SHORT BISECTION IMPLEMENTATION IN MATLAB

LONG CHEN

ABSTRACT. This is the documentation of the local mesh refinement using newest bisection or longest bisection in MATLAB. The new feature of our implementation is the edge marking strategy to ensure the conformity. The short implementation is helpful for the teaching of adaptive finite element methods and programming in more advanced languages.

1. INTRODUCTION

The aim of this report is to document a short implementation of newest vertex bisection and longest edge bisection in 60 lines in MATLAB. This short implementation is helpful for the teaching of adaptive finite element methods. It can be easily adapted to other language, such as Fortran and C. The new feature of our algorithm is a new edge marking strategy for the completion. This work is in the sprit to the "Ten digit, five seconds, and one page" [50] and continuation of a recent trend in the short implementation of algorithms in MATLAB [47, 17, 2, 6, 1, 37].

Adaptive finite element methods (AFEMs) are now widely used in the numerical solution of partial differential equations (PDEs) to achieve better accuracy with minimum degrees of freedom. A typical loop of AFEM through local refinement involves

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

More precisely to get a refined triangulation from the current triangulation, we first solve the PDE to get the solution on the current triangulation. The error is estimated using the solution, and used to mark a set of of triangles that are to be refined. Triangles are refined in such a way to keep two most important properties of the triangulations: shape regularity and conformity.

Recently, several convergence and optimality results have been obtained for adaptive finite element methods on elliptic PDEs [20, 34, 48, 25, 18, 16, 14, 35, 31, 15, 12] which justify the advantage of local refinement over uniform refinement of the triangulations.

In most of those works, newest vertex bisection is used in the **REFINE** step. It has been shown that the mesh obtained by this dividing rule is conforming and uniformly shape regular. In addition the number of elements added in each step is under constroled which is crucial for the optimality of the local refinement. Therefore we mainly discuss newest vertex bisection in this report and include another popular bisection rule, longest edge bisection, as a variant of it.

MATLAB allows one to very quickly implement numerical methods due to its vast predefined mathematical library and compact vector/matrix operations. The

price we pay for this simplicity is the efficiency of the algorithm. Since MATLAB is an interpretive language, the run time can be much greater than that of compiled programming language like Fortran or C. To speed up the algorithm in MATLAB, one should try to avoid `for` loops as much as possible using MATLAB's vectorized addressing and built-in functions. Our code is optimized in this sprit.

The outline of the rest is as the following. In Section 2, we review newest vertex bisection. In Section 3, we discuss the implantation details: data structure, marking strategies and refinement. In Section 4, we provide numerical examples and in Appendix we list the code.

2. Bisections in Adaptive Finite Element Methods

In this section we shall give a review of newest vertex bisection and longest edge bisection. In short, newest vertex bisection always bisects a triangle along the edge opposite to its newest vertex while longest edge bisection always bisects a triangle along the longest edge of a triangle.

Before we get into the details of those bisection methods, we first introduce two important properties of triangulations. A triangulation $\mathcal{T}_h$ (also indicated by mesh or grid) of $\Omega \subset \mathbb{R}^2$ is a decomposition of Ω into a set of triangles. It is called *conforming* or *compatible* if the intersection of any two triangles τ and τ' in $\mathcal{T}_h$ either consists of a common vertex x_i, edge E or empty. An edge of a triangle is called *non-conforming* if there is a vertex in the interior of that edge and that interior vertex is called *hanging node*. See Fig. 1(a) for examples of non-conforming triangles and hanging nodes.

For meshes with hanging nodes, several special basis and more complicated matrix assembly may be required. While for a conforming mesh, only one finite element basis for the reference element is necessary. We would like to keep this property of the triangulations.

A triangulation $\mathcal{T}_h$ is *shape regular* if

$$\max_{\tau\in\mathcal{T}_h} \frac{\operatorname{diam}(\tau)^2}{|\tau|} \leq \sigma \tag{2.1}$$

where $\operatorname{diam}(\tau)$ is the diameter of τ and $|\tau|$ is the area of τ. A sequence of triangulation $\{\mathcal{T}_k, k = 0, 1, \cdots\}$ is called *uniform shape regular* if σ in (2.1) is independent with k.

The shape regularity of triangulations assures that angles of the triangulation remains bounded away from 0 and π which is important to control the interpolation error in H^1 norm [3] and the condition number of the stiffness matrix [26].

Remark 2.1. *We would like to point out that in some applications with boundary layers or interior layers, an optimal triangulation may require a high aspect ratio corresponding to the Hessian matrix of the solution* [36, 23, 24, 22, 21, 27, 28, 19]. *In these cases, we do not need to keep the shape regularity of the triangulation but the conformity is still needed.*

After we marked a set of triangles to be refined, we need to carefully design the rule for dividing the marked triangles such that the refined mesh is still conforming

(a) Bisect a triangle (b) Completion

FIGURE 1. Newest vertex bisection

and shape regular. Such refinement rules include red and green refinement [8], longest edge bisection [40, 39] and newest vertex bisection [46]. In MATLAB's PDE toolbox, the first two refinement methods are implemented. As we point out in the introduction, we will mainly discuss newest vertex bisection and include longest edge bisection as a variant of it.

2.1. **Newest vertex bisection.** Given a shape regular triangulation $\mathcal{T}$ of Ω, for each triangle $\tau \in \mathcal{T}$, we label one vertex of τ as *peak* or *newest vertex*. The opposite edge of the peak is called *base* or *refinement edge*. This process is called a labeling of $\mathcal{T}$. The rule of newest vertex bisection includes:

(1) a triangle is bisected to two new children triangles by connecting the peak to the midpoint of the base;
(2) the new vertex created at a midpoint of a base is assigned to be the peak of the children.

Once the labeling is done for an initial triangulation, the decent triangulations inherit the label by the rule (2) such that the bisection process can proceed.

We now summarize three important properties of newest vertex bisection.

Sewell [46] showed that all the decendants of an original triangle fall into four similarity classes (Fig. 2.1) and hence triangulations obtained by newest vertex bisection is uniformly shape regular.

