorithm to find its neighbors and so on, traversing over the bisection tree is needed.

In summary, the existing coarsening algorithm developed in [21] is element-wise and recursive; it requires tree-related data structure and algorithms. We shall propose a node-wise and non-recursive algorithm which does not store the bisection tree and requires only very simple data structures.

2.4. Compatible bisection and good-for-coarsening nodes

From the previous example discussed in Sections 2.2 and 2.3, we can see that keeping conformity of bisection grids complicates both element-wise refinement and coarsening procedures. It will be more convenient if we change our perspective and view the two procedures node-wise.

We first give a characterization of triangulations in the conforming class $\mathbb{C}(\mathcal{T}_0)$ with some assumptions on the labeling of the initial triangulation $\mathcal{T}_0$.

Let $\mathcal{T}$ be a labeled conforming mesh. Two triangles sharing a common edge are called *neighbors* to each other. A triangle t has at most three neighbors. The neighbor sharing the refinement edge of t is called the *refinement neighbor* and denoted by $F(t)$. Note that $F(t) = \varnothing$ if $E(t)$, the refinement edge of t, is on the boundary of Ω. Although $E(t) \subset F(t)$, the refinement edge of $F(t)$ could be different than $E(t)$. For example, in the triangulation $\mathcal{T}_2$ shown in Figure 2.1, for $t = t_{2,1}$, $F(t) = t_{1,2}$ and they have different refinement edges.

An element t is called *compatible* if $F(F(t)) = t$ or $F(t) = \varnothing$. For a compatible element, its refinement edge e is called a *compatible edge*, and ω_e is called a *compatible patch*. By this definition, if e is a compatible edge, the first ring $\mathcal{R}_e$ is either a pair of two triangles sharing the same refinement edge or one triangle whose refinement edge is on the boundary. In both cases,

bisection of triangles in the first ring $\mathcal{R}_e$ will preserve the conformity (or called compatibility) of a conforming triangulation; we call such a bisection a *compatible bisection*. Mathematically, we define the compatible bisection as a map $b : \mathcal{R}_e \to \mathcal{R}_p$, where $\mathcal{R}_p$ is the first ring of the new point p introduced in the bisection. We note that the inverse map $b^{-1} : \mathcal{R}_p \to \mathcal{R}_e$ can be thought as a coarsening step. It is restricted to the local patch ω_p and thus no conformity issue arises. See Figure 2.2 for an illustration. In this figure, the edges in boldface are the refinement edges and dash-lines represent bisections.

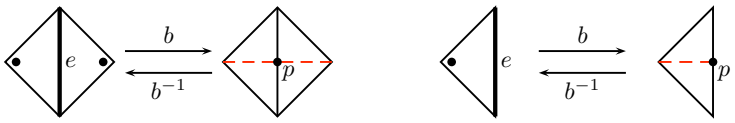


Fig. 2.2. Two examples of compatible bisections. Left: interior edge; right: boundary edge.

If we have access to all compatible bisections, we can easily perform a node-wise coarsening. The question is how to find the node introduced by a compatible bisection without recording all compatible bisections. To this end, we introduce a new concept:

Definition 1 [Good-for-coarsening Node] *For a triangulation $\mathcal{T} \in \mathbb{C}(\mathcal{T}_0)$, a node $p \in \mathcal{N}(\mathcal{T})$ is called a good-for-coarsening node, or a good node in short, if there exists a compatible bisection b and a compatible patch $\mathcal{R}_e$ such that $\mathcal{R}_p = b(\mathcal{R}_e)$. The set of all good nodes in the grid $\mathcal{T}$ is denoted by $\mathcal{G}(\mathcal{T})$.*

2.5. Existence of good nodes on compatibly labeled grids

In general, the set of good nodes $\mathcal{G}(\mathcal{T})$ could be empty; see Figure 2.3 for such an example. This example indicates that the labeling for the initial triangulation cannot be selected freely and it is necessary to impose some conditions on the initial labeling.

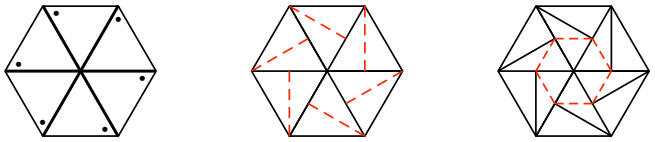


Fig. 2.3. Bisections on a non-compatible triangulation. Left: a non-compatible labeling; middle: one bisection on each triangle; right: bisections for conformity.

If all elements in $\mathcal{T} \in \mathbb{F}(\mathcal{T}_0)$ have the same generation k, $\mathcal{T}$ is called the k-th uniform refinement of $\mathcal{T}_0$ and is denoted by $\overline{\mathcal{T}}_k$. Note that $\overline{\mathcal{T}}_k$ might not be conforming; see Figure 2.3 (middle) for example. The generation of each element in the initial grid $\mathcal{T}_0$ is defined to be 0, and the generation of an element $\tau \in \mathbb{F}(\mathcal{T}_0)$ is 1 plus that of the father. Let

$$\mathscr{P}(\mathcal{T}_0) = \left\{ \cup \mathcal{N}(\mathcal{T}) : \mathcal{T} \in \mathbb{F}(\mathcal{T}_0) \right\}$$

denote the set of all possible nodes. For any node $p \in \mathscr{P}(\mathcal{T}_0)$, we define the *generation* of p, denoted by $g(p)$, to be the minimal integer k such that $p \in \mathcal{N}(\overline{\mathcal{T}}_k)$.

We call a grid $\mathcal{T}$ *compatibly labeled* if every element in $\mathcal{T}$ is compatible and call such a labeling of $\mathcal{T}$ a *compatible labeling*. The following theorem shows the existence of good nodes and gives a practical characterization of good nodes if the initial grid $\mathcal{T}_0$ is compatibly labeled.

Theorem 2.1 (Existence and Characterization of Good Nodes) *Let $\mathcal{T}_0$ be a compatibly labeled conforming triangulation. For any $\mathcal{T} \in \mathbb{C}(\mathcal{T}_0)$ and $\mathcal{T} \neq \mathcal{T}_0$, let*

$$\mathcal{M}_1(\mathcal{T}) := \{p \in \mathcal{N}(\mathcal{T}) : g(p) \geq 1, g(p) = \max_{q \in \mathcal{N}(\mathcal{R}_p)} g(q)\},$$

$$\mathcal{M}_2(\mathcal{T}) := \{p \in \mathcal{N}(\mathcal{T}) : p \notin \mathcal{N}(\mathcal{T}_0) \text{ and } p = V(t)\ \forall t \in \mathcal{R}_p\}.$$

Then $\mathcal{G}(\mathcal{T}) = \mathcal{M}_1(\mathcal{T}) = \mathcal{M}_2(\mathcal{T}) \neq \varnothing$.

Remark 2.1 (Compatible Initial Labeling) The assumption "$\mathcal{T}_0$ is compatibly labeled" is not restrictive. In fact, Mitchell [24] proved that for any conforming triangulation $\mathcal{T}$, there exists a compatible labeling. Biedl et al. [5] give an $\mathcal{O}(N)$ algorithm to find a compatible labeling for a triangulation with N elements. This assumption can be further relaxed by using the longest edge of each triangle as its refinement edge for the initial triangulation $\mathcal{T}_0$; see Kossaczký [21]. Note that such conditions are also needed in the proof of convergence and optimality of adaptive finite element methods [11].

To prove Theorem 2.1, we first study a property on uniform refinements.

Lemma 2.1 (Uniform Refinements on A Compatible Mesh) *If $\mathcal{T}_0$ is conforming and compatibly labeled, then every uniform refinement $\overline{\mathcal{T}}_k$ of $\mathcal{T}_0$ is also conforming and compatibly labeled.*

Proof. We prove it by induction of k. For $k = 0$, $\mathcal{T}_0$ is conforming and compatibly labeled. Suppose $\overline{\mathcal{T}}_{k-1}$ is conforming and compatibly labeled, we will show so is $\overline{\mathcal{T}}_k$. Since $\overline{\mathcal{T}}_{k-1}$ is compatibly labeled, after bisecting every element of $\overline{\mathcal{T}}_{k-1}$, we obtain $\overline{\mathcal{T}}_k$ which is conforming. We only need to prove $\overline{\mathcal{T}}_k$ is also compatibly labeled.

