

# AFEM@MATLAB: A MATLAB PACKAGE OF ADAPTIVE FINITE ELEMENT METHODS

LONG CHEN AND CHEN-SONG ZHANG

## CONTENTS

1. Introduction	2
1.1. Main features	2
1.2. Importance of AFEM@matlab	2
2. Examples	3
2.1. Crack Domain Problem	3
2.2. L-shaped Domain Problem	3
2.3. Moving Circle Problem	3
3. Data Structure	4
3.1. Basic Mesh structure	4
3.2. Triangulation	4
3.3. Boundary Conditions	4
3.4. Type Information	4
3.5. Approximate Solution	5
3.6. Auxiliary Mesh Structures	5
4. Algorithms	7
4.1. FEM: Poisson equation	7
4.2. AFEM	7
4.3. Initial Meshing	8
4.4. Solving	8
4.5. Estimation	8
4.6. Marking	8
4.7. Refinement: the newest vertex bisection	9
4.8. Coarsening	10
References	11

## 1. INTRODUCTION

AFEM@matlab is a MATLAB package of adaptive finite element methods (AFEMs) for stationary and evolution partial differential equations in two spatial dimensions. It contains robust, efficient, and easy-following codes for the main building blocks of adaptive finite element methods. This will benefit not only finite element method education but also future research and method development.

1.1. **Main features.** We summarize and emphasize the main features of our package as the following:

- It is concise and easy to follow. It will be a perfect match for the teaching of the finite element method (FEM) or numerical methods for partial differential equations.
- It includes newest development of AFEM including the design of a convergent adaptive algorithm with optimal complexity.
- It can be easily modified for solving other problems and for implementing in other programming languages. It will dramatically simplify the development of new adaptive methods based on the FEM.

Our package can be useful for education, communication, and research. We list possible benefits below:

- Speeding up program development;
- Facilitating comparisons of ideas and results;
- Improving academic publications.

1.2. **Importance of AFEM@matlab.** The finite element method is one of the most commonly used discretization methods for the numerical simulation of many practical models. The numerical experiments using the FEM need high accuracy to get reliable results. In 1991, an offshore oil platform in Norway sank in the North Sea costing \$ 700,000,000. The post accident investigation traced the error to inaccurate finite element approximation. On the other hand, to achieve high accuracy, it demands more physical memory and CPU time. To speed up numerical simulations, AFEMs are introduced to reduce computational costs while keeping optimal accuracy.

Because of its wide application, the FEM courses are usually offered to engineering and mathematics students in colleges. On the other hand, the programming of the FEM is not trivial for non-experts at all. Coding more complicated AFEM could be very time-consuming. There exist several powerful academic softwares such as ALBERTA [26], PLTMG [6], FEPG, and MC [21, 20]. Those packages are designed to handle a class of problems and thus are structured in a complicated way. The programming language used in those packages are either Fortran or C. Our package can serve as a tutorial to these more powerful but more complicated packages.

MATLAB is an interactive, matrix-based script language for scientific and engineering calculations, which is almost mandatory for engineering and mathematics students nowadays. MATLAB allows one to very quickly implement numerical methods due to its vast predefined mathematical library and compact vector/matrix operations.

The aim of AFEM@matlab is to implement main blocks of AFEMs using MATLAB. It is in the spirit to the “Ten digit, five seconds, and one page” [28]. For the easy of communication and education, our main algorithm will be written in one page long using compact data structures with useful and well laid-out comments. It is good for the academics and it opens the doors wider to non-academics. In spite of its brevity, the package is by no means a “toy” software. All the codes are written and optimized using MATLAB’s vectored addressing and built-in functions. Preliminary numerical tests show that our program can solve a middle size problem (about 10,000 unknowns) in seconds on a desktop PC.

## 2. EXAMPLES

We illustrate the usage of AFEM@MATLAB package by a set of classical test problems used for adaptive finite element methods.

There are two ways to launch test problems. One way is using a MATLAB GUI program

```
>> demo
```

which contains all the test problems:

- (1) **simple**: simple test problem solving the Poisson equation with homogenous Dirichlet boundary condition with uniform mesh;
- (2) **crack**: test problem solving the Poisson equation in a square domain with a crack using adaptive mesh refinements;
- (3) **Lshape**: test problem solving the Poisson equation in a L-shaped domain using adaptive mesh refinements;
- (4) **circle**: test problem with a moving circular singularity demonstrating the ability of refine/coarsening procedure.

The other way is to launch these test problems individually in a MATLAB command window. The test problems as well as how to call them individually are explained as follows.