After the marked triangles are bisected, what in general breaks conformity, to recovery the conformity, bisections are propagated to eliminate the hanging nodes. See Fig. 1(b) for a illustration. This process is called *completion* or *closure*.

One step of completion may produce more hanging nodes and thus we have to show the completion process will terminate. Mitchell [32] and Bansch [11] show the completion will terminate. We shall discuss this issue in Section 3.2.

The last property is not used in this report but is essential for the optimality of the AFEM. It says the completion will not add too many elements comparing to the marked elements.

Binev, Dahmen and DeVore [14] show that if M is the collection of all triangles refined in going from some well labeled (we shall explore this later) initial conforming triangulation $\mathcal{T}_0$ to $\mathcal{T}_k$, then

$$\#\mathcal{T}_k \leq \#\mathcal{T}_0 + C\#M,$$

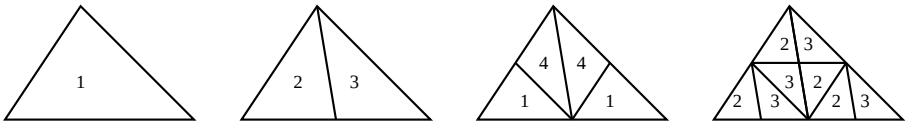


FIGURE 2. Four similarity classes of triangles generated by newest vertex bisection

where $\#A$ denotes the cardinality of the set A. Recently, this result is extended to a bisection scheme of simplices in general spacial dimensions by Stevenson [49].

2.2. **Longest edge bisection.** The longest edge bisection are proposed and studied by Rivara's group [41, 42, 43, 40]. In this method, triangles are always bisected using one of their longest edges.

In the longest edges bisection, every time the largest angle is divided and thus it is reasonable to expect this bisection will maintain the shape regularity. Indeed, it has been proven by Rosenberg and Stenger [44] that the smallest angle is at least half of the smallest angle in the initial triangulation. The termination of the completion process has been shown by Rivara.

We shall view the longest edge bisection as a variant of newest vertex bisection using a different labeling scheme. Namely we label the base of each triangle as its longest edge. Unlike newest vertex bisection, this labeling is performed in the beginning of each loop like (1.1). This viewpoint will unify the implementation of those two bisections methods.

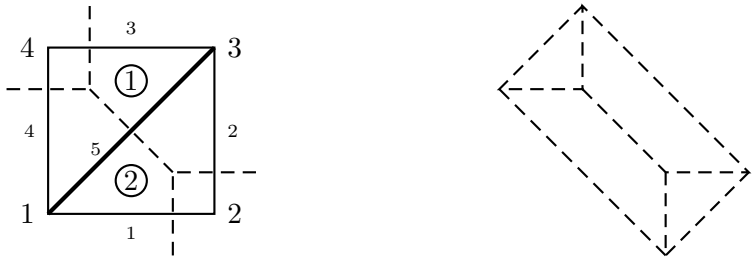
In some cases, those two bisections are equivalent. This happens whenever the longest edge is always opposite newest vertex. One important such example is uniform meshes: meshes obtained by dividing rectangles into triangles using their diagonals. The peaks are always at the right angles and the longest edges are opposite to the peaks. Indeed isosceles right triangles are optimal, in terms of angle conditions, for bisection methods [33].

3. IMPLEMENTATION

In this section, we shall discuss the implementation of newest vertex and longest edge bisection. The new feature is a new edge marking strategy to ensure the conformity which makes the bisection can be implemented in 60 lines in MATLAB.

3.1. **Data Structure.** We shall first discuss the data structure to represent triangulations and facilitate the refinement procedure. There is a diploma for the data structure in the implementation level. If more sophisticated data structure is used to easily traverse on the mesh, for example, to get all triangles surrounding a vertex or an edge, it will simplify the implementation of refinement. On the other hand, once a triangle is bisected, one has to update those sophisticated data structure which complicates the implementation.

We only keep updating two basic data structure `node`, `elem` to represent the triangulation. The following few lines will build other data structure used in the bisection algorithm. Before we explain it line by line, we would like to point out



(a) A simple triangluation (b) Dual graph

FIGURE 3. A triangulation and its dual graph

that since we make use built-in functions in MATLAB, the construction of those data structure is efficient. We thus rebuild those data structure in the beginning of each loop (1.1). If one uses Fortran or C and concerns the optimality of the operations, one has to update those auxiliary data structure whenever performing bisections of triangles.

```
edge=[elem(:,[1,2]); elem(:,[1,3]); elem(:,[2,3])];
edge=unique(sort(edge,2),'rows');
N=size(node,1); NT=size(elem,1); NE=size(edge,1);
dualedge=sparse(elem(:,[1,2,3]),elem(:,[2,3,1]),[1:NT,1:NT,1:NT],N,N);
d2p=sparse(edge(:,[1,2]),edge(:,[2,1]),[1:NE,1:NE],N,N);
```

`node`, `elem` are standard data structure to represent a triangulation. In the node matrix `node`, the first and second rows contain x- and y-coordinates of the nodes in the mesh. In the element matrix `elem`, the three rows contain indices to the vertices of elements, given in anti-clockwise order.

Note that a cyclical permutation of three indices of a triangle represents the same triangle. We shall make use the order of vertices to represent a labeling of a triangle. Namely we assume `elem(t,1)` is always the peak of t. New added elements will follow this rule.

