UNIFIED MULTILEVEL ADAPTIVE FINITE ELEMENT METHODS FOR ELLIPTIC PROBLEMS

BY

WILLIAM F. MITCHELL

B.S., Clarkson College of Technology, 1977
M.S., Clarkson College of Technology, 1979
M.S., Purdue University, 1983

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1988

Urbana, Illinois

PLEASE NOTE

This paper should be referenced as Technical Report UIUCDCS-R-88-1436, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1988.

I am providing this copy of the document as a service to those who prefer the speed of electronic transfer. You may write to the CS department at UIUC for an official copy of the report.

This document was prepared from the same troff files as the original report, with some irrelevant material (vita, etc.) omitted. However, due to differences between troff processors, there are some formatting differences. The most noticable is that the page numbers are different; the page numbers in the table of contents refer to the original page numbers and are incorrect for this document.

The original work was partially funded by the Department of Energy under grant DOE DEFG02-87ER25026.

Please keep a copy of this page with the printed document.

Bill Mitchell, April, 1993.

ABSTRACT

Many elliptic partial differential equations can be solved numerically with near optimal efficiency through the uses of adaptive refinement and multigrid solution techniques. It is our goal to develop a more unified approach to the combined process of adaptive refinement and multigrid solution which can be used with high order finite elements. The basic step of the refinement process is the bisection of a pair of triangles, which corresponds to the addition of one or more basis functions to the approximation space. An approximation of the resulting change in the solution is used as an error indicator to determine which triangles to divide. The multigrid iteration uses a red-black Gauss-Seidel relaxation in which the black relaxations are used only locally. The grid transfers use the change between the nodal and hierarchical bases. This multigrid iteration requires only O(N) operations, even for highly nonuniform grids, and is defined for any finite element space. The full multigrid method is an optimal blending of the processes of adaptive refinement and multigrid iteration. So as to minimize the number of operations required, the duration of the refinement phase is based on increasing the dimension of the approximation space by some fixed factor which is determined to be the largest possible for the given error-reducing power of the multigrid iteration. The result is an algorithm which (i) uses only O(N) operations with a reasonable constant of proportionality, (ii) solves the discrete system to the accuracy of the discretization error, (iii) is able to achieve the optimal order of convergence of the discretization error in the presence of singularities. Numerical experiments confirm this for linear, quadratic and cubic elements. It is believed that the method can also be applied to more practical problems involving systems of PDE's, time dependence, and three spatial dimensions.

TABLE OF CONTENTS

CHAPTER

1 INTRODUCTION 1

2 ADAPTIVE REFINEMENT 19
- 2.1 Newest vertex bisection of a triangle 20
- 2.2 Adaptive refinement using newest vertex bisection 23
- 2.3 Error indicator 32
- 2.4 Selection of next triangle to divide 38
- 2.5 Adaptive refinement algorithm 41

3 MULTIGRID SOLUTION 45
- 3.1 Hierarchical bases 47
- 3.2 Relaxation operator 51
- 3.3 Transfer operators 56
- 3.4 Convergence of the multigrid iteration 65

4 FULL MULTIGRID WITH ADAPTIVE REFINEMENT 72
- 4.1 Switching between refinement and solution 74
- 4.2 An error estimate 79

5 NUMERICAL RESULTS 82
- 5.1 Convergence of the discretization error 84
- 5.2 Effectivity index 88
- 5.3 Convergence of the multigrid iteration 90
- 5.4 Effect of f on the error 92

6 POSSIBLE FUTURE DIRECTIONS 95
- 6.1 Three dimensional problems 96
- 6.2 Time dependent problems 98
- 6.3 Rectangular elements 100
- 6.4 Collocation 102
- 6.5 Parallelism 104
- 6.6 Other possibilities 105

REFERENCES 106

VITA 111

CHAPTER 1
INTRODUCTION

The efficient numerical solution of elliptic partial differential equations has been an important area of research in numerical analysis for several decades. Over the years, many new methods have been discovered in the areas both of discretizing the differential equation and of solving the discrete problem. While every method has restrictions on the subclass of problems to which it is applicable, the efficiency of the methods has increased manyfold since even the best solvers of 25 years ago. A thorough presentation of most practical methods is provided by Birkoff and Lynch [8]. Additionally, the ELLPACK project [27] has provided us with robust software for many of these methods and a sound environment in which to perform numerical experiments to determine the relative merits of each method.

Today, we are at a point where many problems can be solved with near optimal efficiency. Many of the recent improvements have occurred through the uses of adaptive refinement, multigrid solution techniques and parallelism. We will not consider parallelism here but concentrate on adaptive refinement and full multigrid solution. At first glance, the two concepts seem almost meant for each other. Each is a process that alternately performs phases of refinement and solution, with one concentrating on refinement and the other on solution. Yet, when examined more deeply, subtle difficulties arise in combining them. Most researchers who have attempted to join the two have maintained the individual structures of the two phases and then developed strategies to overcome the problems that emerge. It is our goal to develop a more unified approach to the combined process of adaptive refinement and multigrid solution that is so natural that it becomes obvious how to extend these important techniques to more complicated situations, such as with high order methods and for three dimensional problems. This unification and extendability is achieved by interpreting the parts of the method from the viewpoint of the hierarchical basis, in which successive coefficients represent a change in the solution rather than the value of the solution. In particular

(i) adaptive refinement is considered to be the selective enrichment of the approximation space by adding new basis functions, rather than the division of triangles or rectangles. Local relaxations which are identical to those of the multigrid iteration are performed with the addition of each new basis function. The choice of which basis functions to add is based on how much each potential new basis function will reduce the error and is determined by approximating the hierarchical coefficients. This computation uses the equations that will be added to the linear system when the space is enriched by this basis function, and is the same as the local relaxation.

(ii) the multigrid iteration is defined strictly in terms of the hierarchical basis, and is not restricted to the approximation spaces we consider. Grid transfers are achieved through the change between nodal and hierarchical bases. Relaxations are performed in both the nodal and hierarchical bases, essentially supersaturating the approximation space with an excess of directions in which to minimize the error.

(iii) the full multigrid method is a very natural, optimal blending of adaptive refinement with multigrid iterations. The approximation space is enriched with as many new basis functions as is possible with respect to the error-reducing power of the multigrid iteration. This minimizes the number of operations used to obtain a solution whose accuracy is comparable with the discretization error.

We present our method in the context of the second order self-adjoint elliptic problem

$$Lu = (pu_x)_x + (qu_y)_y + ru = f \quad \text{in } \Omega \tag{1.1}$$

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

where Ω is a polygonal domain in $\mathbf{R}^2$ and p, q, r, f and g are functions of x and y. We use the Galerkin finite element method to discretize the problem over a triangular mesh which covers Ω exactly. We assume that the reader is familiar with the finite element method; a good presentation can be found in books by Strang and Fix [35] and Becker [7]. At times we will use the usual space of continuous piecewise linear functions over the triangles, especially when relating our work with that of other authors, but the method will be developed for the higher order spaces of C^0 p^{th} degree piecewise polynomials over triangles, where p is any given positive integer.

In all previous methods that combine adaptive refinement with multigrid solution the individual components of the method (triangle division, error indication, error estimation, prolongation and restriction, relaxation) are not related to each other, but are nevertheless combined to form an effective algorithm. In our approach all the components are closely related resulting in a more unified method in which many computations have multiple purposes. The key to recognizing the relationship is in the interpretation of the components from the viewpoint of hierarchical bases. While some of our components are novel, many are similar to, equivalent to, or in special cases reduce to existing approaches. In these cases we provide an alternative interpretation in terms of hierarchical bases which not only provides us with a new combination of techniques which are closely related, but also provides a deeper understanding of how and why the method works. With this knowledge the techniques are easily extended to other finite element spaces, although there is no existing theory to assure that reasonable convergence rates will be obtained. The method is, in fact, applicable to any finite element space with a hierarchical basis. We will examine the application of the method to spaces of C^0 p^{th} degree polynomials over triangles, and study the convergence numerically. Very little work has been done with either adaptive refinement or multigrid solution for high order methods, and no high order, adaptive refinement, multigrid method has been previously presented.

In Chapter 2 we present the adaptive refinement aspect of the method. As a whole, this is a new method of adaptive refinement. Many of the individual parts are very closely related to existing approaches, but the slight variations on these approaches, the new way in which they are combined, the new global structure of the refinement process and the interpretation of hierarchical bases result in a simpler, more unified and more efficient method which can easily be extended to other finite element spaces. We consider four aspects of the adaptive refinement process:

(i) triangle division,
(ii) maintaining compatibility,
(iii) error indication,
(iv) the overall structure of adaptive refinement.

For triangle division we use bisection. This is similar to the bisection method of Rivara [29, 30] and identical to that of Sewell [32, 33], the difference between the two being in the method for determining which side of the triangle is to be bisected, the longest edge or the side opposite the newest vertex, respectively. For many triangle shapes, including the important case of isosceles right triangles, the longest edge and newest vertex methods are equivalent. When the methods do not agree, the longest edge approach has better angle bounds and hence a smaller interpolation error, but the newest vertex approach does not require the computation of side lengths and

experimental results [24] indicate that the loss of accuracy is small. More importantly, the newest vertex approach provides easily defined hierarchical bases with properties that are necessary for the adaptive refinement and multigrid algorithms. In the longest edge approach the hierarchical bases are not as easily defined and fail to have these necessary properties.

A triangulation used for a finite element mesh should be compatible, i.e., every triangle should not have more than one neighboring triangle along any of its three sides, as in Fig. 1.1. Maintaining compatibility is one of the challenges in developing an adaptive refinement algorithm. The approach we use is unlike any other, and provides an interesting alternative interpretation of adaptive refinement on which we base our error indicator and the global structure of the adaptive refinement process, which in turn allows a new twist on the full multigrid method to be discussed later. Sewell maintains compatibility *before* the refinement process by removing from consideration those triangles whose division would create an incompatibility. It is possible for this to fail abysmally to refine the grid in the right place. Bank and Sherman [3, 4] and Rivara enforce compatibility *after* the refinement process. One divides the desired triangles, and then further divides triangles where incompatibilites have arisen. This results in a two phase refinement algorithm. Bank and Sherman, who use regular division for refinement and bisection for maintaining compatibility, even have a third phase at the beginning of the refinement algorithm that removes the bisection refinements added for compatibility. In our approach we maintain compatibility *during* the refinement process. This is achieved by dividing *pairs* of triangles that share a common edge rather than *individual* triangles. In the terminology of Sewell, the newest vertex is called the *peak*, the side opposite the peak is called the *base*, and a triangle is said to be *compatibly divisible* if its base is also the base of the triangle opposite its peak. When we wish to divide a triangle which is compatibly divisible, we divide both the desired triangle and the triangle opposite its peak as a pair by connecting their peaks through the midpoint of their common base. When we wish to divide a triangle which is not compatibly divisible, we must first divide the triangle opposite the peak (as a pair with the triangle opposite *its* peak) before dividing the pair. This process is easily implemented with a simple and short recursion provided that the assignment of the peaks in the initial triangulation is such that all the triangles are compatibly divisible. We show that such an assignment is always possible. Since we never introduce incompatibilities, *we have eliminated the need for a second phase* to eliminate them, and the resulting refinement algorithm is simpler.

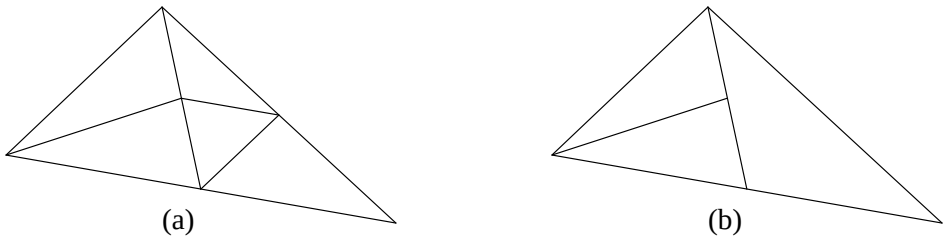


Fig. 1.1. Examples of (a) a compatible triangulation (b) an incompatible triangulation

By always dividing pairs of triangles we are provided with an alternative interpretation of the process of adaptive refinement. Unlike dividing a single triangle, *dividing a pair of triangles corresponds exactly to the addition of a new basis function*. In the context of hierarchical bases, the previously existing basis functions remain unchanged and the support of the new basis function is the pair of triangles divided. Thus we can consider the process of adaptive refinement to be one of enriching the finite element space with the right basis functions rather than one of refining the grid by dividing the right triangles. This provides many interesting and useful interpretations of, not only the components of the adaptive refinement, but also some aspects of the multigrid solution. An interesting, but not very useful, observation is that the restriction imposed on the grid by the condition of compatibility is just a restriction on which basis functions are permitted to be added to the finite element space in terms of which basis functions are already in the space, and our process of maintaining compatibility is one of adding the necessary basis functions before adding the desired basis function.

A more useful interpretation occurs with our error indicator. An error indicator is any value which can be used to indicate which triangles should be divided (in contrast to an error *estimate* which actually approximates the error). Our error indicator is similar to one proposed by Zienkiewicz et al. [39] for bilinear elements and can be formulated in terms of a local problem error estimate in the context of Babuška and Rheinboldt [2]. With the interpretation of adaptive refinement as a way of selecting which basis functions should be added to the space, it will become clear how to define a reasonable error indicator for *any* finite element space. In the hierarchical basis, the coefficient of a basis function of the last level added represents, not the *value* of the solution at the central node as in the nodal basis, but rather how much *change* in the solution has occurred at that node by adding the basis function. If we can estimate the coefficients for each of the basis functions that we are permitted to add to the space, then we can determine which basis functions will create the largest change in the solution, or equivalently, which basis functions will provide the largest reduction of the energy norm of the discretization error. Such an estimate can be obtained by considering the equation that would be added to the linear system if this basis function were added to the space. Solving this equation using the existing coefficients for neighboring basis functions provides us with the required estimate. The error indicator is then the energy norm of the new basis function times its coefficient.

Not only does our error indicator point to the optimal basis function in terms of discretization error reduction, but it is closely related to other aspects of the global algorithm and its computation provides several other useful values, including

(i) the values for the stiffness matrix,
(ii) the first solution value for the new node (equivalent to Gauss-Seidel relaxation),
(iii) values from which a global error estimate can be cheaply computed.

Typically, other error indicators found in the literature are based on a bound on some norm of the error over a triangle, and bear no relationship to the rest of the global algorithm. The computations performed in those error indicators serve no other purpose, except possibly providing a global error estimate. While such error estimate based error indicators provide error estimates over each triangle which asymptotically approach the true error, there is no reason to believe that it is better to divide triangles with the largest error than to add the basis functions that reduce the error the most. In numerical experiments, which approach is better is found to be problem dependent, and there is never a large difference between them.

To see how to extend our error indicator to other finite element spaces, consider the example of C^0 quadratic basis functions over triangles. Here the division of a pair of triangles adds *four*

new nodes and basis functions. Thus any enrichment of the space will be done by four basis functions at a time. So the error indicator will be for the group of four basis functions, and is computed as above but by solving the system of four equations for the four hierarchical coefficients. The error indicator is the energy norm of the function obtained from the four new basis functions and their coefficients. Unlike other error indicators, we have an error indicator which falls naturally out of the finite element space in use. This allows us to define an appropriate error indicator for *any* finite element space.