**2.1. Crack Domain Problem.** Type the following command in MATLAB

```
>> crack
```

We will solve the following crack problem considered in [24, 29]. Let  $\Omega = \{|x|+|y| < 1\} \setminus \{0 \leq x \leq 1, y = 0\}$  with a crack and the solution  $u$  satisfies the Poisson equation

$$(2.1) \quad \begin{cases} -\Delta u = f & \text{in } \Omega \\ u = g_D & \text{on } \Gamma_D \\ \frac{\partial u}{\partial n} = g_N & \text{on } \Gamma_N, \end{cases}$$

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

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

**2.2. L-shaped Domain Problem.** Type the following command in MATLAB

```
>> Lshape
```

In the example, we consider the Poisson equation (2.1) in a L-shaped domain with a reentrant corner. Let  $f = 0$  in  $\Omega$  and choose the Dirichlet boundary condition  $g_D$  such that the exact solution can be written as

$$u(r, \theta) = r^{2/3} \sin\left(\frac{2}{3}\theta\right),$$

in polar coordinates.

**2.3. Moving Circle Problem.** Type the following command in MATLAB

```
>> circle
```

In this time-dependent test problem, we suppose that the exact solution of the problem

$$u(t, r) = \exp\left(-\left(\frac{r - 0.5t}{\epsilon}\right)^2\right),$$

is available to us and we can use the exact solution to estimate the local error and then perform refinement and coarsening procedures based on this error estimator.

### 3. DATA STRUCTURE

**3.1. Basic Mesh structure.** There is a dilemma for constructing data structures in the implementation level. If more sophisticated data structures are used for easily traversing on the mesh, for example, storing all triangles in the star around a vertex or an edge, it will simplify the implementation of refinement and coarsening. On the other hand, once a triangulation is changed, one has to update those sophisticated data structures which complicates the other parts of the implementation.

We try to keep our data structures as simple as possible. Our basic data structure is `mesh` which contains

- `mesh.node`
- `mesh.elem`
- `mesh.Dirichlet`
- `mesh.Newmann`
- `mesh.type`
- `mesh.solu`

**3.2. Triangulation.** `node`, `elem` are standard data structures to represent a triangulation. In the node array `node`, the first and second rows contain  $x$ - and  $y$ -coordinates of the nodes in the mesh. In the element array `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.



FIGURE 1. A triangulation and its dual graph

For the triangulation in Fig. 3.2, 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 3.2.

**3.3. Boundary Conditions.** `mesh.Dirichlet` and `mesh.Newmann` are boundary edges for different boundary conditions. It is a subset of `edge`; See 3.6 for the data structure of edges. Here we give an example for the triangulation in Fig. 3.2. One possibility of `mesh.Dirichlet` and `mesh.Newmann` is

- `mesh.Dirichlet`: (1, 2), (2, 3), (3, 4)
- `mesh.Newmann`: (1, 4)

**3.4. Type Information.** To be able to coarsening the grid, we include an array `type` in `mesh` structure.

- `mesh.type(j)=1`;  $j$ -th node is the node of initial grid or generated by regular uniform refinement which cannot be removed by coarsening procedure.

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. 1(a)

- `mesh.type(k)=2;`  $k$ -th node is the node added by the bisection and eligible to be coarsened.
- `mesh.type(i)=0;`  $i$ -th node is already removed by coarsening. We keep this node information `node` to avoid updating the indices in `elem` array.

It is surprising to us that only this extra data structure is needed for implementing the coarsening. We DO NOT maintain the tree structure (like the generation, parent, or children information for each element). Those are implicitly contained in the order we bisect the elements. This observation dramatically simplify the implementation and improve the performance. This will be explained further later in §4.

**3.5. Approximate Solution.** In `mesh` structure, `solu` is used to store the finite element approximation at each node of the current mesh. We choose to include it in the `mesh` structure since it will be more convenient to interpolate solutions between nested triangulations.

**3.6. Auxiliary Mesh Structures.** We only keep updating `mesh` during the adaptive loop. The following few lines will build other data structures needed in the bisection and coarsening algorithms. Before we explain it line by line, we would like to point out 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 function. If one uses Fortran or C and concerns the optimality of the operations, one has to update those auxiliary data structure whenever performing bisections or coarsening of triangles.

```
>> 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. 3.2, the indices for edges is in small numbers. The edge array is listed in the Table 3.2.

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

It is easy to see that  $N$ ,  $NT$ , 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 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. 1(b) is the dual graph of the triangulation in Fig. 1(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. 1(a). By the definition, `dualedge(i,j)` denotes the element  $\tau$  (if it exists) such that  $v_i v_j$  is an edge of  $\tau$  and `dualedge(j,i)` will give another (if it exists) element  $\tau'$  such that edge  $v_j v_i$  is an edge of  $\tau'$ . If one of them is zero, it implies that this edge is a boundary edge.

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
```

