Adaptive Algorithms and A Posteriori Error Estimates on Partitioned Meshes

Randolph E. Bank

Department of Mathematics, University of California at San Diego, La Jolla, CA 92093.

Jacques Périaux

Dassault Aviation DGT/DEA B.P. 300 92214 St Cloud Cedex France.

1. Introduction

In this paper, we describe some recent enhancements to the software package PLTMG [Ban94]. This package uses piecewise linear finite elements approximations, adaptive meshing procedures and multilevel iterative methods to solve elliptic partial differential equations posed on general regions of the plane. The enhancements described here are mainly concerned with the adaptive mesh generation capabilities of the package.

In the past, the linear equation solvers in PLTMG have been multigrid and hierarchical basis multi level solvers making use of a hierarchy of refined meshes generated through an adaptive mesh refinement process. More recently, the strong connection between HBMG and ILU factorization has been exploited in developing HBMG algorithms that construct the needed hierarchical basis implicitly through an incomplete factorization (grid coarsening) [BX94, BX96]. In terms of efficiency, such hierarchical basis *multigraph* solvers have been seen to be comparable to classical HBMG solvers, and in some cases even better, but make use of information only on the finest mesh.

In any event, with the linear algebra portion of the package able to work efficiently using information from only the finest mesh, it seemed appropriate to consider adaptive

algorithms which do not necessarily produce the usual hierarchy of meshes. This allowed some simple but interesting adaptive refinement, unrefinement and mesh smoothing strategies to be incorporated in the current code.

At the same time, the triangle tree data structures, previously necessary to maintain the hierarchy of meshes, were replaced by simplified data structures which maintain only the current mesh. An additional feature of these new data structures is the capability to deal with partitioned domains. This allows problems with periodic boundary conditions to be solved. This also allows solution of problems posed on more general partitioned domains typical of those arising from domain decomposition algorithms. One hopes that these additional capabilities will enhance the value of the package both as a research tool and in the solution of routine problems. We note that although the PLTMG package is not presently designed for a parallel computing environment, some of the necessary communication links for that environment are now established in the serial version.

The remainder of this manuscript is organized as follows. In Section 2., we briefly describe the data structures. In Section 3., we discuss some details of the resulting systems of linear equations. In Section 4. we describe a few enhancements for the a posteriori error estimates which form the basis of our adaptive procedures. The adaptive procedures themselves are described in Section 5., and a simple numerical example is presented in Section 6..

2. Data Structures

For convenience, we consider the model elliptic boundary value problem

$$-\Delta u + \sigma u = f(x,y) \qquad \text{for } (x,y) \in \Omega, \tag{1}$$

with $\sigma(x,y) \geq 0$ and boundary conditions

$$\begin{array}{rcll} u & = & g_2(x,y) & \text{for } (x,y) \in \partial\Omega_2, \\ \nabla u \cdot n & = & g_1(x,y) & \text{for } (x,y) \in \partial\Omega_1, \\ u & \text{and} & \nabla u \cdot n & \text{continuous for } (x,y) \in \partial\Omega_0. \end{array} \tag{2}$$

The details of the particular partial differential equation have no impact on our specification of the physical domain, so our remarks are relevant for very general systems of nonlinear PDES. The problem formulation (1)-(2) is sufficiently general that it is possible to specify inconsistent and ill-posed problems. This is not our intent here, so we assume that the specification is such that the problem is well posed. Here Ω is a (possibly disconnected) region in $\mathcal{R}^2$ with boundary $\partial\Omega = \partial\Omega_0 \cup \partial\Omega_1 \cup \partial\Omega_2$, and n is the unit normal. The boundary conditions are the usual Dirichlet and Neumann types on part of the boundary. The interesting case is $\partial\Omega_0$, which is best described in terms of a few illustrative examples.

First, suppose that Ω is connected, and we specify periodic boundary conditions on part of the boundary. We can then view corresponding pairs of points on the periodic boundary as being *linked* and interpret the continuity of u and $\nabla u \cdot n$ with respect to these linked boundaries. For example, if Ω is the unit square $(0,1) \times (0,1)$ with

periodic boundary conditions at $x = 0$ and $x = 1$, then the left and right boundaries of Ω are linked as described above.

