TETRAHEDRAL BISECTION AND ADAPTIVE FINITE ELEMENTS

DOUGLAS N. ARNOLD[*] AND ARUP MUKHERJEE[†]

Abstract. An adaptive finite element algorithm for elliptic boundary value problems in $\mathbb{R}^3$ is presented. The algorithm uses linear finite elements, a-posteriori error estimators, a mesh refinement scheme based on bisection of tetrahedra, and a multi-grid solver. We show that the repeated bisection of an arbitrary tetrahedron leads to only a finite number of dissimilar tetrahedra, and that the recursive algorithm ensuring conformity of the meshes produced terminates in a finite number of steps. A procedure for assigning numbers to tetrahedra in a mesh based on a-posteriori error estimates, indicating the degree of refinement of the tetrahedron, is also presented. Numerical examples illustrating the effectiveness of the algorithm are given.

Key words. finite elements, adaptive mesh refinement, error estimators, bisection of tetrahedra

AMS(MOS) subject classifications. 65N50.

1. Introduction.

In this article, we describe an algorithm for the accurate and efficient computation of solutions to elliptic boundary value problems in $\mathbb{R}^3$. The algorithm creates a sequence of adapted tetrahedral meshes starting form an initial coarse tetrahedral mesh, and the solution is computed on each mesh using the hierarchy of meshes below it. Piecewise linear finite elements are used to discretize the problem on a mesh, the resulting linear system is solved using a multi-grid solver, and a-posteriori error estimators based on estimating the residual in an appropriate norm determine which portions of the current mesh need to be refined. Individual tetrahedra marked for refinement are bisected, and further bisection is employed to ensure that the new (adapted) mesh is conforming. We prove that the recursive algorithm that ensures mesh conformity terminates, and also prove that the tetrahedral shapes produced via repeated bisection do not degenerate.

In particular, we consider boundary value problems of the form

$$-\operatorname{div}(\mathcal{A}\nabla u) + bu = f \text{ in } \Omega, \tag{1.1}$$

$$u = g_D \text{ on } \Gamma_D, \tag{1.2}$$

$$\mathcal{A}\nabla u \cdot n + cu = g_N \text{ on } \Gamma_N, \tag{1.3}$$

where $\Omega \subset \mathbb{R}^3$ is an open, bounded, polyhedral domain with boundary $\Gamma = \bar{\Gamma}_D \cup \bar{\Gamma}_N$, and n is the unit outward normal to Γ. The coefficients a_{ij} of the symmetric matrix $\mathcal{A}$ are assumed to be piecewise continuously

[*]Department of Mathematics, The Pennsylvania State University, University Park, PA 16802. Email: dna@math.psu.edu

[†]Department of Mathematics–Hill Center, Rutgers University, 110 Frelinghuysen Rd., Piscataway, NJ 08854-8019. Email: arup@math.rutgers.edu

differentiable while b and c are piecewise continuous with respect to the tetrahedral meshes introduced later. Also assume that f and g_N are piecewise continuous, and g_D is the restriction to Γ_D of a continuous square integrable function u_D. Further, assume that the differential operator in (1.1) is elliptic, and write the weak formulation of (1.1)–(1.3) as:

$$\text{Find } \hat{u} \in V_D \text{ such that } a(\hat{u}, v) = l(v) \text{ for all } v \in V_0, \tag{1.4}$$

where V_0 (resp. V_D) are spaces of H^1 functions on Ω which are zero (resp. equal to g_D) on Γ_D, and the forms $a : H^1(\Omega)\times H^1(\Omega) \to \mathbb{R}$ and $l : H^1(\Omega) \to \mathbb{R}$ are

$$a(u,v) = \int_\Omega [(\mathcal{A}\nabla u)\cdot\nabla v + buv]\,dx + \int_{\Gamma_N} cuv\,ds, \tag{1.5}$$

$$l(v) = \int_\Omega fv\,dx + \int_{\Gamma_N} g_N v\,ds. \tag{1.6}$$

We now introduce some notation that facilitates the description of the discrete problem associated with (1.4) and will be used later. For a tetrahedron τ let $\mathcal{N}(\tau)$, $\mathcal{E}(\tau)$, and $\mathcal{F}(\tau)$ denote the set of its vertices, edges, and faces, respectively. A mesh $\mathcal{T}$ of the domain Ω is a set of closed tetrahedra with disjoint interiors whose union is $\bar{\Omega}$. The vertex, edge, and face sets of $\mathcal{T}$ are denoted by $\mathcal{N}(\mathcal{T})$, $\mathcal{E}(\mathcal{T})$, $\mathcal{F}(\mathcal{T})$ respectively (the Dirichlet, Neumann, and interior nodes of $\mathcal{T}$ are represented by $\mathcal{N}_D(\mathcal{T})$, $\mathcal{N}_N(\mathcal{T})$, and $\mathcal{N}_\Omega(\mathcal{T})$, with similar definitions and for the corresponding faces and edges). A mesh is conforming if the intersection of any two distinct tetrahedra in it is either a common face, a common edge, a common vertex, or empty. Assume that the meshes conform to the subdivision of the boundary into its Dirichlet and Neumann parts in the sense that any face contained in Γ belongs entirely to Γ_D or Γ_N. Denote the diameter of τ by $h(\tau)$ and by $\rho(\tau)$ the diameter of the maximum ball contained in τ, and define by $\sigma(\tau) = h(\tau)/\rho(\tau)$ the shape constant of τ (for a mesh $\mathcal{T}$, the mesh size $h(\mathcal{T})$ is the maximum diameter of all tetrahedra in $\mathcal{T}$ and the shape constant $\sigma(\mathcal{T})$ is defined similarly). A family of meshes $\{\mathcal{T}\}$ is shape regular if $\inf_{\mathcal{T}} h(\mathcal{T}) = 0$ and $\sup_{\mathcal{T}} \sigma(\mathcal{T}) < \infty$. Using the subspaces