But 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 command window for the usage. Notice that if duplicated indices exist, `sparse` will add all the corresponding elements together. This feature is desirable for us. 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 set 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
```

## 4. ALGORITHMS

Now we shall give a brief description and some properties of algorithms implemented in our package.

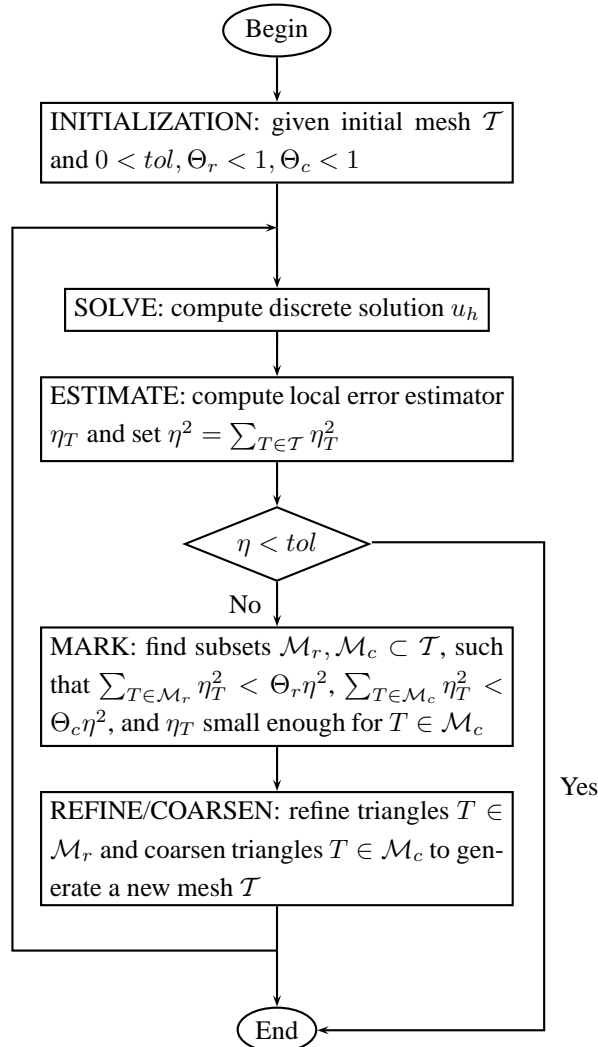
4.1. **FEM: Poisson equation.** See [1]. We shall include basics for completeness.

Several types of equations will be added in the later version including general elliptic equations and parabolic equations. We will not try to include all types of equations. Instead we will leave this to the user which is the main part for user's applications.

4.2. **AFEM.** Adaptive finite element methods are now widely used in numerical simulations of partial differential equations (PDEs) to achieve better accuracy with minimum degrees of freedom. A typical loop of AFEM for stationary problems through local refinement involves

(4.1) **SOLVE**  $\rightarrow$  **ESTIMATE**  $\rightarrow$  **MARK**  $\rightarrow$  **REFINE/COARSEN.**

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 triangles that are to be refined or coarsened. Triangles are refined or coarsened 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 [16, 23, 27, 18, 14, 13, 10, 24, 22, 12, 9] which justify the advantage of local refinement over uniform refinement of the triangulations.

**4.3. Initial Meshing.** A simple initial mesh could be given by hand. For more complicated domains, one can use MATLAB's PDE tool box or **distmesh** [25], a simple mesh generator written in MATLAB, to generate an initial mesh.

Once the `node`, `elem`, `Dirichlet`, `Neumann` are given, we can call the subroutine `getmesh` to generate the initial mesh data structure. Hence, `node`, `elem`, `Dirichlet`, `Neumann` are the only mesh and boundary data structures users need to provide.

We note that for domains with curved boundary and complicated boundary conditions, this step is not trivial. We view this as primarily a mesh generation problem. Robust algorithms to produce well-shaped tetrahedral meshes which are constrained to exactly match some interior embedded two-manifold are available in the literature; for example see [17, 2]

**4.4. Solving.** We use piecewise linear and global continuous finite element to solve this Poisson equation (2.1). 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 [5].

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

When solving the algebraic system generated by the finite element method, we just simply use MATLAB `build` in left matrix divide. For example, to solve  $Ax = b$ , we use  $x = A \backslash b$ . We are working on the fast solver using hierarchical structure of the mesh obtained by the newest vertex bisection. Examples of such optimal solvers are multigrid method or multigrid-based preconditioned conjugate gradient method [11, 19, 31, 30].

**4.5. Estimation.** We shall use the  $W^{2,1}(\tau)$  norm of  $u$  on  $\tau \subset \Omega$ ,  $|u|_{2,1,\tau}$ , 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

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

Furthermore, if the triangulation  $\mathcal{T}_N$  with  $N$  elements equidistributes the  $W^{2,1}$  norm of  $u$  in the sense that  $|u|_{2,1,\tau} \leq CN^{-1}|u|_{2,1,\Omega}$ ,  $\forall \tau \in \mathcal{T}_N$ , then the finite element approximation  $u_N$  based on  $\mathcal{T}_N$  is of optimal approximation order [4]:

$$(4.2) \quad |u - u_N|_{1,\Omega} \leq CN^{-1/2}|u|_{2,1,\Omega}.$$

Since  $u$  is unknown and  $D^2u_h$  is zero almost everywhere, in the estimate function, we first use simplest Zienkiewicz-Zhu recovery [32, 33] to get a piecewise linear approximation of  $\nabla u$ , denoting by  $Ru_h$ , and then use  $\nabla Ru_h$  as a piecewise constant approximation of  $D^2u$ . Sophisticated approximation of  $D^2u$ , for example, the method by Bank and Xu [7, 8] as well as many others, will be added in later versions.

**4.6. Marking.** 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  $\eta_\tau$  associated with a triangle  $\tau$ . The traditional maximum marking strategy is to mark triangulations  $\tau^*$  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 [3] by Babuska and Vogelius and currently used in the MATALBE `adaptmesh` function.

We shall use the bulk marking strategy proposed by Dörfler [18]. With such strategy, one defines the marking set  $M$  such that

$$(4.3) \quad \sum_{\tau \in M} \eta_\tau^2 \geq \theta \eta^2, \quad \text{for some } \theta \in (0, 1).$$

Bigger  $\theta$  will result more refinement of triangles in one loop and smaller  $\theta$  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 (4.1) and thus the local refinement will converge. Furthermore it will give optimal numerical approximation in terms of the number of degrees of freedoms.

**4.7. Refinement: the newest vertex bisection.** See [15]. We only give a brief discussion here. Given a shape regular triangulation  $\mathcal{T}$  of  $\Omega$ , for each triangle  $\tau \in \mathcal{T}$ , we label one vertex of  $\tau$  as *peak* or *newest vertex*. The opposite edge of the peak is called *base* or *refinement edge*. This process is called a labelling 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 labelling is done for an initial triangulation, the decent triangulations inherit the label by the rule (2) such that the bisection process can proceed.

*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
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)];
```