Second, suppose that Ω is decomposed into two subregions $\Omega = \Omega_1 \cup \Omega_2$ with an internal interface Γ. Then the portions of the boundaries of Ω_i lying on the interface comprise $\partial\Omega_0$ and once again are linked, with continuity of u and $\nabla u \cdot n$ again interpreted with respect to the linked boundaries.

There is nothing new in this formulation using linked boundaries, but by viewing the domain as defined in this way, all the necessary information about the linked boundaries is automatically built into the data structures defining the domain, and the necessary communication for the linked boundaries will already be present in a serial implementation. The differences between serial and parallel implementations should thus be diminished.

We now describe the data structures used for our finite element mesh. We suppose that $\mathcal{T}$ is a triangulation of the region Ω. We assume that the triangles are shape regular, although triangles can have curved edges as allowed by isoparametric finite element discretizations. The overall mesh can be nonuniform and unstructured. Along the linked boundary $\partial\Omega_0$ the triangulation must be *matching*; that is, if two boundaries are linked, then the restrictions of the triangulation $\mathcal{T}$ to those boundaries should be congruent.

Our basic data structure consists of lists of vertices, triangles, and edges, and is fairly standard in most respects. Let $(x_k, y_k)_{k=1}^N$ be the set of vertices of $\mathcal{T}$; then the vertex coordinates are stored in real arrays of size N. The only unusual aspect is that along linked boundaries, two vertices may share the same physical coordinates. This would not happen in our example of periodic boundary conditions, but would happen in the example of the linked internal interface Γ. If there are so-called *cross points*, more than two distinct vertices could have the same physical coordinates.

A triangle $t \in \mathcal{T}$ is characterized in terms of four integers; three global vertex numbers (essentially pointers to the arrays storing the (x_k, y_k)) and a fourth optional pointer (called a *label*), which could, for example, point at a database of material properties associated with the region containing t. This is realized (in FORTRAN) as a $4 \times NT$ integer array, where $NT \approx 2N$ is the number of triangles. This is again a fairly standard approach.

We now turn to the definition of edges, which is the most intricate part of the data structure. All boundary edges must be defined in our data structure, which is the standard case. Internal edges of the mesh are defined within the data structure *only* if they are curved; internal straight edges are implicitly defined through the triangulation. As a practical matter, we expect that all or almost all internal edges will be straight, and thus we anticipate that $NB = O(\sqrt{N})$, where NB is the number of edges defined in the data structure.

An edge b is characterized by five integers; two are global vertex numbers for the endpoints of the edge. The third characterizes the geometry of the edge. It is zero for straight edges. It is positive for curved edges, and points at a database which allows one to compute points along the curve as needed.

The fourth parameter is an edge *type*. We specify $type = 2$ for Dirichlet boundary edges, $type = 1$ for Neumann boundary edges, and $type = 0$ for internal edges. Edges lying on linked boundaries have a negative *type*. Suppose that b_i and b_j are a pair of corresponding edges along a linked boundary. The $type = -j$ for b_i and

type $= -i$ for b_j. Thus it is the *type* parameter (and only the *type* parameter) which establishes communication along linked boundaries. We note that it is this simple plan for communication which requires the triangulation to be matching along linked boundaries.

The fifth parameter is an optional *label* parameter, which could be used to point at a database of material properties associated with the given edge. This data structure is realized as a $5 \times NB$ integer array.

3. The Linear System

In this section we describe briefly the Lagrange multiplier formulation we use to construct the system of equations to be solved. To some extent this aspect of the problem is peripheral to the main thrust of this paper, but we include a short discussion for the sake of completeness. For simplicity, we consider the case of two subregions with one linked boundary, since the generalization to the case of many regions and linked boundaries is immediately apparent. Also, we assume the case of piecewise linear finite element approximation, since the standard degrees of freedom correspond to solution values at the vertices, and this will also simply our discussion.

