ALGORITHMS AND DATA STRUCTURES FOR ADAPTIVE MULTIGRID ELLIPTIC SOLVERS

John Van Rosendale
Institute for Computer Applications in Science and Engineering

ABSTRACT

With the advent of multigrid iteration, the large linear systems arising in numerical treatment of elliptic boundary value problems can be solved quickly and reliably. This frees the researcher to focus on the other issues involved in numerical solution of elliptic problems; adaptive refinement, error estimation and control, and grid generation. Progress is being made on each of these issues and the technology now seems almost at hand to put together general purpose elliptic software having reliability and efficiency comparable to that of library software for ordinary differential equations.

This paper looks at the components required in such general elliptic solvers and suggests new approaches to some of the issues involved. One of these issues is adaptive refinement and the complicated data structures required to support it. These data structures must be carefully tuned, especially in three dimensions where the time and storage requirement of algorithms are crucial. Another major issue is grid generation. The options available seem to be curvilinear fitted grids, constructed on iteractive graphics systems, and unfitted Cartesian grids, which can be constructed automatically. On several grounds, including storage requirements, the second option seems preferrable for the well behaved scalar elliptic problems considered here. A variety of techniques for treatment of boundary conditons on such grids have been described previously and are reviewed here. A new approach, which may overcome some of the difficulties encountered with previous approaches, is also presented.

Research reported in this paper was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-17070 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

1. Introduction

Library software for ordinary differential equations has been around for many years and is now highly refined. The overwhelming majority of ordinary differential equation (ODE) problems encountered are readily handled by standard library software, such as the Episode program of Alan Hindmarsh. The situation for partial differential equation (PDE) problems is quite different. For the majority of PDE problems encountered, no library software is available, programs must be constructed almost entirely from scratch.

There seem to be a number of reasons for this dicotomty. The large sparse linear systems arising in most PDE discretizations are difficult to solve. Applying boundary conditions in PDE problems is a more complex and central problem than the analogous problem for ODEs. But the principal reason why robust general purpose library software is not available for the bulk of simple commonly occurring PDE problems seems to be the difficulty in representing complicated two and three dimensional domains and generating grids on them.

This is especially true now with the advent of fast multigrid solvers for the large sparse linear systems arising. Similarly, adaptive refinement strategies are now quite well understood. Further research is needed on error estimates for adaptive refinement, and on the complex data structures involved in adaptive multigrid algorithms, but these are not the main issue. Grid generation seems to be the bottle neck.

This paper looks at the components which would be required for the construction of flexible and reliable solvers for simple elliptic problems in geometrically complex three dimensional domains. There are now a number of research efforts aimed at creating analogous two dimensional software. We mention the Ellpack project, Houstis and Rice [1980], the Fears project, Zave

and Rheinboldt [1979], and especially the adaptive multigrid finite element code, PLTMG, Bank and Sherman [1978].

These projects provide a model for the development of similar three dimensional software. But there are a number of differences complicating the development of analogous three dimensional software. The extra dimension greatly magnifies the cost of any inefficiencies, and as is well known, the grid generation problem is far more acute in three dimensions.

The plan of this paper is as follows. Section 2 treats the knotty problem of grid generation. Several alternative approaches are examined, but it is argued that unfitted grids can provide the greatest efficiency and simplest user interfaces for well behaved elliptic boundary value problems. Section 3 treats data structures that support efficient adaptive refinement algorithms. Finally Section 4 looks at numerical issues. It examines multigrid algorithms tuned to the data structures described in Section 3. Error estimates for adaptive refinement are also briefly discussed.

2. Geometry Modelling and Grid Generation

As argued, grid generation is at the heart of the problem of constructing reliable ellptic software for general three dimensional domains. This is so, primarily because the other issues involved in solving elliptic problems are now fairly well resolved. Fortunately, the problem of grid generation is less severe for elliptic problems than for hyperbolic or parabolic problems, for a variety of reasons. First, though there can be sharp transitions or boundary layers in elliptic problems, these are usually much less severe than those in hyperbolic problems. Second, any sharp fronts occurring in elliptic problems are stationary, so simple adaptive strategies can resolve them well. Finally,

finite element discretications work well for elliptic problems, even on grids that are quite badly distorted. By contrast, no comparably flexible discretizations exist for hyperbolic problems, which can be extremely difficult to solve accurately even on Cartesian grids.

There are a number of potential ways of constructing grids for general three dimensional domains. Several possibilities are examined in the next subsection. Following that, we focus on unfitted grids, which may be the most viable alternative for general three dimensional elliptic software.

2.1 Alternate Approaches to Grid Generation

The most widely used grid generation technique for curved domain is to construct a mapping from a uniform rectangular grid onto the given domain. This approach has the advantage of having very simple data structures, and is more flexible than one might first think. However, there are several disadvantages. First, construction of the grid mapping can be a complicated process requiring sophisticated interactive graphics. Second, grids generated in this way are often highly distorted. Even with elliptic problems, severe grid distortions are undesirable. Finally, this simple approach is not universally applicable; some geometries cannot be treated in this way.

There are several alternative approaches one might consider. The first is the use of simplicial or tetrahedral grids, Figure 2.1. Any polyhedron can be decomposed into a union of tetrahedrons, so this approach is completely general. It may also be possible to generate such grids automatically, with no user intervention, though this possibility seems not to have been researched.

There are two main disadvantages with this simplicial approach. First, the data structures are complex and costly. With each grid point it is necessary to keep a list of all neighboring grid points, together with the corresponding finite element matrix elements. This might entail storing 50 or more pointers and coefficients per mesh point. The storage requirements can already be a problem with the two dimensional adaptive finite element code PLTMG, which uses triangular grids.

The second problem with the simplicial approach is that there is no natural way to perform refinement or construct multigrid levels in this approach. The two dimensional code PLTMG refines triangles by the "regular" refinement process shown in Figure 2.2. There is no direct analog of this process for tetrahedrons; every decomposition of a regular tetrahedron into subtetrahedrons generates irregular subtetrahedrons with sharper angles than the original tetrahedron.