We pick up a triangle $t \in \overline{\mathcal{T}}_k$. If $F(t) = \varnothing$, t is compatible by definition. We now consider the case $F(t) \neq \varnothing$. Denoted by the father of a triangle t by $father(t)$. Since t is refined

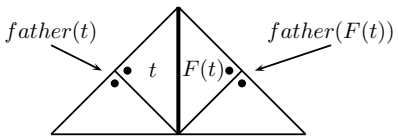


Fig. 2.4. Refinement edges of t and $F(t)$.

from its father, $E(t) \neq E(father(t))$. By the same reason, $E(F(t)) \neq E(father(F(t)))$. By the conformity of $\overline{\mathcal{T}}_{k-1}$, $E(t)$ is an edge of both $father(t)$ and $father(F(t))$. In the triangle $father(F(t))$, it is evident that after the bisection, the refinement edge of $F(t)$ is also $E(t)$; see Figure 2.4 for an illustration of one possible configuration of t and $F(t)$. □

If we begin with a compatibly labeled initial triangulation $\mathcal{T}_0$, then for any $p \in \mathcal{N}(\overline{\mathcal{T}}_k)$ with $k = g(p)$, $k \geq 1$, p is introduced by a compatible bisection from $\overline{\mathcal{T}}_{k-1}$ and thus p must be a good node. More precisely, if we denote the ring of p in $\overline{\mathcal{T}}_k$ as $\overline{\mathcal{R}}_{k,p}$ with $k = g(p)$ and the first ring of $e \in \mathcal{E}(\overline{\mathcal{T}}_{k-1})$ by $\overline{\mathcal{R}}_{k-1,e}$, then $b_e : \overline{\mathcal{R}}_{k-1,e} \to \overline{\mathcal{R}}_{k,p}$ is the compatible bisection which introduces p.

Remark 2.2 (Generation of Elements) Let $\mathcal{T}_0$ be a compatibly labeled triangulation and $\mathcal{T} \in \mathbb{C}(\mathcal{T}_0)$, but $\mathcal{T} \neq \mathcal{T}_0$. For any $t \in \mathcal{T} \backslash \mathcal{T}_0$, $V(t)$, the newest vertex of t, is introduced later than other vertices of t and thus $g(V(t)) > g(p)$, for any vertex p of t and $p \neq V(t)$. Since $t \in \overline{\mathcal{T}}_{g(t)}$ and $t \notin \overline{\mathcal{T}}_k$ for $k < g(t)$, we conclude that $g(t) = g(V(t))$.

Now we are at the position to complete the proof of Theorem 2.1.

Proof of Theorem 2.1. Let $p^* \in N(\mathcal{T})$ such that $g(p^*) = \max_{q \in \mathcal{N}(\mathcal{T})} g(q)$. Then $p^* \in \mathcal{M}_1(\mathcal{T})$ and thus $\mathcal{M}_1(\mathcal{T})$ is non-empty.

We then show the equivalence of $\mathcal{M}_1(\mathcal{T})$ and $\mathcal{M}_2(\mathcal{T})$. It is obvious that $g(p) \geq 1$ is equivalent to $p \notin \mathcal{N}(\mathcal{T}_0)$. Let us pick up a node $p \in \mathcal{M}_1(\mathcal{T})$. Suppose there is a triangle $t \in \mathcal{R}_p$ and $V(t) \neq p$. Then we have $g(V(t)) > g(p)$ which contradicts with the choice of p. So we conclude p is the newest vertex of all triangles in $\mathcal{R}_p$. On the other hand, if p is the newest vertex of all elements in $\mathcal{R}_p$, then $g(p) = g(V(t)) \geq g(q)$ for any $t \in \mathcal{R}_p, q \in \mathcal{N}(t)$, i.e., $g(p)$ will be a local maximum in $\mathcal{R}_p$. This finishes the proof of the equivalence of $\mathcal{M}_1(\mathcal{T})$ and $\mathcal{M}_2(\mathcal{T})$.

The proof of $\mathcal{G}(\mathcal{T}) \subseteq \mathcal{M}_2(\mathcal{T})$ is straightforward. Since p is introduced by a compatible bisection, p should be the newest vertex of all $t \in \mathcal{R}_p$.

To complete the proof, we now prove $\mathcal{M}_1(\mathcal{T}) \subseteq \mathcal{G}(\mathcal{T})$. Let $p \in \mathcal{M}_1(\mathcal{T})$. For any $t \in \mathcal{R}_p$, $g(t) = g(V(t)) = g(p)$ and consequently $\mathcal{R}_p \subseteq \overline{\mathcal{R}}_{k,p}$. Since ω_p is hormophism to a disk (interior node) or half disk (boundary node) with the center p, we conclude $\mathcal{R}_p = \overline{\mathcal{R}}_{k,p}$ and thus p is a good node. □

Remark 2.3 (Characterization in 2-D) In $\mathbb{R}^2$, there are only two possibilities for compatible bisections, for $p \in \mathcal{G}(\mathcal{T})$, $\#\mathcal{R}_p = 4$ or $\#\mathcal{R}_p = 2$. This characterization will help us to find out all good nodes without recording the generation; see Section 3.2.

Remark 2.4 (Generalization to 3-D) Existence of good nodes (Theorem 2.1) can be easily generalized to three or higher dimensions, if we can choose an initial labeling of $\mathcal{T}_0$ such that all uniform refinement $\overline{\mathcal{T}}_k, k \geq 1$ are conforming. However, we need to record the generation of nodes and extra information; see [13] for details.

2.6. A node-wise coarsening algorithm

Formally, our new coarsening algorithm simply reads:

```
ALGORITHM COARSEN (T)
   Find all good nodes G(T)  of T.
   For each good node p ∈ G(T)
     Replace the first ring R_p  by b_e^{-1}(R_p).
END
```

We postpone the discussion on implementation of this algorithm to the next section and continue theoretical discussions on the coarsening algorithm. An important question is whether we can finally obtain the initial grid back by repeatedly applying this coarsening algorithm. The answer is positive and a rigorous discussion is given in the following theorem.

Theorem 2.2 (Coarsening Theorem) *Let $\mathcal{T}_0$ be a compatibly labeled conforming triangulation. For any $\mathcal{T} \in \mathbb{C}(\mathcal{T}_0)$, there exists a positive integer $L \leq \#\mathcal{N}(\mathcal{T}) - \#\mathcal{N}(\mathcal{T}_0)$ such that by applying the algorithm* COARSEN *at most L times, we can recover $\mathcal{T}_0$.*

Proof. If $\mathcal{T} = \mathcal{T}_0$, we can simply choose $L = 0$. When $\mathcal{T} \neq \mathcal{T}_0$, by the existence of good nodes (Theorem 2.1), we obtain a new grid $\mathcal{T}' = \texttt{COARSEN}(\mathcal{T})$ with $\#\mathcal{N}(\mathcal{T}') < \#\mathcal{N}(\mathcal{T})$ and $\mathcal{T}' \in \mathbb{C}(\mathcal{T}_0)$ since only good nodes are removed.

If $\mathcal{T}' \neq \mathcal{T}_0$, we can continue applying the COARSEN algorithm on $\mathcal{T}'$. Therefore with at most $L = \#\mathcal{N}(\mathcal{T}) - \#\mathcal{N}(\mathcal{T}_0)$ steps, we obtain $\mathcal{T}_0$. □

Remark 2.5 (Refinement Length) In the theorem above, the worst case scenario is $L = \#\mathcal{N}(\mathcal{T}) - \#\mathcal{N}(\mathcal{T}_0)$. Our numerical examples strongly indicates that at each step the decrease of the number of nodes is at the ratio about 0.5. Namely for most bisection triangulations, half of the nodes are good nodes. See Section 4 (Table 4.1) for some numerical evidence.

Remark 2.6 (Coarsening and Refinement) It is possible that the algorithm COARSEN applied on the current grid $\mathcal{T}$ gives a grid which is not in the adaptive history. Indeed our coarsening algorithm may remove nodes added in several different stages of the adaptive procedure.

3. Data Structures and Implementation

In this section, we present a MATLAB implementation of the proposed algorithm, COARSEN.