The global system of equations is denoted $AU = F$, with A the global stiffness matrix, U the solution vector (including Lagrange multipliers as necessary) and F the right hand side. Let B and C denote the usual finite element stiffness matrices for Ω_1 and Ω_2, respectively. For convenience in presentation, in Ω_1 we assumed that the vertices along the linked boundary Γ are ordered last. In Ω_2, we do the same, with that additional requirement that the linked vertices are given the same relative order as in Ω_1. As a practical matter, vertices in both regions can be ordered arbitrarily and independently, but that generality would make our current discussion notationally cumbersome. With our imposed ordering, the matrices B and C have the simple block structure

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \qquad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

The blocks B_{22} and C_{22} are of the same order, which is equal to the number of vertices along the linked boundary.

To construct our linear system, we impose (pointwise) continuity at all vertices along the linked boundary, and introduce Lagrange multipliers for each pair of linked vertices. These Lagrange multipliers have a physical interpretation in terms of the normal component of the gradient at the linked vertices. Then the overall linear system $AU = F$ has the block structure

$$\begin{pmatrix} B_{11} & B_{12} & 0 & 0 & 0 \\ B_{21} & B_{22} & 0 & 0 & I \\ 0 & 0 & C_{11} & C_{12} & 0 \\ 0 & 0 & C_{21} & C_{22} & -I \\ 0 & I & 0 & -I & 0 \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \\ W_1 \\ W_2 \\ \Lambda \end{pmatrix} = \begin{pmatrix} G_1 \\ G_2 \\ H_1 \\ H_2 \\ 0 \end{pmatrix}, \tag{3}$$

where V and W are the solution vectors for Ω_1 and Ω_2, respectively, and Λ is the vector of Lagrange multipliers. This linear system has the standard form of a saddle point

problem. One could weakly impose continuity using an appropriate (one dimensional) finite element space defined on Γ; the main effect is that the identity matrices in (3) would be replaced by mass matrices.

Because of the simple structure we can easily eliminate the Lagrange multipliers and either V_2 or W_2 from (3) and have the reduced system

$$\begin{pmatrix} B_{11} & 0 & B_{12} \\ 0 & C_{11} & C_{12} \\ B_{21} & C_{21} & B_{22}+C_{22} \end{pmatrix} \begin{pmatrix} V_1 \\ W_1 \\ W_2 \end{pmatrix} = \begin{pmatrix} G_1 \\ H_1 \\ G_2+H_2 \end{pmatrix}. \tag{4}$$

Equation (4) is the linear system which would have resulted from the finite element discretization of the original problem, without the introduction of Γ.

4. A Posteriori Error Estimates

Of central importance to the adaptive procedures is the computation of a posteriori local error estimates [BR78, AO93, Ver95]. In this section we briefly summarize our a posterior error estimation procedure for the case of piecewise linear finite element approximation. In our procedure, we compute an approximate error $e_h \approx u - u_h$ as a discontinuous piecewise quadratic polynomial which is zero at the vertices of the mesh. The piecewise polynomial e_h is found by solving a small (of order 3) Neumann problem in each element of the mesh as described in [BW85, Ban96]. The unknowns of the problem associated with triangle t are the values of the quadratic polynomial approximating the error at the midpoints of the three edges of t. The data for the problem in triangle t is the residual of the partial differential equation in t, and the boundary data is based on the jump in normal direction of the solution across the edges of t. This is a completely standard scenario, with only a small modification for linked boundary edges, where the solution for the two elements sharing a pair of linked edges is used to compute the jump in normal derivative.