For the triangulation in Fig. 3.1, the indices of nodes is indicated by bigger numbers and the indices of triangles is given by circled numbers. Suppose it represents the square $[0,1] \times [0,1]$. Then the `node`, `elem` matrix are given in the Table 3.1.

```
edge=[elem(:,[1,2]); elem(:,[1,3]); elem(:,[2,3])];
edge=unique(sort(edge,2),'rows');
```

In the edge matrix `edge`, the first and second rows contain indices of the starting and ending point. The row of `edge` is sorted in the way that for every edge k, `edge(k,1)<edge(k,2)`. For the triangulation in Fig. 3.1, the indices for edges is in small numbers. The `edge` array is listed in the Table 3.1.

```
N=size(node,1); NT=size(elem,1); NE=size(edge,1);
```

`N`, `NT`, and `NE` are number of nodes, triangles, and edges, respectively.

```
dualedge=sparse(elem(:,[1,2,3]),elem(:,[2,3,1]), [1:NT,1:NT,1:NT],N,N);
```

For a given triangulation $\mathcal{T}$, its dual graph is defined as follows. Triangles in $\mathcal{T}$ are interior dual nodes in the dual graph and those nodes are connected if two

t	elem(t,1)	elem(t,2)	elem(t,3)
1	2	3	1
2	4	1	3

v	node(v,1)	node(v,2)
1	0	0
2	1	0
3	1	1
4	0	1

k	edge(k,1)	edge(k,2)
1	1	2
2	2	3
3	3	4
4	1	4
5	1	3

(i,j)	dualedge(i,j)	d2p(i,j)
(3,1)	2	5
(1,3)	1	5
(1,2)	2	1
(2,1)	0	1
(2,3)	2	2
(3,2)	0	2
(3,4)	1	3
(4,3)	0	3
(4,1)	1	4
(1,4)	0	4

TABLE 1. Data structure for the triangulation in Fig. 3(a)

triangles are neighbors i.e. they share a common edge. We also introduce boundary dual nodes in the dual graph for boundary edges and connect dual boundary nodes to interior dual nodes if the corresponding edges of a boundary dual node is an edge of the triangle corresponding to the interior dual node. Two boundary nodes are connected if the corresponding edges share a common vertex. Fig. 3(b) is the dual graph of the triangulation in Fig. 3(a).

`dualedge` is an $N \times N$ sparse non-symmetric matrix which stores the non-boundary edges of the dual graph of the triangulation. Namely the dash line in Fig. 3(a). It can be formed by the following loop.

```
for t=1:NT
    dualedge(elem(t,1),elem(t,2))=t;
    dualedge(elem(t,2),elem(t,3))=t;
    dualedge(elem(t,3),elem(t,1))=t;
end
```

By the definition, `dualedge(i,j)` denotes the element τ (if it exists) such that v_iv_j is an edge of τ and `dualedge(j,i)` will give another (if it exists) element τ' such that edge v_jv_i is an edge of τ'. If one of them is zero, it implies that this edge is a boundary edge.

One should avoid `for` loop as much as possible when coding in MATLAB since each line in the loop will be interpreted in each iteration. This can quickly add significant overhead when dealing with large systems (as is often the case with finite element codes). To this end, we use the `sparse` function because of the nice summation property for duplicated indices. Try `help sparse` in MATLAB for the usage. It will become quite nature and useful after you get used to it.

```
d2p=sparse(edge(:,[1,2]),edge(:,[2,1]),[1:NE,1:NE],N,N);
```

`d2p` is an $N \times N$ symmetric sparse matrix which denotes the index map between dual edge and primary edge set. Again we use `sparse` command which does the same as the following loop

```
for k=1:NE
    i=edge(k,1); j=edge(k,2);
    d2p(i,j)=k; d2p(j,i)=k;
end
```

3.2. **Initial labeling.** In this subsection, we shall discuss labeling algorithms for the initial triangulation which is crucial to ensure the completion will terminate in finite steps.

We begin with some concepts. Given a labeled triangulation $\mathcal{T}$, an edge is *compatible* if it is the base of all triangles sharing it. A triangle is *compatible* if its base is compatible. A labeled triangulation $\mathcal{T}$ is called *compatible* if every triangle in $\mathcal{T}$ is compatible and the label of $\mathcal{T}$ is called a *compatible label*. Fig. 3.2 shows a compatible (Fig. 4(b)) and non-compatible (Fig. 4(a)) label for a triangulation. In this figure, the peak of each triangle is denoted by a small dot near the vertex. It is obvious that the completion for a compatible triangulation is terminated in one step.

Theorem 3.1 (Mitchell 1988)**.** *For any conforming triangulation $\mathcal{T}$, there exist a compatible label scheme.*

Proof. For simplicity, we assume Ω is simply connected. It is easy to see that the dual graph of $\mathcal{T}$ is a 3-regular graph, i.e. each dual node has degree 3. And the dual graph has no cut edges, i.e. removing one dual edge, the dual graph is still connected. Note that a compatible label corresponds to a perfect matching of the dual graph. By the graph theory [30], which can be traced back to Petersen (1891) [38], every 3-regular graph without cut edges has a perfect matching. □

The above theorem only ensures the existence of a compatible label. The next question is on algorithms for this compatible label. Although for general k-regular graph (every node has k-degree), the problem to find a perfect matching is N-P hard, for 3-regular planar graph, there does exist polynomial time algorithms to find a perfect matching. Recently in [13] it has been shown that this even can be done in $O(N)$ operations for a triangulation with N elements.

Theorem 3.2 (Biedl et al 1999)**.** *For a triangulation $\mathcal{T}$ with N triangles, there exists an $O(N)$-time algorithm which give a compatible label of $\mathcal{T}$.*

It is a temptation to put such labeling algorithm inside newest vertex bisection. Namely we always call a labeling algorithm in the beginning of loop (1.1) such that the current triangulation is compatible. Then no extra effort is required to ensure the conformity. However since we change the peaks in every loop, we cannot prevent the angle to be unacceptably large or small. Furthermore, Mitchell [32] shows that if the the refined triangulations are always compatible, the only possibility is to perform the uniform refinement of an initial compatible triangulation.

(a) Non-compatible label (b) Compatible label

FIGURE 4. Different labeling for a triangulation

In this sense, the complexity of the completion for non-compatible triangulation is the price we have to pay for the efficiency of the local refinement.

It is desirable that the initial labeling can be done in an optimal way. But the algorithm presented in [13] is kind of complicate. Note that for newest vertex bisection we only need to labeling the initial triangulation which has, in practice, small number of elements. We would like to sacrifice the complexity for the simplicity.

In our implementation, we shall choose a longest edge as the base for each triangle in the initial triangulation $\mathcal{T}_0$. Such labeled triangulation may not be compatibly divisible. But we shall show the completion will termiante for every triangulation refined from such labeled $\mathcal{T}_0$ using newest vertex bisection in Section 3.3.