3.1. Data structures

There is a dilemma when designing data structures in the implementation level. Sophisticated data structures can be used to facilitate traversing on the mesh more easily; for example, saving all elements surrounding a node p makes finding $\mathcal{R}_p$ simple. On the other hand, if we do so, after each bisection and coarsening step, we have to update these data structures which in turn makes the computational overhead heavier and complicates the implementation. We decide to use minimal data structures for the mesh and regenerate auxiliary data structures when necessary.

3.1.1. Basic data structure

The matrices `node(1:N,1:2)` and `elem(1:NT,1:3)` are used to represent a two dimensional triangulation, where N is the number of vertices and NT is the number of elements. In the node matrix `node`, the first and second columns contain x- and y-coordinates of the nodes in the mesh. In the element matrix `elem`, the three columns contain indices to the vertices of elements. These two matrices represent two different structures of a triangulation: `elem` for the topological connectivity and `node` for the geometric embedding of vertices. As an example, `node` and `elem` matrices to represent a triangulation of the L-shape domain $\Omega = (-1,1) \times (-1,1) \backslash ([0,1] \times [0,-1])$ are given in the Figure 3.1 (a) and (b).

3.1.2. Assumptions on ordering

An important feature of our implementation is that we only maintain `node` and `elem` matrices. At a first glance, one might think it is impossible to coarsen an adaptive mesh without storing the refinement history. Our trick is: a tree structure of the adaptive procedure can be built into the `elem` matrix by the ordering. We shall make it more precisely in the follows.

Suppose p_1, p_2, and p_3 are three vertices of a triangle t and p_4 is the midpoint of the refinement edge $E(t)$. After t is bisected, we name the new element with vertices p_1, p_2, and

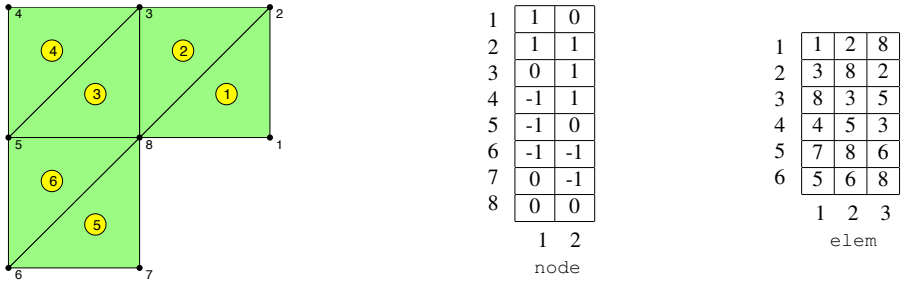


Fig. 3.1. (Left) is a triangulation of the L-shape domain $(-1,1)\times(-1,1)\backslash([0,1]\times[0,-1])$ and (Right) is its representation using `node` and `elem` matrices.

p_4 the *left* child and the other the *right* child. (Left or right is with respect to the direction walking from p_4 to p_1.) For example, in Figure 2.1, the left children always appear left to their brothers (right children).

Any permutation of vertices of t will represent the same triangle. By a convention, three vertices of a triangle are ordered counter-clockwise such that the signed once is positive. Even with such ordering requirement, an even permutation of vertices is still allowed. Also a permutation of indices of triangles, i.e., rows of `elem` matrix, still represents the same triangulation. The ordering of the row and column indices of `elem` matrix provides a room to store additional information.

We impose the following three assumptions on the ordering of `node` and `elem` matrices which can easily be built into the bisection procedure.

(O1) `elem(t,1)` stores the newest vertex of element `t`.

(**O2**) When a triangle is bisected, its left child is stored in a position prior to its right child in the new `elem` matrix.

(**O3**) The nodes in the initial triangulation $\mathcal{T}_0$ is stored in the range `node(1:N0,:)`.

3.1.3. Auxiliary data structures

We introduce two auxiliary data structures: `valence` and `edge2elem`. We do not maintain these auxiliary data structures during refinement nor coarsening. Instead, we rebuild `edge2elem` and `valence` in the beginning of our algorithm. Since we make use of built-in functions in MATLAB, the construction of those data structure is simple yet efficient.

First, we use the array `valence` to record the number of the triangles in the first ring of a node. It will be used to find all good nodes.

Second, we use an `N` $\times$ `N` sparse matrix `edge2elem` to store the mapping from edges to element; see Table 3.1. If p_ip_j is an edge of t, then `edge2elem(i,j)=t`. Due to the ordering of vertices, for an interior edge, `edge2elem(j,i)` will give another (if it exists) element t' such that p_jp_i is an edge of t'. If one of them is zero, it implies that this edge is on the boundary.

3.2. Code and explanation

We now present our MATLAB code for the coarsening algorithm and then explain the code in detail part by part.

Table 3.1: `edge2elem` matrix for the L-shape mesh in Figure 3.1.

1	0	1	0	0	0	0	0	0
2	0	0	2	0	0	0	0	1
3	0	0	0	4	3	0	0	2
4	0	0	0	0	4	0	0	0
5	0	0	4	0	0	6	0	3
6	0	0	0	0	0	0	5	6
7	0	0	0	0	0	0	0	5
8	1	2	3	0	6	5	0	0
	1	2	3	4	5	6	7	8

```
function [node,elem] = coarsen(node,elem,N0)
%--------------- Part 1: construct auxiliary data structures ---------------
N = size(node,1);  NT = size(elem,1);   % number of nodes and elements
edge2elem = sparse( elem(:,[1,2,3]),elem(:,[2,3,1]),[1:NT,1:NT,1:NT]);
valence = accumarray(elem(:),ones(3*NT,1),[N 1]);
%--------------- Part 2: find good-for-coarsening nodes ---------------------
newestNode = unique(elem((elem(:,1)>N0),1));    % newest vertices
isGood = find((valence(newestNode) == 2) | (valence(newestNode) == 4));
goodNode = newestNode(isGood);          % all good nodes
marker(goodNode) = 1;                   % marker for good nodes
%--------------- Part 3: coarsen good nodes ---------------------------------
for t = 1:NT                            % for loop over all elements
    p = elem(t,1);                      % newest vertex of element t
    if (p > 0)                          % p could be removed already
       if (marker(p)==1)                % p should be a good node
           brother = edge2elem(elem(t,2),p);    % brother of t
           elem(t,1) = elem(t,2);       % keep t
           elem(t,2) = elem(t,3);
           elem(t,3) = elem(brother,2);
           elem(brother,1) = 0;         % discard brother
       end                              % end of if (marker(p)==1)
    end                                 % end of if (p > 0)
end                                     % end of for loop
%--------------- Part 4: clean node and elem --------------------------------
elem((elem(:,1) == 0),:) = [];          % remove empty entries in elem
node(isGood,:) = [];                    % remove empty entries in node
indexMap = zeros(N,1);                  % initialize of indexMap
indexMap(¬isGood)= 1:size(node,1);      % index map from old node to new node
elem = indexMap(elem);                  % shift the nodal index
```

In the first part, we construct auxiliary data structures `edge2elem` and `valence`. To improve the efficiency and concise of the code, we use the `sparse` and `accumarray` functions to avoid using `for` loops. We refer to MATLAB manual (`help sparse` and `help accumarray` in the command window) for the detailed usage of these two functions.

The second part finds all good nodes in the current mesh. It is just algorithmic description of good nodes. The last part is to remove the empty entries in `elem` and `node` arrays. The clearance of `node,elem` arrays is relatively easy. But we should also shift the indices in `elem` to reflect to the change of node indices. So we build an index map from the old nodal index to the new and shortened nodal index. Then `elem` is shifted by the index map. Note that we cannot coarsen the node in the initial triangulation thus **(O3)** remains hold.

We now explain in detail the third part of our algorithm. After we find out all good nodes, we traverse in the `elem` array to find elements containing good nodes. Let t be such an element. We need to find out the brother of t which can be glued with t. By **(O2)**, if we go through all elements from 1 to `NT`, we always meet a left child before its (right) brother. It is easy to find the brother of a left child t by `brother = edge2elem(elem(t,2),elem(t,1))`.