The error estimates are further processed for use in the adaptive algorithms. Suppose that in each element t the true solution u is well approximated by a quadratic polynomial. In particular, assume that $e_h \approx u_2 - u_1$ where u_2 is the local quadratic interpolant based on vertices and midpoints, and u_1 is the linear interpolant based on vertices. Let (x_m, y_m) denote the midpoint of the edge connecting vertices (x_i, y_i) and (x_j, y_j). Then the coefficient of the nodal quadratic basis function for midpoint (x_m, y_m) for the quadratic polynomial $u_2 - u_1$ is

$$\begin{aligned} u(x_m,y_m) - \frac{1}{2}u(x_i,y_i) &- \frac{1}{2}u(x_j,y_j) \\ &\approx -\frac{1}{8}\begin{bmatrix} x_i - x_j \\ y_i - y_j \end{bmatrix}^t \begin{bmatrix} u_{xx} & u_{xy} \\ u_{xy} & u_{yy} \end{bmatrix} \begin{bmatrix} x_i - x_j \\ y_i - y_j \end{bmatrix}. \end{aligned}$$

Our specific assumption for the true solution u is that the 2×2 matrix of second derivatives is (approximately) constant in each element t. Our a posteriori error estimates provide approximations $e_h(x_m, y_m)$ of the coefficient of the quadratic basis function for the midpoint (x_m, y_m). These values are related to the second derivatives

by the relation

$$e_h(x_m, y_m) = -\frac{1}{8} \begin{bmatrix} x_i - x_j \\ y_i - y_j \end{bmatrix}^t \begin{bmatrix} u_{xx} & u_{xy} \\ u_{xy} & u_{yy} \end{bmatrix} \begin{bmatrix} x_i - x_j \\ y_i - y_j \end{bmatrix}, \tag{5}$$

For each triangle t, we solve a 3×3 set of equations using (5), where m is in turn each of the three midpoints of element t, for the (assumed) constants $u_{xx}(t)$, $u_{xy}(t)$, and $u_{yy}(t)$.

Of course, the initial computation of a posteriori estimates yields directly error estimates for the original elements t in the finite element mesh, without the necessity of solving for the second derivatives using (5). However, it is the second derivatives for each element which are actually used to estimate errors in new elements created through the adaptive processes. For example, if an element is refined, one can use the second derivatives for the parent element in (5) to evaluate $e_h(x_m, y_m)$ for the edge midpoints of the refined elements. Once these midpoint values are known, one can directly compute error estimates for the refined elements.

5. Adaptive Algorithms

Our adaptive algorithms all operate on the data structures defined in Section 2.; in some sense they can be regarded as *operators* which map an input triangulation data structure to an output triangulation data structure. Because our multilevel solution techniques no longer require a hierarchy of meshes to be efficient, our adaptive procedures no longer maintain a history of the adaptive process (e.g. a refined element tree), and hence have become both less complicated and more flexible. Fundamentally, all our adaptive algorithms are hueristics with the goals of minimization of the local error, maintenance of the shape regularity of the elements, and efficiency in implementation.

Let t be a triangle with area a and side lengths h_1, h_2, and h_3. The quality of t, $q(t)$, is measured using the formula

$$q(t) = 4\sqrt{3}a/(h_1^2 + h_2^2 + h_3^2). \tag{6}$$

The function $q(t)$ is normalized to equal one for an equilateral triangle and to approach zero for triangles with small angles, and is used for controlling the shape regularity of elements in the mesh.

5.1. Refinement and Unrefinement.

We begin with a discussion of our adaptive refinement and unrefinement algorithms. Our goal here is to begin with a mesh with N vertices, and create a new mesh with (approximately) N' vertices. If $N' > N$, then adaptive refinement takes place; if $N < N'$, then unrefinement takes place.

A third possibility is to combine both procedures. Given $\bar{N} < N$, one can first unrefine the mesh to $\bar{N}$ vertices. The one can then refine to again obtain a mesh with N vertices. The output triangulation thus has approximately the same number of

vertices as the input triangulation, but the topology of the mesh and the distribution of mesh points can be quite different.

We begin with a brief summary of our refinement procedure, which is based on the longest edge bisection algorithm of Rivara [Riv84, Mit89]. All current elements are placed in a heap data structure according to the size of the error estimates. The element with largest error estimate is at the root of the heap. This element is selected for refinement, and is bisected along its longest edge. The neighbor element sharing the longest edge is also bisected along its longest edge. If the result is a triangulation (i.e. the longest edge for both elements is the same) the process stops. Otherwise, it is recursively applied to the longest edge neighbors of all refined elements. An example is shown is Figure 1. This process is known to have finite termination, typically in a very small number of steps. When the longest edge bisection process finally results in a triangulation, the new elements are created, and added to the triangulation data structures. New elements inherit the second derivative information from their parents, so error estimates can be computed and the heap updated. Using the updated heap, the refinement process continues, until a mesh with approximately N' vertices is created.