We now consider the global structure of the adaptive refinement process and how it fits in with the solution process. In the usual approach, illustrated by Alg. 1.1, the refinement phase is the division of a set of triangles to obtain a new grid in which each triangle has been divided at most once. This is typically done in four steps -- compute error indicators for each triangle, determine which triangles are to be divided, divide triangles and enforce compatibility. As noted earlier, Bank and Sherman have an additional step at the beginning where they remove the bisection refinements used to enforce compatibility. The determination of which triangles are to be divided is usually done by the approach outlined by Babuška and Rheinboldt [2] where one searches the triangles for those whose error indicators are larger than some threshold value which depends on the largest error indicator. For an efficient full multigrid method, it is necessary that the dimensions of the successive finite element spaces grow exponentially. To achieve this, the solution phase is performed only if the number of vertices has grown by some factor, typically 4 which is the growth one would get with uniform refinement. For highly nonuniform grids, the number of refinement phases between solution phases may grow exponentially. When a solution phase is not performed, it may be possible to omit some of the steps of the refinement phase. Certainly, many of the error indicators may remain unchanged, but then one must decide which triangles need a new error indicator. It is also possible that some of the refinements to enforce compatibility can be postponed. We note that, with highly nonuniform grids, *just searching every triangle during each refinement phase requires an unacceptably large number of operations*. So any algorithm of this type must not compute the error indicator for every triangle, or even examine every triangle to determine which ones have large error indicators. As none of the authors address this problem, it is unknown how (or if !) this problem is dealt with.

Algorithm 1.1. The usual global structure

```
repeat
  remove bisection refinements (Bank and Sherman only)
  compute error indicators
  S ← set of triangles with large error indicators
  divide the triangles in S
  divide triangles for compatibility
  if the number of vertices has been increased by the factor 4 then
    apply multigrid iteration(s)
  endif
until done
```

We take a different approach to the global structure of the refinement phase and how it fits into the full multigrid method. Our approach is illustrated in Alg. 1.2. Because we interpret adaptive refinement as the addition of new basis functions, we are freed from the bonds associated with dividing a predetermined set of triangles to obtain a new grid in which each triangle has been divided at most once. Instead, we can add basis functions to the finite element space *for as long as we like*, provided we maintain the solution values and error indicators and have a way of determining which basis functions have the largest error indicators. The purpose of the solution phase in the full multigrid algorithm is to keep the error in the approximate solution of the discrete system less than the discretization error. We will consider this in more depth later, but for now we state that this can be achieved by performing a solution phase every time the dimension of the space is increased by some factor (but not necessarily 4 as is typically used). So, in our approach we continue to enrich the space with new basis functions until the dimension of the space reaches a predetermined value. Contrast this to the usual approach where, if that predetermined dimension is not met, another complete refinement phase is performed which could exceed the predetermined dimension considerably.

It is necessary in our approach to maintain the solution values and error indicators during the refinement process. Reasonable solution values are necessary to obtain reasonable error indicators, and without reasonable error indicators the refinement may be degraded. To maintain our solution, with the addition of each new basis function we perform a relaxation at each of the nodes which are most strongly affected by the addition of that basis function, i.e., those nodes associated with basis functions which are not orthogonal to the new basis function. (Recall that the coefficient for the new basis function itself comes directly from the error indicator.) This is sufficient to obtain reasonable error indicators, but does not remove the need for the solution phase. However, as will be seen later, these relaxations are very closely related to the multigrid iteration and can actually be considered to be part of the solution phase, again demonstrating the strong relationship of all components of this method. The addition of a new basis function and the local relaxations that follow it affect only a few neighboring error indicators, which are easily updated at this time.

Algorithm 1.2. Our global structure

```
repeat
  repeat
    pick a basis function to add
    add that basis function †
  until the number of vertices has been increased by some given factor
  apply multigrid iteration(s)
until done
```

---

† the addition of a basis function includes adding other basis functions first (if necessary), local relaxations and updating affected error indicators

This leaves us only with the problem of knowing which potential new basis functions have the largest error indicators. Ideally we would want to know which one has the largest, but we cannot search them all in an acceptable number of operations. We might create a linked list ordered by the size of the error indicator, but here, too, we cannot make an insertion in O(1) operations. Instead, we will be satisfied to add a basis function whose error indicator is *close* to the largest. Toward this end we construct a set of linked lists each of which contains those potential new basis functions whose error indicators fall in a given range of values with the ranges determined so as to guarantee that by the end of the refinement phase all the basis functions with the largest error indicators have been added. The maintenance of these lists can be performed in O(1) operations.

In Chapter 3 we present and analyze the multigrid iteration used to solve the linear system of equations. This multigrid uses a V-cycle, a restricted form of red-black Gauss-Seidel relaxation, and restriction and prolongation operators which arise naturally from the hierarchical basis. In the case of uniform grids the method is equivalent to that studied by Braess [10, 11, 12] and is related to the MGR methods [22, 28]. It is also very closely related to the hierarchical basis multigrid method recently developed by Bank, Dupont and Yserentant [6].

The main difficulty with multigrid iterations for nonuniform grids is in producing a method for which the number of operations for one cycle is proportional to N, the number of nodes, while obtaining an error reduction factor which is bounded away from 1 independent of N. Several approaches to this problem have been taken. Bank and Sherman [3] use a form of level compression where several levels are treated as one to obtain a geometric growth in the number of nodes in each level. The problem with this is that the number of levels compressed into one level may grow exponentially. This means that the relaxation must damp more than just the high frequency components of the error, hence *either the number of relaxations required is not independent of N or there is no guarantee of an adequate error reduction factor*. To overcome this problem, Rivara [31] uses a local (rather than global) transfer and relaxation process for the levels where the number of nodes has not grown geometrically. Her description of this process is vague, making it difficult to tell, not only how to do this process, but also whether or not it will work. In the method of Bank, Dupont and Yserentant [6], the number of operations for one V-cycle is guaranteed to be O(N) for nonuniform grids by performing only the red part of a red-black Gauss-Seidel relaxation, the red nodes being those that are in grid k but not grid $k-1$. However, *the error reduction factor is not independent of N*, and the number of V-cycles required grows proportional to the number of levels. For a uniform grid this is O($\log N$) and for nonuniform grids it can be as bad as O(N). In the method of Braess the relaxation uses only the red nodes before coarse grid correction and red followed by black after. This reduces the error sufficiently, but for nonuniform grids could require as much as O(N^2) operations for one V-cycle. We use an intermediate approach which uses only red relaxation before coarse grid correction and both red and black after, *but the black relaxation is performed only at black nodes that are neighbors of red nodes*. As with the relaxations performed during the refinement, these are the nodes that are most strongly affected by the change of the values at the red nodes. This guarantees O(N) operations for one V-cycle and appears to be sufficient to maintain an error reduction factor which is independent of N. We have no mathematical proof of this, but we present supporting numerical evidence. Thus we have apparently achieved *both* O(N) operations per cycle and an N-independent error reduction factor.

The multigrid method presented is easily extended to higher order finite element spaces. Little has been achieved in the multigrid solution of linear systems that arise when using high order methods. One of the biggest difficulties is in determining appropriate restriction and prolongation operators. Our transfer operators fall naturally out of the conversion of the nodal basis

to the hierarchical basis, making it easy to determine appropriate operators for *any* finite element space. For some low order spaces they degenerate to commonly used transfers. Thus we have a way of defining, for any finite element space, multigrid transfer operators which are of the correct order of accuracy, natural to the space in question, and degenerate to the usual operators for some spaces previously considered.

The multigrid iteration is used periodically in a full multigrid algorithm to keep the error in the approximate solution smaller than the discretization error. In our full multigrid, the multigrid iteration is used whenever the number of nodes has been multiplied by some given constant. The frequency at which multigrid iterations must occur depends on the error reduction factor of one V-cycle. Thus it is important to have a good bound on that factor to determine how often to switch between refinement and solution. In the case of the usual model problem (Poisson's equation, linear elements and uniform grids) our method degenerates to that anayzed by Braess [10, 11, 12]. Braess showed that the error reduction factor for the V-cycle is bounded by ½ for certain polygonal domains. In our analysis, we use some results from linear algebra to show how one can compute the error reduction factor for any problem, not just the model problem. With the model problem on the unit square we find that the error reduction factor for the V-cycle is apparently bounded by 1/8, the same value found for the 2-grid iteration by Fourier analysis. It will be shown that this means one V-cycle reduces the error sufficiently to increase the number of nodes by the factor 32.5 between multigrid iterations, in contrast to the usual factor 4. We also examine the error reduction factor for 3^{rd} and 4^{th} order finite elements with the model problem and finally consider, through examples, the effects of nonuniform grids and reentrant corners.

In Chapter 4 we present the full multigrid which combines the adaptive refinement procedure with the multigrid iteration into a very efficient unified solution method that is easily extended to high order finite element spaces. The full multigrid method consists of alternately performing refinement and solution phases. For the usual full multigrid method for uniform grids the multigrid iteration (solution phase) is performed after each refinement to the next level grid. But with nonuniform grids this can result in more than O(N) operations, as much as O(N^2). The usual extensions to nonuniform grids maintain the concept of one refinement phase resulting in a grid in which each triangle has been refined at most once, and skip some of the grids for solution phases, possibly performing some local solution process, to obtain the O(N) operation full multigrid. In contrast, we use the properties of the convergence of the discretization error to justify basing the frequency of solution phases, *not on the levels of refinement, but on the increase in the number of nodes*. Thus, as earlier noted, the refinement phase continues until the number of nodes has been increased by some given factor. We derive a formula to determine how large this factor can be in terms of the error reduction factor of the multigrid iteration and the convergence rate of the discretization error for the finite element space being used.

We also present a new error estimate in Chapter 4. This error estimate is not as accurate as some available estimates, such as that of Bank and Weiser [5]. Moreover, it may not satisfy some of the requirements of a "good" error estimate as outlined by De et al. [16], in particular that the effectivity index (the ratio of the error estimate to the actual error) be greater than 1 and approach 1 as N approaches infinity. But, from the values of the error indicator we use, the estimate is very cheap to compute, requiring less than $2N$ multiplications, and appears to be reasonably accurate in practice with effectivity indices typically between .9 and 1.2. More importantly, the estimate can be extended to high order finite elements providing a reasonable error estimate for arbitrarily high order finite element spaces.

The full multigrid method presented is an optimal combination of adaptive refinement with multigrid iterations. The hierarchical basis multigrid iteration is guaranteed to use only O(N)

operations, and the adaptive refinement procedure produces a grid over which the optimal order of convergence of the discretization error is achieved. The frequency at which multigrid iterations occur guarantees that the entire algorithm requires only O(N) operations. All components of the method are closely related to each other, providing a unified O(N) algorithm with adaptive refinement and multigrid solution for high order finite element spaces.

CHAPTER 2
ADAPTIVE REFINEMENT

The use of adaptive refinement to obtain a grid for the discretization of a partial differential equation has been the subject of much research in the past decade [2, 3, 16, 24, 25, 30, 32, 39]. The idea is to automatically construct a grid which is coarse where the solution is well behaved, fine near singularities, boundary layers, etc., and has a smooth transition between the coarse and fine parts. Such a grid can dramatically reduce the number of nodes needed to obtain an accurate solution for marginally smooth problems, and can recover the optimal order of convergence for nonsmooth problems.

Central to any adaptive refinement algorithm for a finite element grid is a method for dividing (refining) the elements, triangles in our case. There are two major methods for dividing triangles in adaptive refinement algorithms. Regular division divides a triangle into four similar triangles by connecting the midpoints of the sides. Bank and Sherman [3, 4] show how to use regular division in an adaptive refinement algorithm. Bisection division connects one of the vertices of the triangle to the midpoint of the opposite side. Two approaches are in use regarding the selection of the vertex to be divided. Rivara [29, 30] chooses the vertex opposite the longest edge. Sewell [32, 33] chooses the "newest" vertex. This is the method we use, and it will be fully explained in §2.1.

Another critical element of any adaptive refinement algorithm is the error indicator, which is used to determine which triangles should be divided. Several good error indicators have been proposed [2, 3, 5, 39]. Most of these are based on estimating the discretization error over each triangle. Our approach is slightly different. We attempt to determine which basis functions that can be added would reduce the discretization error the most. This becomes a type of error indicator for *pairs* of triangles. Mitchell [24] surveyed several error indicators and triangle division methods and compared them in a numerical experiment. He found that, among the methods considered, there is no universally "best" adaptive refinement method, and that most of the methods performed approximately the same. The methods we use performed well in that experiment.

We begin the presentation of our adaptive refinement algorithm by describing the process of bisecting a triangle by the newest vertex, and giving some properties of the resulting triangles. We then show how to use this bisection in an adaptive refinement algorithm. This is followed by the definition of our error indicator, and finally we present the adaptive refinement algorithm. For the most part, adaptive refinement is independent of the space of functions to be defined over the triangulation. The exception to this is in the error indicator, which we define first for piecewise linear functions, and then for piecewise C^0 p^{th} degree polynomials.

2.1 Newest vertex bisection of a triangle

The basic building block of an adaptive refinement algorithm is a method for dividing a triangle. The method we use, which we call *newest vertex bisection*, is nearly identical to a method presented by Sewell [32]. Much of the terminology of this section is due to Sewell.

In bisection division a triangle is divided to form two new triangles by connecting one of the vertices, called the *peak*, to the midpoint of the opposite side called the *base*, as in Fig. 2.1. The original triangle is called the *parent*, and the two new triangles are called the *children*. The children are said to have *generation* i+1 where i is the generation of the parent. The initial triangle is assigned generation 1. The assignment of the peak for the initial triangle will be examined in

Fig. 2.1. Propagation of the peak with newest node bisection

§2.2. The new vertex created at the midpoint of the base is assigned to be the peak of the children, hence the name newest vertex bisection.

It is important that the angles be bounded away from 0 and π [1, 18]. Notice that by assigning the peak this way, no angle is divided more than once. Intuitively, this should prevent the angles from getting too small. Indeed, it is shown by Sewell [32] that there are only four similarity classes of triangles created by this method, as in Fig. 2.2, and hence the angles are bounded.