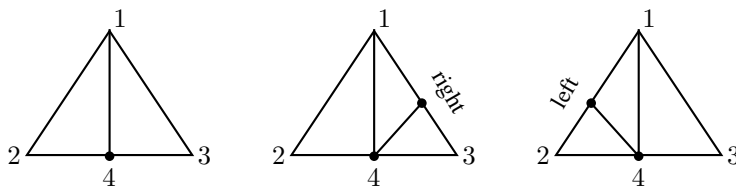


FIGURE 2. Divide a triangle according to the marked edges

We first explain the `divide` function.  $\tau$  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. 2). We modify the current triangle  $\tau$  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. 2 for an illustration.

**4.8. Coarsening.** In this section, we shall present our simple coarsening algorithm. We first observe that the coarsening can be only from good points that is remove this point will not destroy the conformity of the triangulation.

**Definition 4.1.** *A point is called a good point if it is the newest node in each of the elements with it as a node.*

The following characterization of is important for our coarsening.

**Theorem 4.2.** *A point is a good point if and only if*

- (1) *It is not a node in the initial triangulation;*
- (2) *It is the newest node of some element*
- (3) *Its valence is 4 (for an interior node) or 2 (for a boundary node).*

*Proof.* to be added. □

After we find out all good nodes, we traverse in the `elem` array to find the elements containing good points. Let  $t$  is such an element. Its brother will be found by using

```
>> brother = dualEdge(mesh.elem(t,2),mesh.elem(t,1));
```

This is because when we bisect an element, the left child is always keep in a prior of right child. Thus if we go from 1 to  $N$ , we always meet left children first. For an interior vertex, one many worry about the other two elements containing the same good point. Again like refinement, the coarsening is purely locally. The other two elements will be visited anyway.