We use $\mathcal{T}_3$ in Figure 2.1 as an example to illustrate how our coarsening algorithm works. There is only one good node p in $\mathcal{T}_3$; see Figure 3.2. We mark this node and traverse the element array `elem`. In the element array of $\mathcal{T}_3$, elements are stored in a possible order indicated by the

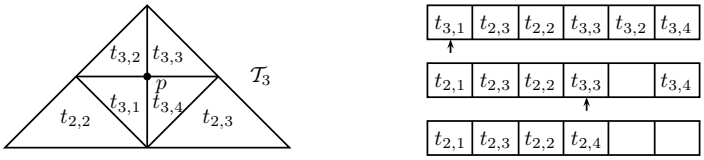


Fig. 3.2. Good nodes and coarsening procedure.

first row in the right of Figure 3.2. We will encounter firstly $t_{3,1}$ which use p as its newest vertex. We use the `edge2elem` to find its brother $t_{3,2}$ and then glue these two elements together to get $t_{2,1}$ back. The place of $t_{3,1}$ is used to store its father $t_{2,1}$ and place of $t_{3,2}$ is marked to be discarded by setting its newest vertex as 0; see the second row in Figure 3.2. After this step, p will be left as a hanging node as it is still the newest vertex of $t_{3,3}$ and $t_{3,4}$. However the traverse of element array will continue and encounter $t_{3,3}$ and glue it with $t_{3,4}$ and thus recovery the conformity. Their father $t_{2,4}$ will be stored in the place of $t_{3,3}$ which is behind $t_{2,3}$. In this way, the ordering of the coarse grid still satisfies the condition (**O2**). Finally, we get a mesh $\mathcal{T}_2'$ which is different than $\mathcal{T}_2$ in Figure 2.1 but still in the class of $\mathbb{C}(\mathcal{T}_0)$; so we can proceed as before.

4. Application in Multilevel Preconditioning

In this section, we apply the proposed coarsening algorithm to construct multilevel preconditioners. We note that most existing multilevel preconditioners on adaptive grids are developed for the regular refinement [1,9,16]. We shall make use of the special structure of bisection grids to construct a simple but efficient preconditioner.

4.1. Preliminary

Let Ω be a polygonal bounded domain in $\mathbb{R}^2$ and consider the following Dirichlet problem:

$$\begin{cases} -\mathrm{div}(\mathcal{A}(x)\nabla u) = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases} \tag{4.1}$$

where $f \in L^2(\Omega)$ and $\mathcal{A}(x)$ is a uniformly bounded and positive definite symmetric matrix function defined in Ω. The weak formulation of (4.1) reads: find $u \in H_0^1(\Omega)$ such that

$$(\mathcal{A}(x)\nabla u, \nabla v) = (f, v) \qquad \text{for all } v \in H_0^1(\Omega), \tag{4.2}$$

where $(\cdot,\cdot)$ is the L^2 inner product in Ω and $H_0^1(\Omega)$ is the usual Sobolev space of function with square integrable weak derivatives and vanishing boundary trace in Ω. We approximate (4.2)

by the linear finite element discretization. Let $\mathcal{T}$ be a conforming and shape-regular grid of the polygonal domain Ω. We define

$$\mathbb{V} = \mathbb{V}(\mathcal{T}) := \left\{ v \in H_0^1(\Omega) \,:\, v|_t \text{ is affine, for all } t \in \mathcal{T} \right\},$$

and look for a discrete solution $u_h \in \mathbb{V}$ such that

$$(\mathcal{A}(x)\nabla u_h, \nabla v_h) = (f, v_h) \quad \text{for all } v_h \in \mathbb{V}. \tag{4.3}$$

Let $\{\phi_i\}_{i=1}^N$ be the set of piecewise linear nodal basis functions for interior nodes and $u_h = \sum_{i=1}^N u_i \phi_i$. With an abuse of notation, we still denote the vector $(u_1, u_2, \ldots, u_N)^T$ by u and $(f_1, f_2, \ldots, f_N)^T$ by f. Let $A = (a_{i,j})_{i,j=1}^N \in \mathbb{R}^{N\times N}$ with $a_{ij} = (\mathcal{A}(x)\nabla\phi_j, \nabla\phi_i)$ be the stiffness matrix. We then end up with the following algebraic system

$$Au = f. \tag{4.4}$$

We shall apply the preconditioned conjugate gradient (PCG) method to solve (4.4), namely use the conjugate gradient method to solve the preconditioned system $BAu = Bf$, where B is a symmetric positive definite (SPD) matrix and known as a preconditioner. Note that we do not have to form B explicitly. Instead, given a vector r, we only need the action of B on r, i.e., the vector Br. A good preconditioner is a balance of the following two considerations:

- the conditioner number $\kappa(BA)$ is small compared with $\kappa(A)$;
- the action of B is relatively cheap to compute.

We shall construct multilevel preconditioners using the framework of space decomposition and subspace correction methods [33]. Let $\mathbb{V} = \sum_{k=0}^L \mathbb{V}_k$ be a decomposition of $\mathbb{V}$, where $\mathbb{V}_k \subset \mathbb{V}$ $(k = 0, \ldots, L)$ are subspaces of $\mathbb{V}$. Let $I_k : \mathbb{V}_k \mapsto \mathbb{V}$ be the natural inclusion operator (often known as prolongation) and $I_k^T : \mathbb{V} \mapsto \mathbb{V}_k$ be its adjoint in L^2 inner product (often known as restriction.) Let $A_k : \mathbb{V}_k \mapsto \mathbb{V}_k$ be the restriction of A on the subspace $\mathbb{V}_k$. By choosing a local subspace solver, often known as a smoother, $R_k \approx A_k^{-1}$, we obtain an additive preconditioner of the form

$$B = \sum_{k=0}^L I_k^T R_k I_k. \tag{4.5}$$

It is well-known, e.g. [33,34] that, when R_k is SPD on $\mathbb{V}_k$ under L^2 inner product, the operator B defined by (4.5) is also SPD on $\mathbb{V}$ under L^2 inner product and can be used as a preconditioner.

We now discuss the choice of R_k. Let D_k be the diagonal matrix of A_k. We choose

$$R_0 = A_0^{-1} \quad \text{and} \quad R_k = D_k^{-1}. \tag{4.6}$$

Notice that we use a direct solver on the coarsest space $\mathbb{V}_0$ since the dimension of $\mathbb{V}_0$ is small and the computational cost is negligible.

With such a choice of smoothers, the preconditioner is uniquely determined by the space decomposition. We now present several classical and new preconditioners in Section 4.2.

4.2. Space decomposition and preconditioning

Starting with $\mathcal{T}_L = \mathcal{T} \in \mathbb{C}(\mathcal{T}_0)$, we apply our coarsening algorithm iteratively, i.e., $\mathcal{T}_{k-1} = \texttt{COARSEN}(\mathcal{T}_k)$ to obtain a sequence of nested grids. Let $\mathbb{V}_k = \mathbb{V}(\mathcal{T}_k)$ be the linear finite element

space on $\mathcal{T}_k$ and $\mathcal{N}_k$ as the set of interior nodes of $\mathcal{T}_k$ for $k = 0, 1, \ldots, L$. When $k = L$, the subscript will be skipped. For a given node $x_{k,i} \in \mathcal{N}_k$, we use $\phi_{k,i}$ to denote the canonical nodal basis function at $x_{k,i}$ in $\mathcal{T}_k$. We shall denote by $\mathbb{V}_{k,i} = \text{span}\{\phi_{k,i}\}$ the one dimensional space spanned by the nodal basis on $\mathcal{T}_k$.

4.2.1. Hierarchical basis preconditioner

Recall that $\mathcal{G}_k \subset \mathcal{N}_k$ denote sets of good nodes. Let $\mathbb{W}_0 = \mathbb{V}_0$. The so-called hierarchical basis (HB) preconditioner B_{HB} by Yserentant [35,36] is obtained using the hierarchical decomposition

$$\mathbb{V} = \bigoplus_{k=0}^{L} \mathbb{W}_k \quad \text{with} \quad \mathbb{W}_k = \bigoplus_{x_{k,i} \in \mathcal{G}_k} \mathbb{V}_{k,i}, \quad k = 1, \cdots, L. \tag{4.7}$$

Since our coarsening algorithm will remove all good nodes in the current level, we have

$$\mathcal{N}_k = \mathcal{N}_{k-1} \cup \mathcal{G}_k. \tag{4.8}$$

Thus the first decomposition for $\mathbb{V}$ in (4.7) is a direct sum. Let $p_{k,i}, p_{k,j} \in \mathcal{G}_k$ be two different good nodes in $\mathcal{T}_k$. Suppose that there exists an element $t \in \mathcal{R}_{k,i} \cap \mathcal{R}_{k,j}$, then both $p_{k,i}$ and $p_{k,j}$ are the newest vertices of t which is a contradiction. So $\mathcal{R}_{k,i} \cap \mathcal{R}_{k,j} = \varnothing$, and the second decomposition in (4.7) for each $\mathbb{W}_k$ is also a direct sum. Note that this is the special property of nested bisection grids obtained by our coarsening algorithm and may not be true for nested bisection grids using the tree structure and adaptive grids obtained by the regular refinement.