In the future it may be useful to know, not only that the angles are bounded away from 0 and π, but also exactly what the angles are. It is easily seen that only eight angles arise, as illustrated in Fig. 2.3. If α, β and γ are the angles of the initial triangle and γ is the angle at the peak, the other angles are given by

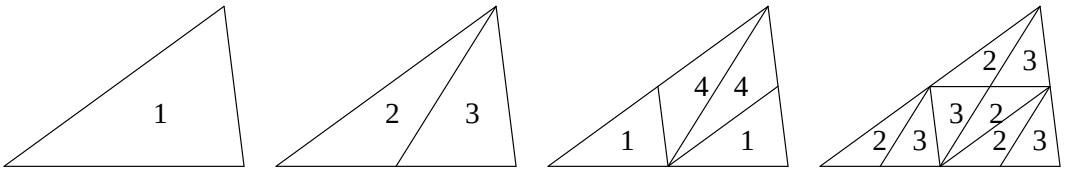


Fig. 2.2. Four similarity classes of triangles generated by newest vertex bisection

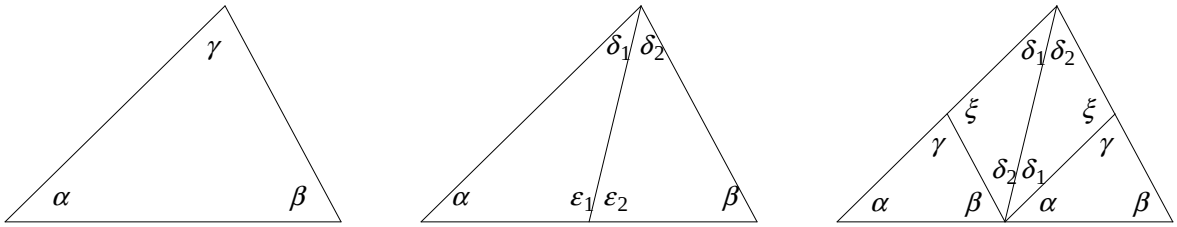


Fig. 2.3 Angles that arise during bisection

$$\varepsilon_1 = \tan^{-1} \frac{2 \sin\alpha \sin\beta}{\sin(\alpha - \beta)}$$

$$\varepsilon_2 = \pi - \varepsilon_1$$

$$\delta_1 = \varepsilon_2 - \alpha$$

$$\delta_2 = \varepsilon_1 - \beta$$

$$\xi = \pi - \gamma$$

The only angle which is not obvious is ε_1, which we derive here. In the original triangle, let b and c be the lengths of the sides opposite the angles β and γ, respectively. Then by the law of sines

$$\frac{\sin\beta}{\sin\gamma} = \frac{b}{c} \quad \text{and} \quad \frac{\sin\varepsilon_1}{\sin\delta_1} = \frac{b}{c/2}$$

thus

$$\sin\varepsilon_1 = \frac{2 \sin\beta \sin\delta_1}{\sin\gamma}$$

Since $\delta_1 = \pi - \alpha - \varepsilon_1$, $\sin\delta_1 = \sin(\alpha + \varepsilon_1)$, and thus

$$\sin\varepsilon_1 = \frac{2 \sin\beta}{\sin\gamma} (\sin\alpha \cos\varepsilon_1 + \cos\alpha \sin\varepsilon_1)$$

$$(1 - \frac{2 \sin\beta \cos\alpha}{\sin\gamma}) \sin\varepsilon_1 = \frac{2 \sin\beta \sin\alpha}{\sin\gamma} \cos\varepsilon_1$$

$$\tan\varepsilon_1 = \frac{2 \sin\beta \sin\alpha / \sin\gamma}{1 - 2 \sin\beta \cos\alpha / \sin\gamma} = \frac{2 \sin\beta \sin\alpha}{\sin\gamma - 2 \sin\beta \cos\alpha}$$

Since $\gamma = \pi - \alpha - \beta$,

$$\tan\varepsilon_1 = \frac{2 \sin\alpha \sin\beta}{\sin\alpha \cos\beta + \cos\alpha \sin\beta - 2 \sin\beta \cos\alpha} = \frac{2 \sin\alpha \sin\beta}{\sin(\alpha - \beta)}$$

2.2 Adaptive refinement using newest vertex bisection

Dividing an individual triangle is only one aspect of refining a triangulation. The process of adaptive refinement is one of dividing triangles such that

(i) the angles are bounded away from 0 and π,

(ii) the grid is fine in the right places,

(iii) the triangulation is compatible (defined below),

(iv) the process requires only O(number of triangles) operations.

In the previous section we showed that the angles will be bounded away from 0 and π. To make

certain that the grid is fine in the right places, one uses an error indicator which specifies which triangles should be divided. We present our error indicator in §2.3, and for now assume the existence of such an error indicator. In this section we develop an approach to adaptive refinement using newest vertex bisection such that the third and fourth requirements are also satisfied.

One of the difficulties in adaptive refinement is that of maintaining compatibility of the triangulation. A triangulation is said to be *compatible* if for any two triangles t_i and t_j, $t_i \cap t_j$ is either empty, a common vertex, or a common side. Other authors, most notably Rivara [30] and Bank et al. (e.g. [4]), have taken the approach of dividing some set of triangles with large error indicators, producing an incompatible triangulation, and then performing a second process to regain compatibility by dividing more triangles. The approach of Bank even includes a third process of removing some of the extra divisions before the next refinement phase because he uses bisection for triangles divided to maintain compatibility and regular division for the triangles divided because of a large error indicator. In our approach we never have an incompatible triangulation. Compatibility is maintained *during* the refinement process, rather than after, by dividing *pairs* of triangles rather than individual triangles. Thus we have eliminated the need for a separate follow-up process to recover compatibility.

A triangle is said to be *compatibly divisible* if its base is either the base of the triangle that shares that side or part of the boundary of the domain. If a triangle is compatibly divisible, then we divide the triangle and the neighbor opposite the peak (if such a neighbor exists) simultaneously as a pair. If a triangle is not compatibly divisible, then after a single bisection of the neighbor opposite the peak, it will be. So in this case, we first divide the neighbor by the same process, and then divide the triangle and neighbor opposite the peak simultaneously. This is illustrated in Fig. 2.4. We note that this process always divides a pair of compatibly divisible triangles, or a triangle whose base is part of the boundary, and that the triangulation is never incompatible. This leads to the recursive Algorithm 2.1, which is easily implemented in FORTRAN by constructing a stack of the triangles that need to be divided.

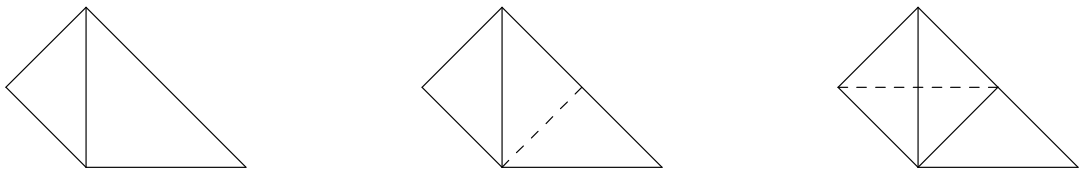


Fig. 2.4. Maintaining compatibility during refinement

Algorithm 2.1 divide_triangle(t)

```
if t is not compatibly divisible then
  divide_triangle(neighbor of t opposite peak)
endif
divide the triangle pair t and the neighbor opposite the peak of t
return
```

It is important that this recursion be finite and not very large. We will show that the length of the recursion is bounded by the generation of the triangle. To do this we must assume that in the initial triangulation every triangle is compatibly divisible. Theorem 2.1 shows that this assumption is reasonable.

Theorem 2.1. Given any triangulation, there exists a choice of peaks such that every triangle is compatibly divisible.

We postpone the proof of this theorem until the end of this section, where the theorem is restated as Theorem 2.9. We also need the following lemma which relates the generations of neighboring triangles.

Lemma 2.2. Let T_0 be an initial triangulation in which every triangle is compatibly divisible, T be a refinement of T_0 and $t_0 \in T$ have generation g. Let t_1 be the triangle opposite the peak of t_0 (if it exists) and t_2 and t_3 be the other neighbors of t_0 (if they exist). Then

(i) if t_0 is compatibly divisible, the generation of t_1 is g,
(ii) if t_0 is not compatibly divisible, the generation of t_1 is g-1,
(iii) if t_0 is the triangle opposite the peak of t_i, i=2 or 3, the generation of t_i is g+1,
(iv) if t_0 is not the triangle opposite the peak of t_i, i=2 or 3, the generation of t_i is g.

Proof. We prove this by induction. The conclusion holds for T_0 since every triangle is compatibly divisible and has generation g=1. Suppose the conclusion holds for T a refinement of T_0, and consider the triangulation $\tilde{T}$ obtained by dividing one pair of compatibly divisible triangles as in Fig. 2.5 (the case of dividing a single boundary triangle is nearly identical). Let g be the generation of t_0 and t_1 in T. Then the triangles t_{01}, t_{02}, t_{11} and t_{12} all have generation g+1 in $\tilde{T}$. It suffices to examine the generations of the three neighbors of t_{01} in $\tilde{T}$. t_{01} is not opposite the peak of t_{02} (and t_{11}) and the generation of t_{01} is the same as the generation of t_{02} (and t_{11}), so the conclusion holds for the triangles that are not opposite the peak. For the triangle opposite the peak there are two cases. Suppose that t_0 is opposite the peak of t_2 in T. Then by the inductive hypothesis t_2 has generation g+1, we see that t_{01} is compatibly divisible in $\tilde{T}$, and so the conclusion holds. If

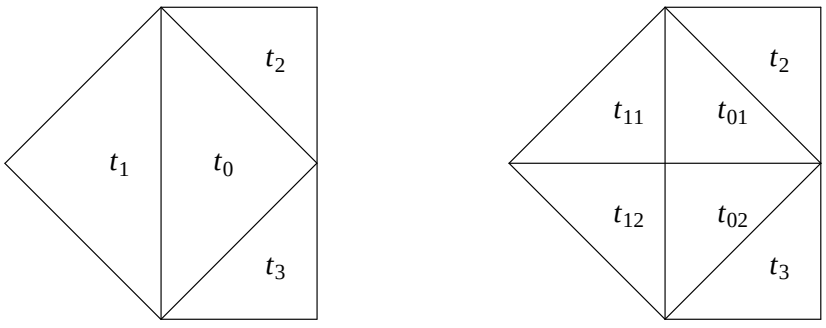


Fig. 2.5 Triangles for Lemma 2.2

t_0 is not opposite the peak of t_2 in T, then t_2 has generation g and t_{01} is not compatibly divisible, so again the conclusion holds. □

Theorem 2.3. The length of the recursion in Alg. 2.1 is bounded by the generation of the triangle t.

Proof. The recursive call occurs as long as the triangle passed in is not compatibly divisible. From Lemma 2.2 we see that the triangle opposite the peak of a triangle which is not compatibly divisible has one less generation. Thus the generation of the triangle decreases with each recursive call, and since the minimum generation is 1, the number of recursive calls is bounded by the generation of the first triangle. □

Since some extra effort is required to divide triangles which are not compatibly divisible, it is not unreasonable to ask whether or not we can avoid such triangles. Since we can always assign the peaks in such a way that all triangles are compatibly divisible, we could change the peaks to achieve this. However, we would no longer be performing newest vertex bisection and this would surely result in angles which are unacceptably large or small. Without changing the peaks, the only way to eliminate triangles which are not compatibly divisible is to perform some extra refinements. The following corollary of Lemma 2.2 asserts that any such attempt would result in a uniform refinement, and hence for adaptive refinement we necessarily have triangles that are not compatibly divisible.

Corollary 2.4. If every triangle in the initial triangulation is compatibly divisible, then for any refinement either there exists a triangle which is not compatibly divisible, or every triangle has the same generation.

The remainder of this section is devoted to the proof of Theorem 2.1. We call an assignment of peaks for a triangulation such that every triangle is compatibly divisible a *perfect matching* of the triangles. This is related to perfect matching in graph theory, and in fact we will use graph theory to prove that every triangulation has a perfect matching.

We begin with some definitions from graph theory. A *graph*, G, is a nonempty set, V, of *vertices* and a set, E, of *edges* which are unordered pairs of vertices. To avoid confusion between the vertices of a triangle and the vertices of a graph, we use the term *G-vertex* to refer to a vertex of a graph. If $e=(u,v) \in E$, then the G-vertices u and v are said to be *adjacent* and are called the *endpoints* of e. The *degree* of a G-vertex v is the number of edges in E containing v. A graph is *k-regular* if all vertices have degree k. A *walk* is a nonempty sequence $v_0v_1v_2\cdots v_n$ such that $(v_i, v_{i+1}) \in E$. A *cycle* is a walk in which $v_0 = v_n$ and $v_i \neq v_j$ for all other $i \neq j$. A graph is said to be *connected* if for every $u,v \in V$ there exists a walk with $v_0 = u$ and $v_n = v$. An edge e of a connected graph is a *cut edge* if the removal of e from G results in a graph which is not connected. A *perfect matching* in G is a set $M \subseteq E$ such that each G-vertex in V is an endpoint of exactly one edge in M.

We are now ready to define $G(T)$, the graph of a triangulation T. The G-vertex set consists of two parts. With each triangle of T we associate an *interior G-vertex*. With each side of a triangle which is part of the boundary of the domain (boundary side) we associate a *boundary G-vertex*. V consists of all interior and boundary G-vertices. The edges of $G(T)$ are of three forms. If $u,v \in V$ are both interior G-vertices, then $(u,v) \in E$ iff the corresponding triangles share a

common side. These edges correspond to interior sides of triangles. If one of u or v is an interior G-vertex and the other is a boundary G-vertex, then $(u,v) \in E$ iff the corresponding boundary side is a side of the corresponding triangle. These edges correspond to boundary sides. Finally, if u and v are both boundary G-vertices, then $(u,v) \in E$ iff the corresponding boundary sides share a common vertex. These edges correspond to boundary vertices. Fig. 2.6 illustrates a simple triangulation and its graph. We use squares for boundary G-vertices and circles for interior G-vertices.

For simplicity, we assume that the boundary of the domain is a continuous simple closed curve. Then it is easily seen that the graph has a cycle containing exactly the boundary G-vertices and the edges that correspond to boundary vertices of the triangulation. This cycle is found by traversing the boundary of the domain. The results that follow hold also for more complicated domains, but this complicates the proofs. If the boundary is not continuous, then there is more than one cycle of boundary G-vertices. If the boundary is not a simple curve, then the edge set between boundary G-vertices must be modified slightly so that each boundary G-vertex is adjacent to exactly two other boundary G-vertices. From the definition of the graph of a triangulation, the next lemma is obvious.

Lemma 2.5. The graph of a triangulation is 3-regular.

Theorem 2.6. An edge of a connected graph is a cut edge iff it is contained in no cycles.

This is a well known theorem from graph theory. See, for example, Bondy and Murty [9] p. 27.

Lemma 2.7. The graph of a triangulation contains no cut edges.

Proof. Let e be any edge in $G(T)$. We will show that e is contained in a cycle. If e connects two boundary G-vertices, then e is contained in the cycle of boundary G-vertices found by traversing the boundary of the domain. Otherwise, let s be the triangle side corresponding to e and let v be a vertex of T that is an endpoint of s. Let $\{s_i\}$ be the set of all triangle sides that have v as an endpoint, and $\{e_i\}$ be the corresponding edges in $G(T)$ plus also the edge corresponding to v if v is a boundary vertex. It is easily seen that, when properly ordered, the edges e_i form a cycle in $G(T)$ since their endpoints correspond to triangles, boundary sides and boundary vertices that are adjacent in T. Since e is one of the e_i's, we have a cycle containing e. □

Fig. 2.6 A simple triangulation and its graph

Theorem 2.8. Every 3-regular graph without cut edges has a perfect matching.

For a proof of this see, for example, Bondy and Murty [9] p. 79.

Theorem 2.9. Every triangulation has a perfect matching.