Local edge swapping and mesh smoothing algorithms [BSar] are employed to locally optimize the shape regularity of the of the final mesh in terms of the quality measure (6).

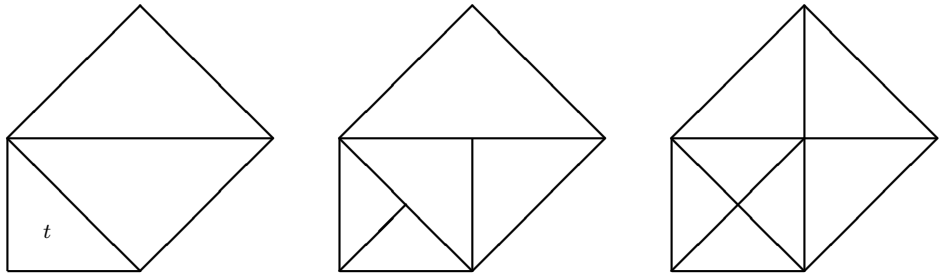

Figure 1 Element t is refined by the longest edge bisection method. The original mesh is on the left. The first step of bisection (middle) does not yield a compatible triangulation. However, the second step (right) does yield a triangulation.

In the case of unrefinement, the basic step consists of deleting vertices from the mesh, rather then directly unrefining elements. Each vertex v is associated with a region Ω_v, as illustrated in Figure 2. The error associated with vertex v is the largest error of any element contained in Ω_v. With these definitions, the unrefinement procedure is quite analogous to the refinement procedure described above. All the vertices are placed in a heap based on their errors, with the vertex of smallest error at the root. Certain vertices, which are critical to the geometric integrity of the domain as a whole (e.g, corner vertices on the boundary of the region) are given artificially large errors. Vertices of low degree have their errors reduced a bit to favor their elimination.

In the elimination step, the root vertex of the heap is eliminated from the mesh.

The region Ω_v associated with this mesh is then triangulated using the boundary vertices, as shown in Figure 2. The newly created elements inherit second derivative information from the original elements in Ω_v (through suitable averaging), and error estimates are computed for the new elements. The vertices lying on $\partial\Omega_v$ have their errors updated as required, and the heap is updated. The process is continued until a mesh with N' vertices is achieved. As in the case of refinement, local edge swapping and mesh smoothing are used to improve the shape regularity of the final mesh in terms of the quality measure (6).

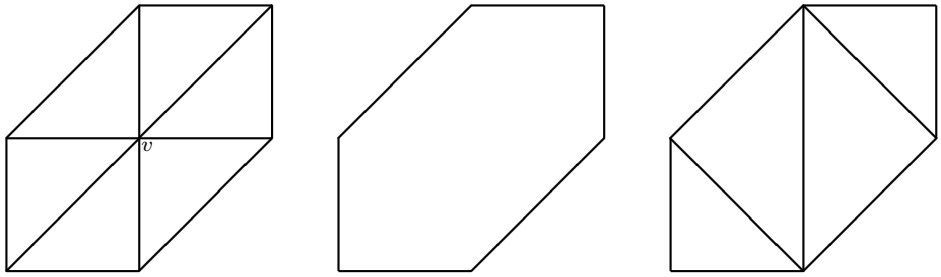

Figure 2 On the left is the subregion Ω_v, associated with vertex v. To unrefine the mesh, vertex v and all its incident edges are removed from the triangulation (middle). The region Ω_v is then triangulated using the boundary vertices (right).