The hierarchal decomposition (4.7) is of optimal computational complexity and easy to implement. It is well known [4, 35] that the preconditioner B_{HB} based on (4.7) is almost optimal in the sense that

$$\kappa(B_{\text{HB}}A) \leq CL|\log h_{\min}|, \tag{4.9}$$

where $h_{\min} = \min_{t \in \mathcal{T}} \text{diam}(t)$.

4.2.2. BPX preconditioner

To stabilize the HB preconditioner, Bramble, Pasciak and Xu [10] propose to use the so-called BPX preconditioner B_{BPX} based on the space decomposition

$$\mathbb{V} = \sum_{k=0}^{L} \mathbb{V}_k \quad \text{with} \quad \mathbb{V}_k = \sum_{x_{k,i} \in \mathcal{N}_k} \mathbb{V}_{k,i}, \quad k = 1, \ldots, L. \tag{4.10}$$

It is well known [16, 29, 33] that there exists a constant C independent of the problem size such that

$$\kappa(B_{\text{BPX}}A) \leq C. \tag{4.11}$$

The decomposition (4.10), however, has lots of overlapping. For adaptive grids, it is possible that $\mathbb{V}_k$ results from $\mathbb{V}_{k-1}$ by just adding a handful of basis functions (maybe even only one). Thus smoothing on both $\mathbb{V}_k$ and $\mathbb{V}_{k-1}$ leads to a lot of redundancy. In the worst scenario, the complexity of smoothing could be as bad as $\mathcal{O}(N^2)$ [26].

4.2.3. Three-point hierarchical basis preconditioner

We note that the difference between (4.7) and (4.5) is that the nodes set for finite element spaces in the k-th level. One natural idea is to choose nodal sets $\mathcal{S}_k$ such that

$$\mathcal{G}_k \subset \mathcal{S}_k \subset \mathcal{N}_k, \quad k = 1, \ldots, L. \tag{4.12}$$

On the one hand, $\mathcal{S}_k$ is chosen more than $\mathcal{G}_k$ to stabilize the decomposition. On the other hand, $\mathcal{S}_k$ should have the same order of cardinality as $\mathcal{G}_k$ to preserve optimal complexity.

Let $p_i \in \mathcal{G}_k$ be the midpoint of E_i. Let $p_{i,1} = p_i$ and $p_{i,2}, p_{i,3}$ be the two end nodes of E_i (or the so-called parents of p_i). We define $\mathcal{S}_k := \{p_{i,1}, p_{i,2}, p_{i,3} \,|\, p_i \in \mathcal{G}_k\}$, $\widetilde{\mathbb{W}}_0 = \mathbb{V}_0$, and the decomposition

$$\mathbb{V} = \sum_{k=0}^{L} \widetilde{\mathbb{W}}_k \quad \text{with} \quad \widetilde{\mathbb{W}}_k = \sum_{p \in \mathcal{S}_k} \mathbb{V}_p, \quad k = 1, \cdots, L, \tag{4.13}$$

where $\mathbb{V}_p = \operatorname{span}\{\phi_p\}$ is the space spanned by the nodal basis functions on $\mathcal{T}_k$.

We call the corresponding multilevel preconditioner. Three-point smoothing (TPS) preconditioner and denoted by B_{TPS}. It is obvious that $\#\mathcal{S}_k = 3\#\mathcal{G}_k$ and thus the computational complexity of B_{TPS} is at most three times of B_{HB}. In [14], it is proved that

$$\kappa(B_{\mathrm{TPS}} A) \leq C. \tag{4.14}$$

We conclude that (4.13) is a stable decomposition with optimal complexity.

4.2.4. Locally orthogonal hierarchical basis preconditioner

We propose another improvement over the HB decomposition (4.7) by enhancing the coarse space. For convenience of presentation, we now give a local index of the vertices in ω_{p_i}; see Figure 4.1 for a pictorial description.

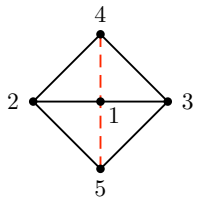


Fig. 4.1. Local indices of nodes for a compatible bisection.

Consider the local patch ω_{p_i} and the piecewise linear function space

$$\mathbb{V}_{k,i} := \mathbb{V}_k(\omega_{p_i}) = \operatorname{span}\{\phi_{i,j} \,|\, j = 1, \ldots, J_i\},$$

where $J_i = 5$ if p_i is an interior point and $J_i = 3$ if p_i is on the boundary. We define

$$\psi_{i,j} := \phi_{i,j} + \alpha_{i,j}\phi_{i,1} \in \mathbb{V}_k(\omega_{p_i}) \quad \text{and} \quad \alpha_{i,j} = -\frac{(\phi_{i,j}, \phi_{i,1})_A}{(\phi_{i,1}, \phi_{i,1})_A},$$

such that $(\psi_{i,j}, \phi_{i,1})_A = 0$. We construct a local A-orthogonal decomposition

$$\mathbb{V}_{k,i} = \operatorname{span}(\phi_{x_i}) \oplus \mathbb{Q}_{k-1,i}, \tag{4.15}$$

where $\mathbb{Q}_{k-1,i} := \{\psi_{i,2}, \psi_{i,3}, \ldots, \psi_{i,J_i}\} \subset \mathbb{V}_k(\omega_{p_i})$. We name this preconditioner corresponding to (4.15) as *Locally Orthogonal Hierarchical Basis* (LOHB) preconditioner. This simple modification gives a much better hierarchical basis preconditioner.

In fact, we should point out the equivalence of LOHB preconditioner and hierarchical basis multigrid (HBMG) [4], where the special hierarchical structure of bisection grids using our coarsening algorithm plays an important role. Thus we can estimate the condition number of $\kappa(B_{\mathrm{LOHB}}A)$ using the results by Bank, Dupont and Yserentant [4]. More precisely, in our setting, suppose $\mathcal{A}(x)$ is piecewise constant on the coarse mesh $\mathcal{T}_0$, then

$$\kappa(B_{\mathrm{LOHB}}A) \leq CL|\log h_{\min}|, \tag{4.16}$$

and the constant C is independent of the jump of diffusion coefficients and the size of the linear system. Although (4.16) still depends on the mesh size, the computational results show that the LOHB preconditioner outperforms the other preconditioners.

Remark 4.1 (Implementation of Prolongation and Restriction) The local prolongation operator, $\mathcal{J}_{k-1}^k : \mathbb{Q}_{k-1,i} \to \mathbb{V}_{k,i}$ is given by

$$(\mathcal{J}_{k-1}^k u)(p_{i,1}) = \begin{cases} \displaystyle\sum_{j=2}^{J_i} \alpha_{i,j} u(p_{i,j}) & p_{i,1} \in \mathcal{G}_k \\ u(p_{i,1}) & p_{i,1} \in \mathcal{N}_k \backslash \mathcal{G}_k. \end{cases}$$

The restriction operator will be the transpose of the prolongation operator. Algorithmically, it is a simple modification of HB preconditioner.