Proof. From Lemma 2.5, Lemma 2.7 and Theorem 2.8, $G(T)$ has a perfect matching M. We define a perfect matching for T as follows. For each $e \in M$ for which both endpoints are interior G-vertices, select the peaks of the corresponding triangles so that the bases of those two triangles are their common side. Then both of these triangles are compatibly divisible. For each $e \in M$ for which one endpoint is an interior G-vertex and the other endpoint is a boundary G-vertex select the peak of the corresponding triangle so that the base is the corresponding boundary side of the triangle. Then this triangle is compatibly divisible. Since (i) every triangle corresponds to an interior G-vertex, (ii) M is a perfect matching, and (iii) we have considered every interior G-vertex that is an endpoint of an edge in M, we have assigned a peak for every triangle and every triangle is compatibly divisible. □

Theorem 2.9 says that given any triangulation we can find a way to assign the initial peaks such that every triangle is compatibly divisible. It would be nice to develop an algorithm which automatically assigns the peaks such that we have a perfect matching. However, the perfect matching problem is known to be NP-Complete [20]. It is possible that it can be solved in polynomial time for the special case of 3-regular graphs, but we have been unable to find any algorithm for this. In any case, it is probably not a good idea to develop such an algorithm, at least not without incorporating further guidelines. The perfect matching is not unique, and a poor choice can result in angles which are unnecessarily small. Whenever possible, it is best to use the largest angle as the peak. Finding the optimal perfect matching in terms of angle conditions is nontrivial. Usually, though, the initial triangulation contains a small number of triangles and it is easy for the user to find a good perfect matching.

2.3 Error indicator

To guide the adaptive refinement, it is necessary to have some sort of error indicator which determines which triangles should be refined. Many error indicators have been proposed [2, 3, 5, 39]. Mitchell [24] performed a numerical experiment to compare the effectiveness of several indicators. The method we describe here performed well in those experiments. This method is similar to that proposed by Zienkiewicz et al. [39] for bilinear rectangular elements. Our interpretation of the indicator makes it possible to define an error indicator for any finite element space. We will concentrate on the spaces of C^0 p^{th} degree polynomials over bisected triangles. We consider first linear elements, and then show how to extend to arbitrary degree.

At this point we must make the distinction between an error indicator and an error estimate. By an *error indicator* we mean a nonnegative real number assigned to each triangle, or small groups of triangles, which has its largest values in the triangles whose refinement would be most beneficial for reducing the discretization error. An *error estimate*, on the other hand, can be defined either locally or globally and should be a good approximation of the discretization error in some norm. An error indicator is used to guide adaptive refinement; an error estimate can be used as a termination criterion for a program, or just to give the user some idea of how accurate

the solution is. Usually, if an error estimate is defined locally it can be used as an error indicator, but it is not clear that error estimates make the best error indicators.

Our error indicator is not an error estimate (although we present an error estimate based on this error indicator in §4.2). Instead of attempting to divide the triangles over which the error is the largest (which is what an error estimate based error indicator does), we attempt to divide the triangles for which the division thereof makes the greatest change in the solution. Since this change in the solution reduces the error, we are attempting to divide the triangles which make the greatest reduction in the error, in other words, reduce the error the fastest for the number of divisions performed. If these numbers were known exactly this would provide the optimal choice of which triangles to divide. Of course, we are approximating these values so we do not claim this to be the optimal method.

To approximate how much of a change in the solution will occur, we use the hierarchical basis, which we define in §3.1. For what follows it suffices to know that when a function is expanded using the hierarchical basis, the coefficients represent displacements rather than nodal values as with the usual nodal basis. Fig. 2.7 illustrates this for a simple case in one dimension. We consider the interpolation of a function f, defined over the unit interval, by piecewise linear functions f_2 and f_3 which have 2 and 3 nodes, respectively. Let $f_3 = \alpha_1\phi_1 + \alpha_2\phi_2 + \alpha_3\phi_3$ for some basis $\phi = \{\phi_1, \phi_2, \phi_3\}$. If ϕ is the usual nodal basis, $\alpha_2 = f(\frac{1}{2})$. But, if ϕ is the hierarchical basis, $\alpha_2 = f(\frac{1}{2}) - f_2(\frac{1}{2})$, i.e., α_2 represents how much change occurs in the approximating function when we refine the grid by adding the new node. Then $|| \ \alpha_2\phi_2 \ ||$ tells us the norm of the change in the approximation. This is the basis of our error indicator. These principles can be extended not only to linear bases over bisected triangles, but to any finite element space, even rectangles, 3-D, etc. We will consider in detail the spaces of C^0 p^{th} degree polynomials over bisected triangles, starting with the linear case.

The division of a pair of triangles as in Fig. 2.8 by newest vertex bisection corresponds to the addition of one new basis function. We let v_i and ϕ_i, i=1,2,3 and 4, be the vertices of the triangles and corresponding hierarchical basis functions, and v_5 and ϕ_5 be the new vertex and basis function. We wish to approximate how much change would occur if this division were to be performed. To do this, we approximate the coefficient of ϕ_5, α_5, by assuming that α_i, the coefficients of ϕ_i, remain unchanged for i=1,2,3 and 4. Then

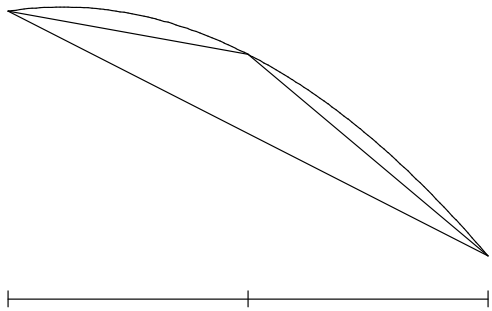


Fig. 2.7. A smooth function and its 2-node and 3-node linear interpolants

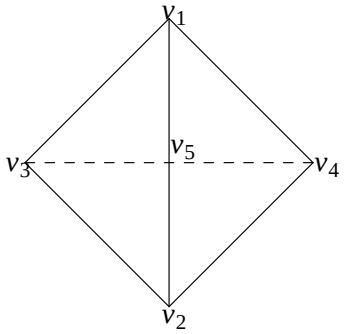


Fig. 2.8 Triangle pair for error indicator

$$\alpha_5 = ((f, \phi_5) - \sum_{i=1}^{4} \alpha_i < \phi_i, \phi_5 >)/ < \phi_5, \phi_5 > \tag{2.1}$$

where $< \cdot, \cdot >$ and $(\cdot, \cdot)$ are the usual inner products used to obtain the stiffness matrix and load vector, respectively, and f is the right hand side of the differential equation. This corresponds to one step of a Gauss-Seidel iteration for the linear system we would have if we divided this pair of triangles. If $|| \cdot ||$ is the energy norm defined by $|| u ||^2 = < u, u >$, $|| \alpha_5\phi_5 ||^2$ is the amount by which the square of the energy norm of the error is reduced by adding $\alpha_5\phi_5$ to the approximate solution, so we use $|| \alpha_5\phi_5 ||$ as the error indicator for this pair of triangles.

We comment that one should not be concerned about the number of operations required to compute these inner products since these are precisely the values we need to define the new row of the stiffness matrix and load vector if this pair of triangles is actually divided. These values can be stored and later copied into the matrix and right hand side when the triangle division occurs. We can also store α_5 to be used for the first approximation of the solution at v_5.

We assumed above that we are computing an error indicator for a compatibly divisible pair of triangles. We must also have error indicators for triangles whose base is on the boundary and triangles that are not compatibly divisible. Boundary triangles are treated conceptually the same: α_5 is determined by a Gauss-Seidel relaxation of the (boundary condition) equation that would be added to the linear system if this triangle were divided. In the case of Dirichlet boundary conditions, α_5 is simply the difference between the boundary condition and approximate solution at the prospective new vertex. When one has a triangle that is not compatibly divisible, such as triangle $v_1v_2v_3$ in Fig. 2.9, we introduce the vertex $\hat{v}_4$ (the vertex which must be added before v_5) with corresponding basis function $\hat{\phi}_4$ and coefficient $\hat{\alpha}_4 = (\alpha_1 + \alpha_4)/2$ and replace α_4 and ϕ_4 by $\hat{\alpha}_4$ and $\hat{\phi}_4$ in Eqn. 2.1.

The extension of this error indicator to C^0 p^{th} degree polynomials is straight forward. The only difference is that now there are p^2 new basis functions rather than just one, so the amount of change depends on more than one basis function. But we still have a hierarchical basis whose coefficients represent change, and in principle the error indicator is computed the same way. Now Eqn. 2.1 is replaced by a linear system of p^2 equations in p^2 unknowns. This system can be solved by Cholesky decomposition in an acceptable number of operations, and we can compute

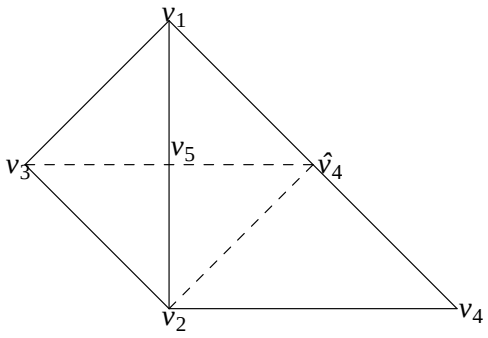


Fig. 2.9 Triangle pair for error indicator when not compatibly divisible

$|| \Sigma \alpha_i \phi_i ||$ where the sum is over the p^2 new nodes associated with the prospective new vertex. It can be seen that this error indicator can be further extended to any finite element space with a hierarchical basis.

The way in which we compute our error indicator presents a very interesting interpretation of the role of the error indicator. We could imagine a situation in which we have an *infinitely* refined uniform grid with the hierarchical basis. Recall that, with the hierarchical basis, the coefficients represent how much *change* in the solution occurs by including the corresponding basis functions. The process of adaptive refinement is now one of discarding, from our infinite number of basis functions, those whose coefficients are very close to zero, i.e., those basis functions that do not make a significant contribution to the solution. The computation of our error indicator is a form of relaxation for some of the basis functions that have not yet been added to the finite dimensional subspace. Thus, in effect, we are using a larger approximation space, and are ignoring those basis functions which do not make a significant contribution to the solution.

2.4 Selection of next triangle to divide

Given the error indicators for every triangle (or pairs of compatibly divisible triangles) we would ideally want to select the next triangle to divide by choosing the triangle with the largest error indicator. However, for our algorithm to use only O(N) operations it is imperative that the selection of the next triangle to divide requires only O(1) operations. This means that we do not have time to search every triangle to find one with the largest error indicator. So instead we will be satisfied to find a triangle whose error indicator is close to the largest.

Let e be the largest error indicator at the beginning of the refinement phase. We partition the triangles (only including one triangle from each pair of compatibly divisible triangles) into Q sets such that each set contains all the triangles whose error indicators fall in a certain range. Specifically, for a given $0<c<1$, a triangle is in the q^{th} set iff its error indicator is between $c^{q-1}e$ and $c^q e$ for $1 \le q \le Q-1$ and is in the Q^{th} set if its error indicator is less than $c^{Q-1}e$. The first set contains all the triangles whose error indicator is larger than ce, and we will select any one of these triangles as the next triangle to be divided.

To represent the sets, we use a doubly linked list for each set. It is then easy to do any of the following processes in O(1) operations. After a solution phase, the error indicator is computed for every triangle. At this time e is determined. The lists are then set to be empty and each triangle is inserted at either the head or tail (see below) of the appropriate list. During refinement, the head of the first list is selected as the next triangle to divide. If the first list is empty, then e is replaced by ce and all the lists are "shifted to the left" by shifting the head and the tail pointers, for example, head(1) $\leftarrow$ head(2). The Q^{th} list is left empty. When a triangle is divided it is removed from the list. After division, error indicators are computed for the new triangles and these triangles are added to the appropriate list. It is possible that the new error indicators could be larger than e. In this case e is replaced by e/c and the lists are "shifted to the right" with the $Q-1^{st}$ and Q^{th} lists merged into one. If necessary, this is repeated, but it is highly unlikely that more than one shift to the right would occur for reasonable choices of c. The error indicators for neighboring triangles are also updated, and these triangles are removed from the lists and added back to the lists.

By allowing insertion to occur at either the head or tail, we can improve the resolution of the partition, essentially doubling the number of sets. To do this, insert at the head if the error indicator is larger than the midpoint of the range and at the tail if it is smaller than the midpoint.

Note. When the discretization error converges like O($N^{-\alpha}$) and the number of nodes (or triangles) is increased by the factor f, we would expect the largest error indicator to be reduced by approximately the factor $f^{-\alpha-1/2}$. To see this, we use the result from §4.2 which says that $e^2 \approx \gamma \sum \varepsilon_i^2$ where e is the discretization error, ε_i are the error indicators, γ is some constant which depends only on the degree of the approximating polynomials, and the summation is over all error indicators. From this and the order of convergence of the discretization error we have $\varepsilon_{T+1}^2 + \varepsilon_{T+2}^2 \approx (1/2)^{2\alpha}\varepsilon_1^2$ where ε_1 is the largest error indicator, T is the number of triangles, and ε_{T+1} and ε_{T+2} are the "children" of ε_1. We also expect $\varepsilon_{T+1} \approx \varepsilon_{T+2}$, hence $\varepsilon_{T+1}^2 \approx (1/2)^{2\alpha+1}\varepsilon_1^2$. With f=2, ε_{T+1} has approximately the largest value after refinement, and we see the claimed reduction. For f=4, there would be another refinement associated with ε_{T+1} which, by the same argument, gives $\varepsilon_{3T+1}^2 \approx (1/2)^{2\alpha+1}\varepsilon_{T+1}^2 \approx (1/4)^{2\alpha+1}\varepsilon_1^2$. A similar argument holds when f is any power of 2. Interpolating between powers of 2 gives the desired result for any f. Numerical computations support this convergence of the maximum error indicator.

The selection of c depends on how close to the largest error indicator you want to be. The following should suffice. If the discretization error converges like O($N^{-\alpha}$) and the refinement phase increases the number of nodes by a factor f, then, as noted, we would expect the largest error indicator to be reduced by a factor of $f^{-\alpha-1/2}$. By using $c = f^{-\alpha-1/2}$, the first list should, over the course of the refinement phase, contain approximately the triangles that will be divided. If it is not completely emptied, we will at least have refined the most important triangles since we insert at both the head and tail. If it is emptied before refinement is complete, we will start the second list but probably will not get very far into it. With this value of c, Q=4 should be enough lists. The second list may be used, but it is highly unlikely we would get to the third list. The reason for having four lists rather than three is so that the second list remains undisturbed if we should need to shift to the right.

2.5 Adaptive refinement algorithm

Combining the material presented in this chapter into an adaptive refinement algorithm results in the rather simple looking Alg. 2.2. What we mean by "enough refinement" will be

discussed in Chapter 4. Determining which triangle to divide was considered in §2.4 and the process of dividing a triangle is given by Alg. 2.1 in §2.2. However, we were perhaps a bit under detailed in Alg. 2.1 when we said "divide the triangle pair". Much happens when a triangle pair is divided. We present the process of dividing a triangle pair in Alg. 2.3 and devote the rest of this section to a discussion of each of the steps involved.