Although the completion will terminate for longest edge labeling, the refinement of an element can enforce the bisection of remote elements. Here we use this method for its simplicity. If one implement bisection using Fortran or C, one had better give a compatible labeling for the initial triangulation. For a simple algorithm, we refer to Zikatanov [54].

We now list and explain our code for the labeling.

```
function elem=label(node,elem)
edgelength(:,1)=(node(elem(:,3),1)-node(elem(:,2),1)).^2 ...
               +(node(elem(:,3),2)-node(elem(:,2),2)).^2;
edgelength(:,2)=(node(elem(:,1),1)-node(elem(:,3),1)).^2 ...
               +(node(elem(:,1),2)-node(elem(:,3),2)).^2;
edgelength(:,3)=(node(elem(:,3),1)-node(elem(:,2),1)).^2 ...
               +(node(elem(:,3),2)-node(elem(:,2),2)).^2;
[temp,I]=max(edgelength,[],2);
elem((I==2),[1 2 3])=elem((I==2), [2 3 1]);
elem((I==3),[1 2 3])=elem((I==3), [3 1 2]);
```

The first 3 lines is to compute the edge lengths. Then we find the longest edge and switch the nodes of each triangle such that `elem(t,1)` is always opposite to the longest edge.

Note that for newest vertex bisection, once the peak (or base) for the initial triangulation is chosen, the peak or base for decent triangulations are assigned by newest vertex bisection rule. Thus we only need to call this subroutine for the initial triangulation.

If we relabel the longest edge as the base in each loop, then it becomes longest edge bisection. Thus we include a `method` input argument in our bisection algorithm.

```
function [node,elem,marker,d2p] = bisection(node,elem,eta,theta,method)
if (method==1), elem = label(node,elem); end
```

When `method==1`, we call the label subroutine and the bisection becomes longest edge bisection. Otherwise it is newest vertex bisection.

3.3. **Marking strategy.** In this subsection, we shall discuss marking strategies used in our algorithm. We need to distinguish two types of marking: one is to mark triangles for reducing the error and another one is to mark edges for the conformity.

Marking triangles for reducing the error. Let us first discuss the marking strategy used with the error indicator. Let

$$\eta^2 = \sum_{\tau\in\mathcal{T}_h} \eta_\tau^2$$

be an error indicator with local contributions η_τ associated with a triangle τ. The traditional maximum marking strategy is to mark triangulations τ^* such that

$$\eta_{\tau^*} \geq \theta \max_{\tau\in\mathcal{T}_h} \eta_\tau, \quad \text{for some } \theta \in (0,1).$$

This marking strategy is proposed in the pioneering work [4] by Babuska and Vogelius and currently used in the MATALBE `adaptmesh` function.

We shall use the bulk marking strategy proposed by Dörfler [25]. With such strategy, one defines the marking set M such that

$$\sum_{\tau\in M} \eta_\tau^2 \geq \theta\, \eta^2, \quad \text{for some } \theta \in (0,1). \tag{3.1}$$

Bigger θ will result more refinement of triangles in one loop and smaller θ will result more optimal grid but more refinement loops. Usually we choose $\theta = 0.2 - 0.5$.

The advantage of the bulk marking strategy is that one can prove for elliptic problems, with other reasonable assumptions, the approximation error is decreased by a fixed factor for each loop (1.1) and thus the local refinement will convergence. Furthermore it will give optimal numerical approximation in terms of the number of degrees of freedoms. For details we refer to [20, 34, 48, 25, 18, 16, 14, 35, 31, 15, 12].

Completion. After those marked triangles are bisected, the crucial problem now is maintaining of the mesh conformity. We first survey two basic approaches for the completion and then propose a new edge marking strategy.

A standard iterative algorithm of the completion is the following. Let M denotes the set of triangles to be refined.

Iterative algorithm
SUBROUTINE completion
WHILE $M \neq \emptyset$ **DO**

Bisect each $t \in M$;
Let now M be the set of non-conforming triangles.
END WHILE

A more efficient recursive algorithm is proposed by Mitchell [32, 33] and generalized by Kossaczký [29] to 3-D. Because of the efficiency, it is implemented in the adaptive finite element package ALBERTA [45]. This approach is based on an observation that if a triangle is not compatible, then after a single division of the the neighbor opposite the peak, it will be. Of course, it may be possible that the neighboring triangle is also not compatible, so the algorithm recursively check the neighboring triangle until a compatible triangle is found. The recursion occurs before the division, so it always bisect a pair of compatible triangles (except near the boundary) and thus the conformity is ensured.

Recursive algorithm
SUBROUTINE divide_triangle(τ)
IF τ is not compatible **THEN**
divide_triangle(neighbor of τ opposite the peak)
divide the triangle τ and the neighbor opposite the peak of τ.
END IF

One has to show the recursion will terminate. Indeed there exists a labeled triangulation such that the recursion does not terminate. See Fig. 4(a) for such an example. Note that the labeled triangulation in Fig. 4(a) is non-compatible. Mitchell [32, 33] proved that if the initial triangulation is compatible labeled, then the recursion will terminate. Let the triangles of the initial triangulation be assigned generation 1. Let children have generation $i + 1$ where i is the generation of the parent.

Theorem 3.3 (Mitchell 1988)**.** *If the initial triangulation is compatible, the length of the recursion of* **divide_triangle***(t) in* **Recursive algorithm** *is bounded by the generation of τ.*

The proof of this theorem based on the fact that if a triangle is not compatible, the triangle opposite its peak has one less generation provided that the initial triangulation is compatible. And thus the generation of the triangles decreases with each recursive call. Since the minimum generation is 1, the number of recursive calls is bounded by the generation of the first triangle.

We can relax the compatible requirement for the initial triangulation. For example, we can use longest edge labeling. Indeed Kossaczký justify this choice.