$$\begin{aligned} X(\mathcal{T}) &= \left\{ v \in H^1(\Omega) : \, v|_\tau \in P_1(\tau) \text{ for every } \tau \in \mathcal{T} \right\}, \\ V_0(\mathcal{T}) &= \left\{ v \in X(\mathcal{T}) : \, v(\nu) = 0 \text{ for all } \nu \in \mathcal{N}_D(\mathcal{T}) \right\}, \\ V_D(\mathcal{T}) &= \left\{ v \in X(\mathcal{T}) : \, v(\nu) = g_D(\nu) \text{ for all } \nu \in \mathcal{N}_D(\mathcal{T}) \right\}, \end{aligned}$$

where $P_1(\tau)$ is the space of linear functions on τ, the discrete problem for (1.4) is:

$$\text{Find } u_{\mathcal{T}} \in V_D(\mathcal{T}) \text{ such that } a(u_{\mathcal{T}}, v) = l(v) \text{ for all } v \in V_0(\mathcal{T}). \tag{1.7}$$

It is well known (see, for example, [11]) that if the bilinear form a is coercive, (1.4) has a unique solution $\hat{u}$. Moreover, for a family $\{\mathcal{T}\}$ of conforming, shape regular meshes of Ω, (1.7) has a unique solution $u_{\mathcal{T}}$ for every mesh in the family, these converge to $\hat{u}$ in H^1 as the mesh size goes to zero, and there exists a constant C such that if $\hat{u} \in H^2(\Omega)$, then

$$\|\hat{u} - u_{\mathcal{T}}\| \leq Ch(\mathcal{T})\|\hat{u}\|_2. \tag{1.8}$$

In general, locally adapted meshes contain tetrahedra with widely varying diameters and the mesh diameter is not a useful measure of the size of such meshes. It is common to use the number of nodes in the mesh instead. A finite element method is said to be of order α if the discretization error decreases like $N^{-\alpha/3}$ where $N = \mathrm{card}(\mathcal{N}(\mathcal{T}))$, in the H^1 or energy norm. Since for a uniform mesh $\mathcal{T}$, $N^{-1/3} = O(h(\mathcal{T}))$, our algorithm should have an optimal order of 1.

Figure 1 shows the structure of the algorithm used for solving (1.1)–(1.3). It produces a sequence of nested meshes $\mathcal{T}_k$ and corresponding finite element solutions $u_{\mathcal{T}_k}$. The meshes are adapted to the solution of the problem being solved via the error estimator.

1. Begin with an initial coarse conforming tetrahedral mesh $\mathcal{T}_0$, and set $k = 0$.
   Repeat until accurate solution is obtained:
2. On the current mesh $\mathcal{T}_k$ discretize the problem using piecewise linear finite elements.
3. Solve the problem (1.7) using full multi-grid to obtain the approximate solution $u_{\mathcal{T}_k}$ (on the coarse mesh $\mathcal{T}_0$, $u_{\mathcal{T}_0}$ is found using an exact solver).
4. Assign error indicators η_τ to each tetrahedron in the current mesh and a number q_τ based on η_τ indicating the degree of refinement needed for τ.
5. Unless stopping criterion is met, refine the mesh as indicated. Refine further to restore conformity thus obtaining the next finer mesh $\mathcal{T}_{k+1}$.

FIG. 1. *Structure of the code.*

Remark: A simple modification the code can be used to solve semi-linear elliptic boundary value problems of the form

$$-\mathrm{div}(\mathcal{A}\nabla u) + f(x, u) = 0 \text{ in } \Omega, \tag{1.9}$$

with the boundary conditions (1.2)–(1.3). The generation of initial data for computing the gravity waves emanating from the spiraling coalescence of two black holes (and many other applications) involves the solution of such boundary value problems. Using Newton's iteration to reduce the semi-linear problem (1.9) to a sequence of linear boundary value problems, and solving these linear problems by the algorithm described in figure 1, we have efficiently computed accurate initial data for binary black hole systems (for a more detailed description of the equations and physical model together with some results, see [2]).

In section 2, we describe the residual error estimator used in the code together with an algorithm for determining the numbers q_τ. The numbers q_τ are chosen so as to ensure that the sequence of meshes have geometric increase in the number of nodes, and the error is approximately equidistributed on each mesh. The mesh refinement algorithm used is presented in section 3. The basic building block for mesh refinement is bisection of tetrahedra. We show that given a conforming mesh $\mathcal{T}$ and a subset $\mathcal{S} \subset \mathcal{T}$ of tetrahedra in it, bisection can be effectively used to generate a conforming mesh $\mathcal{T}'$ such that all tetrahedra in $\mathcal{S}$ have been bisected. The tetrahedral shapes do not degenerate (even after an arbitrary number of bisections of a tetrahedron), and the recursive algorithm that ensures conformity of the adapted mesh $\mathcal{T}'$ terminates without over refining the mesh $\mathcal{T}$. We also discuss the relationship of our mesh refinement algorithm with other bisection based algorithms. Finally, we present some numerical results in section 4.