The second general approach which may be considered is the use of block structured grids. See, for example, Rubbert and Lee [1982], Figure 2.3 shows such a grid. In this approach one decomposes the domian into a union of cells or regions, each of which is a distorted code. A rectangular grid may be imposed on each mapped rectangular cell by standard algebraic grid generation techniques.

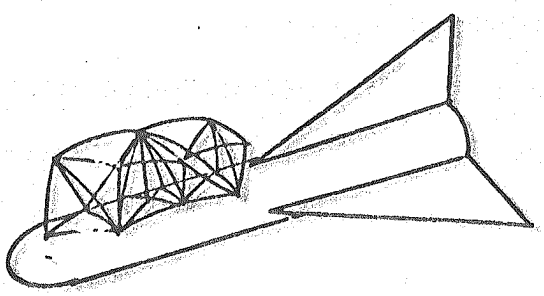


Figure 2.1. Simplicial Grid

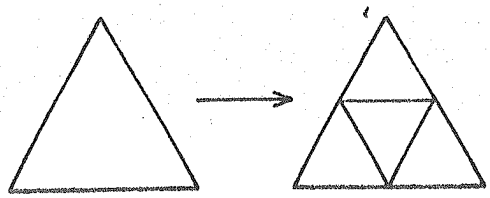


Figure 2.2. Regular Refinement of a Triangle

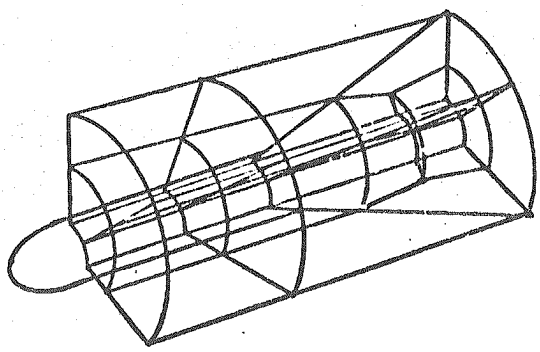


Figure 2.3. Block Structured Grid

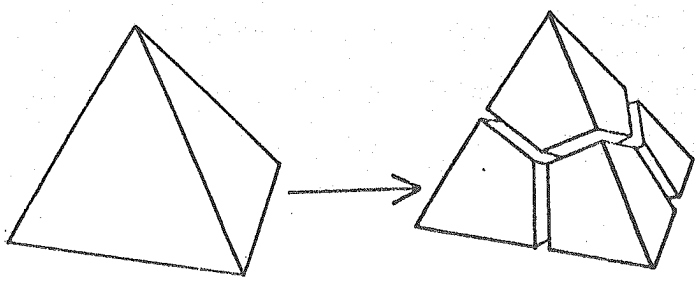


Figure 2.4. Decomposition of a Tetrahedron

This approach is extremely flexible and has a number of advantages. Figure 2.4 shows that a tetrahedron can be decomposed into four distorted cubes. Thus any polyhedral domain can be viewed as a union of mapped rectangular cells.

Another advantage is that the data structures are far less expensive than those required for simplicial grids. Data structure information need be stored only for each cell, which may contain thousands of mesh points, rather than separately for every mesh point. It is also quite easy to perform adaptive refinement with this grid structure and to construct multigrid grid levels.

However, there seem to be three problems with this approach. First, the construction of such grids is complex, requiring sophisticated interactive graphics. Second, it is extremely hard to generate grids of this type which are not severely distorted. In complicated regions, strong grid distortions which severely limit solution accuracy are almost inevitable. Finally, though less storage is needed than for simplicial grids, the storage requirements are still substantial. Typically 20 or 30 variables per mesh point are required with second order finite elements, and somewhat less for finite difference formulas.

Figure 2.5 shows an unfitted grid, which is the third general alternative we wish to examine. There are two major advantages to this type of grid. First, grid generation can be completely automatic. No complex interactive graphics are needed. Second, the use of a Cartesian grid reduces storage requirements greatly. This is especially true for constant coefficient problems, but holds for variable coefficient problems as well.

The problems with this type of grid are well known. The major problem is the difficulty in imposing boundary conditions. The next two subsections are

devoted to this issue. It is also somewhat harder to perform adaptive refinement in this setting, because of the boundary treatments, although there seem to be no serious obstacles here.

Unfitted grids have inherent disadvantages in resolving sharp boundary layers. But for the well behaved elliptic problems we are considering, their advantages seem to outweigh their disadvantages. The cost per mesh point is significantly lower than for curved grids, the user interface is comparatively trivial, and for problems with smooth solutions high order accuracy can be achieved.

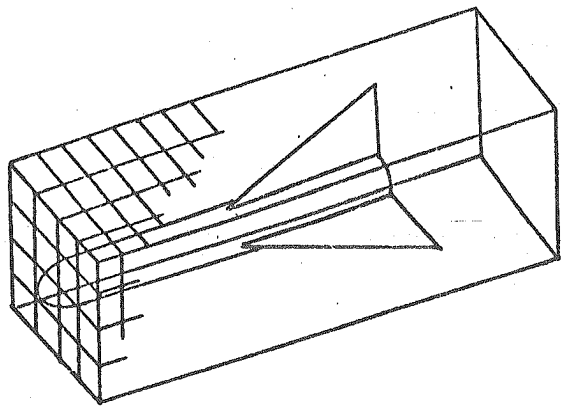


Figure 2.5. Unfitted Cartesian Grid

2.2 Treatment of Boundary Conditions on Unfitted Grids

Though unfitted grids have significant advantages, as we have argued, there are major problems with them as well. The principal problems are that applying boundary conditions on these grids is quite complicated, and it is more difficult to approximate boundary conditions to high order on these grids. A number of approaches to imposing boundary conditions on such grids have been suggested. Three of these approaches are reviewed in this section. A new approach, which seems to offer a number of advantages, is

described in the next section.