The details of changing the grid specification depend on the particular data structures used to represent the grid, so we will not go into this in depth. Basically, one has to indicate that there is another vertex, p^2 more nodes and that two triangles have been replaced by four new triangles.

The size of the linear system is increased by the addition of p^2 new equations. The coefficients and right side for these equations are actually already available from the error indicator computation. So the addition of the new equations is just a matter of copying the values into the data structures used to represent the linear system. One may also need to do a basis change if the hierarchical basis representation was used. The process of changing bases is detailed in §3.1.

During the refinement, the discrete problem is represented with the nodal basis. Thus when a pair of triangles is divided, some of the neighboring basis functions change, and thus the equations associated with those basis functions also change. The coefficients and right side that we have before dividing the pair of triangles are actually those that come from the hierarchical basis after dividing the pair of triangles. Thus, to keep these equations in the nodal basis we need only perform a basis change as described in §3.1. There is, however, one complication that prevents it from being this simple. If we just did this from the beginning of the program until the end, the equations associated with the initial basis functions would still have quadrature errors with the order of accuracy of the initial grid, which would totally destroy the accuracy of the solution. Thus we must replace the integration for these inner products, which was performed over the two triangles divided, by a quadrature over the four new triangles. We can then perform the basis change to get the corrected old equations with a quadrature error of the correct order.

Algorithm 2.2. Adaptive refinement

```
repeat
  determine which triangle to divide
  divide the triangle
until enough refinement has occured
```

Algorithm 2.3. Divide triangle pair

```
change grid specification
add new equation(s) to the linear system
change other affected equations
block relaxation for new nodes
Gauss-Seidel point relaxation for neighboring old nodes
compute error indicators for new triangles and neighboring triangles
```

With the new linear system defined, we need a first approximation to the solution at the new nodes. There are p^2 new nodes, for p^{th} degree polynomials, with an associated $p^2 \times p^2$ symmetric positive definite submatrix (or block) on the diagonal of the matrix representing the linear system. To obtain our first solution for these new unknowns we perform a block relaxation, i.e., one step of a block Gauss-Seidel iteration. Actually, this need not be computed here. These values were computed in determining the error indicator, and could be stored then and just copied over at this time. This is the same as one step of the relaxation at the red nodes in the multigrid iteration of §3.2.

Since it is possible for several generations of triangle divisions to occur in the same area before the refinement phase is complete, it may also be necessary to improve the solution at the old nodes that neighbor the new nodes to obtain a sufficiently accurate error indicator. Thus we extend the partial Gauss-Seidel iteration to include point relaxations at the old nodes whose associated basis functions are not orthogonal to the new basis functions. There are $(p+1)^2$ of these, so this process uses only O(1) operations. This is the same as the local black relaxation in the multigrid iteration of §3.2.

Finally, we must compute the first error indicators for the new triangles. Moreover, we must recompute the error indicators for the neighboring triangles whose error indicators involve the old nodes whose solution values were changed during the point relaxation. There are O(1) of these.

CHAPTER 3
MULTIGRID SOLUTION

The multigrid method has recently established itself as perhaps the most efficient method for solving the linear systems that arise from the discretization of differential equations. The popularity of this method can be attributed to the fact that it is optimal in the sense that one multigrid iteration can reduce the norm of the error of the approximate solution of the linear system by a factor that is bounded away from 1 independent of N, the size of the linear system, while using only O(N) operations. The multigrid method was popularized by Brandt in the late 70's [13, 14] and has since been studied by many researchers. In this chapter we present and analyze a multigrid iteration suitable for the linear systems that arise from using the finite element method with low or high order bases on the triangulations generated by the adaptive refinement algorithm of Chapter 2. In the special case of linear elements, uniform grids and certain domains, our multigrid iteration is equivalent to that studied by Braess [10, 11, 12] and the MGR methods [22, 28]. In this special case it can also be presented in terms of standard relaxation and transfer operators. However, we will develop the method in terms of hierarchical bases. From this approach it will be easy for us to extend the method to nonuniform grids, more general domains and high order bases.

Bank, Dupont and Yserentant [6] have recently presented a hierarchical basis multigrid method that is similar to ours, but with two major differences. In order to guarantee that the multigrid iteration uses only O(N) operations for nonuniform grids it it necessary to restrict the amount of relaxation that is performed on each grid. Bank et al. restrict this so far that the factor by which the error is reduced is no longer independent of N, and O($\log N$) iterations are required, hence their method is suboptimal. We use a weaker restriction on how much relaxation occurs so that we obtain both O(N) operations and an apparently N-independent error reduction. Secondly, Bank et al. use regular refinement, where we use bisection refinement. With regular refinement, the hierarchical basis functions of the same level are not orthogonal, which means that the principle submatrix corresponding to the basis functions of one level is not diagonal. This necessitates the use of what they call "inner iterations" during the relaxation process to solve the principle subsystem. Essentially this means that relaxation consists of several red phases of a red-black Gauss-Seidel iteration with no black phases. With newest vertex bisection refinement and linear elements, the hierarchical basis functions of the same level *are* orthogonal, hence the red phase of red-black Gauss-Seidel solves this subsystem exactly, and we have eliminated the need for inner iterations. For high order elements the principle submatrix is block diagonal and can also be solved exactly.

We begin our presentation of the multigrid iteration by defining the hierarchical basis and examining some properties of the hierarchical matrix. We then present the relaxation and transfer operators, and finally the multigrid iteration algorithm. In most cases we will first present the method for linear finite elements, and then show how this can be extended to higher order finite elements. In the last section of this chapter we examine the convergence properties of the multigrid iteration using a new approach which uses a combination of theoretical results and numerical computations. With this approach we are able to examine the convergence rate with high order finite elements and other situations that the current theories do not cover.

3.1 Hierarchical bases

Our multigrid iteration will make use of the hierarchical basis, so we begin by defining the hierarchical basis. The use of hierarchical bases for finite elements has been considered by Zienkeiwicz et al. [39], Yserentant [37, 38], and more recently by Bank, Dupont and Yserentant [6].

The usual nodal basis, $\{\phi_i\}_{i=1}^N$, for a space of piecewise polynomials can be defined on a given grid by

$$\phi_i = \begin{cases} 1 & \text{at node } i \\ 0 & \text{at all other nodes} \end{cases}$$

In contrast, the hierarchical basis is defined using the family of nested grids, $\{T_i\}_{i=1}^L$, from the refinement process. The hierarchical basis begins with the nodal basis on the initial grid, T_1. As refinement proceeds, with each division one or more new nodes are added, and for each node we add a new basis function defined so that it has the value 1 at the new node and 0 at all other nodes, *but the existing basis functions remain unchanged*. Fig. 3.1 illustrates the nodal and hierarchical basis for the simple case of piecewise linear elements in one dimension with a 3-level grid. The *level* of a basis function is the same as the generation of the elements created when the basis function is added, so by *higher level basis functions* we mean those that are associated with smaller elements.

We can also define hierarchical bases which do not use all levels of the grid. Such a basis is defined by starting with the nodal basis for the grid T_i for some $1 \le i \le L$ and defining the higher level basis functions as above. If there are $k = L - i + 1$ levels of the grid used in this definition, we call this the *k-level hierarchical basis* (or just *k-level basis*). In this context the hierarchical basis is the L-level basis and the nodal basis is the 1-level basis. We will be concerned mainly with the nodal, hierarchical and 2-level bases, and will use the superscripts (N), (H) and (2) to indicate which basis is in use. Thus, $\phi_i^{(\mathrm{N})}$, $\phi_i^{(\mathrm{H})}$ and $\phi_i^{(2)}$ represent basis functions from the nodal, hierarchical and 2-level bases, respectively.

Any function, f, which lies in our space of piecewise polynomial functions on T_L has an expansion in terms of any of the bases. We use forms of α to denote the coefficients in this expansion. Thus we have

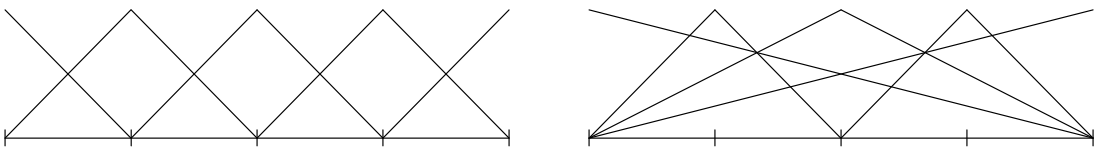


Fig. 3.1. Nodal and hierarchical bases for piecewise linear functions in one dimension

$$f = \sum_{i=1}^{N} \alpha_i^{(\mathrm{N})} \phi_i^{(\mathrm{N})} = \sum_{i=1}^{N} \alpha_i^{(\mathrm{H})} \phi_i^{(\mathrm{H})} = \sum_{i=1}^{N} \alpha_i^{(2)} \phi_i^{(2)}$$

Conversion between bases is a linear process. We use S to denote the matrix that converts the coefficient vector of the hierarchical basis to the nodal basis, thus $\alpha^{(\mathrm{N})} = S\alpha^{(\mathrm{H})}$ and $\alpha^{(\mathrm{H})} = S^{-1}\alpha^{(\mathrm{N})}$. S_l denotes the conversion from the 2-level basis to the nodal basis on an l level grid. If we order the rows and columns of S_l so that rows corresponding to nodes of the same level are grouped together and the lower level rows come first, and partition S_l into two parts corresponding to levels 1 through l-1 and level l, then S_l has the form

$$S_l = \begin{bmatrix} \mathrm{I} & 0 \\ s & \mathrm{I} \end{bmatrix}$$

and

$$S_l^{-1} = \begin{bmatrix} \mathrm{I} & 0 \\ -s & \mathrm{I} \end{bmatrix}$$

where $s_{ij} = \phi_j^{(2)}(x_i, y_i)$ and x_i and y_i are the coordinates of the i^{th} node. We see that the conversion between bases, i.e., multiplying a vector by S_l or S_l^{-1}, is an easy process. If N_{l-1} is the number of nodes in the first $l-1$ levels, we have

$$\alpha_i^{(\mathrm{N})} = \begin{cases} \alpha_i^{(2)} & \text{if node } i \text{ does not have level } l \\ \alpha_i^{(2)} + \sum_{j=1}^{N_{l-1}} \phi_j^{(2)}(x_i, y_i)\, \alpha_j^{(2)} & \text{if node } i \text{ has level } l \end{cases}$$

Most of the $\phi_j^{(2)}$ are zero at node i, so the sum is very short. In fact, with newest vertex bisection refinement and p^{th} degree piecewise polynomials, there are at most $(p+1)^2$ nonzeroes in the sum. Moreover, the value of $\phi_j^{(2)}(x_i, y_i)$ depends only on p and the relative placements of nodes i and j in the same triangle, and is independent of the triangle shapes and sizes, problem being solved, etc. Thus one can construct a small ($(p+1)^2 \times p^2$) table of these values to be used whenever a basis change is desired.

We will also need to multiply a vector by S_l^{T} and $S_l^{-\mathrm{T}}$. This is similar, except now we distribute the value of $\alpha_i^{(2)}$ over neighboring nodes when node i has level l rather than collecting values from the neighboring nodes. We summarize the process of multiplying by S_l and S_l^{T} in Alg. 3.1 and Alg. 3.2. Here s_{ij} represents $\phi_j^{(2)}(x_i, y_i)$. Multiplication by S_l^{-1} and $S_l^{-\mathrm{T}}$ are the same but with $+s_{ij}$ changed to $-s_{ij}$.

As with S, the rows and columns of the stiffness matrix, A, are ordered so that rows corresponding to nodes of the same level are grouped together, and smaller levels come first. The stiffness matrix will be called the *nodal matrix* or *hierarchical matrix* when we need to indicate which basis is in use. Basis changes for the matrix are possible using S_l. Yserentant [38] showed that we can get the hierarchical matrix from the nodal matrix by $A^{(\mathrm{H})} = S^{\mathrm{T}} A^{(\mathrm{N})} S$. Alg. 3.3 shows how to change the basis of the matrix from nodal to 2-level.

Algorithm 3.1. Multiply α by S_l

```
for each node i with level l
  for each neighbor j of i with level < l
    α_i ← α_i + s_ij α_j
  next j
next i
```

Algorithm 3.2. Multiply α by S_l^{T}

```
for each node i with level l
  for each neighbor j of i with level < l
    α_j ← α_j + s_ij α_i
  next j
next i
```

Algorithm 3.3. Replace A by $S_l^{\mathrm{T}} A S_l$

```
for each node i with level l
  for each neighbor j of i with level < l
    row j ← row j + s_ij * row i
    column j ← column j + s_ij * column i
  next j
next i
```

3.2 Relaxation operator

We are now ready to develop the components of the multigrid iteration. We begin with the relaxation, or smoothing, operator. This is first presented for linear elements and uniform grids, then extended to nonuniform grids and finally to higher order elements.

The basis of our relaxation operator is the red-black Gauss-Seidel iteration. This is among the simplest and most commonly used relaxation operators. We always do the red phase first, where the red nodes are those that are in the current grid, but not in the next coarser grid. As is common in multigrid methods, we perform ν_1 iterations of the relaxation operator before coarse grid correction and ν_2 iterations after. We allow ν_1 and ν_2 to be multiples of ½ where by half an iteration we mean only the red phase. Bank, Dupont and Yserentant [6] use $\nu_1 = \nu_2 = ½$, i.e., they perform relaxations at the red nodes only. This V-cycle is equivalent to a symmetric Gauss-Seidel iteration using the hierarchical matrix. The condition number of the hierarchical matrix is $O(L^2)$ where L is the number of refinement levels [37, 38]. Since $L \geq \log N$ and the number of Gauss-Seidel iterations depends on the condition number we see that their method requires at

least O($\log N$) iterations to reduce the error by a given factor. To overcome this difficulty we use $\nu_1 = ½$ and $\nu_2 = 1$ which adds in relaxation at the black nodes after the coarse grid correction. This is a special case of the values of ν_1 and ν_2 considered by Braess [12]. He uses $\nu_1 = \nu_2 - ½$, but performs the black phase first if ν_1 is an integer. Braess shows that by using $\nu_1 = ½$ and $\nu_2 = 1$, the V-cycle reduces the error by at least a factor of .5 independent of N for certain convex polygonal domains. In §3.4 we provide strong evidence that for a square domain and uniform grid, the error is reduced by a factor of at least .125.