2. Error Estimator. If the residual functional, $R : H^1(\Omega) \to V_0^*$, is defined as

$$\langle R(u), v\rangle = a(u,v) - l(v) = a(u-\hat{u}, v) \text{ for } u \in H^1(\Omega) \text{ and } v \in V_0,$$

then for any $u \in V_D$, $\|\hat{u}-u\|_1 \le C\|R(u)\|_{V_0^*}$. Integrating $\langle R(u), v\rangle$ by parts, taking $u = u_\mathcal{T}$ and using the equality $\langle R(u_\mathcal{T}), v_\mathcal{T}\rangle = 0$ for any $v_\mathcal{T} \in V_0(\mathcal{T})$, and choosing the Clément interpolant for $v_\mathcal{T}$ leads to the estimate

$$\begin{aligned}\|\hat{u} - u_\mathcal{T}\|_1 \le C\Bigg(&\sum_{\tau\in\mathcal{T}} h(\tau)^2 \| - \operatorname{div}(\mathcal{A}\nabla u_\mathcal{T}) + b u_\mathcal{T} - f\|_{0;\tau}^2 \\ &+ \sum_{\mu\in\mathcal{F}_\Omega(\mathcal{T})} h(\mu)\,\|[\mathcal{A}\nabla u_\mathcal{T}\cdot n_\mu]_\mu\|_{0;\mu}^2 \\ &+ \sum_{\mu\in\mathcal{F}_N(\mathcal{T})} h(\mu)\,\|\mathcal{A}\nabla u_\mathcal{T}\cdot n_\mu + c u_\mathcal{T} - g_N\|_{0;\mu}^2 \Bigg)^{1/2},\end{aligned} \tag{2.1}$$

where C is a constant depending on the shape constant of the mesh and $[\varphi]_\mu$ is the jump of φ across μ. The integrals involved in (2.1) may be calculated using either quadrature or by replacing the data by simpler functions (projection onto appropriate spaces) and then using exact integration. Note that on any $\tau \in \mathcal{T}$, $\operatorname{div}(\mathcal{A}\nabla u_\mathcal{T}) = \operatorname{div}\mathcal{A}\cdot\nabla u_\mathcal{T}$, since $u_\mathcal{T}$ is piecewise linear and div is applied to $\mathcal{A}$ column-wise. Letting φ_τ (resp. φ_μ) denote the projection of a continuous function φ onto the space $P_0(\tau)$ (resp. $P_0(\mu)$), we define the residual error estimator η_τ as the easily computable quantity

$$\eta_\tau = \Big(h(\tau)^2 \| - (\operatorname{div}\mathcal{A})_\tau \cdot \nabla u_\mathcal{T}) + b_\tau u_\mathcal{T} - f_\tau\|_{0;\tau}^2$$

$$(2.2)\qquad + \frac{1}{2} \sum_{\mu \in \mathcal{F}(\tau) \cap \mathcal{F}_\Omega(\mathcal{T})} h(\mu) \, \|[\mathcal{A}_\mu \nabla u_\mathcal{T} \cdot n_\mu]_\mu\|_{0;\mu}^2$$
$$+ \sum_{\mu \in \mathcal{F}(\tau) \cap \mathcal{F}_N(\mathcal{T})} h(\mu) \, \|\mathcal{A}_\mu \nabla u_\mathcal{T} \cdot n_\mu + c_\mu u_\mathcal{T} - (g_N)_\mu\|_{0;\mu}^2 \Bigg)^{1/2}.$$

Observing that each interior face is counted twice in a summation over all tetrahedra in the mesh, we see that

$$\|\hat{u} - u_\mathcal{T}\|_1 \le C \Bigg(\sum_{\tau \in \mathcal{T}} \eta_\tau^2 + \text{ consistency terms } \Bigg).$$

Verfürth [14] has shown that η_τ is bounded by the H^1 norm of the error on a set of tetrahedra neighboring τ together with the L^2 norms of some consistency terms. The expression (2.2) for η_τ is used in the code (note that it suffices to evaluate φ_τ (resp. φ_μ) at a single point on τ (resp. μ) and we choose the barycenter). Residual error estimators like η_τ were introduced by Babuška and Rheinboldt [3], and our analysis and computation follows the outlines provided by Verfürth [14]. Details of the derivations may be found in [10].

The estimator η_τ is used to assign numbers q_τ to every tetrahedron in $\mathcal{T}$ so that the refined mesh $\mathcal{T}'$ obtained after bisecting each tetrahedron in $\mathcal{T}$ q_τ times (some additional bisection may be required to ensure a conforming mesh) meets the following requirements.

- The number of nodes in $\mathcal{T}'$ are approximately $2\mathcal{N}(\mathcal{T})$ (this makes the multi-grid solver efficient).
- The error is approximately equidistributed on $\mathcal{T}'$.

The two goals stated above are achieved in practice by using the expression

$$(2.3)\qquad q_\tau = -\frac{\ln[(\bar{\eta}^{\mathcal{T}'})^2 - \eta_\tau^2]}{\ln 2^{5/3}}, \text{ where } \bar{\eta}^{\mathcal{T}'} = \frac{2^{-2/3}}{2\text{card}(\mathcal{T})} \sum_{\tau \in \mathcal{T}} \eta_\tau^2.$$

For an asymptotic analysis leading to the expression (2.3), see [10].

3. Mesh Refinement. The basic component of our mesh refinement algorithm is tetrahedral bisection (introducing a new vertex on a selected edge–called the *refinement edge*—of the tetrahedron and creating two new edges joining the new vertex to the two vertices that do not lie on the refinement edge). An alternative is to use octasection (cutting a single tetrahedron into eight in a particular pre-determined manner) to sub-dividing individual tetrahedra. Bey [6], generalizing the work of Bank [5] in $\mathbb{R}^2$, uses octasection combined with bisection to generate adapted tetrahedral meshes. A case-by-case analysis of the many intermediate strategies is needed to guarantee that the meshes produced are conforming and that the tetrahedra in them do not degenerate. Algorithm `LocalRefine` introduced in this section produces a conforming mesh $\mathcal{T}'$ from a conforming mesh $\mathcal{T}$ and a subset $\mathcal{S}$ of tetrahedra to be sub-divided, and uses only bisection of individual tetrahedra. Moreover, in applications where a sequence of nested meshes $\mathcal{T}_k$ are produced from a coarse conforming mesh $\mathcal{T}_0$, `LocalRefine` guarantees that only a finite number of similarity classes are produced for every tetrahedron in the initial mesh.