We present the following algorithm for the LOHB preconditioner B_{LOHB}. We shall use e and r to indicate that we are solving the residual equation $Ae = r$ in each subspace. The algorithm will compute Br for a given vector r.

```
Algorithm e = LOHB(r)
r_L = r
for k = L : 1
    r_{k-1} = (J_{k-1}^k)^t r_k        % restriction
end
e_0 = A_0^{-1} r_0                     % exact solver
for k = 1 : L
    e_k = R_k r_k                      % local smoother
    e_k = e_k + J_{k-1}^k e_{k-1}      % prolongation
end
e = e_L
END Algorithm
```

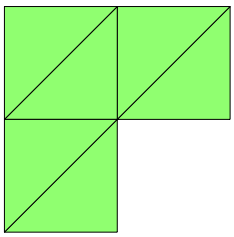
4.3. Numerical examples

We use the residual-type error estimator introduced by Babuška and Miller [2] for general second-order elliptic equations. The bulk marking strategy by Dörfler [17] with $\theta = 0.3$ is used in our simulation for marking. For comparison, we always start the preconditioned conjugate gradient (PCG) methods from the zero initial guess and the stopping criteria is the relative residual error is less than $\mathrm{tol} = 10^{-6}$. All numerical experiments are performed with MATLAB 7.0 on a PC with Intel Pentium IV 1.0GHz and 1GB RAM.

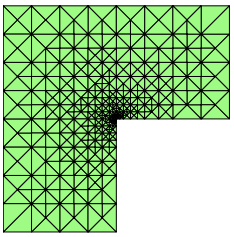
Example 1: Poisson equation on L-shaped domain

In this example, we consider the second-order elliptic equation (4.1) with $\mathcal{A} \in \mathbb{R}^{2\times 2}$ being the identity matrix and $f = 0$ on a L-shaped domain $\Omega := (-1,1)^2 \backslash \{[0,1) \times (-1,0]\}$ with a reentrant corner. We choose the Dirichlet boundary condition g such that the exact solution to be $u(r,\theta) = r^{\frac{2}{3}} \sin(\frac{2}{3}\theta)$ in polar coordinates. It is well-known that the solution $u \in H^s(\Omega)$ for $s < \frac{5}{3}$ has a corner singularity at the origin.

We start the adaptive finite element method from a compatibly labeled initial grid $\mathcal{T}_0$ (Figure 4.2(a)). An example adaptive grid is given in Figure 4.2(b). Since there is a point-singularity,



(a) Initial grid: isosceles triangles.



(b) Adaptive grid obtained by newest vertex bisections.

Fig. 4.2. Initial and refined meshes used in Example 1.

the adaptive refinement are done very locally (see Figure 4.2(b)). It is interesting to find out how many good-for-coarsening nodes we actually have on each level for highly graded adaptive grids generated by the AFEM loop (1.1). The results are reported in Table 4.1, from which we can see that degree of freedom (DOF) on each level (generated by our coarsening algorithm) decreases geometrically as for the uniform refinement case. Since the decay rates $\alpha = \mathrm{DOF}_{k-1}/\mathrm{DOF}_k$ are almost constant for any two consecutive levels, we only show the rates for the last two levels in the table.

Table 4.1 suggests that we only need to call the coarsening algorithm iteratively a few times to obtain a coarse enough grid. For example, for this problem, after 5 steps of coarsening, the degree of freedom left is only about 3% of the original number of unknowns.

Table 4.1: Number of good nodes on each level in five different adaptive grids for Example 1.

DOF	9628	13339	18648	26097	36528
level: J	4586	6365	8934	12532	17793
level: $J-1$	2402	3393	4699	6572	9164
level: $J-2$	1276	1749	2425	3448	4733
level: $J-3$	684	943	1293	1765	2425
level: $J-4$	375	486	696	945	1293
Decay rate	0.548	0.515	0.538	0.535	0.533

We present number of iterations for PCG with CPU time in the bracket using different preconditioners in Table 4.2. From this table, we have a couple of observations: (1) The iteration number of HB preconditioner increases slightly as DOF increases because the decomposition (4.7) is not stable. (2) The BPX preconditioner is uniform, but it could take more CPU time than the HB preconditioner. The advantage of the former is more significant for

Table 4.2: Number of iterations (CPU time in seconds) by PCG (initial guess $u_0 = 0$ and tol $= 10^{-6}$) with different preconditioners for Example 1.

DOF	9628	13339	18648	26097	36528
HB	31 (0.37)	31 (0.48)	31 (0.67)	36 (0.98)	36 (1.42)
BPX	23 (0.48)	23 (0.71)	22 (0.92)	25 (1.48)	25 (2.32)
TPS	18 (0.35)	19 (0.50)	18 (0.71)	20 (0.87)	20 (1.40)
LOHB	10 (0.21)	10 (0.29)	10 (0.37)	11 (0.54)	11 (0.79)

large problems. (3) The TPS is a good balance of HB and BPX. (4) The LOHB is the best among the four in terms of CPU time and iteration steps.

Example 2: Discontinuous coefficient problem

In this example, we employ a test example designed by Kellogg [20] with discontinuous diffusion coefficient. Consider the partial differential equation (4.1) with $\Omega = (-1, 1)^2$ and the coefficient matrix $\mathcal{A}$ is piecewise constant: in the first and third quadrants, $\mathcal{A} = a_1 I$; in the second and fourth quadrants, $\mathcal{A} = a_2 I$. For $f = 0$, the exact solution in polar coordinates has been chosen to be $u(r, \theta) = r^\gamma \mu(\theta)$, where

$$\mu(\theta) = \begin{cases} \cos\left((\frac{\pi}{2} - \sigma)\gamma\right) \cos\left((\theta - \frac{\pi}{2} + \rho)\gamma\right) & \text{if } 0 \le \theta \le \frac{\pi}{2}, \\ \cos(\rho\gamma) \cos\left((\theta - \pi + \sigma)\gamma\right) & \text{if } \frac{\pi}{2} \le \theta \le \pi, \\ \cos(\sigma\gamma) \cos\left((\theta - \pi - \rho)\gamma\right) & \text{if } \pi \le \theta \le \frac{3\pi}{2}, \\ \cos\left((\frac{\pi}{2} - \rho)\gamma\right) \cos\left((\theta - \frac{3\pi}{2} - \sigma)\gamma\right) & \text{if } \frac{3\pi}{2} \le \theta \le 2\pi, \end{cases}$$

and the constants

$$\gamma = 0.1,\ \rho = \pi/4,\ \sigma = -14.9225565104455152,\ a_1 = 161.4476387975881,\ a_2 = 1.$$

We see that the solution u produces a very strong singularity at the origin (barely in $H^1(\Omega)$). See Figure 4.3 for an example of adaptive grids and its associated finite element solution.

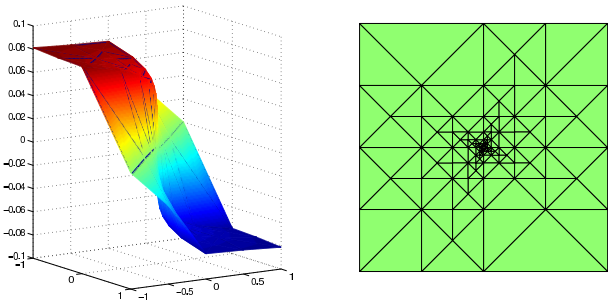


Fig. 4.3. Finite element solution u_h (left) on the adaptive grid with DOF = 164 (right).

The point singularity in the example is much stronger than the one in the previous test example. The adaptive grids are extensively concentrated at the origin. Due to this effect, the number of marked elements are quite small each iteration. The number of good nodes on first 5 levels are shown in Table 4.3. The decay rate of the number of DOF is slight worse than before but still close to a constant. The number of iterations required for each method are listed in

Table 4.3: Number of good nodes on each level in five adaptive grids for Example 2.

DOF	7095	9708	13726	19821	28956
level: J	2351	3280	4792	7068	10569
level: $J-1$	1780	2364	3245	4624	6648
level: $J-2$	1067	1443	2029	2884	4147
level: $J-3$	662	873	1234	1765	2554
level: $J-4$	393	552	788	1132	1666
Decay rate	0.593	0.632	0.638	0.641	0.652

Table 4.4: Number of iterations (CPU time in seconds) by PCG (initial guess $u_0 = 0$ and tol $= 10^{-6}$) with different preconditioners for Example 2.

DOF	7095	9708	13726	19821	28956
HB	26 (0.26)	25 (0.40)	27 (0.62)	27 (1.0)	33 (1.51)
BPX	24 (0.48)	24 (0.60)	24 (0.85)	24 (1.48)	27 (1.93)
TPS	19 (0.29)	19 (0.42)	19 (0.56)	19 (0.89)	21 (1.26)
LOHB	10 (0.17)	10 (0.23)	10 (0.32)	10 (0.59)	10 (0.68)

Table 4.4. We observe similar behaviors as in Example 1. Especially the LOHB is the best among the four in terms of CPU time and iteration steps.