While the use of red-black Gauss-Seidel with $\nu_1 = ½$ and $\nu_2 = 1$ is an effective relaxation operator for uniform grids, it presents a problem with nonuniform grids, because the number of nodes might not grow exponentially with the number of levels. Suppose that the number of new nodes in each level, n_l, grows polynomially, i.e., $n_l = O(l^{p-1})$ for some power $p \geq 1$. Then the total number of nodes in each level, N_l, satisfies $N_l = O(l^p)$. The number of operations used for relaxation with $\nu_1 = ½$ and $\nu_2 = 1$ is $\sum_{l=1}^{L} O(l^{p-1}) + \sum_{l=1}^{L} O(l^p) = O(L^{p+1}) = O(N_L^{1+1/p})$. This can be as bad as $O(N_L^2)$, which is unacceptable. To overcome this problem we must restrict the amount of relaxation performed so that the number of operations used for the relaxation on one grid is proportional to the number of red nodes in that grid, not the total number of nodes. To achieve this, Bank et al. perform the relaxation only at the red nodes. However, as noted earlier, this restriction is too strong and destroys the N-independence of the convergence. We propose the weaker restriction of performing the black phase only at black nodes that are immediate neighbors of red nodes. We call this *local black relaxation*. Since each red node has at most four black neighbors, the number of operations in the relaxation is proportional to the number of red nodes. Moreover, the black nodes at which we correct the solution value are exactly those that are most strongly affected by the change at the red nodes. The basis functions at the other black nodes are orthogonal to the basis functions at the red nodes, and hence the change there is only a second order effect through the black nodes that neighbor the red nodes. Intuitively, this may be sufficient to maintain the N-independent convergence rate. We examine this numerically in §5.3.

In order to perform the local black relaxation we must know which black nodes are neighbors of red nodes. To search all the black nodes to determine this would require more than O(N) operations. To avoid this search, we could perform the relaxation at the black neighbors of a red node immediately after the relaxation at each red node, but this would result in relaxations at many of the black nodes more than once. Instead, we construct a linked list of the black neighbors during the red relaxation. Also, to avoid duplication of the black relaxations we need a logical vector to indicate which black nodes are already on the list. This would be set to .true. during the red relaxation and set to .false. during the black relaxation so as to avoid initializing the entire vector every time. The construction of the "black list" is included in the algorithm for red relaxation.

The relaxation operator is easily extended to higher order finite elements with only one minor change. To be specific, we will consider the spaces of C^0 p^{th} degree polynomials over triangles. The difference for the higher order spaces is that the basis functions of the same level are not mutually orthogonal, as with the linear basis, so a simple red phase of red-black Gauss-Seidel does not solve the subsystem exactly. However, for the spaces we consider the submatrix is block diagonal with blocks of size p^2, so we can still solve the subsystem exactly if we are willing to solve many small systems. For high order finite elements the bisection of a pair of triangles adds one new vertex, p^2 new nodes and p^2 new basis functions. These new nodes are the red nodes in the relaxation. The p^2 basis functions associated with the new vertex are not orthogonal to each

other, but are orthogonal to all other basis functions of the same level, hence the block diagonal structure of the submatrix. Since the size of the symmetric positive definite diagonal blocks depends only on p, these small subsystems can be solved using Cholesky decomposition in a constant (w.r.t N) number of operations and we maintain the O(N) operation count for relaxation. The use of an iterative solver for these small systems would provide little, if any, reduction in the operation count, and could damage the convergence properties if not used carefully. All other aspects of the relaxation operator remain the same for these spaces of high order finite elements. We should emphasize, however, that for the local black relaxation, relaxation occurs at all black nodes for which the associated basis function is not orthogonal to any of the red basis functions. There are $(p+1)^2$ of these associated with each group of p^2 red basis functions, as illustrated in Fig. 3.2, where we show the red and black nodes associated with a new vertex in the case of cubic elements. Alg. 3.4 and Alg. 3.5 give the red and black relaxation algorithms, respectively.

3.3 Transfer operators

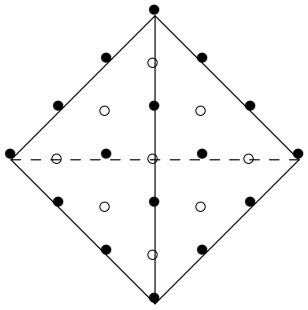


Fig. 3.2. Red nodes (○) and black nodes (•) associated with a new vertex (cubic elements)

Algorithm 3.4. Red relaxation

```
black list ← empty
for each vertex of level l
  set up and solve system for p^2 associated red nodes
  if black relaxation will follow then
    for each associated black node
      if black node is not on the black list then
        add to black list
      endif
    next black node
  endif
next vertex
```

Algorithm 3.5. Black relaxation

```
for each black node on the black list of Algorithm 3.4
  point Gauss-Seidel relaxation at this node
next node
```

The other main parts of a multigrid algorithm are the two transfer operators which are used to move between fine grids and coarse grids. The restriction operator, I_f^c, transfers the problem from the fine grid to the coarse grid and the prolongation operator, I_c^f, transfers the problem from the coarse grid to the fine grid. The transfer operators we use turn out to be those of the Galerkin approach [36]. In this approach the restriction and prolongation operators are adjoint and the coarse grid operator, A_c, is related to the fine grid operator, A_f by $A_c = I_f^c A_f I_c^f$. We will use the change between the nodal and 2-level hierarchical bases to describe the transfer processes. From this it is not clear what the transfer operators are, or even that our method *is* a multigrid method, so we also present a second derivation in terms of the usual multigrid approach to show that our method fits into the conventional multigrid framework. Since we depend only upon the basis change to define the transfers, this method applies to any finite element space with a hierarchical basis. The resulting operators are very natural and of the correct order of accuracy for the approximation space being used. We immediately present this for arbitrary spaces without first considering linear elements.

Let the nodal matrix for the linear system be

$$A = \begin{bmatrix} A_{11} & A_{12}^{\mathrm{T}} \\ A_{12} & A_{22} \end{bmatrix}$$

and the nodal solution vector and right side be $x = [x_1\ x_2]^T$ and $b = [b_1\ b_2]^T$ where the partition is such that the second part contains values corresponding to basis functions of the highest level. Let $\tilde{A}$, etc., be the corresponding entities using the 2-level basis, and let S be the matrix that converts from the 2-level basis to the nodal basis. As in §3.1, s is the lower left submatrix of S. From $\tilde{A} = S^{\mathrm{T}} A S$ and the equivalence of $Ax = b$ and $S^{\mathrm{T}} A S S^{-1} x = S^{\mathrm{T}} b$, we have $\tilde{x} = S^{-1} x$ and $\tilde{b} = S^{\mathrm{T}} b$. Since the lower level of the 2-level basis is the nodal basis of the coarse grid, $\tilde{A}_{11}$ is the nodal matrix for the coarse grid. It turns out that to obtain the problem we will solve on the coarse grid we use the lower level part of the 2-level basis fine grid problem, i.e., we extract the equations corresponding to the coarse grid nodes. This is

$$\begin{bmatrix} \tilde{A}_{11}\ \tilde{A}_{12}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix} = \begin{bmatrix} \tilde{b}_1 \end{bmatrix}$$

So the problem we solve on the coarse grid is

$$\tilde{A}_{11} \tilde{x}_1 = \tilde{b}_1 - \tilde{A}_{12}^{\mathrm{T}} \tilde{x}_2$$

We summarize this process in Alg. 3.6. Recall that the algorithms for the basis changes were given in §3.1.

Algorithm 3.6. Restriction

$A \leftarrow S^{\mathrm{T}} A S$
$x \leftarrow S^{-1} x$
$b \leftarrow S^{\mathrm{T}} b$
$b_1 \leftarrow b_1 - A_{12}^{\mathrm{T}} x_2$
coarse grid problem is $A_{11} x_1 = b_1$

From this derivation, it is not clear that the problem we solve on the coarse grid is equivalent to the standard multigrid coarse grid problem. Nor is it clear what the transfer operators are. To clarify these points, we provide a second derivation. In the usual multigrid methods, the fine grid *residual* is restricted to the coarse grid to be used as the right hand side of the linear system. The solution of this system then approximates the *error* at the coarse grid nodes and is used as a correction. We have a coarse grid problem which approximates the *solution* at the coarse grid nodes, making our method a *full approximation scheme* [36], but we will show the equivalence. In the Galerkin approach, the coarse grid matrix is given by $\mathrm{I}_{\mathrm{f}}^{\mathrm{c}} A_{\mathrm{f}} \mathrm{I}_{\mathrm{c}}^{\mathrm{f}}$, so the usual coarse grid problem is

$$(\mathrm{I}_{\mathrm{f}}^{\mathrm{c}} A_{\mathrm{f}} \mathrm{I}_{\mathrm{c}}^{\mathrm{f}})(\tilde{x}_1^{\mathrm{new}} - \tilde{x}_1^{\mathrm{old}}) = \mathrm{I}_{\mathrm{f}}^{\mathrm{c}} (b - A_{\mathrm{f}} x^{\mathrm{old}}) \tag{3.1}$$

We will begin with this and derive our coarse grid problem. First note the following relationships between the nodal and 2-level matrices and vectors:

$$\begin{aligned}
\tilde{A}_{11} &= A_{11} + s^{\mathrm{T}} A_{12} + A_{12}^{\mathrm{T}} s + s^{\mathrm{T}} A_{22} s \\
\tilde{A}_{12} &= A_{12} + A_{22} s \\
\tilde{x}_1 &= x_1 \\
\tilde{x}_2 &= x_2 - s x_1 \\
\tilde{b}_1 &= b_1 + s^{\mathrm{T}} b_2 \\
\tilde{b}_2 &= b_2
\end{aligned}$$

We see that

$$\tilde{A}_{11} = \begin{bmatrix} \mathrm{I} & s^{\mathrm{T}} \end{bmatrix} A \begin{bmatrix} \mathrm{I} \\ s \end{bmatrix}$$

so we have the transfer operators given by

$$\mathrm{I}_{\mathrm{f}}^{\mathrm{c}} = \begin{bmatrix} \mathrm{I} & s^{\mathrm{T}} \end{bmatrix} \quad \text{and} \quad I_c^f = \begin{bmatrix} I \\ s \end{bmatrix}$$

Then

$$\mathrm{I}_{\mathrm{f}}^{\mathrm{c}}(b - Ax^{\mathrm{old}}) = \begin{bmatrix} \mathrm{I} & s^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} b_1 - A_{11}x_1^{\mathrm{old}} - A_{12}^{\mathrm{T}}x_2^{\mathrm{old}} \\ b_2 - A_{12}x_1^{\mathrm{old}} - A_{22}x_2^{\mathrm{old}} \end{bmatrix}$$

$$= b_1 - A_{11}x_1^{\mathrm{old}} - A_{12}^{\mathrm{T}}x_2^{\mathrm{old}} + s^{\mathrm{T}}b_2 - s^{\mathrm{T}}A_{12}x_1^{\mathrm{old}} - s^{\mathrm{T}}A_{22}x_2^{\mathrm{old}}$$

Since we are solving for the solution, rather than the correction, we move $\tilde{A}_{11}x_1^{\mathrm{old}}$ to the right hand side of Eqn. 3.1 to obtain the coarse grid problem

$$\begin{aligned} \tilde{A}_{11}\tilde{x}_1^{\mathrm{new}} &= b_1 - A_{11}x_1^{\mathrm{old}} - A_{12}^{\mathrm{T}}x_2^{\mathrm{old}} + s^{\mathrm{T}}b_2 - s^{\mathrm{T}}A_{12}x_1^{\mathrm{old}} - s^{\mathrm{T}}A_{22}x_2^{\mathrm{old}} \\ &\quad + A_{11}\tilde{x}_1^{\mathrm{old}} + s^{\mathrm{T}}A_{12}\tilde{x}_1^{\mathrm{old}} + A_{12}^{\mathrm{T}}s\tilde{x}_1^{\mathrm{old}} + s^{\mathrm{T}}A_{22}s\tilde{x}_1^{\mathrm{old}} \\ &= b_1 + s^{\mathrm{T}}b_2 - (A_{12}^{\mathrm{T}} + s^{\mathrm{T}}A_{22})(x_2^{\mathrm{old}} - sx_1^{\mathrm{old}}) \\ &= \tilde{b}_1 - \tilde{A}_{12}^{\mathrm{T}}\tilde{x}_2^{\mathrm{old}} \end{aligned}$$

which is the same as the problem we solve on the coarse grid.

The prolongation process is simply to add back in the part of the solution due to the high level basis functions and return the system back to the nodal basis. The changes made in x_1 by solving the coarse grid problem are carried into x_2 as corrections during the basis change $x \leftarrow S\tilde{x}$. We note, however, that in our case this step is unneccessary. Only x_2 is affected by this change, and the next step will be a red relaxation which redefines x_2 without using x_2. We include this step in the algorithm because these algorithms can also be used with approximation spaces in which the red relaxation does use x_2. We summarize the prolongation processes in Alg. 3.7.

As with the restriction process, we show that this gives the same result as a standard multigrid approach. There, the error computed on the coarse grid is prolongated to the fine grid and added as a correction, i.e.,

$$x^{\mathrm{new}} = x^{\mathrm{old}} + \mathrm{I}_{\mathrm{c}}^{\mathrm{f}}(\tilde{x}_1^{\mathrm{new}} - \tilde{x}_1^{\mathrm{old}})$$

We have

$$x^{\mathrm{new}} = \begin{bmatrix} x_1^{\mathrm{old}} \\ x_2^{\mathrm{old}} \end{bmatrix} + \begin{bmatrix} \mathrm{I} \\ s \end{bmatrix}(\tilde{x}_1^{\mathrm{new}} - \tilde{x}_1^{\mathrm{old}})$$

Algorithm 3.7. Prolongation

$b_1 \leftarrow b_1 + A_{12}^{\mathrm{T}}x_2$
$b \leftarrow S^{-\mathrm{T}}b$
$x \leftarrow Sx$
$A \leftarrow S^{-\mathrm{T}}AS^{-1}$

$$= \begin{bmatrix} x_1^{\text{old}} + \tilde{x}_1^{\text{new}} - \tilde{x}_1^{\text{old}} \\ \tilde{x}_2^{\text{old}} + s\tilde{x}_1^{\text{old}} + s\tilde{x}_1^{\text{new}} - s\tilde{x}_1^{\text{old}} \end{bmatrix} = \begin{bmatrix} \tilde{x}_1^{\text{new}} \\ \tilde{x}_2^{\text{old}} + s\tilde{x}_1^{\text{new}} \end{bmatrix} = S \begin{bmatrix} \tilde{x}_1^{\text{new}} \\ \tilde{x}_2^{\text{old}} \end{bmatrix}$$

So $x = S\tilde{x}$ provides the desired prolongation of the correction.

Notice that in our restriction and prolongation algorithms we actually perform the basis changes for the matrix. This could be of some concern for two reasons: (i) it requires several operations, in fact, it is the dominant part of the operation count for high order elements, and (ii) it could introduce excessive roundoff errors, although we never experience this in the numerical computations of Chapter 5. Unfortunately, it is necessary to perform these basis changes. First, if we have a highly nonuniform grid, then we cannot store all the representations of the stiffness matrix in O(N) space. Of course, we are only using O(N) of those values, and it may be possible to find a (probably complicated) scheme to store only the useful values. Second, even if we can store all the necessary values in O(N) space, we have a problem with quadrature errors. If we were to keep the originally computed inner products of the low level basis functions, these values would contain quadrature errors on the order of the very coarsest grid, which would destroy the accuracy of the solution on the finest grid. We would have to correct these quadrature errors at the time we refine an element, just as we do with our nodal basis matrix. But to do this with every representation would require the correction of at least O(log N) values with the addition of each new basis function. So we *must* perform these basis changes in the restriction and prolongation algorithms. One way to reduce the accumulation of roundoff errors (at the expense of essentially doubling the amount of storage) would be to keep two copies of the stiffness matrix: a working copy in which the basis changes are performed, and a backup copy which is copied into the working copy before each V-cycle.