In the coarsening procedure, it is possible that we will need to remove one node from  $\mathcal{N}$ . To make this updating process as simple as possible, we keep the index of this node. In such a way, there is no need to modify most part of the `elem` except those rows corresponding to the elements in the patch around  $z$ . To distinguish a node in use and an empty entry, we let the corresponding entry in `type` to be 0 to denote this node has been removed.

To take into account the coarsening error, we separate the marking and coarsening. But this time, the marker is for node.

## REFERENCES

- [1] J. Albery, C. Carstensen, and S. A. Funken. Remarks around 50 lines of Matlab: short finite element implementation. *Numerical Algorithms*, 20:117–137, 1999.
- [2] P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun. Variational tetrahedral meshing. *ACM Transactions on Graphics*, 24(3):617–625, 2005.
- [3] I. Babuška and M. Vogelius. Feedback and adaptive finite element solution of one-dimensional boundary value problems. *Numerische Mathematik*, 44:75–102, 1984.
- [4] C. Bacuta, L. Chen, and J. Xu. An adaptive grid refinement procedure and its asymptotic convergence estimate. *Preprint*, 2005.
- [5] 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.
- [6] R. E. Bank. *Pltmg: A software package for solving elliptic partial differential equations users' guide 9.0*. Department of Mathematics, University of California at San Diego, 2004.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] P. Binev, W. Dahmen, and R. DeVore. Adaptive finite element methods with convergence rates. *Numerische Mathematik*, 97(2):219–268, 2004.
- [11] J. H. Bramble and X. Zhang. The analysis of multigrid methods. In *Handbook of numerical analysis, Vol. VII*, pages 173–415. North-Holland, Amsterdam, 2000.
- [12] C. Carstensen and R. H. Hoppe. Error reduction and convergence for an adaptive mixed finite element method. *Mathematics of Computation*, 2005.
- [13] C. Carstensen and R. H. W. Hoppe. Convergence analysis of an adaptive nonconforming finite element methods. *Numerische Mathematik*, 103(2):251–266, 2006.
- [14] C. Carstensen, A. Orlando, and J. Valdman. A convergent adaptive finite element method for the primal problem of elastoplasticity. *Preprint*, 2005.
- [15] L. Chen. Short bisection implementation in MATLAB. *report*, 2006.
- [16] L. Chen, M. Holst, and J. Xu. Convergence and optimality of adaptive mixed finite element methods. (submitted), 2006.
- [17] L. Chen and M. J. Holst. Mesh adaptation based on Optimal Delaunay triangulations. *Preprint*, 2006.
- [18] W. Dörfler. A convergent adaptive algorithm for Poisson's equation. *SIAM Journal on Numerical Analysis*, 33:1106–1124, 1996.
- [19] W. Hackbusch. *Multigrid Methods and Applications*, volume 4 of *Computational Mathematics*. Springer-Verlag, Berlin, 1985.
- [20] M. Holst. *MCLite: An adaptive multilevel finite element MATLAB package for scalar nonlinear elliptic equations in the plane*. UCSD Technical report and guide to the MCLite software package, 2000.
- [21] M. Holst. Adaptive numerical treatment of elliptic systems on manifolds. *Advances in Computational Mathematics*, 15:139–191, 2001.
- [22] 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.
- [23] P. Morin, R. Nochetto, and K. Siebert. Data oscillation and convergence of adaptive FEM. *SIAM Journal on Numerical Analysis*, 38(2):466–488, 2000.
- [24] P. Morin, R. H. Nochetto, and K. G. Siebert. Convergence of adaptive finite element methods. *SIAM Review*, 44(4):631–658, 2002.
- [25] P.-O. Persson and G. Strang. A simple mesh generator in matlab. *SIAM Review*, 46(2):329–345, 2004.
- [26] A. Schmidt and K. G. Siebert. *The finite element toolbox ALBERTA*. Springer-Verlag, Berlin, 2005.
- [27] R. Stevenson. Optimality of a standard adaptive finite element method. *Department of Mathematics*, 2005.
- [28] L. N. Trefethen. Ten digit algorithms. Mitchell Lecture, June 2005.
- [29] H. Wu and Z. Chen. Uniform convergence of multigrid v-cycle on adaptively refined finite element meshes for second order elliptic problems. *Preprint*, 2003.
- [30] J. Xu. Iterative methods by space decomposition and subspace correction. *SIAM Review*, 34:581–613, 1992.
- [31] J. Xu and L. Zikatanov. The method of alternating projections and the method of subspace corrections in Hilbert space. *Journal of The American Mathematical Society*, 15:573–597, 2002.
- [32] 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.

- [33] 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.