The oldest method of applying boundary conditions on unfitted grids is to use special difference or interpolation formulas at the boundary. For a good overview of this approach see Forsythe and Wasow [1960]. This approach is simple and effective for Dirichlet boundary conditions. Suppose, for example, we wish to impose the Dirichlet condition

$$u = q \quad \text{on} \quad \partial\Omega$$

with the two dimensional geometry shown in Figure 2.6. One approach used is to compute the solution value at mesh points near the boundry, such as point 0 shown, by an interpolation formula rather than by the finite difference on finite element formula used at other interior points. See Collatz [1955]. A related approach is to apply a modified difference formula at mesh points adjacent to the boundary. For example, at point 0 shown, the Laplacian could be approximated by the five point star difference formula for variable mesh spacing:

$$\Delta u_0 \tilde{=} \frac{2}{h_1 + h_3}\left[\frac{u_1 - u_0}{h_1} + \frac{u_3 - u_0}{h_3}\right] + \frac{2}{h_2 + h_4}\left[\frac{u_2 - u_0}{h_2} + \frac{u_4 - u_0}{h_4}\right]$$

Here u_1 and u_2 are known Dirichlet boundary values.

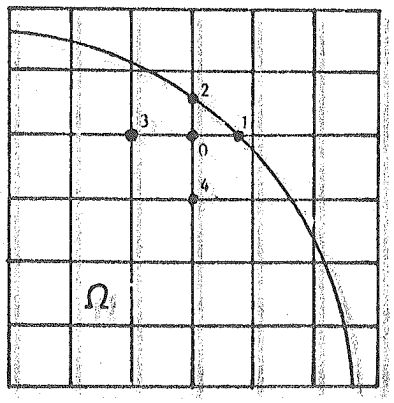


Figure 2.6. Boundary Treatment on Two Dimensional Unfitted Grid

Related but far more complicated approaches can be used for Neumann or mixed boundary conditions. See Forsythe and Wasow [1960] for details. Though this simple approach works well for Dirichlet problems, it is quite complicated and problematical for mixed or Neumann conditions. Alternatives are clearly needed.

Two elegant alternative approaches have been derived in the finite element context. The simplest of these is the penalty method. See Strang and Fix [1973], page 132, or Babusha [1971] for details. To see how this approach works consider the model problem

$$\Delta u = f \quad \text{on} \quad \Omega,$$

(2.2.1)

$$u = g \quad \text{on} \quad \partial\Omega.$$

In the penalty method, one seeks the function u^h in a finite element space M^h which minimizes the functional

$$I_\lambda(u) = \iint_\Omega \left(u_x^2+u_y^2-2fu\right) + \lambda \int_{\partial\Omega} (u-g)^2. \tag{2.2.2}$$

where $\lambda > 0$ is a penalty parameter. Taking the first variation, the minimizing discrete solution must satisfy

$$\iint_\Omega \left(u_x v_x + u_y v_y - 2wfv\right) + \lambda \int_{\partial\Omega} (u-g)v = 0$$

for all test function v in the finite element space M^h. Clearly, if one lets $\lambda \to \infty$, the discrete solution u^h is forced to closely satisfy the Dirichlet boundary condition.

There seem to be two basic problems with this method. From a practical point of view it is quite complex, requiring element integrals both in the domain interior and along the boundary. Since the interior integrals must be done only over Ω, quadrature on elements meeting the boundary is quite complicated to perform.

There is also a mathematical difficutly with this method. The true solution of the model problem (2.2.1) does not minimize the functional (2.2.2) for any finite λ. Thus in addition to the usual truncation error, one now has an error term due to the penalty parameter. Because of this, the optimal finite element rate of convergence is not usually attained by this method.

This mathematical difficutly is overcome by a more subtle finite element treatment of unfitted boundaries using the idea of Lagrange multipliers. See Babuska [1973], Babuska and Aziz [1972] or Strang and Fix [1973]. Applying this method to the model problem (2.2.2), one seeks the stationary point of the indefinite functional

$$F(u,\Lambda) = \iint_{\Omega}\left(u_x^2+u_y^2-2fu\right) - 2\int_{\partial\Omega}\Lambda(u-g).$$

The Lagrange multiplier here runs over all admissible functions on the boundary.

To apply this method, one must construct two finite element spaces, the usual spline space M^h on the domain Ω, and a spline space B^h of functions defined on the bondary. Moreover, the boundary spline space B^h must consist of "coarser" elements than those of the interior space M^h.

This Lagrange multiplier method achieves the optimal finite element order of convergence, but is extremely complex to program. In two dimensions the programming complexities are manageable, but for general three dimensional

geometries the complexity of the required programming seems prohibitive. For three dimensional problems, one must first construct a grid on the boundary surface. Then one must compute both the usual interior quadratures and the quadratures of surface elements against interior elements. For the latter, one must perform quadratures on the intersection of each curvilinear surface element with every interior element it meets. This is clearly a highly complex computational geometry task.

2.3 Least Squares Boundary Conditions for Unfitted Grids

A number of other approaches to imposing boundary conditions on unfitted grids have been described previously. For example, approaches specific to a given physical problem, such as transonic flow, have been described. However, the three approaches reviewed in the last section seem to be the most fully developed, and are fairly representative.

Though there are some numerical difficulties with the approaches described, any of these approaches could, in principle, yield any order of accuracy desired. The real problem with these methods is that the programming required is very complex and the resulting code would be quite inefficient. Even the simple finite difference approach described requires geometric information which is difficult to extract. One must compute the intersections of each mesh line with the boundary surface. This ordinarily requires the use of a Newton-like method to solve the nonlinear equations involved.

A new finite element approach, designed to overcome these geometric difficulties, is proposed here. The idea of the method is first to extend the computational domain beyond the true domain. This is done by including in the computational domain all mesh points which are adjacent to the boundary but

outside the domain. These points are shown in Figure 2.7. One then applies the usual finite element formulas at all interior points. Simultaneously, one asks that the boundary conditions be satisfied in the least squares sense at all exterior points.