Theorem 3.4 (Kossaczký 1994)**.** *Suppose that each triangle of the initial triangulation has the unique longest edge. Let the base of each triangle of an initial triangulation $\mathcal{T}_0$ be the longest edge of the triangle. Then* **divide_triangle***(t) stops on $\mathcal{T}_0$ and also on every refinement of $\mathcal{T}_0$, created by this method.*

The above theorem need a well-defined tie-breaking for the comparison of the edge length. Due to the round-off error, the example in Fig. 4(a) is still possible. Note that many mesh generators will generate grids containing a sub-grid like

Fig. 4(a). To apply the recursive algorithm, it is safe to get a compatible initial triangulation.

Marking edges for the conformity. We shall propose a new approach to address the conformity issue. Note that in the output mesh, the new points are always middle points of some edges of the input mesh. Instead of operating on triangles, we mark a set of edges. The marking strategy is that if an edge is marked, then the base of the triangles sharing that edge should be marked too.

This marking strategy can be implemented in a recursive way using a minor modification of **divide_triangle(t)**.

Recursive edge marking algorithm
SUBROUTINE mark(τ)
mark the base of τ
τ' = neighbor of τ opposite the peak
IF base of τ' is not marked **THEN**
  mark(τ')
END IF

The termination of the recursion is obvious since every recursion will mark an edge and the number of edges for the current triangulation are finite.

To control the stack and easily access to the data structure, we implement a non-recursive edge marking strategy. We list our code below.

```
total = sum(eta); [temp,ix] = sort(-eta);
current = 0; marker = zeros(NE,1);
for t = 1:NT
    if (current > theta*total), break; end
    index=1; ct=ix(t);
    while (index==1)
       base = d2p(elem(ct,2),elem(ct,3));
       if (marker(base)>0), index=0;
       else
          current = current + eta(ct);
          N = size(node,1)+1;
          marker(d2p(elem(ct,2),elem(ct,3))) = N;
          node(N,:) = mean(node(elem(ct,[2 3]),:));
          ct = dualedge(elem(ct,3),elem(ct,2));
          if (ct==0), index=0; end
       end
    end
end
```

The first line is to compute the total sum of the error estimator η and sort the index by the value of η in decent order. `current` is used to denote the current summation of the error indicator contributed by marked triangles. We do a loop for `elem` matrix, and mark the triangles. Line 4 is the bulk marking strategy.

The array `marker` is used to denote wether the edge is marked or not. If `marker(k)=0`, it means the k-th edge is not marked. Otherwise `marker(k)` denotes the global index of the new nodes (the middle point of the k-th edge).

For current triangle `ct`, we first get its base using `d2p` matrix. If the base of `ct` is not marked, we mark this edge, introduce the new nodes and add the coordinates

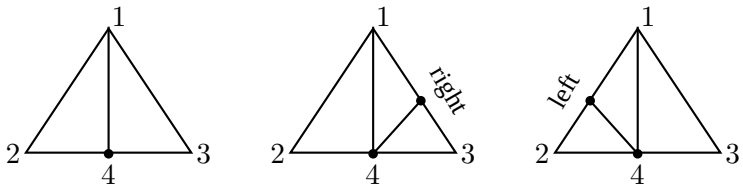


FIGURE 5. Divide a triangle according to the marked edges

of this new nodes is added into the `node` matrix. Then we take the neighbor containing its base and repeat the process. The `while` loop ends until the base of `ct` is already marked or it is a boundary edge.

We only append new nodes in the `node` matrix during the marking. Since we do not bisect any triangles, we can keep using the auxiliary data structure `edge`, `dualedge`,`d2p`.

3.4. **Refinement.** Refinement is short and easy since the conformity is ensured in the marking step. It is purely local in the sense that we only need to divide each triangle according to how many edges are marked.

```
for t=1:NT
    base=d2p(elem(t,2),elem(t,3));
    if (marker(base)>0)
       p=[elem(t,:), marker(base)];
       elem=divide(elem,t,p);
       left=d2p(p(1),p(2)); right=d2p(p(3),p(1));
       if (marker(right)>0)
           elem=divide(elem,size(elem,1), [p(4),p(3),p(1),marker(right)]);
       end
       if (marker(left)>0)
          elem=divide(elem,t,[p(4),p(1),p(2),marker(left)]);
       end
    end
end
%-------------------------------------------------------------------------
function elem=divide(elem,t,p)
elem(size(elem,1)+1,:)=[p(4),p(3),p(1)]; elem(t,:)=[p(4),p(1),p(2)];
```

We first explain the `divide` function. `t` is the current triangle to be divided. Its vertices are $p(1), p(2), p(3)$, and $p(4)$ is the new vertex added in its base (Fig. 5). We modify the current triangle `t` and add one new element to `elem` array. Note that in those elements, the first node is changed to $p(4)$, newest vertex added by the bisection.

We do a loop for `elem` matrix. For each triangle, we first check if its base is marked. If so we divide it and then check the other two edges. If one of them is marked, we then divide children with suitable order. See Fig. 5 for an illustration.

4. NUMERICAL EXAMPLE

In this section, we shall present a numerical example to show how to cooperate our bisection algorithm in adaptive finite element methods.

We will solve the following crack problem consider in [35, 51]. Let $\Omega = \{|x| + |y| < 1\} \backslash \{0 \le x \le 1, y = 0\}$ with a crack and the solution u satisfies the Poisson equation

$$-\Delta u = f, \text{ in } \Omega \quad u = u_D \text{ on } \Gamma_1, \quad \text{and} \quad \frac{\partial u}{\partial n} = g \text{ on } \Gamma_2,$$

where $f = 1$, $\Gamma_1 = \partial\Omega$, and $\Gamma_2 = \emptyset$. We choose u_D such that the exact solution u in polar coordinates is

$$u(r, \theta) = r^{\frac{1}{2}} \sin \frac{\theta}{2} - \frac{1}{4} r^2.$$

We use piecewise linear and global continuous finite element to solve this Poisson equation. The initial grid is given by hand. For general domains, one can use MATLAB's PDE tool box or use distmesh [37], a simple mesh generator written in MATLAB, to generate an initial mesh and call `label` for the initial labeling.

For the assembling and solving of Poisson equation, we use the short finite element implementation in [1]. We optimize the code by avoiding the `for` loop. It is much faster than the original implementation in [1].