From the experiments above, we have already seen that the LOHB preconditioner performs the best. Now we want to check how sensitive it is to the magnitude of the jump in the coefficient matrix $\mathcal{A}$. We use the same domain with $f = 1$ and $g = 0$. We keep $a_1 = 1$ and change a_2 from 1 to 10^4 on the same grid (uniform refinement by newest vertex bisections.) From Table 4.5, we can see that the preconditioner B_{LOHB} is robust with respect to the size of jumps.

Table 4.5: Number of iterations by PCG (initial guess $u_0 = 0$ and tol $= 10^{-6}$) with the LOHB preconditioner for Example 2.

DOF	961	1985	3969	8065	16129
$a_2 = 1$	10	8	9	11	9
$a_2 = 10$	10	9	10	11	10
$a_2 = 10^2$	10	9	10	11	10
$a_2 = 10^3$	10	9	10	11	10
$a_2 = 10^4$	10	9	10	11	10

5. Application in Time Adaptive Mesh Refinement

When solving time dependent problems with local features, it is usually difficult if even possible to design optimal meshes a priori. Hence, adaptive mesh refinement and adaptive time stepping are important to achieve optimal complexity. In order to obtain nearly optimal meshes for time dependent problems, the **COARSEN** step in (1.1) is crucial as local features often move in time. Standard coarsening algorithms (see [31] for details) requires data structures to store a refinement tree in order to keep shape regularity after coarsening. As we have seen

before, the proposed new coarsening algorithm, on the contrary, does not need refinement tree information.

We take an example from Chen and Jia [15] to test the performance of the proposed coarsening algorithm when applied to adaptive mesh refinement. Consider the heat equation with Dirichlet boundary condition in two spatial dimensions for $u(x, s)$:

$$\frac{du}{ds} - \Delta u = f \qquad x \in \Omega,\ s \in (0, T], \tag{5.1}$$

where $\Omega := (-1, 1) \times (-1, 1)$ and $T = 1$. We choose the right hand side function $f(x, s)$ such that the exact solution

$$u(x, s) = \beta(s) \exp(-25|x - \alpha(s)|^2)$$

with

$$\alpha(s) = s - 0.5, \quad \beta(s) = 0.1 \left(1 - \exp(-10^4 \alpha(s)^2)\right).$$

A posteriori error estimations and adaptive algorithms for linear parabolic problems have been discussed by many researchers [6, 7, 15, 18, 19, 22, 23, 28, 30]. Traditionally we write a posterior error estimators in element-wise which is more convenient for marking elements with large local error for refinement. We could also easily rewrite error estimators side-wise or node-wise. There are also error estimators which are intrinsically node-wise; see [27] for example. A genuine a posterior error estimators for parabolic problems can be written as follows

$$\int_0^T \|\!| u - U_h |\!\|_\Omega^2 \, ds \le C \Big\{ \eta_{\text{init}}^2 + \sum_{n=1}^{N} k_n \Big((\eta_{\text{space}}^n)^2 + (\eta_{\text{time}}^n)^2 + (\eta_{\text{coarse}}^n)^2 \Big) \Big\},$$

where, u is the solution to (5.1), U_n is an finite element approximation of u, and, as their names suggest, η_{init} is an initial error estimator, η_{space}^n is a spatial error estimator, η_{time}^n is a time error estimator, and η_{coarse}^n measures error introduced by coarsening; see [31] for details.

We now briefly discuss the node-wise time-space adaptive mesh refinement scheme for time dependent problems. Note that we assume that the newest vertex bisection as our refinement algorithm, see Algorithm 5.1.

Remark 5.1 (An example of error indicators for the heat equation) For completeness, we now give node-wise error indicators adapted from [27] for the heat equation (5.1):

$$\begin{cases} \eta_{\text{init}}(p) = \|U_h^0 - u(0)\|_{\omega_p} \\ \eta_{\text{space}}^n(p) = \|h^{\frac{1}{2}} J_h^n\|_{\gamma_p} + \|h(f^n - \widetilde{f}_p^n)\|_{\omega_p} \\ \eta_{\text{coarse}}^{n-1}(p) = \|\nabla(U_h^{n-1} - I^n U_h^{n-1})\|_{\omega_p} \\ \eta_{\text{time}}^n = \|\nabla(U_h^n - I^n U_h^{n-1})\|_\Omega, \end{cases}$$

where the superscript n refers to the time level. $I^n : \mathbb{V}(\mathcal{T}^{n-1}) \to \mathbb{V}(\mathcal{T}^n)$ is the standard transfer operator, h is the local mesh size, γ_p is the set all interior edges of ω_p, J_h^n is the jump of gradient of U_h^n over interior edges, and $\widetilde{f}_p^n$ is the average of f^n on the local patch ω_p.

Algorithm 5.1 (Adaptive Algorithm for Evolution Problems)
Start with initial time step size k_0, initial mesh $\mathcal{T}_0$, and initial solution U_h^0. Set $n = 1$ and $s_n = k_0$.
(i) Compute initial error indicator η_{init}.
If η_{init} is too large, refine the patch $\mathcal{R}_p$
if $\eta_{\text{init}}(p)$ is large; goto (i).
While $s_n \le T$, do (a)–(e):

(a) Solve for U_h^n and compute time error indicator η_{time}^n.
If η_{time}^n is too large, reduce time step k_n, update s_n, and goto (a).

(b) For every $p \in \mathcal{N}(\mathcal{T}_n)$, compute spatial and coarsening error indicators:
if $\eta_{\text{space}}^n(p)$ is too large, refine $\mathcal{R}_p$;
if $\eta_{\text{space}}^n(p) + \eta_{\text{coarse}}^n(p)$ is too small, coarsen $\mathcal{R}_p$ (if p is a good node).

(c) If the mesh was changed in (b):
solve for U_h^n and compute error indicators again;
if η_{time}^n is too large, goto (a);
if η_{space}^n is too large, goto (b);
otherwise, accept the current solution U_h^n.

(d) If η_{time}^n is small, enlarge k_{n+1}.

(e) Let $s_{n+1} = s_n + k_{n+1}$ and $n = n + 1$.

Now we show the performance of our coarsening algorithm. We report energy error at different time in Figure 5.1 together with number of degrees of freedom (DOF) and time step size. From Figure 5.1, we find the adaptive refined mesh and time step are adapted to the exact very well. In particular, when the solution becomes very smooth in space but changes very fast in time around time level 0.5, the time stepsize becomes small and spatial mesh size becomes large. We also show two sample meshes in Figures 5.2 and 5.3.

Acknowledgments. The first author was supported in part by NSF Grant DMS-0811272, and in part by NIH Grant P50GM76516 and R01GM75309. The second author was supported by NSF Grant DMS-0915153.

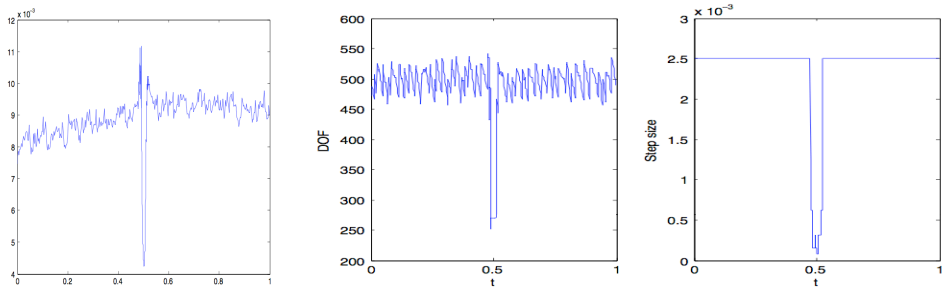


Fig. 5.1. History of energy error, spatial DOF and time stepsize. Left: energy error; middle: spatial degree of freedom; right: time stepsize.