To make these ideas precise, consider the model Dirichlet problem (2.2.1). Let M^h be the spline space of bilinear finite elements on the domain shown in Figure 2.7. Let M_0^h be the subspace of M^h consisting of functions vanishing at the darkened boundary points in Figure 2.7. Finally let Ω be the true domain and let $\overline{\Omega}$ be the extended computational domain shown in this figure. Then the method proposed is to compute the discrete solution $u^h \varepsilon M^h$ minimizing

$$\int_{\partial\Omega} (u^h - g)^2 \tag{2.3.1}$$

subject to

$$\iint_{\overline{\Omega}} (u^h{}_x v^h_x + u^h_y v^h_y - 2f u^h) = 0 \quad \text{for all} \quad v^h \varepsilon M_0^h. \tag{2.3.2}$$

Since the solution u^h is sought in the space M^h, while the test function v^h range over the subspace M_0^h, equation (2.3.2) constitutes an underdetermined linear system. Minimization of the boundary integral (2.3.1) determines a unique solution.

Though theoretical error bounds for this approach are not available, this method offers a number of practical advantages, and appears to offer the potential of high order convergence. To see the practical advantages, note first that equation (2.3.2) involves only integrals on the extended domain $\overline{\Omega}$. Thus element integrals on partially cutoff elements near the boundary are

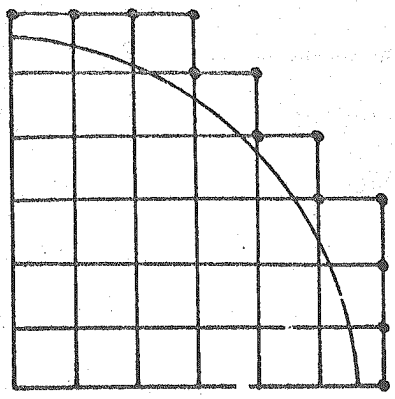


Figure 2.7. Least Squares Boundary Treatment

avoided. This is clearly a violation of the finite element philosophy, but the alternative, computing integrals on partially cutoff elements, is impractical.

The second practical advantage is that the boundary integral (2.3.1) here is quite easy to approximate and no boundary grid generation is required. Consider a two dimensional model problem of the form (2.2.1) and suppose the boundary is given parametrically as

$$\partial\Omega = \{\gamma(t), t\in[0,1]\}$$

for some mapping

$$\gamma\colon R^1 \to R^2.$$

Then we may take as quadrature points the point set

$$\{\gamma(\tfrac{i}{N}),\ i=1,2,\cdots N\}.$$

If we are using a Cartesian grid, as in Figure 2.7, it is easy to decide which element each quadrature point falls in. Thus the computation of the element

intergrals needed for (2.3.1) is simple and efficient. Exactly the same conclusions hold for the analogous method in three dimensions.

Notice that only low order quadrature can be performed in this way, since the quadrature is done without regard to the boundaries between elements. The order of quadrature could be increased by shifting to elements with higher inter-element continuity, such as C^1 or C^2 cubic elements, but this is probably unnecesary. The accuracy of the quadrature may be of relatively little importance, and in any case, large numbers of quadrature points can be used since the cost per quadrature point is minimal.

The last practical advantage of this appproach is that the least squares minimization required in (2.3.1) and (2.3.2) is easy to perform when the interior equation (2.3.2) is solved iteratively. After each relaxation sweep on the interior equations, one performs one or more relaxation sweeps on the boundary equations (2.3.1). Numerical experiments designed to assess the speed of different boundary iterations, and the impact of different quadrature approximations are currently under way.

3. Data Structure for Adaptive Block Structured Grids

In this section, we look at data structures equally applicable to the block structured fitted grids discussed in subsection 2.1, or to block structured adaptive unfitted grids. The fundamental premise here is that while adaptive refinement is important, keeping track of data structure information for every mesh point or finite element is far too expensive. Instead we track cells, which are small rectangular grids, typically about 10 by 10 by 10. Such cells can be viewed in the finite element context as large macro-elements. Such a cell is shown in Figure 3.1.

The main advantage in the use of adaptive grids based on such blocks or cells is that the required data structures are relatively inexpensive. While each cell can require 1000 to 30000 real variables, only about 100 data structure variables are required per cell. There can also be advantages to the use of this type of grid from the point of view of vector or parallel processing, Gannon and Van Rosendale [1983].