Special care is needed in choosing the refinement edges of the children produced via bisection in order to ensure that the tetrahedral shapes do not degenerate. One approach, which follows its analogue in $\mathbb{R}^2$, is to always choose the longest edge of a tetrahedron as its refinement edge. Rivara and coworkers [12, 13] conjecture and provide numerical evidence that this does not lead to element degeneration in $\mathbb{R}^3$. The algorithm of Liu and Joe [7] is also motivated by longest edge bisection. Their algorithm uses a mapping between an arbitrary tetrahedron and a tetrahedron of equal volume having a special shape. The special tetrahedron and its children are bisected using longest edge bisection, with the mapping determining the corresponding refinement edges for the tetrahedron and its children. They prove a bound of 168 for the number of similarity classes of tetrahedra produced. Below we shall give a sharp bound of 72 for our algorithm.

Other approaches for tetrahedral bisection are all based on generalizations of opposite-edge (or newest-vertex) bisection of triangles in $\mathbb{R}^2$. Opposite-edge bisection guarantees that repeated bisection of an arbitrary triangle yields at most four similarity classes, and may be described as follows: for every triangle in the initial mesh choose an arbitrary edge as its refinement edge, and for each child created via bisection, choose the edge opposite the newest vertex to be its refinement edge. A naive generalization to $\mathbb{R}^3$ quickly leads to degenerate shapes as the same edges get cut repeatedly. Bänsch [4] developed an algorithm for local tetrahedral mesh refinement based on the generalization of opposite-edge bisection. He shows that the tetrahedral shapes produced do not degenerate but does not prove finiteness of similarity classes. Our algorithm is essentially equivalent to that of Bänsch, but uses a formulation that we find easier to state and analyze. A data structure called the *marked tetrahedron*, which is key to our formulation is introduced. In particular, a marked tetrahedron τ is a 4-tuple $(\mathcal{N}(\tau), r_\tau, (m_\varphi)_{\varphi\in\mathcal{F}(\tau)}, f_\tau)$ where

- $\mathcal{N}(\tau)$ is the vertex set of τ;
- $r_\tau \in \mathcal{E}(\tau)$ is the refinement edge of τ;
- $m_\varphi \in \mathcal{E}(\varphi)$ is the marked edge of φ, with $m_\varphi = r_\tau$ if $r_\tau \subset \varphi$;
- $f_\tau \in \{0, 1\}$ is the flag for τ.

The faces of τ containing r_τ are the *refinement faces* and r_τ is taken as the marked edge for both refinement faces. The two non-refinement faces also have a marked edge and each tetrahedron has a boolean flag f_τ. Each marked non-refinement edge of a marked tetrahedron is either adjacent of opposite to the refinement edge. All marked tetrahedra are classified into types as follows (cf. Figure 2).

- Type P, planar: the marked edges all lie on a plane. This is

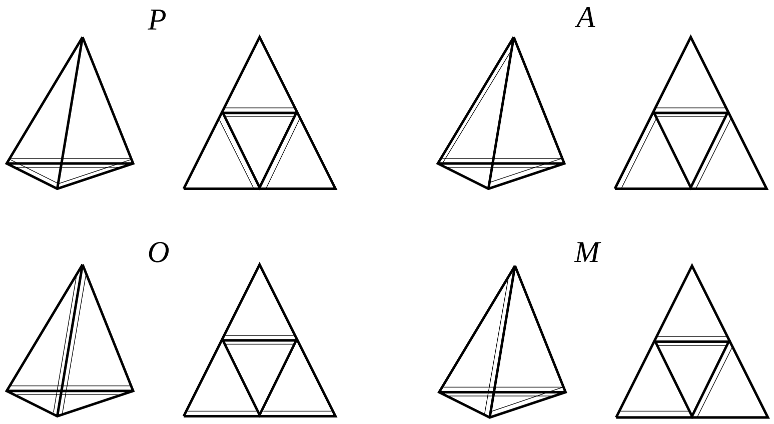

FIG. 2. *The four types of marked tetrahedra. Each marked edge is indicated by a double line and the refinement edge is marked for both faces containing it. Each tetrahedron is shown in three dimensions and cut open (unfolded) into two dimensions.*

further sub-classified into type P_f (Planar-flagged) or P_u (Planar-unflagged) according to whether f_τ is 1 or 0.

- Type A, adjacent: the marked edges for the non-refinement faces are adjacent to r_τ, but are not coplanar.
- Type O, opposite: the marked edges of the non-refinement faces do not intersect r_τ. In this case, a pair of opposite edges are marked in the tetrahedron–one for the two refinement faces intersecting it and another for the two non-refinement faces intersecting it.
- Type M, mixed: the marked edge of exactly one non-refinement face intersects r_τ.

  We impose that the flag $f_\tau = 0$ for types A, O, and M. Thus, the flag is only relevant for planar tetrahedra.

If τ_1 and τ_2 are the children of τ, a face $\varphi \in \mathcal{F}(\tau_i)$ is called an *inherited face* if $\varphi \in \mathcal{F}(\tau)$, a *cut face* if $\varphi \subset \varphi'$ for some $\varphi' \in \mathcal{F}(\tau)$, and a *new face* otherwise.

Algorithm `BisectTet` is described in figure 3 and figure 4 shows the types of tetrahedra output by `BisectTet`. The two children τ_1 and τ_2 output by `BisectTet` always have the same type. Moreover, types M and O are never output by `BisectTet`—thus, in an adaptive mesh refinement algorithm that uses `BisectTet`, these can only occur in the initial mesh.

Maubach [8] generalizes opposite-edge bisection of triangles to `BisectSimplex`, an algorithm for bisecting n-simplices in $\mathbb{R}^n$. In [1], we prove the following theorem bounding the number of similarity classes produced by the repeated bisection of an arbitrary n-simplex.