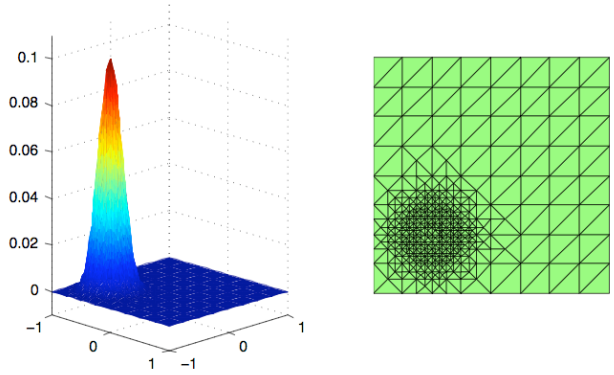


Fig. 5.2. Solution and automatically generated mesh at the initial time ($t = 0.0$).

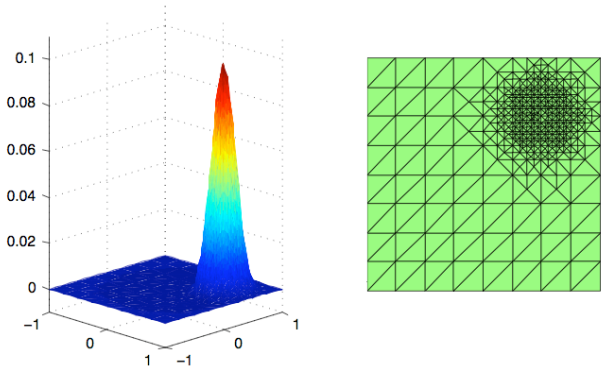


Fig. 5.3. Solution and automatically generated mesh at the final time ($t = 1.0$).

References

[1] B. Aksoylu, S. Bond, and M. Holst, An adyssey into local refinement and multilevel preconditioning III: Implementation and numerical experiments, *SIAM J. Sci. Comput.*, **25**:2 (2003), 478-498.

[2] I. Babuška and A. Miller, The post-processing approach in the finite element method. Part 3: A posteriori error estimates and adaptive mesh selection, *Int. J. Numer. Meth. Eng.*, **20** (1984), 2311-2324.

[3] W. Bangerth, R. Hartmann, and G. Kanschat, deal.ii — a general purpose object oriented finite element library, *ACM T. Math. Software*, **33**:4 (2007), 24, Aug. Article 24, 27 pages.

[4] R.E. Bank, T. Dupont, and H. Yserentant, The hierarchical basis multigrid method, *Numer. Math.*, **52** (1988), 427-458.

[5] T.C. Biedl, P. Bose, E.D. Demaine, and A. Lubiw, Efficient algorithms for Petersen's matching theorem, *J. Algorithm.*, **38**:1 (2001), 110-134.

[6] M. Bieterman and I. Babuška, The finite element method for parabolic equations. I. A posteriori error estimation, *Numer. Math.*, **40**:3 (1982), 339-371.

[7] M. Bieterman and I. Babuška, The finite element method for parabolic equations. II. A posteriori error estimation and adaptive approach, *Numer. Math.*, **40**:3 (1982), 373-406.

[8] P. Binev, W. Dahmen, and R. DeVore, Adaptive finite element methods with convergence rates,

Numer. Math., **97**:2 (2004), 219-268.

[9] F.A. Bornemann and H. Yserentant, A basic norm equivalence for the theory of multilevel methods, *Numer. Math.*, **64** (1993), 455-476.

[10] J.H. Bramble, J.E. Pasciak, and J. Xu, Parallel multilevel preconditioners, *Math. Comput.*, **55**:191 (1990), 1-22.

[11] J.M. Cascon, C. Kreuzer, R.H. Nochetto, and K.G. Siebert, Quasi-optimal convergence rate for an adaptive finite element method, *SIAM J. Numer. Anal.*, **46**:5 (2008), 2524-2550.

[12] L. Chen, Short implementation of bisection in MATLAB, In P. Jorgensen, X. Shen, C.-W. Shu, and N. Yan, editors, Recent Advances in Computational Sciences – Selected Papers from the International Workship on Computational Sciences and Its Education, pages 318-332. World Scientific Pub Co Inc, 2007.

[13] L. Chen, *i*FEM: an integrated finite element methods package in MATLAB. *Technical Report, University of California at Irvine*, 2009.

[14] L. Chen, R.H. Nochetto, and J. Xu, Local multilevel methods on graded bisection grids, *In Preparation*, 2009.

[15] Z. Chen and F. Jia, An adaptive finite element algorithm with reliable and efficient error control for linear parabolic problems, *Math. Comput.*, **73** (2004), 1167-1194.

[16] W. Dahmen and A. Kunoth, Multilevel preconditioning, *Numer. Math.*, **63** (1992), 315-344.

[17] W. Dörfler, A convergent adaptive algorithm for Poisson's equation, *SIAM J. Numer. Anal.*, **33** (1996), 1106-1124.

[18] K. Erickson and C. Johnson, Adaptive finite element methods for parabolic problems. i. a linear model problem, *SIAM J. Numer. Anal.*, **28**:1 (1991), 43-77.

[19] K. Eriksson and C. Johnson, Adaptive finite element methods for parabolic problems II: Optimal error estimates in $l_\infty l_2$ and $l_\infty l_\infty$, *SIAM J. Numer. Anal.*, **32**:3 (1995), 706-740.

[20] R.B. Kellogg, On the Poisson equation with intersecting interface, *Appl. Aanal.*, **4** (1975), 101-129.

[21] I. Kossaczký, A recursive approach to local mesh refinement in two and three dimensions, *J. Comput. Appl. Math.*, **55** (1994), 275-288.

[22] O. Lakkis and C. Makridakis, Elliptic reconstruction and a posteriori error estimates for fully discrete linear parabolic problems, *Math. Comput.*, **75**:256 (2006), 1627-1658 (electronic).

[23] C. Makridakis and R.H. Nochetto, Elliptic reconstruction and a posteriori error estimates for parabolic problems, *SIAM J. Numer. Anal.*, **41**:4 (2003), 1585-1594.

[24] W.F. Mitchell, Unified Multilevel Adaptive Finite Element Methods for Elliptic Problems, PhD thesis, University of Illinois at Urbana-Champaign, 1988.

[25] W.F. Mitchell, A comparison of adaptive refinement techniques for elliptic problems, *ACM Transactions on Mathematical Software (TOMS) archive*, **15**:4 (1989), 326-347.

[26] W.F. Mitchell, Optimal multilevel iterative methods for adaptive grids, *SIAM J. Sci. Comput.*, **13** (1992), 146-167.

[27] K.-S. Moon, R.H. Nochetto, T. von Petersdorff, and C.-S. Zhang, A posteriori error analysis for parabolic variational inequalities. *Mathematical Modelling and Numerical Analysis (M2AN)*, **41**:3 (2007), 485-511.

[28] R.H. Nochetto, A. Schmidt, and C. Verdi, A posteriori error estimation and adaptivity for degenerate parabolic problems, *Math. Comput.*, **229**:220 (1999), 1-24.

[29] P. Oswald, Multilevel Finite Element Approximation, Theory and Applications, Teubner Skripten zur Numerik. Teubner Verlag, Stuttgart, 1994.

[30] M. Picasso, Adaptive finite elements for a linear parabolic problem, *Comput. Method. Appl. M.*, **167**:3-4 (1998), 223-237.

[31] A. Schmidt and K.G. Siebert, Design of adaptive finite element software, volume 42 of *Lecture Notes in Computational Science and Engineering*, Springer-Verlag, Berlin, 2005. The finite element toolbox ALBERTA, With 1 CD-ROM (Unix/Linux).

[32] E.G. Sewell, Automatic generation of triangulations for piecewise polynomial approximation, In

Ph. D. dissertation. Purdue Univ., West Lafayette, Ind., 1972.

[33] J. Xu, Iterative methods by space decomposition and subspace correction, *SIAM Rev.*, **34** (1992), 581-613.

[34] J. Xu and L. Zikatanov, The method of alternating projections and the method of subspace corrections in Hilbert space, *J. Am. Math. Soc.*, **15** (2002), 573-597.

[35] H. Yserentant, On the multi–level splitting of finite element spaces, *Numer. Math.*, **49** (1986), 379-412.

[36] H. Yserentant, Two preconditioners based on the multi-level splitting of finite element spaces, *Numer. Math.*, **58** (1990), 163-184.