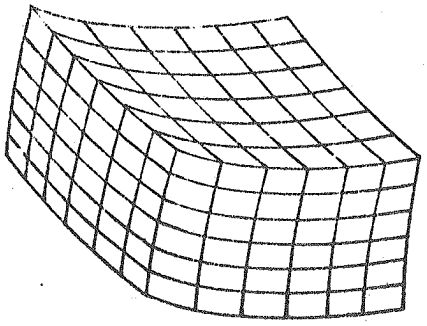


Figure 3.1. Macro-element Cell

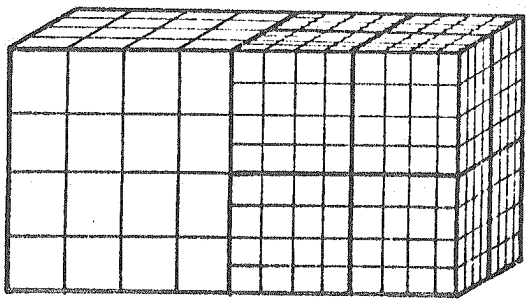


Figure 3.2. Adjacent Cells with Different Mesh Size

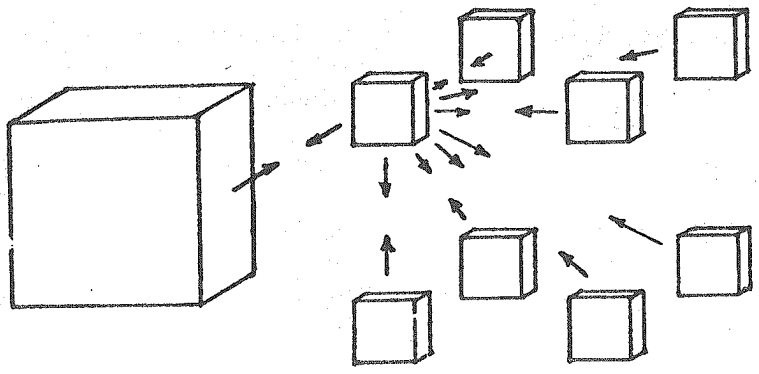


Figure 3.3. Explicit Pointer Scheme

Adaptive grids can be constructed with such cells, since cells of differenct mesh size can be abutted, as in Figure 3.2. Using finite elements, there are no mathematical difficulties here. Finite element theory requires only that the spline spaces constructed maintain C^0 continuity. It also turns out that multigrid iteration is quite natural in this context. The remainder of this section focuses on the data structure issues involved. Section 4 looks at a multigrid adaptive algorithm which can be used with this data structure.

3.1 Adjacency Information

With the block structured grids considered here the hardest problem is keeping track of which cells are in contact. Dynamic allocation of cells for adaptive refinement is quite simple, but generating and maintaining adjacency information is not.

Two basic schemes for tracking adjacency information have been considered and programmed. For convenience, we will refer to these as the implicit and explicit schemes. In the explicit scheme each cell maintains pointers to all cells it abutts. Thus for the geometry in Figure 3.2 we would have the pointer structure shown in Figure 3.3.

Though this explicit data structure is natural, it tends to be quite complex to use. In the multigrid algorithm described in Section 4, we permit a cell to be on more than one multigrid grid level. This reduces storage requirements, but complicates this explicit scheme. A cell will have one set of pointers for its neighbors on one grid level and another for its neighbors on another level.

There is also a problem with this scheme if the grids are not logically Cartesian. For example, in Figure 2.4 where the decomposition of a tetrahedron into cells is illustrated, only four cells meet at the interior vertex instead of the usual eight. In such cases, the adjacency pointers must also carry orientation information.

An alternative scheme, which overcomes many of these difficulties,is available. In this scheme, we associate with each cell 27 vertices. These vertices are the 8 corners, the midpoints of the edges and faces, and cell center. If these vertices are globally numbered then all adjacency information is implicit. For example, if one cell has an edge containing vertices 23 and 96 and another cell has an edge containing these same two vertices then these two cells are in contact along that edge. The use of this data structure for the geometry of Figures 3.2 and 3.3 is illustrated in Figure 3.4.

This implicit scheme overcomes both of the problems cited regarding the explicit scheme. To work, it requires that adjacent cells in the same grid differ by at most a factor of two in mesh size. More precisely, if binary refinement is performed, two cells in the same multigrid grid level that share an edge or face must differ by at most one in their level of refinement.

This implicit scheme is surprisingly easy to program. One needs to be able to perform dynamic allocation of cells, allocation of vertex numbers, and one needs a procedure:

procedure pair_find (var cells: cell_list; v_1,v_2: integer);

Given the integer labels of two vertices, this procedure produces a list of all cells sharing those vertices. This procudure can be performed quite efficiently if associated with each labelled vertex we store a list of all

cells that vertex belongs to. Then procedure pair_find only needs to merge the lists of the two vertices it is passed.

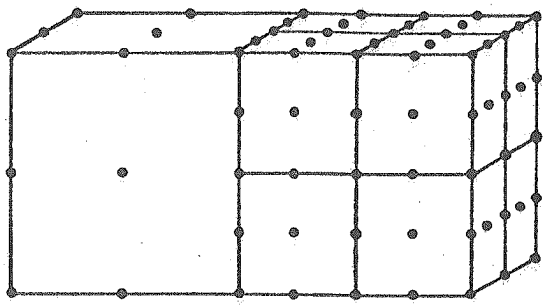


Figure 3.4. Implicit Vertex Labelling Scheme

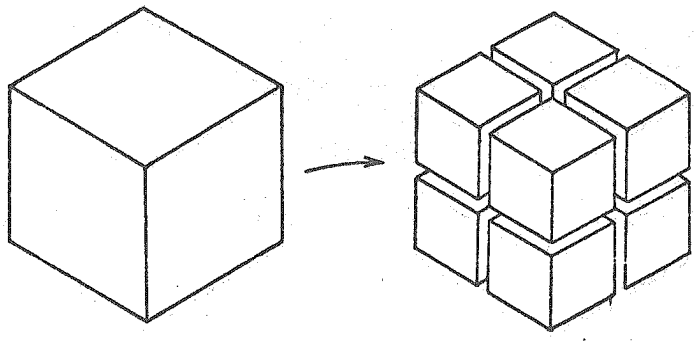


Figure 3.5. Subdivision of a Cell

3.2 Adaptive Refinement

The data structure using labelled vertices just described is designed to keep track of cell adjacencies during adaptive refinement. It is also designed to support multigrid interation although additonal information must be stored in order to permit this. In this subsection the data manipulations needed to perform adaptive refinement are considered, and the additional data structure information necessary for multigrid iteration is described.

To perform adaptive refinement, whenever the truncation error on a cell is too large we refine it, using the subdivision shown in Figure 3.5. This refinement operation creates 8 new cells, which we call the "children" of the

original "parent" cell. The natural data structure for tracking this refinement process is the oct-tree structure. In this data structure each parent cell contains pointers to its eight children, and each child contains a back pointer to its parent. The multigrid projection and injection operations, carrying data between coarser and finer grid levels, follow these pointers.

Only one other type of data structure information needs to be stored. It is necessary to know which grids belong to each multigrid grid level. We wish here to allow cells to belong to more than one multigrid grid level. Thus there are two reasonable ways of keeping trach of which cells belong to each grid level:

1. For each grid level we can maintain a list of all cells constituting that grid level.
2. For each cell we can maintain two integers, min_lev and max_lev, which indicate the range of grid levels this cell belongs to.