THEOREM 3.1. *When an arbitrary n-simplex is bisected repeatedly using* `BisectSimplex`*, at most $2^{n-2}n!$ similarity classes arise in each generation and the set of similarity classes depends on the generation modulo*

```
Algorithm {τ_1, τ_2} = BisectTet(τ)
    input: marked tetrahedron $\tau$
    output: marked tetrahedra $\tau_1$ and $\tau_2$
1. Bisect $\tau$ by joining the midpoint of its refinement edge to
   each of the two vertices not lying on the refinement edge.
   This defines $\mathcal{N}(\tau_i)$ for $i = 1$ and 2.
     Mark the faces of the children as follows:
2. The inherited face inherits its marked edge from the par-
   ent, and this marked edge is the refinement edge of the
   child.
3. On the cut faces of the children mark the edge opposite
   the new vertex with respect to the face.
4. The new face is marked the same way for both children. If
   the parent is type $P_f$, the marked edge is the edge connect-
   ing the new vertex to the new refinement edge. Otherwise
   it is the edge opposite the new vertex.
5. The flag is set in the children if and only if the parent is
   type $P_u$.
```

Fig. 3. *Algorithm BisectTet.*

n. Thus in two dimensions there are only two classes of each generation and only four total. In three dimensions the corresponding numbers 12 and 36. By computation on a particular tetrahedron we showed that these numbers are sharp [1]. Maubach recently proved that the result is sharp for all n [9]. Further, for the particular case $n = 3$, we construct a 2 to 1 and onto mapping from the subset of 3-simplices which can be input into `BisectSimplex` to the subset of all marked tetrahedra with adjacent markings (the marked edges for the non-refinement faces are adjacent to r_τ–the tetrahedra may be planar or of type A). This shows that the repeated application of `BisectTet` to an arbitrary marked tetrahedron produces a maximum of 36 similarity classes. Since one application of `BisectTet` to a tetrahedron of type M or O produces children of type P_u, the repeated bisection of an arbitrary marked tetrahedron will produce at most 72 similarity classes.

Thus, even though our algorithm is closely related to that of Bänsch, we can prove that only a finite number of similarity classes ever arise and this number is optimal. The marked tetrahedron data-structure is also useful in proving that `LocalRefine`, the recursive algorithm used to generate adapted sequence of meshes, terminates.

In order to successfully implement step 5 of the algorithm in figure 1, an algorithm that begins with a conforming tetrahedral mesh and a subset

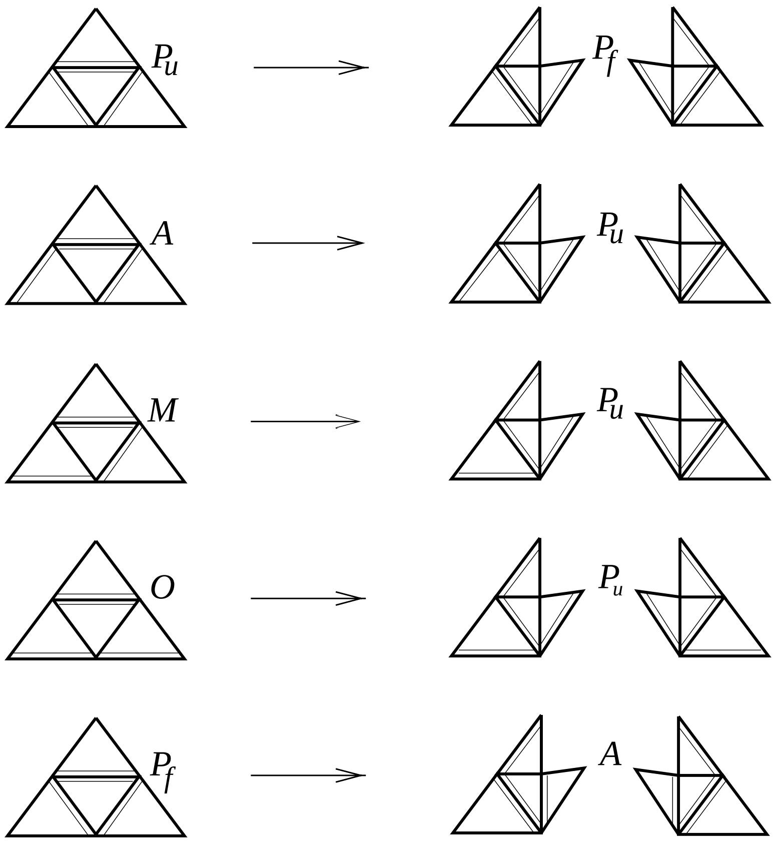

FIG. 4. *Bisection rules for marked tetrahedra.*

of tetrahedra pre-selected for refinement and returns a conforming tetrahedral mesh in which all of the pre-selected tetrahedra have been sub-divided is needed. Algorithm `LocalRefine`, based on `BisectTet`, performs this duty. If ν is a vertex of some tetrahedron in a mesh and ν belongs to another tetrahedron τ but is not a vertex of τ, we say that ν is a *hanging node* of τ (i.e., if $\tau \in \mathcal{T}$ and $\nu \in \mathcal{N}(\mathcal{T})$, ν is a hanging node of τ if $\nu \in \tau \setminus \mathcal{N}(\tau)$). A mesh is conforming if no tetrahedron in it is has a hanging node and every face of every tetrahedron in the mesh either belongs to the boundary or is a face of another tetrahedron in the mesh. A mesh will be called *marked* if each tetrahedron in it is marked. A marked conforming mesh is *conformingly-marked* if each face has a unique marked edge (that is, when a face is shared by two tetrahedra, the marked edge is the same for both). The tetrahedra in any conforming mesh may be marked so as to yield a conformingly-marked mesh. For example, this may be accomplished by the following procedure. Strictly order the edges in the mesh in an arbitrary but fixed manner, e.g., by using length with a well-defined tie-breaking rule. Then choose the maximal edge of each tetrahedron as its refinement edge and the maximal edge of each face as its marked edge. Unset the flags on all the tetrahedra in the mesh.

```
**Algorithm** $\mathcal{T}' = \texttt{LocalRefine}(\mathcal{T}, \mathcal{S})$
    *input*: conformingly-marked mesh $\mathcal{T}$ and $\mathcal{S} \subset \mathcal{T}$
    *output*: conformingly-marked mesh $\mathcal{T}'$
1. $\overline{\mathcal{T}} = \texttt{BisectTets}(\mathcal{T}, \mathcal{S})$
2. $\mathcal{T}' = \texttt{RefineToConformity}(\overline{\mathcal{T}})$
```