For some approximation spaces our transfer operators reduce to commonly used transfer operators when we have a uniform grid on a square domain. Thus we could consider our operators to be a generalization of those operators to other spaces, other domains, and nonuniform grids. Since the prolongation operator is the adjoint of the restriction operator, we will consider only the restriction operator. The commonly used operators we consider are all described in Stüben and Trottenberg [36]. We have two cases where, with the proper approximation space, our restriction operator is the same as a commonly used restriction operator. If the space of piecewise linear functions over triangles which were refined by *regular* division is used, the restriction operator is the same as the *7-point* operator of [36]; if the space of piecewise bilinear functions over rectangles is used, the restriction operator is the same as the *full weighting* operator. However, another important restriction operator, the *half weighting* operator, is not produced by any approximation space. And, the restriction operators from piecewise polynomial functions over triangles which were refined by *bisection* are not the same as any of the restriction operators in [36].

We now have all the necessary components for a multigrid iteration. Alg. 3.8 performs one V-cycle using $\nu_1 = \frac{1}{2}$ and $\nu_2 = 1$. L is the number of levels in the grid. The exact solve on level 1 can be performed by Cholesky decomposition or a sufficient number of Gauss-Seidel iterations.

It is not sufficient to just know that a method requires only O(N) operations; if the constant of proportionality is extremely large, say on the order of 10000, the method is practically useless. We provide in Table 3.1 the asymptotic constants of proportionality for our multigrid iteration. We provide the constants for each of the parts of the method, and for the complete V-cycle. We make the following comments:

Algorithm 3.8. V-cycle

for level = L downto 2
  red relaxation
  restriction
next level
exact solve on level 1
for level = 2 to L
  prolongation
  red relaxation
  black relaxation
next level

Table 3.1. Number of multiplications per node for the multigrid iteration

	Poisson		linear	quadratic	cubic
	mults	adds	elements	elements	elements
red relaxation	1	4	5	18	37 5/9
local black relaxation	1 - 4	4 - 16	9 - 36	14 - 37 1/4	19 2/3 - 43 1/9
right side basis change	1	2	2	4	6 2/9
solution basis change	1	2	2	4	6 2/9
matrix basis change	0	0	12	56	167 5/9
V-cycle total	6 - 9	18 - 30	49 - 76	174 - 197 1/4	448 5/9 - 472

(i) In addition to the general problem using linear, quadratic and cubic elements, we consider Poisson's equation on a square domain with linear elements. We can take advantage of the fact that every relaxation is an averaging of four neighbors to reduce the operation count considerably.

(ii) Red relaxation, right hand side basis change, and matrix basis change are each performed twice in one V-cycle; solution basis change and local black relaxation are performed once in one V-cycle.

(iii) For the general cases, the given value is the number of multiplications for each node; the number of additions is approximately the same. For the special Poisson case, the number of additions and multiplications differ, so we provide both.

(iv) The number of operations for local black relaxation depends on how many red neighbors each black node has. We provide lower and upper bounds. The best case is when all black nodes are completely surrounded by red nodes, i.e., a uniform grid. The worst case is when each black node that has a red neighbor has only one neighboring red vertex, i.e., when the red vertices are all widely separated. Since adaptive refinement usually occurs in areas, not widely separated points, we would expect to be closer to the lower bound in practice.

(v) Although there are $(p+1)^2$ black nodes and p^2 red nodes associated with each red vertex, so that there is at most $p^2 \times (p+1)^2$ nonzeroes in S for each red vertex, many of the black basis functions are zero at many of the red nodes. In fact, there are 2, 16 and 58 nonzeroes

in S for each red vertex for linear, quadratic and cubic elements, respectively. We take advantage of this in our operation count.

(vi) For red relaxation, we include the operations for factoring the principle submatrix by Cholesky decomposition. One may think that this need only be done once, but because of quadrature errors it must be done during every solution phase. It need only be done on the "downward" pass of the first V-cycle, but we have included it for every red relaxation. When omitted, the operation count for red relaxation is reduced by 4 and 12 1/9 for quadratic and cubic elements, respectively.

(vii) The special Poisson case with a uniform grid is very fast indeed. We will show in Chapter 4 that the entire solution process can be done in a number of operations equivalent to 65/63 V-cycles. We note that we can solve the system with $5N$ nonzeroes in the matrix with only about $6.19N$ multiplications. As a further indication of how fast this is, we compare our operation count with that of FFT (see, e.g. [8]). If one operation is a multiplication and an addition, FFT can solve the linear system with $2N \log N$ operations. If we assume that multiplication and addition are equivalent, then one cycle of our method uses $12N$ operations. The entire solution process uses about $12.38N$ operations. This means that our method is faster than FFT when there are more than 74 nodes. It may also be worth noting that all of these multiplications are divisions by 2 or 4.

3.4 Convergence of the multigrid iteration

In order to determine how many V-cycles are required to keep the *solution error* (the difference between our current solution and the exact solution of the discrete problem) of the same order as the *discretization error* (the difference between the exact solution of the discrete problem and the true solution of the continuous problem), it is necessary to know the factor by which one iteration reduces the error. In particular, we are interested in the worst case reduction of the error in the energy norm. By using the energy norm we will be able to relate the error-reducing power of the multigrid iteration to the convergence of the discretization error through the orthogonality of the discretization error to the approximation space. We will use this in Chapter 4. Many researchers use the spectral radius, ρ, of the iteration operator as the measure of the convergence of the multigrid iteration. This is not of interest to us because we do only one V-cycle and so the limiting behavior is not very relevant. The error reduction depends on many factors (elliptic operator, domain, grid, etc.); so as is typical of convergence analysis, we will determine the rate of convergence for the model problem of Poisson's equation on the unit square with a uniform grid. Proposition 3.1 shows how we can numerically compute the rate of convergence of the multigrid iteration. This holds for any self-adjoint elliptic operator, domain, grid and approximation space. In this section we perform the numerical computation for the model problem using linear, quadratic and cubic elements. We will consider some other cases in Chapter 5. For linear elements and the model problem, it is known that the rate of convergence for the 2-grid iteration is 1/8 for square domains, and Braess [12] showed that for certain polygonal domains the rate of convergence for the V-cycle is bounded by 1/2. We determine that for linear elements and the model problem, the rate of convergence of the V-cycle appears to be 1/8, the same as the 2-grid iteration. The convergence rate for higher order elements is slower.

Let V be the iteration operator for an l-level V-cycle, i.e., $e_{\text{new}} = Ve_{\text{old}}$ where e_{old} and e_{new} are the solution errors before and after the V-cycle, respectively. Let $||\cdot||$ be the energy norm defined by $||x||^2 = x^{\mathrm{T}}Ax$ where A is the stiffness matrix. Also let $||\cdot||$ denote the subordinate matrix norm. We define σ_l to be $||V||$, and σ, the *rate of convergence of the multigrid*

iteration, to be $\sigma = \sup_{l \geq 1} \sigma_l$. σ is a bound on the amount by which the energy norm of the error will be reduced by one V-cycle.

Proposition 3.1. Let x_0 not be orthogonal to the dominant eigenspace of $V^{\mathrm{T}}AVA^{-1}$, and $x_{i+1} = V^{\mathrm{T}}AVA^{-1}x_i$ for $i \geq 0$. Then

$$\lim_{i \to \infty} \frac{x_{i+1}^{\mathrm{T}} x_i}{x_i^{\mathrm{T}} x_i} = || V ||^2$$

Proof. We recognize the limit in the conclusion as the power method for computing $\rho(V^{\mathrm{T}}AVA^{-1})$, so we need only show that the power method converges and that $|| V ||^2 = \rho(V^{\mathrm{T}}AVA^{-1})$. Since $V^{\mathrm{T}}AVA^{-1}$ is similar to $A^{-½}V^{\mathrm{T}}AVA^{-½} = (A^{½}VA^{-½})^{\mathrm{T}}(A^{½}VA^{-½})$ which is symmetric and positive semidefinite, $V^{\mathrm{T}}AVA^{-1}$ has a complete set of eigenvectors and, since all the eigenvalues are real and nonnegative, even if the dominant eigenvalue is a multiple eigenvalue, no other eigenvalues have the same modulus. It is known (see, e.g., Stewart [34]) that under these conditions, the power method will converge to the spectral radius. That $|| V ||^2 = \rho(V^{\mathrm{T}}AVA^{-1})$ follows immediately from the definition of $|| V ||$ and the fact that A has a symmetric positive definite square root as follows:

$$|| V ||^2 = \sup_{x \neq 0} \frac{x^{\mathrm{T}}V^{\mathrm{T}}AVx}{x^{\mathrm{T}}Ax} = \sup_{x \neq 0} \frac{x^{\mathrm{T}}A^{-½}V^{\mathrm{T}}AVA^{-½}x}{x^{\mathrm{T}}x} = \rho(A^{-½}V^{\mathrm{T}}AVA^{-½})$$

Using a similarity transform, we get $|| V ||^2 = \rho(V^{\mathrm{T}}AVA^{-1})$. □

As a consequence of this proposition we have that σ_l can be computed by using the power method, provided that we can multiply a vector by $V^{\mathrm{T}}AVA^{-1}$. It is clear how one would multiply by A^{-1}, V, and A, but since we do not have the matrix V available it is not obvious how to multiply a vector by V^{T}. However, it is possible to compute this product in O(N) operations, even for high order elements, nonuniform grids, etc. We outline the process here.

A V-cycle consists of a sequence of several linear operations, thus we know V as the product of several matrices. If we let R_l, B_l, S_l, and S_l^{-1} denote the matrices that represent the operations of red relaxation, black relaxation, conversion from the 2-level hierarchical basis to the nodal basis and conversion from the nodal basis to the 2-level hierarchical basis on the l level grid, respectively, and E_1 denote the matrix representing exact solution on the coarsest grid, then

$$V = B_L R_L S_L B_{L-1} R_{L-1} S_{L-1} \cdots B_2 R_2 S_2 E_1 S_2^{-1} R_2 S_3^{-1} R_3 \cdots S_L^{-1} R_L$$

Thus

$$V^{\mathrm{T}} = R_L^{\mathrm{T}} S_L^{-\mathrm{T}} \cdots R_2^{\mathrm{T}} S_2^{-\mathrm{T}} E_1^{\mathrm{T}} S_2^{\mathrm{T}} R_2^{\mathrm{T}} B_2^{\mathrm{T}} \cdots S_L^{\mathrm{T}} R_L^{\mathrm{T}} B_{\mathrm{L}}^{\mathrm{T}}$$

and we can multiply a vector by V^{T} if we can multiply a vector by each of $R_{\mathrm{l}}^{\mathrm{T}}$, $B_{\mathrm{l}}^{\mathrm{T}}$, $S_{\mathrm{l}}^{\mathrm{T}}$, $S_{\mathrm{l}}^{-\mathrm{T}}$ and E_1^{T}. Multiplication by $S_{\mathrm{l}}^{\mathrm{T}}$ and $S_{\mathrm{l}}^{-\mathrm{T}}$ were explained in §3.1. We consider the multiplication by $R_{\mathrm{l}}^{\mathrm{T}}$ here. $B_{\mathrm{l}}^{\mathrm{T}}$ and E_1^{T} are similar. R_l is given by a matrix of the form

$$R_l = \begin{bmatrix} \mathrm{I} & 0 \\ -A_{22}^{-1}A_{12} & 0 \end{bmatrix}$$

so

$$R_l^{\mathrm{T}} = \begin{bmatrix} \mathrm{I} & -A_{12}^{\mathrm{T}}A_{22}^{-1} \\ 0 & 0 \end{bmatrix}$$

We see that to compute

$$R_l^{\mathrm{T}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

we must set

$$x_1 \leftarrow x_1 - A_{12}^{\mathrm{T}}A_{22}^{-1}x_2$$

$$x_2 \leftarrow 0$$

With a procedure to multiply by V^{T}, we can use the result of Proposition 3.1 to compute the rate of convergence σ_l. The matrix V in Proposition 3.1 can be any linear operator, so we can also use this procedure to compute the rate of convergence of the 2-grid iteration, which can be compared to the known theoretical 2-grid error reduction.

We determine the rate of convergence here for the usual model problem: Poisson's equation on the unit square with Dirichlet boundary conditions. We use a uniform refinement of the initial triangulation of Fig. 3.3.

For the model problem, the error reduction in the energy norm for the 2-grid iteration is known to be bounded by 1/8 independent of h, the grid spacing, or L, the number of levels in the grid. More precisely, for an L-level refinement of the initial triangulation in Fig. 3.3,

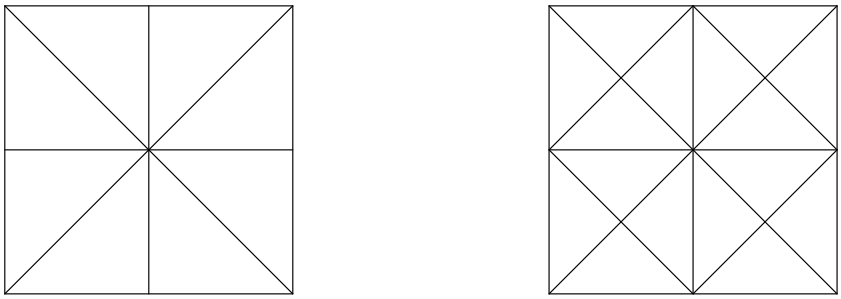


Fig. 3.3. Initial triangulation (1-level grid) and first refinement (2-level grid) used for the model problem

$$\text{error reduction by } 2-\text{grid iteration} = \begin{cases} \frac{1}{8} & \text{if } L \text{ is even} \\ \frac{1}{8}\cos^3 \pi h & \text{if } L \text{ is odd} \end{cases}$$

where $h = 2^{-\frac{L+1}{2}}$.

In Table 3.2 we present the theoretical and computed 2-grid iteration error reductions and the computed V-cycle error reduction. We see that the computed 2-grid values agree with the theoretical values. For odd numbers of levels, the V-cycle does not reduce the error by as much as the 2-grid iteration, but is still bounded by 1/8 and, in fact, the sequence is converging to 1/8.

Proposition 3.1 applies, not only to linear elements, but to high order elements as well. In Table 3.3 we show the reduction of the energy norm of the error by one V-cycle for linear, quadratic and cubic elements. We see that the rate of convergence slows as the order of the elements is increased. It is difficult to determine, from the number of levels used, the precise value of σ for quadratic and cubic elements. However, we can be quite confident that σ is approximately .31 for quadratics and .38 for cubics.