Both of these schemes work well, so the choice between them is unimportant.

We now look in detail at the operation of adaptive refinement, which is the most complex and important data structure operation to be performed. The basic steps required are:

1. Estimate the truncation error for each cell in the fine grid.
2. Flag all fine grid cells with excessive truncation error.
3. Flag neighboring cells of flagged cells, where necessary, so that after refinement adjacent cells will never differ by more than one in their level of refinement.
4. Refine all flagged cells.
5. Adjust the data structure so that fine grid cells which were not refined become part of the new fine grid.

Step 4 here, refining flagged cells, is clearly the most difficult. The rest of this subsection focuses on this step in detail.

In order to perform step 4 we require a new basic procedure. The procedure is:

procedure find_mid (var v_0: integer; v_1, v_2: integer);

Given two vertices, v_1 and v_2, this procedure searches through the data structure to see if any cell has a label for the midpoint of the line segment joining these two vertices. If v_1 and v_2 are the ends of an edge of a cell, or are diagonally opposite corners of a face, find_max returns the label of the center of that edge or face. Given procedure pair_find already discussed, procedure find_mid is quite easy to program.

Using procedure find_mid, the refinement operation is straight forward. Associated with each cell we have 27 labelled vertices, as shown in Figure 3.4. Thus when the 8 children are dynamically allocated, the labels for their corner vertices are known. The remaining 19 (= 27-8) vertices of each child may or maynot have labels already allocated in the data structure. This is where procedure find_mid is used.

On each of the eight childern we first allocate a label for the center vertex, since that vertex cannot occur anywhere else in the data structure. Secondly, on each child we look at each edge and face in turn. For each edge, we call procedure find_mid using the labelled corner vertices at its ends, to find a label for the edge midpoint. Similarly, for each face we call procedure find_mid using diagonally opposite corner vertices on that face to find a label for the face center. Whenever find_mid cannot locate a label for an edge midpoint or face center, none exists and one must be allocated.

4. Adaptive Multigrid Solution Algorithm

The data structures described in the last section can be used to solve elliptic boundary value problems adaptively using any iterative algorithm. However, the arguments in favor of using multigrid iteration for regular meshes are compounded for the locally refined grids created by adaptive refinement. For elliptic problems, the condition number of the finite element stiffness matrix K ordinarily satisfies

$$\kappa(k) \tilde{=} h_{min}^{-2}.$$

Thus, on the locally refined grids generated by adaptive refinement, the stiffness matrix can be quite badly conditioned even when the total number of unknowns is not particularly large.

All iterative methods, except multigrid, seems quite sensitive to the condition number of the linear system being solved. Even preconditioned conjugate gradient iteration has not performed well on locally refined grids. However, experimental evidence suggests that multigrid algorithms perform almost as well on locally refinds grids as they do on uniform grids. Thus multigrid iteration is clearly the iterative algorithm of choice for adaptive elliptic solvers.

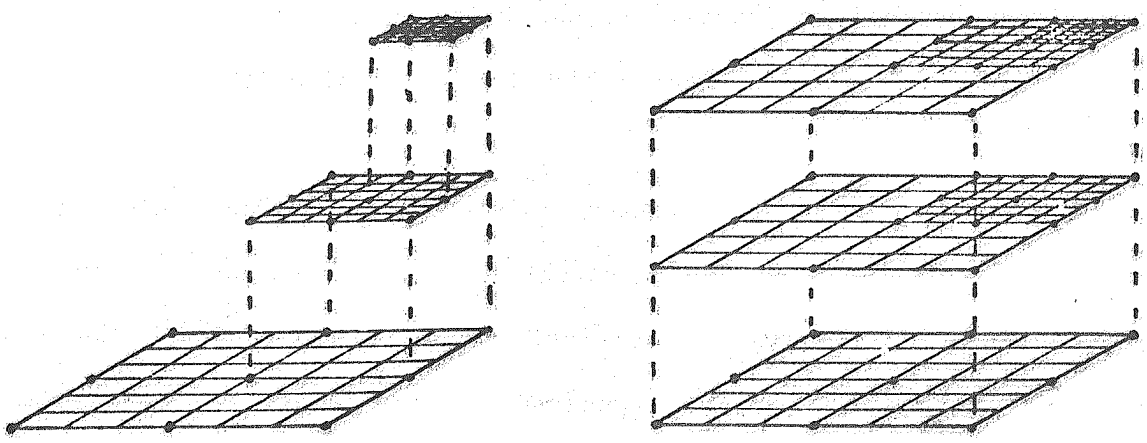
This section looks at adaptive multigrid algorithms which can be effectively imbedded in the data structures just described. The first subsection considers the type of multigrid algorithm to employ. The second looks at the data movements required in residual or interpolation calculations. The last subsection considers briefly error bounds required for adaptive refinement.

4.1 Multigrid Solution Algorithm

A variety of multigrid algorithms have been applied to locally refined finite element grids, Bank and Sherman [1978], Gannon and Van Rosendale [1983], with good experimental results. This section describes one such algorithm which has been found to work well.

The first major question in applying multigrid iteration to locally refined grids is the way grid levels are defined. The two major alternatives are indicated in Figure 4.1, where simple two dimensional grids are shown. In Figure 4.1a, the multigrid levels are local grids, while in Figure 4.1b all grids are global. Theoretical arguments lead one to believe the global grids in Figure 4.1b are superior, although local grids, as in Figure 4.1a, seem to perform about as well in practice. We follow here the more justifiable approach of using globally defined grids.

There are several other problems to deal with here. First there is the problem of calcuating residuals on locally refined grids like the fine grid in Figure 4.1b or the grid in Figure 3.2. This issue is dealt with in the next subsection.



a) local grids b) global grids

Figure 4.1. Locally Refined Multigrid Grid Levels for Locally Refined Grids

Second there is the problem of choosing the type of multigrid cycle. The adaptive finite element program PLTMG uses a recursive algorithm in which many more smoothing iterations are performed on coarse grids, during each cycle, than on fine grids. This type of algorithm is also known as a W-cycle. The alternative is an V-cycle in which equally many smoothing iterations are performed on each grid level.