Fig. 5. *Algorithm LocalRefine*

The algorithm in the first step of figure 5, `BisectTets` is trivial: we simply bisect each tetrahedron in $\mathcal{S}$:

$$\texttt{BisectTets}(\mathcal{T}, \mathcal{S}) = (\mathcal{T} \setminus \mathcal{S}) \cup \bigcup_{\tau \in \mathcal{S}} \texttt{BisectTet}(\tau). \tag{3.1}$$

In the second step, we perform further refinement as necessary to obtain a conforming mesh (algorithm `RefineToConformity` is described in figure 6).

```
**Algorithm** $\mathcal{T}' = \texttt{RefineToConformity}(\mathcal{T})$
    *input*: marked mesh $\mathcal{T}$
    *output*: marked mesh $\mathcal{T}'$ without hanging nodes
1. set $\mathcal{S} = \{\tau \in \mathcal{T} \,|\, \tau$ has a hanging node $\}$
2. if $\mathcal{S} \neq \emptyset$ then
       $\overline{\mathcal{T}} = \texttt{BisectTets}(\mathcal{T}, \mathcal{S})$
       $\mathcal{T}' = \texttt{RefineToConformity}(\overline{\mathcal{T}})$
3. else
       $\mathcal{T}' = \mathcal{T}$
```

Fig. 6. *Algorithm RefineToConformity*

The recursion in figure 6 could conceivably continue forever. Moreover, even if the recursion terminates, the output mesh may not be conforming (a mesh without hanging nodes can nonetheless be non-conforming; cf., Figure 7). However, the following theorem, which is proved in [1] ensures that the recursion does terminate in the application of `RefineToConformity` in algorithm `LocalRefine` and that the resulting output mesh is conformingly-marked. Moreover, it gives a bound on the amount of refinement which can occur before termination. To state the theorem precisely, we consider an initial marked mesh $\mathcal{T}_0$, set $\mathcal{Q}_0 = \mathcal{T}_0$, and $\mathcal{Q}_{k+1} = \texttt{BisectTets}(\mathcal{Q}_k, \mathcal{Q}_k)$, for $k = 0, 1, \ldots$. Thus, $\mathcal{Q}_1$ consists of all children of tetrahedra in the initial mesh, $\mathcal{Q}_2$ of all grandchildren, etc. We assign *generation* k to all tetrahedra in $\mathcal{Q}_k$. The proof of the theorem depends on the classification of the edges that arise from an unflagged marked tetrahedron and uses the intermediate result that the types of tetrahedra and the generation of the edges occuring in $\mathcal{Q}_k$ can be obtained explicitly and that the meshes $\mathcal{Q}_{3k}$ are conformingly-marked (for details see [1]).

THEOREM 3.2. *Let $\mathcal{T}_0$ be a conformingly-marked mesh with no flagged tetrahedra. For $k = 0, 1, \ldots$, choose $\mathcal{S}_k \subset \mathcal{T}_k$ arbitrarily, and set $\mathcal{T}_{k+1} = \texttt{LocalRefine}(\mathcal{T}_k, \mathcal{S}_k)$. Then for each k, the application of* `RefineToConformity` *from within* `LocalRefine` *terminates producing a conformingly-marked mesh, and each tetrahedron in $\mathcal{T}_k$ has generation at most $3k$. Moreover, if the maximum generation of a tetrahedron in $\mathcal{T}_k$ is less than $3m$ for some integer m, then the maximum generation of a tetrahedron in $\mathcal{T}_{k+1}$ is less than or equal to $3m$.*

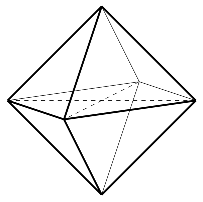

FIG. 7. *A non-conforming mesh without hanging nodes (the barycenter is NOT a vertex of the mesh).*

The marked tetrahedron data structure is essential to guarantee conformity since the assignment of a marked edge ensures that two tetrahedra sharing a common face are not bisected inconsistently. Also, the flag plays a key role in the analysis. If the requirement that the planar marked tetrahedra in the initial mesh are unflagged is removed, Theorem 3.2 need not be valid.

4. Numerical Results.

The code was run on a variety of problems with known solutions to test its performance. In this section, we report on the typical performance for two problems posed on $[0,1]^3$. The coarse mesh $\mathcal{T}_0$ is taken to be a uniform mesh having 96 tetrahedra and 35 nodes. For the first problem we set $\mathcal{A} = \mathcal{I}$, the identity matrix, $b = 0$, $\Gamma_N = \emptyset$, and $g_D = u_{\text{ex}}$ in (1.1)–(1.3) with

$$u_{\text{ex}} = (x^2 - x)(y^2 - y)(z^2 - z)e^{-\alpha[(x-a)^2+(y-b)^2+(z-c)^2]}. \tag{4.1}$$

In particular, we solve $-\Delta u = f$ on $[0,1]^3$ where f is chosen such that $\hat{u}$ satisfies the equation (the exact solution is smooth but strongly peaked at