We shall use $|u|_{2,1,\tau}$, the $W^{2,1}$ norm of u, as our error indicator. By the embedding theorem and the optimality of the finite element solution u_h based on the triangulation $\mathcal{T}_h$, it is easy to show that

$$|u - u_h|_1^2 \le C \sum_{\tau \in \mathcal{T}_h} |u|_{2,1,\tau}^2.$$

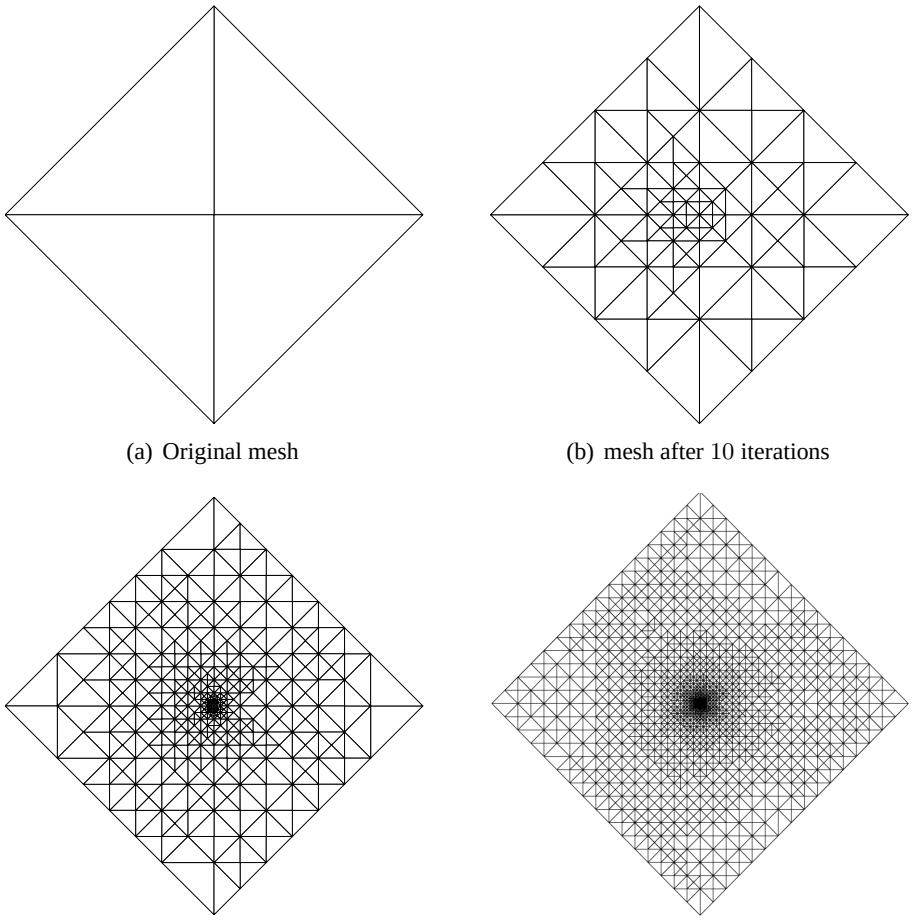
Furthermore, if the triangulation $\mathcal{T}_N$ with N elements equidistributes the $W^{2,1}$ norm of u in the sense that $|u|_{2,1,\tau} \le CN^{-1}|u|_{2,1,\Omega}$, $\forall \tau \in \mathcal{T}_N$, then the finite element approximation u_N based on $\mathcal{T}_N$ is of optimal approximation order [5]:

$$|u - u_N|_1 \le CN^{-1/2}|u|_{2,1,\Omega}. \tag{4.1}$$

Since u is unknown and $D^2 u_h$ is zero almost everywhere, in the estimate function, we first use simplest Zienkiewicz-Zhu recovery [52, 53] to get a piecewise linear approximation of ∇u, denoting by Ru_h, and then use ∇Ru_h as a piecewise constant approximation of $D^2 u$. Sophisticated approximation of $D^2 u$, for example, the method by Bank and Xu [9, 10], can be also implemented efficiently in MATLAB.

After the current triangulation is refined, one needs to update the boundary edges. Thanks to our data structure, the updating of boundary edges can be simply done by the following function. In the function, `bdedge` is a subset of `edge` array.

```
function bdedge=updatebd(marker,bdedge,d2p)
ND=size(bdedge,1);
for k=1:ND
    i=bdedge(k,1); j=bdedge(k,2);
    if marker(d2p(i,j)) >0
       bdedge(k,:)=[i,marker(d2p(i,j))];
       bdedge(size(bdedge,1)+1,:)=[j,marker(d2p(i,j)];
   end
end
```



(a) Original mesh (b) mesh after 10 iterations

(c) mesh after 15 iterations (d) mesh after 20 iterations

FIGURE 6. Meshes after different iterations

For domains with curved boundaries, one needs to project the middle points on the boundary edges to the boundary. To this end, `bdedge` should store more information; See, for example, the data structure used in PLTMG [7].

Fig. 6 displays the grid for several loops of (1.1). The MATLAB code for the crack problem is listed in Appendix.