Numerical tests do not show large differences between these approaches, Gannon and Van Rosendale [1973]. However, the recursive W-cycle can run into a problem on highly refined grids. If the number of multigrid levels is large, because of adaptive refinement, an excesive number of iterations on the coarsest grid occurs, and can dominate the computation time. Thus, the non-recursive V-cycle seems the more practical.

A final issue relates to allowing cells to belong to more than one grid level. In most multigrid algorithms, an approximation to the solution is computed on the finest grid. On the second finest grid corrections to the fine grid solution are computed. And in general, on the i-th grid a correction to the (i+1)st grid solution is required. When a cell belongs to a number of grid levels, it would thus need to store a number of different types of solution vectors.

There is a simple solution to this problem. One can use the full approximation storage scheme (FAS scheme) of Brandt [1977]. Though designed primarily for nonlinear problems, this scheme greatly simplifies code design for locally refined grids. In this scheme, coarse grid levels contain approximations to the solution, rather than correction vectors. Thus there is no difficulty in having cells belong to more than one grid level.

4.2 Interpolation and computation of Residuals

The basic steps in the multigrid algorithm here are injection and projection between grid levels, and performance of relaxation iterations. We consider here the weighted injection and projection operations natural in the finite element context. For the relaxation interations we suppose a simple Jacobe on simultaneous desplacement iteration is employed. These operations are quite simple and easy to program for uniform rectangular grids. However, with the data structures for adaptive refinement described in Section 3, these operations become relatively involved. This section describes how such operations are performed.

Of the three operations, injection, projection and calculation of residuals, the simplest by far is injection. The complex data structures needed for local refinement have little effect on it, and the programming required is similar to that required in the uniform grid case.

Projection and residual computation are more complex operations. As it turns out, these two operations are quite similar. A residual calculation involves the application of a finite difference stencil, or its finite element analog, at every mesh point. The projection operation is begun exactly the same, except the weights or coefficients in the stencils used are different. Following this, the projection operation is completed by transfering the results to the next coarser grid.

Because of this similarity, we content ourselves with describing in some detail the residual calculation. This calculation is formally the computation

$$r^{(k)} = A^{(k)} u^{(k)} - f^{(k)},$$

where $r^{(k)}$, $u^{(k)}$ and $f^{(k)}$ are vectors corresponding to functions in the

finite element space $M^{(k)}$. This is the finite element space for the k-th multigrid level and $A^{(k)}$ is the corresponding stiffness matrix.

It is inefficient to actually assemble the sparse martix $A^{(k)}$. Instead we form only the matrices corresponding to the separate cells in the block structured grid. Since cells amy be thought of as finite element macro-elements, we can form the element matrices for these macro-elements. Let $\{C_i\}_{i=1}^m$ be the cells in the grid and let $\{A_i^{(k)}\}_{i=1}^m$ be the corresponding macro-element stiffness matrices. Then we have

$$A^{(k)}u^{(k)} = \sum_{i=1}^{m} Q_i^{(k)} A_i^{(k)} P_i^{(k)} u^{(k)}$$

for certain matrices $\{Q_i^{(k)}\}_{i=1}^m$ and $\{P_i^{(k)}\}_{i=1}^m$. On uniform grids, the matrices $\{Q_i^{(k)}\}$, $\{P_i^{(k)}\}$ simply relate the global mesh point numbering to the local numbering in each cell, a standard operation in finite element programming. On locally refined grids, these matrices also perform the interpolations on cell boundaries required to enforce inter-element continuity.

These matters become fairly tranparent when viewed in the right way. It is helpful to think of "residual" as a fluid generated in each finite element and squirted to its corners. As shown in Figure 4.2, element residuals are combined to give partial residuals on each cell. Doing this, we get residuals at all points in the cells including the circled boundary points, even though there are no nodal variables at these circled points as that would violate the inter-element continuity requirement. Residual data at these circled points is then moved to adjacent boundary points, following the curved arrows shown. Finally residual information is shared along the dashed lines to complete the computation of $A^{(k)}u^{(k)}$.

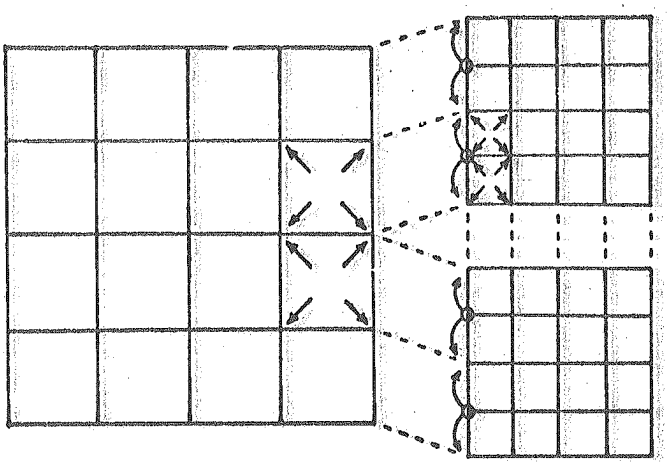


Figure 4.2. Data Movement During Residual Calculation

Though Figure 4.2 is two dimensional, there is no difference in treating three dimensions. There is also no real difference in treating the projection operation, except that less computation is required, since the value of the projection is required only at the points of the next coarser grid.

Notice that there is some wasted effort here, since residuals, on the value of the projection operator are computed redundantly on cell boundaries. Both cells sharing a face, or all cells sharing a vertex compute the same results. With three dimensional cells of size 10 by 10 by 10, approximately 30% of the computation time is wasted in redundant calculation. Using larger cells, the percentage of redundant computation would drop, but adaptive refinement would become less effective.