5.2. Mesh Smoothing.

Our adaptive mesh smoothing algorithm does no refinement or unrefinement of the mesh, but rather adjusts the (x, y) coordinates of the mesh points for a fixed topology in an attempt to optimize the mesh. The procedure consists of a Gauss-Seidel like iteration on the vertices in the mesh, where the location of each vertex is locally optimized with all other vertices held fixed. The procedure is described in detail in [BSar]. Typically a given vertex v is allowed to move withing the region Ω_v as shown in Figure 2. Not all vertices in the mesh are allowed to move. Some boundary and interface vertices must remain fixed to preserve the definition of the region. These vertices are called *corners*. Vertices on the boundary or on interfaces that are not designated corners are allowed to move only along the boundary or interface. The remaining vertices, called *interior* vertices, are allowed to move freely within Ω_v. As in our refinement algorithms, some local mesh smoothing based on (6) is used to locally optimize the shape regularity of the mesh.

For each vertex $v = (x, y)$ in the mesh, we solve the minimization problem

$$\min_{x,y} \|\nabla e_h\|^2_{\mathcal{L}^2(\Omega_v)} \tag{7}$$

of order two by a damped Newton's method. As noted above, we assume the second derivatives are constant in each element t having v as a vertex, leading to an

overall piecewise constant approximation of the second derivatives on Ω_v. All other dependencies on $v = (x, y)$ are taken into account by Newton's method. Boundary and interface vertices have an additional constraint equation, so an appropriately constrained version of problem (7) is solved for those vertices. Besides its usual task of insuring sufficient decrease, the damping strategy for Newton's method is also used to insure that the point (x, y) remains well within Ω_v, so that all triangles are always well defined. It is interesting to note that the function $\|\nabla e_h\|_{\mathcal{L}^2(\Omega_v)}$ contains a natural barrier function that becomes infinite as (x, y) approaches $\partial\Omega_v$.

6. Numerical Example

In this section we present a simple numerical example to illustrate the minimal effect of linked boundaries on our adaptive algorithms. We solve the equation $-\Delta u = 1$ with homogeneous boundary conditions in the region shown in Figure 3 (the profiles are NACA0012). The initial mesh has 644 elements and 375 vertices. We refined this mesh to one with $N = 24000$ using three refinement steps ($N = 1500$, $N = 6000$ and $N = 24000$). A detail of the final mesh is shown in Figure 4 (the complete mesh is too dense to be resolved in such a small picture). For comparison we solve the same problem except the domain is now partitioned into two subdomains as illustrated in Figure 3. The initial mesh still has 644 elements but now has 398 vertices due to duplicate vertices along the linked boundary. We again refine this mesh to one with $N = 24000$ vertices; a detail is shown in Figure 4. From this we see that while the meshes for the two cases are not identical, they are quite similar.

In Figure 5, we present a log-log plot of the a posteriori error estimates for the $\mathcal{H}^1$ norm as a function of N. Again we see quite similar behavior. The final relative error estimate is $3.01e(-2)$ for the first case and $3.01e(-2)$ for the partitioned mesh. For the $\mathcal{L}^2$ norm (not illustrated) the final relative error estimates were $2.27e(-4)$ and $2.25e(-4)$, respectively.

REFERENCES

[AO93] Ainsworth M. and Oden J. T. (1993) A unified approach to a posteriori error estimation using element residual methods. *Numerische Mathematik* 65: 23–50.

[Ban94] Bank R. E. (1994) *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 7.0.* Frontiers in Applied Mathematics, Vol. 15, SIAM, Philadelphia.

[Ban96] Bank R. E. (1996) Hierarchical bases and the finite element method. In *Acta Numerica 1996 (A. Iserles, ed.)*, pages 1–43. Cambridge University Press.

[BR78] Babuška I. and Rheinboldt W. C. (1978) Error estimates for adaptive finite element computations. *SIAM J. Numer. Anal.* 15: 736–754.

[BSar] Bank R. E. and Smith R. K. (to appear) Mesh smoothing using a posteriori error estimates. *SIAM J. Numerical Analysis* .

[BW85] Bank R. E. and Weiser A. (April 1985) Some a posteriori error estimators for partial differential equations. *Math. Comp.* 44: 283–301.