APPENDIX A: MATLAB CODE FOR BISECTION

```
function [node,elem,marker,d2p] = bisection(node,elem,eta,theta,method)
if (method==1), elem = label(node,elem); end
edge = [elem(:,[1,2]); elem(:,[1,3]); elem(:,[2,3])];
edge = unique(sort(edge,2),'rows');
N = size(node,1); NT = size(elem,1); NE = size(edge,1);
dualedge = sparse(elem(:,[1,2,3]),elem(:,[2,3,1]),[1:NT,1:NT,1:NT],N,N);
d2p = sparse(edge(:,[1,2]),edge(:,[2,1]),[1:NE,1:NE],N,N);
total = sum(eta); [temp,ix] = sort(-eta);
current = 0; marker = zeros(NE,1);
for t = 1:NT
    if (current > theta*total), break; end
    index=1; ct=ix(t);
    while (index==1)
       base = d2p(elem(ct,2),elem(ct,3));
       if (marker(base)>0), index=0;
       else
          current = current + eta(ct);
          N = size(node,1)+1;
          marker(d2p(elem(ct,2),elem(ct,3))) = N;
          node(N,:) = mean(node(elem(ct,[2 3]),:));
          ct = dualedge(elem(ct,3),elem(ct,2));
          if (ct==0), index=0; end
       end
    end
end
for t = 1:NT
    base = d2p(elem(t,2),elem(t,3));
    if (marker(base)>0)
        p = [elem(t,:), marker(base)];
        elem = divide(elem,t,p);
        left = d2p(p(1),p(2)); right = d2p(p(3),p(1));
        if (marker(right)>0)
            elem = divide(elem,size(elem,1),[p(4),p(3),p(1),marker(right)]);
        end
        if (marker(left)>0)
            elem = divide(elem,t,[p(4),p(1),p(2),marker(left)]);
        end
  end
end
hold off; trimesh(elem,node(:,1),node(:,2),zeros(size(node,1),1));
view(2),axis equal,axis off
%---------------------------------------------------------------------
function elem = divide(elem,t,p)
elem(size(elem,1)+1,:) = [p(4),p(3),p(1)];
elem(t,:) = [p(4),p(1),p(2)];
%---------------------------------------------------------------------
function elem=label(node,elem)
edgelength(:,1)=(node(elem(:,3),1)-node(elem(:,2),1)).^2 ...
                + (node(elem(:,3),2)-node(elem(:,2),2)).^2;
edgelength(:,2)=(node(elem(:,1),1)-node(elem(:,3),1)).^2 ...
                + (node(elem(:,1),2)-node(elem(:,3),2)).^2;
edgelength(:,3)=(node(elem(:,3),1)-node(elem(:,2),1)).^2 ...
                + (node(elem(:,3),2)-node(elem(:,2),2)).^2;
[temp,I]=max(edgelength,[],2);
elem((I==2),[1 2 3])=elem((I==2),[2 3 1]);
elem((I==3),[1 2 3])=elem((I==3),[3 1 2]);
```

APPENDIX B: MATLAB CODE FOR THE CRACK PROBLEM

```
function crack
clear all
% Initial Mesh
node = [1,0; 0,1; -1,0; 0,-1; 0,0; 1,0];
elem = [5,1,2; 5,2,3; 5,3,4; 5,4,6];
Dirichlet = [1,2; 2,3; 3,4; 4,6; 1,5; 5,6];
Neumann = [];
% Adaptive Finite Element Method
for step = 1:20
    % Step1: Solver
    u = Poisson(node, elem, Dirichlet, Neumann, @f, @u_D, @g);
    % Step2: Estimate
    eta = estimate(node,elem,u);
    % Step3: Refine
    [node,elem,marker,d2p] = bisection(node,elem,eta.^2,0.4,0);
    % refine boundary edges
    Dirichlet = updatebd(Dirichlet,marker,d2p);
    Neumann = updatebd(Neumann,marker,d2p);
end
u = Poisson(node, elem, Dirichlet, Neumann, @f, @u_D, @g);

% -------------------------------------------------------------------------
function z = f(p) % data
z = 1;
% -------------------------------------------------------------------------
function z = u_D(p) % Dirichlet boundary condition
r = sqrt(sum(p.^2,2));
z = sqrt(0.5*(r-p(:,1)))-0.25*r.^2;
% -------------------------------------------------------------------------
function z = g(p) % Neumann boundary condition
z = 0;
```

REFERENCES

[1] J. Alberty, C. Carstensen, and S. A. Funken. Remarks around 50 lines of Matlab: short finite element implementation. *Numerical Algorithms*, 20:117–137, 1999.

[2] J. Alberty, C. Carstensen, S. A. Funken, and R. Klose. Matlab implementation of the finite element method in elasticity. *Computing*, 69(3):239–263, 2002.

[3] I. Babuška and A. K. Aziz. On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, 13(2):214–226, 1976.

[4] I. Babuška and M. Vogelius. Feeback and adaptive finite element solution of one-dimensional boundary value problems. *Numerische Mathematik*, 44:75–102, 1984.

[5] C. Bacuta, L. Chen, and J. Xu. An adaptive grid refinement procedure and its asymptotic convergence estimate. *Preprint*, 2005.

[6] C. Bahriawati and C. Carstensen. Three matlab implementations of the lowest-order Raviart-Thomas MFEM with a posteriori error control. *Computational Methods In Applied Mathematics*, 5(4):333–361, 2005.

[7] R. E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 8.0*. Software, Environments and Tools, Vol. 5, SIAM, Philadelphia, 1998.

[8] R. E. Bank, A. H. Sherman, and A. Weiser. Refinement algorithms and data structures for regular local mesh refinement. In *Scientific Computing*, pages 3–17. IMACS/North-Holland Publishing Company, Amsterdam, 1983.

[9] R. E. Bank and J. Xu. Asymptotically exact a posteriori error estimators, Part I: Grids with superconvergence. *SIAM Journal on Numerical Analysis*, 41(6):2294–2312, 2003.