$(a, b, c) \in \mathbb{R}^3$ for large values of α). For our second example, we solve the semi-linear problem $-\Delta u + u^3 = h$ with $\Gamma_N = \emptyset$, $g_D = \hat{u} = (xyz)^{\alpha/2}$ and h chosen so that the exact solution satisfies the equation (for moderately large values of α most of the variation in the exact solution occurs in the vicinity of $(1, 1, 1)$). On each mesh $\mathcal{T}_k$, the semi-linear problem is linearized around the current discrete solution and the resulting linear problem is solved. The process is repeated until a final (good) approximation is obtained on the current mesh and this final approximation is used for the linearization process on the next mesh (obtained by employing the error estimators for the linear problem and using `LocalRefine`). All integrals involved in the computation of errors, as well as those appearing in the computation of the stiffness matrices and the right hand sides, are evaluated using quadrature rules. The linear systems are solved using a standard V-cycle multi-grid solver. For the numerical tests, the stopping criterion is that we compute solutions using the highest possible number of nodes allowed on our machine (we performed our calculations on a 1993 DEC 3000 model 500 with a single 150 MHz Alpha processor).

We show for both problems that the algorithm in figure 1 converges with optimal order 1 in the H^1 (equivalently energy) norm. We also report the convergence rate in the L^2 norm (under some additional assumptions on the boundary value problem (1.1)–(1.3) this is expected to be 2 and these smoothness assumptions are satisfied by the problems we solve). The error plots are on a log-log scale and we plot the error against $N^{-1/3}$ where N is the number of nodes in the mesh. Figures 8 and 9 show the errors and rates for the two problems (lines with slopes 1 and 2 are shown for easy comparison). Using uniform refinement in problem 1, the finest mesh has $68,705$ nodes and a relative percentage energy error of approximately 15.85% while the maximum number of nodes using adaptive refinement are $62,738$ and the corresponding relative percentage energy error is 4.95%. The numbers for problem 2 are 14.24% using uniform refinement, and 2.3% using the finest adapted mesh with $59,323$ nodes.

In figure 10 we show how well the meshes adapt to the different features of the solutions for the two problems. For problem 1 the intersections of the tetrahedra with a plane are shown slightly shrunk to improve visibility while the figure for problem 2 shows the edges of the tetrahedra intersecting the boundary. The other three faces of the unit cube show a uniform mesh for problem 2 (this is expected–the solution and its derivatives are zero on these faces).

Finally, figure 11 shows a plot of the CPU time versus the number of nodes in the mesh for problem 2. The plot shows that the computation time is nearly proportional to the degrees of freedom (as is to be expected of a fast multi-grid solver).

REFERENCES

[1] D. N. Arnold, A. Mukherjee, and L. Pouly, *Locally adapted tetrahedral meshes using bisection*, Submitted to SIAM J. Sci. Comp. (1997). (available at http://www.math.psu.edu/dna/publications.html)

[2] D. N. Arnold, A. Mukherjee, and L. Pouly, *Adaptive finite elements and colliding black holes*, Numerical analysis 1997: Proceedings of the 17th Biennial Conference on Numerical Analysis, Addison Wesley, D. F. Griffits and G. A. Watson, eds. (1998).

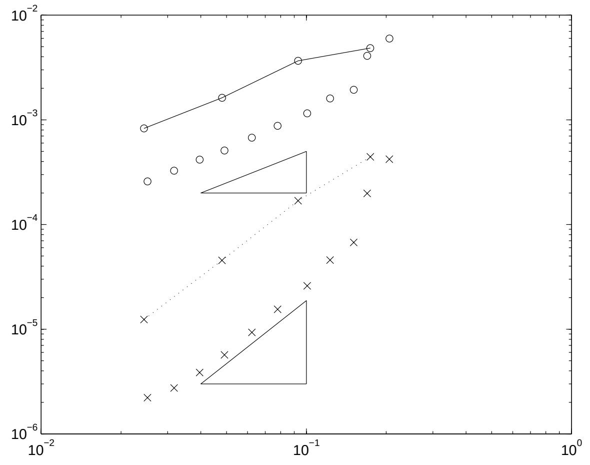

Fig. 8. *Problem 1 with $\alpha = 100$, $(a,b,c) = (0.25, 0.25, 0.25)$. The energy ($\circ$) and L^2 ($\times$) errors and rates. The plots with lines joining the $\circ$ (resp. $\times$) show the energy (resp. L^2) errors for uniform refinement while the disjoint ones are for adaptive refinement.*

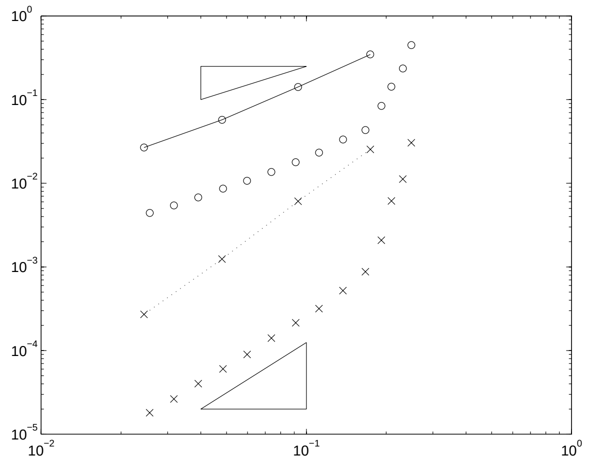

Fig. 9. *Problem 2 with $\alpha = 20$. The energy ($\circ$) and L^2 ($\times$) errors and rates. The plot with lines joining the $\circ$ (resp. $\times$) show the energy (resp. L^2) errors for uniform refinement while the disjoint ones are for adaptive refinement.*