[BX94] Bank R. E. and Xu J. (1994) The hierarchical basis multigrid method and incomplete LU decomposition. In *Seventh International Symposium on Domain Decomposition Methods for Partial Differential Equations (D. Keyes and J. Xu, eds.)*, pages 163–173.

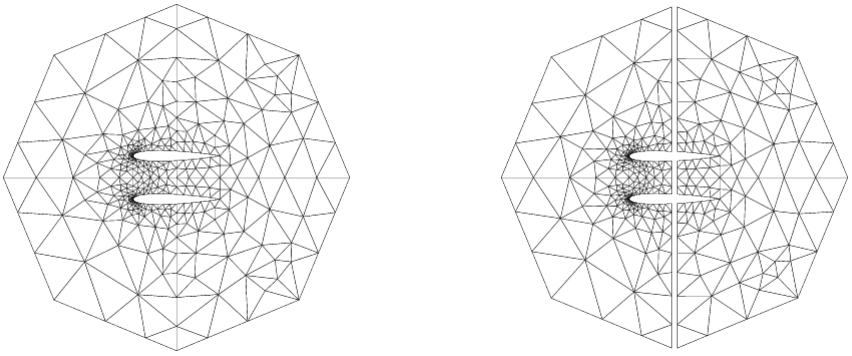

Figure 3 The initial grids.

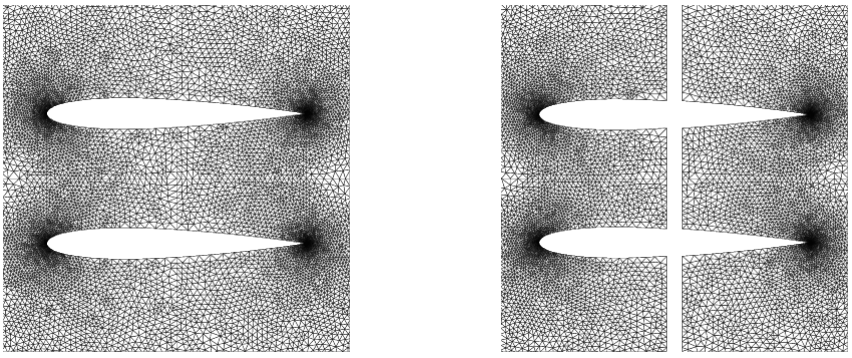

Figure 4 Detail of the final grids.

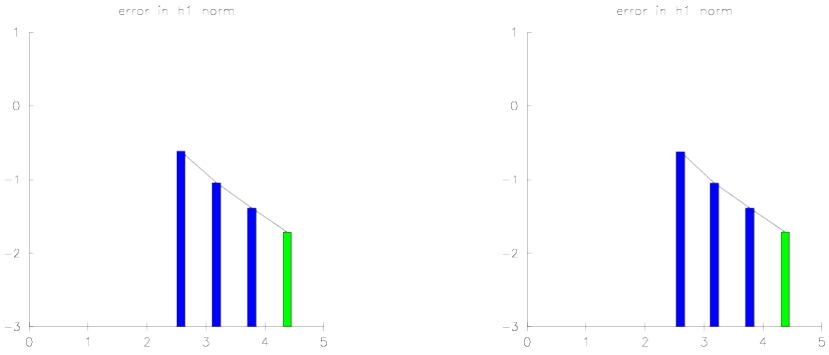

Figure 5 Convergence histories in the $\mathcal{H}^1$ norm.

AMS, Providence, Rhode Island.
[BX96] Bank R. E. and Xu J. (1996) An algorithm for coarsening unstructured meshes. *Numerische Mathematik* 73: 1–36.
[Mit89] Mitchell W. F. (1989) A comparison of adaptive refinement techniques for elliptic problems. *ACM Trans. Math. Soft.* 15: 326–347.
[Riv84] Rivara M. C. (1984) Mesh refinement processes based on the generalized bisection of simplices. *SIAM J. Numer. Anal.* 21: 604–613.
[Ver95] Verfürth R. (1995) *A Posteriori Error Estimation and Adaptive Mesh Refinement Techniques.* Teubner Skripten zur Numerik, B. G. Teubner, Stuttgart.