[10] R. E. Bank and J. Xu. Asymptotically exact a posteriori error estimators, Part II: General unstructured grids. *SIAM Journal on Numerical Analysis*, 41(6):2313–2332, 2003.
[11] E. Bänsch. Local mesh refinement in 2 and 3 dimensions. *Impact of Computing in Science and Engineering*, 3:181–191, 1991.
[12] E. Bänsch, P. Morin, and R. H. Nochetto. An adaptive Uzawa FEM for the stokes problem: Convergence without the inf-sup condition. *SIAM Journal on Numerical Analysis*, 40(4):1207–1229, 2002.
[13] T. C. Biedl, P. Bose, E. D. Demaine, and A. Lubiw. Efficient algorithms for petersen's matching theorem. In *Symposium on Discrete Algorithms*, pages 130–139, 1999.
[14] P. Binev, W. Dahmen, and R. DeVore. Adaptive finite element methods with convergence rates. *Numerische Mathematik*, 97(2):219–268, 2004.
[15] C. Carstensen and R. H. Hoppe. Error reduction and convergence for an adaptive mixed finite element method. *Mathematics of Computation*, 2005.
[16] C. Carstensen and R. H. W. Hoppe. Convergence analysis of an adaptive nonconforming finite element methods. *Numerische Mathematik*, 103(2):251–266, 2006.
[17] C. Carstensen and R. Klose. Elastoviscoplastic finite element analysis in 100 lines of Matlab. *J. Numer. Math.*, 10(3):157–192, 2002.
[18] C. Carstensen, A. Orlando, and J. Valdman. A convergent adaptive finite element method for the primal problem of elastoplasticity. *Preprint*, 2005.
[19] L. Chen. *Robust and Accurate Algorithms for Solving Anisotropic Singularities*. PhD thesis, Department of Mathematics, The Pennsylvania State University, 2005.
[20] L. Chen, M. Holst, and J. Xu. Convergence and optimality of adaptive mixed finite element methods. (submitted), 2006.
[21] L. Chen, P. Sun, and J. Xu. Optimal anisotropic simplicial meshes for minimizing interpolation errors in L^p-norm. *Mathematics of Computation*, 76(257):179–204, 2007.
[22] L. Chen and J. Xu. Optimal Delaunay triangulations. *Journal of Computational Mathematics*, 22(2):299–308, 2004.
[23] E. F. D'Azevedo and R. B. Simpson. On optimal interpolation triangle incidences. *SIAM Journal on Scientific and Statistical Computing*, 6:1063–1075, 1989.
[24] E. F. D'Azevedo and R. B. Simpson. On optimal regular meshes for minimizing the gradient error. *Numerische Mathematik*, 59:321–348, 1991.
[25] W. Dörfler. A convergent adaptive algorithm for Poisson's equation. *SIAM Journal on Numerical Analysis*, 33:1106–1124, 1996.
[26] I. Fried. Condition of finite element matrices generated from nonuniform meshes. *AIAA J.*, 10:219–221, 1972.
[27] W. Huang. Variational mesh adaptation: isotropy and equidistribution. *Journal of Computational Physics*, 174:903–924, 2001.
[28] W. Huang. Mathematical principles of anisotropic mesh adaptation. *Commun. Comput. Phys.*, 1:276–310, 2006.
[29] I. Kossaczký. A recursive approach to local mesh refinement in two and three dimensions. *Journal of Computational and Applied Mathematics*, 55:275–288, 1994.
[30] L. Lovász and M. D. Plummer. *Matching theory*. Akademiai Kiado (Budapest) and North-Holland Publishing Company (Amsterdam), 1986.
[31] K. Mekchay and R. Nochetto. Convergence of adaptive finite element methods for general second order linear elliptic PDE. *SIAM Journal on Numerical Analysis*, 43(5):1803–1827, 2005.
[32] W. F. Mitchell. *Unified Multilevel Adaptive Finite Element Methods for Elliptic Problems*. PhD thesis, University of Illinois at Urbana-Champaign, 1988.
[33] W. F. Mitchell. A comparison of adaptive refinement techniques for elliptic problems. *ACM Transactions on Mathematical Software (TOMS) archive*, 15(4):326 – 347, 1989.
[34] P. Morin, R. Nochetto, and K. Siebert. Data oscillation and convergence of adaptive FEM. *SIAM Journal on Numerical Analysis*, 38(2):466–488, 2000.
[35] P. Morin, R. H. Nochetto, and K. G. Siebert. Convergence of adaptive finite element methods. *SIAM Review*, 44(4):631–658, 2002.

[36] E. Nadler. *Piecewise Linear Approximation on Triangulations of a Planar Region*. PhD thesis, Brown University, 1985.
[37] P.-O. Persson and G. Strang. A simple mesh generator in matlab. *SIAM Review*, 46(2):329–345, 2004.
[38] J. P. C. Petersen. Die theorie der regulären graphs (the theory of regular graphs). *Acta Math.*, 15:193–220, 1891.
[39] M. C. Rivara. Design and data structure for fully adaptive, multigrid finite element software. *ACM Trans. Math. Soft.*, 10:242–264, 1984.
[40] M. C. Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis*, 21:604–613, 1984.
[41] M.-C. Rivara and P. Inostroza. Using longest-side bisection techniques for the automatic refinement fo Delaunay triangulations. *International Journal for Numerical Methods in Engineering*, 40:581–597, 1997.
[42] M. C. Rivara and G. Iribarren. The 4-triangles longest-side partition of triangles and linear refinement algorithms. *Mathemathics of Computation*, 65(216):1485–1501, 1996.
[43] M. C. Rivara and M. Venere. Cost analysis of the longest-side (triangle bisection) refinement algorithms for triangulations. *Engineering with Computers*, 12:224–234, 1996.
[44] I. G. Rosenberg and F. Stenger. A lower bound on the angles of triangles constructed by bisecting the longest side. *Mathemathics of Computation*, 29:390–395, 1975.
[45] A. Schmidt and K. G. Siebert. *The finite element toolbox ALBERTA*. Springer-Verlag, Berlin, 2005.
[46] E. G. Sewell. Automatic generation of triangulations for piecewise polynomial approximation. In *Ph. D. dissertation*. Purdue Univ., West Lafayette, Ind., 1972.
[47] O. Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, 2001.
[48] R. Stevenson. Optimality of a standard adaptive finite element method. *Department of Mathematics*, 2005.
[49] R. Stevenson. The completion of locally refined simplicial partitions created by bisection. *Technique Report*, 2006.
[50] L. N. Trefethen. Ten digit algorithms. Mitchell Lecture, June 2005.
[51] H. Wu and Z. Chen. Uniform convergence of multigrid v-cycle on adaptively refined finite element meshes for second order elliptic problems. *Science in China: Series A Mathematics*, 49(1):1–28, 2006.
[52] O. C. Zienkiewicz and J. Z. Zhu. The superconvergence patch recovery and a posteriori error estimates. Part 1: The recovery techniques. *International Journal for Numerical Methods in Engineering*, 33:1331–1364, 1992.
[53] O. C. Zienkiewicz and J. Z. Zhu. The superconvergence patch recovery and a posteriori error estimates. Part 2: Error estimates and adaptivity. *International Journal for Numerical Methods in Engineering*, 33:1365–1382, 1992.
[54] L. Zikatanov. Private communication, 2006.