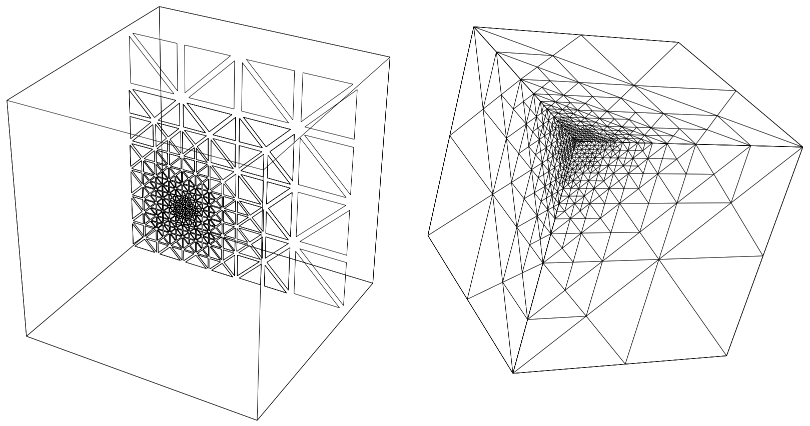

FIG. 10. *Cut along the plane $x = 1/4$ for Problem 1 (the mesh has* $11,418$ *tetrahedra and* $2,116$ *nodes) and a view of the locally adapted mesh for Problem 2 (*$5,988$ *tetrahedra and* $1,321$ *nodes)—the $x = 1$, $y = 1$, and $z = 1$ faces are shown.*

[3] I. BABUŠKA AND W. C. RHEINBOLDT, *Error estimates for adaptive finite element computations*, SIAM J. of Numer. Anal., **15** 736–754 (1978).

[4] E. BÄNSCH, *Local mesh refinement in 2 and 3 dimensions*, Impact of Comp. in Sci. and Engrg. **3** 181–191 (1991).

[5] R. BANK *PLTMG: a software package for solving elliptic partial differential equations. User,s guide 7.0*, SIAM, Philadelphia, 1994.

[6] JÜRGEN BEY, *Tetrahedral grid refinement*, Computing **55** 71–288 (1995).

[7] A. LIU AND B. JOE *Quality local refinement of tetrahedral meshes based on bisection*, SIAM J. Sci. Comput. **16(6)** 1269–1291 (1995).

[8] J. M. MAUBACH *Local bisection refinement for n-simplical grids generated by reflection*, SIAM J. Sci. Comput. **16(1)** 210–227 (1995).

[9] J. M. MAUBACH *The amount of similarity classes created by local n-simplical bisection refinement*, preprint (1997).

[10] A. MUKHERJEE, *Ph.d. Thesis*, The Pennsylvania State University, 1996. (available at http://www.math.rutgers.edu/∼arup/publications.html)

[11] A. QUARTERONI AND A. VALLI, *Numerical approximation of partial differential equations*, Springer-Verlag, 1994.

[12] M. C. RIVARA, *Local modification of meshes for adaptive and/or Multi-grid finite element methods*, J. Comput. Appl. Math. **36** 79–89 (1991).

[13] M. C. RIVARA AND C. LEVIN *A 3-D refinement algorithm suitable for adaptive and multi-grid techniques*, Comm. in App. Num. Meth. **8** 281–290 (1992).

[14] R. VERFÜRTH, *A review of a posteriori error estimation and adaptive mesh refinement techniques*, Technical report, Lecture notes of a Compact Seminar at TU Magdeburg, June 2–4, 1993.

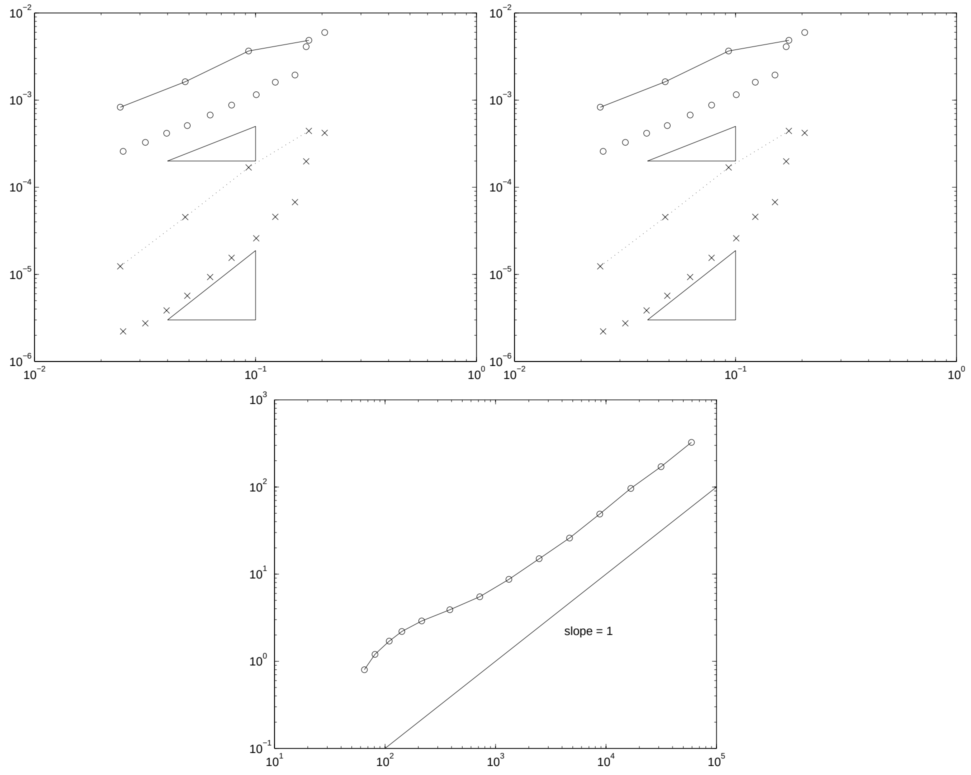

FIG. 11. *Total CPU time in seconds (y axis) versus number of nodes (x axis)*