4.3 Error Bounds for Adaptive Refinement

The accurate estimation of the error present in a numerical solution is a subtle mathematical problem, frequently as difficult as generating the numerical solution in the first place. In adaptive finite element algorithms,

one produces a sequence of trial solutions $u^{(1)},u^{(2)},\cdots,u^{(k)},\cdots$, converging to the exact solution. Each trial solution $u^{(k)}$ is on a different grid G^k, and belongs to the corresponding finite element space M^k. We need error estimates here both to determine how to construct the successively refined grids $G_1,G_2,\cdots$, and to decide when a sufficiently accurate numerical solution has been obtained so the computation can be terminated.

Ideally, one would like to have three kinds of error estimates. First, one would like an extimate at the global error as a termination criterion. Second, one needs an estimate of the local truncation error, or error in energy at each point, since this is essential in deciding which points of the grid to refine. Finally, one would also like an estimate of the pointwise convergence rate of the descrete solutions to the exact solution, so that one can optimally allocate grid refinement.

The motivation for this third type of estimate is more subtle than the motivations for the other two. Our goal in adaptive refinement is to meet a given global error tolerance at least cost. Since it is nearly impossible to know which sequence of refinements will achieve this, we look instead for the refinement at each step which will minimize some weighted average of local errors at the next step. Since the rate of convergence is poorer in regions where the solution is singular, we can only select near optimal refinements if we know the approximate rate of convergence of the discrete solutions at each point. With out this information, excessive refinement will be concentrated in singular regions.

Considerable research has been focused on local and global error estimation. For approaches to global error estimation see Pereyra [1968], Lentini and Pereyra [1975], Lindberg [1976] or Stetter [1974]. These references also discuss local error estimation, since this is a necessary

component in global error estimation. For other references on local error estimation especially relevant to adaptive refinement see Babuska and Rheinboldt [1978A], [1978B].

5. Conclusion

The data structures, adaptive refinement strategies, unfitted grid generation approach, and multigrid solution algorithms described in this paper constitute building blocks for general numerical software for elliptic PDEs in complex three dimensional domains. Two dimensional experience with the Fears and Ellpack projects, and with the adaptive multigrid code PLTMG suggest the potential of this line of research. The difficulty of constructing three dimensional software makes the need for general purpose three dimensional elliptic solvers that much more urgent. Such software might prove as general and reliable as current ODE software, at least for simple scalar elliptic problems such as the Poisson and Helmholtz equations, on general three dimensional domains.

Acknowledgement

The author wishes to thank George Fix and Mac Hyman for valuable discussions of boundary condition treatment on unfitted grid. Dennis Gannon also deserves appreciation for his clarification of several issues regarding data structures and adaptive refinement.

References

Babuska, I. [1971]. Finite element method with penalty, Report No. BN-710, University of Maryland.

Babuska, I. and Aziz, A. K. [1972]. Survery lectures on the mathematical foundations of the finite element method, in The Mathematical Foundtions of the Finite Element Method with Applications to Partial Differential Equations, Academic Press, pp. 3-359.

Babuska, I. [1973]. The finite element method with lagrangion multipliers, Numer. Math., Vol. 20, pp. 179-192.

Babuska, I. and Rheinboldt, W. C. [1978A]. Error estimates for adaptive finite element computation, SIAM J. Numer. Anal., Vol. 15, pp. 736-754.

Babuska, I. and Rheinboldt, W. C. [1978B]. A-posteriori error estimates for the finite element method, J. Numer. Math. in Engrg., Vol. 12, pp. 1597-1615.

Bank, R. E. and Sherman, A. H. [1978]. Algorithmic aspects of the multi-level solution of finite element equations, CNA-144, Center for Numerical Analysis, The University of Texas at Austin.

Brandt, A. [1977]. Multi-level adaptive solution to boundary value problems, Math. Comp., Vol. 31, pp. 333-390

Gannon, D. B. [1980]. Self adaptive methods for parabolic partial differential equations, PhD. thesis, University of Illinois, Computer Science Technical Report UIUCDCS-R-80-1020.

Gannon, D. B. and Van Rosendale, J. [1983]. Solving elliptic PDE problems on parallel processors: experiments with locally refined grids, in preparation, ICASE, NASA Langley Research Center.

Houstis, E. N. and Rice, J. R. [1980]. An experimental design for the computational evaluation of elliptic partial differential equaiton solvers, The Production and Assessment of Numerical Software, M. A. Hennell, ed., Academic Press.

Lentini, M. and Pereyra, V. [1975]. An adaptive finite difference solver for nonlinear two-point boundary vaule problems with mild boundary layers, Report STAN-CS-75-530, Computer Science Department., Stanford University.

Lindberg B. [1976]. Error estimation and iteration improvement for the numerical solution of operator equations, Technical Report UIUCDCS-R-76-820, University of Illinois U-C, Urbana, Illinois.

McCormick, S. [1982]. Multigrid methods for variational problems: the V-cycle, Proc. IMACS World Cong. Sys. Sym. Sci. Comp., Montreal, Canada.

Pereyra, V. [1968]. Iterated deferred corrections for nonlinear boundary value problems, Numer. Math., Vol. 11, pp. 111-125.

Rheinboldt, W. C, and Mesztenyi, C. K. [1980]. On a data structure for adaptive finite element mesh refinements, ACM Trans. Math. Software, Vol. 6, No. 2, pp. 166-187.

Rubbert, P. E. and Lee, K. D. [1982]. Patched coordinate systems with block structure in Numerical Grid Generation, Proceedings and Symposium on The Numerical Generation of Curvilinear Coordinate Systems and Their Use in the Numerical Solution of Partial Differential Equations, J. Thompson, ed., North Holland.

Strang, G. and Fix, G. [1973]. An Analysis of the Finite Element Method, Prentice Hall.

Zave, P. and Rheinboldt, W. [1979]. Design of an adaptive, parallel finite element system, ACM Trans. Math. Software, Vol. 5, No. 1, pp. 1-77.