Table 3.2. Reduction of energy norm of error by one cycle for the model problem using linear elements.

level of grid	2-grid theoretical	2-grid computed	V-cycle
2	.12500	.12500	.12500
3	.04419	.04419	.07329
4	.12500	.12500	.12500
5	.09857	.09857	.10297
6	.12500	.12500	.12500
7	.11793	.11793	.11823
8	.12500	.12500	.12500
9	.12320	.12320	.12330
10	.12500	.12500	.12500
11	.12455	.12455	.12463

Table 3.3. Reduction of energy norm of error by one V-cycle for the model problem using higher order elements.

level of grid	linear elements	quadratic elements	cubic elements
2	.125	.289	.333
3	.073	.291	.349
4	.125	.297	.363
5	.103	.301	.371
6	.125	.302	.375
7	.118	.303	.377

CHAPTER 4
FULL MULTIGRID WITH ADAPTIVE REFINEMENT

The multigrid iteration provides a method for reducing the error between our approximate solution and the exact solution of the discrete problem by a factor which is bounded away from 1 independent of N using O(N) operations. From an arbitrary initial guess, however, it would take O(log$N$) iterations to reduce this error to the order of the discretization error. The *full multigrid method* is a way of obtaining a more accurate initial guess for the final grid so that the *solution error* (the difference between our approximate solution and the exact solution of the discrete system) is of the same order as the *discretization error* (the difference between the exact solution of the discrete problem and the true solution of the continuous problem) and the total number of operations is O(N). The basic idea of the full multigrid method [36] is to begin with a very coarse grid and alternately perform refinement and solution phases. At the end of each solution phase the solution error should be less than the discretization error.

For uniform grids, full multigrid is now a well established method. The refinement phase consists of one uniform refinement of the grid (divide each triangle once) to obtain a grid of one level higher. This reduces the grid spacing h by a factor of $\sqrt{2}$ or 2, depending on the type of refinement used, and increases the number of nodes by about a factor of 2 or 4, respectively. Using linear elements and a refinement which cuts h in half as an example, we see that the discretization error is cut in half and so the solution phase must perform enough cycles to cut the solution error in half (this is not quite true, as we will see in the next section) which is done by a fixed number of cycles of the multigrid iterations. Since the number of nodes grows like 4^l where l is the number of levels, the total amount of work for this is about 4/3 the amount of work done on the final grid. Thus we have used O(N) operations and have the solution error less than the discretization error.

For adaptive grids, the full multigrid method is not that well established. The problem is that the number of nodes need not grow geometrically with the number of levels. If one used the method the way it is used for uniform grids, the number of operations can be larger than O(N). In the worst case where the number of nodes is proportional to the number of levels, the operation count is O(N^2). The usual approach used to overcome this [4, 31] follows the uniform grid approach closely. The grid is refined to get one level higher, but if the number of nodes has not been increased by a factor of 2 or 4 (depending on the type of refinement used) the refinement is repeated rather than moving on to the solution phase. This approach is probably necessary because the refinement phase follows the usual approach to adaptive refinement. In that approach one determines which triangles should be refined, refines them, and enforces compatibility to obtain a grid of one level higher. While this approach does result in an O(N) algorithm, it does not flow smoothly, has unnecessary overhead in starting and ending refinements and may overshoot the target increase factor of 2 or 4.

Since the basic step of our refinement is the division of one pair of triangles and compatibility is always present, we can take a more elegant approach to the full multigrid method. The refinement phase proceeds until the number of nodes (or vertices or triangles) has been increased by exactly some given factor f. Moreover, there is no reason for f to be the factor 2 or 4 from uniform refinement, so we allow f to be any real number larger than 1. The solution phase consists of performing v V-cycles (multigrid iterations) where v is large enough to keep the solution error smaller than the discretization error. We summarize this in Alg. 4.1.

Algorithm 4.1. Full multigrid

initializations
repeat
  refine until the number of nodes has been increased by a given factor f
  apply v V-cycles
until some termination criterion is met

In the next section we consider how one determines when to switch between refinement and solution phases, i.e., what are good values for f and v. In §4.2 we present an error estimate which approximates the energy norm of the discretization error. Such an estimate could be used as the termination criterion in Alg. 4.1.

4.1 Switching between refinement and solution

Since the full multigrid method is just a process of alternating refinement and solution phases, the crucial missing element of the algorithm is a method for determining when to switch from one phase to the other. As explained earlier, we refine until the number of vertices or nodes has been increased by some factor, f, and then perform v multigrid iterations (V-cycles) for the solution phase. What we need is a way of determining reasonable values for f and v. In this section we will determine the most efficient values of f and v in terms of α, the rate of convergence of the discretization error, and σ, the rate of convergence of the multigrid iteration. We begin by finding out how much reduction in error must be obtained by the solution phase, and from this determine the most efficient value of f in terms of α, σ and v. We then give the number of operations used by the full multigrid method, and from this show how to determine v.

For uniform grids, the rate of convergence of the discretization error is usually given in terms of h, a measure of the size of the triangles. A method is said to have order 2α if the energy norm of the error decreases like $O(h^{2\alpha})$ as h gets small. For adaptively refined grids, h is not such a meaningful entity, but we can use N to measure the rate of convergence of the discretization error. For a uniform grid, $N = O(h^{-2})$ so the error decreases like $O(N^{-\alpha})$ as N gets large. The beauty of adaptive refinement is that for many problems we can maintain the $O(N^{-\alpha})$ rate of convergence even when the uniform grid does not provide the $O(h^{2\alpha})$ convergence of "nice" problems. Thus, we say that α is the *rate of convergence of the discretization error* if α is the largest value such that the discretization error is $O(N^{-\alpha})$, i.e., $|| u - u_N || \sim cN^{-\alpha}$ for some constant c where u is the true solution of the differential equation, u_N is the exact solution of the discrete problem with N nodes, and $||\cdot||$ is the energy norm. Normally, α=½, 1 and 3/2 for linear, quadratic and cubic elements, respectively. As in Chapter 3, σ is the *rate of convergence of the multigrid iteration* defined to be a bound on the factor by which the energy norm of the solution error is reduced in one V-cycle of the multigrid iteration, where the solution error is $\tilde{u}_N - u_N$ and $\tilde{u}_N$ is our approximate solution.

Theorem 4.1. Let α be the rate of convergence of the discretization error and f be the factor by which the refinement phase increases the number of nodes. Then, asymptotically, the solution error will remain less than the discretization error if the solution phase reduces the solution error

by a factor of at least $(2f^{2\alpha}-1)^{-\frac{1}{2}}$.

Proof. The true solution of the partial differential equation, u, lies in a Hilbert space $\mathbf{H}$ endowed with the energy inner product $<\cdot,\cdot>$ and the subordinate energy norm $||\cdot||$. Let $\mathbf{S}_N \subseteq \mathbf{H}$ be the space of C^0 piecewise p^{th} degree polynomials over the triangulation with N nodes, and $\mathbf{S}_{fN}$ be the space associated with the refined triangulation with fN nodes. We have $\mathbf{S}_N \subseteq \mathbf{S}_{fN}$ since the triangulation with fN nodes is a refinement of the triangulation with N nodes. Let u_N be the exact solution of the discrete problem in $\mathbf{S}_N$, i.e., u_N is the unique function in $\mathbf{S}_N$ such that $<u-u_N, w>=0 \ \forall \ w \in \mathbf{S}_N$, and let u_{fN} be the exact solution of the discrete problem in $\mathbf{S}_{fN}$. Let $\tilde{u}_N$ be our approximate solution in $\mathbf{S}_N$. We assume that $||\ \tilde{u}_N - u_N\ || \le ||\ u-u_N\ ||$. We then wish to find a $\tilde{u}_{fN} \in \mathbf{S}_{fN}$ such that $||\tilde{u}_{fN} - u_{fN}\ || \le ||\ u-u_{fN}\ ||$. Thus the solution phase is designed to keep the solution error smaller than the discretization error.

From the definition of α

$$\frac{||\ u-u_{fN}\ ||}{||\ u-u_N\ ||} \sim \frac{c(fN)^{-\alpha}}{cN^{-\alpha}} = f^{-\alpha}$$

Thus $||\ u-u_N\ || \sim f^{\alpha}||\ u-u_{fN}\ ||$.
Since $\tilde{u}_N \in \mathbf{S}_N$ and $\tilde{u}_N \in \mathbf{S}_{fN}$ we have the Pythagorean identities

$$||\ \tilde{u}_N - u_N\ ||^2 + ||\ u-u_N\ ||^2 = ||\ \tilde{u}_N - u\ ||^2$$

and

$$||\ \tilde{u}_N - u_{fN}\ ||^2 + ||\ u-u_{fN}\ ||^2 = ||\ \tilde{u}_N - u\ ||^2$$

and thus, using $||\ \tilde{u}_N - u_N\ || \le ||\ u-u_N\ ||$

$$||\ \tilde{u}_N - u_{fN}\ ||^2 = ||\ \tilde{u}_N - u_N\ ||^2 + ||\ u-u_N\ ||^2 - ||\ u-u_{fN}\ ||^2$$

$$\le 2||\ u-u_N\ ||^2 - ||\ u-u_{fN}\ ||^2$$

$$\sim 2f^{2\alpha}||\ u-u_{fN}\ ||^2 - ||\ u-u_{fN}\ ||^2$$

$$= (2f^{2\alpha}-1)||\ u-u_{fN}\ ||^2$$

Therefore, to insure that $||\ \tilde{u}_{fN} - u_{fN}\ || \le ||\ u-u_{fN}\ ||$ we must reduce the error $||\ \tilde{u}_N - u_{fN}\ ||$ by a factor of $(2f^{2\alpha}-1)^{-1/2}$. □

Corollary 4.2. For given α, σ and number of V-cycles v, f must be bounded by

$$f \le \left(\frac{\sigma^{-2v}+1}{2}\right)^{1/2\alpha}$$

Proof. Since one V-cycle reduces the solution error by a factor of σ, v iterations reduce the error by a factor of σ^v. To keep the solution error smaller than the discretization error, we must have

$$\sigma^{\nu} \le (2f^{2\alpha} - 1)^{-1/2}$$

Rearranging this inequality gives the desired result. □

Let $N_r = c_1 f^r$ be the number of nodes in the triangulation after r refinement phases, $c_2 N_r$ be the (asymptotic) number of operations used by one V-cycle on a triangulation with N_r nodes and the final triangulation be the result of R refinement phases. Then, the number of operations used by the full multigrid solution is

$$\sum_{r=1}^{R} \nu c_2 N_r = \sum_{r=1}^{R} \nu c_2 c_1 f^r = \nu c_2 c_1 \frac{f(f^R - 1)}{f - 1} \sim \nu \frac{f}{f-1} c_2 c_1 f^R = \nu \frac{f}{f-1} c_2 N_R$$

We see that for given ν, the operation count is minimized by using the largest possible f. Thus we should choose $f = ((\sigma^{-2\nu} + 1)/2)^{1/2\alpha}$. The most efficient choice of ν is then given by the ν which minimizes $\nu \frac{f}{f-1}$ with this choice of f. Clearly, this is an increasing function of ν when ν is sufficiently large. Thus, for given α and σ, one can compute $\nu \frac{f}{f-1}$ for a few small positive integer values of ν to determine the most efficient choice for f and ν. Usually, $\nu = 1$ is best. In Table 4.1 we give these values for linear, quadratic and cubic elements using the multigrid rate of convergence for the model problem as determined in §3.4. In the last column we give the value of $\nu \frac{f}{f-1}$. This represents the amount of work required with respect to one V-cycle on the finest grid, e.g., with quadratic elements the full multigrid is slightly faster than 2 V-cycles.

One final note. One can use the number of vertices or number of triangles as the quantity to be increased by the factor f. If V and T are the number of vertices and triangles, respectively, and we are using p^{th} degree piecewise polynomials, each division of a pair of triangles adds one vertex, two triangles and p^2 nodes, thus $V \sim T/2 \sim N/p^2$. So V, T and N are asymptotically linearly related and increasing any one of them by a factor f increases the others by (approximately) the same factor.

4.2 An error estimate

With any software package for the solution of partial differential equations it is desirable, though not necessary, to have a reasonable error estimate. The distinction between an error indicator and an error estimate is made by Zienkiewicz et al. [39]. An error indicator is used to determine where the grid should be refined and need not necessarily be an accurate estimate of the error. An error estimate, on the other hand, should be a good approximation of the error in some

Table 4.1. Optimal choices for ν and f for the model problem

type of elements	α	σ	ν	f	$\nu f/(f-1)$
linear	.5	.125	1	32.5	1.03
quadratic	1.0	.31	1	2.39	1.72
cubic	1.5	.38	1	1.58	2.72

norm and may be used as a termination criterion for the program, or just to give the user an idea of how accurate the solution is. Most of the available error estimates, including the one presented here, estimate the error in the energy norm. The accuracy of an error estimate is measured by the *effectivity index*, defined to be the ratio of the error estimate to the norm of the actual error. Zienkiewicz et al. present a set of requirements for a good error estimate, the most important of which are:

(i) it should be easy to compute,
(ii) the effectivity index should be greater than 1,
(iii) the effectivity index should asymptotically approach 1 as $N \to \infty$.

There is no guarantee that our error estimate will satisfy these last two conditions, and, in fact, we do not claim that this estimate is competitive (in terms of accuracy) with the best available error estimates, such as that of Bank and Weiser [5]. However, in our numerical examples of Chapter 5 we find that the effectivity index for linear elements is typically between .9 and 1.2, very respectable indeed. *It is also reasonably accurate for high order elements, whereas other error estimates are only defined for linear elements*. The most attractive feature of our error estimate is the speed with which it can be computed when used in conjunction with the adaptive refinement algorithm of Chapter 2. Given the error indicators, it requires only N/p^2 to $2N/p^2$ operations, depending on how many triangles are not compatibly divisible.

The derivation of our error estimate contains several crude approximations and should be considered to be somewhat heuristic. This results in the inability to make any claims on the accuracy of this estimate. We once again point out that the error estimate is not a crucial part of the algorithm and that our objective is just to find a reasonable estimate of the error that is very easy to compute.

Let u_N be our approximation of u using N nodes and suppose we refined the grid to fN nodes to get a new approximate solution u_{fN}. Since the error $e_{fN} = u - u_{fN}$ is orthogonal to the approximation space,

$$|| e_N ||^2 = || e_{fN} ||^2 + || u_{fN} - u_N ||^2$$

Also, since $|| e_N || = O(N^{-\alpha})$ we have $|| e_{fN} ||^2 \approx f^{-2\alpha} || e_N ||^2$, hence

$$(1 - f^{-2\alpha}) || e_N ||^2 \approx || u_{fN} - u_N ||^2$$

or

$$|| e_N ||^2 \approx \frac{f^{2\alpha}}{f^{2\alpha} - 1} || u_{fN} - u_N ||^2$$

If we were to perform a uniform refinement of our adaptive grid (divide each triangle once) then the number of nodes is approximately doubled, so we use f=2. Then if we can estimate how much change occurs in the norm of the solution, we have an error estimate. Such an estimate can be obtained from the error indicators. Each error indicator is an estimate of the change in the energy norm of the solution if the pair of triangles is divided. For triangles that are not compatibly divisible, we only count one of the triangles in the pair, and so divide the error indicator by 2. Thus, if S_1 is the set of compatibly divisible pairs of triangles and S_2 is the set of triangles that are not compatibly divisible and ε_i are the error indicators

$$|| \, u_{fN} - u_N \, ||^2 \approx \sum_{S_1} \varepsilon_i^2 + \tfrac{1}{2} \sum_{S_2} \varepsilon_i^2$$

Since $\alpha = p/2$, we then have

$$|| \, e_N \, ||^2 \approx \frac{2^p}{2^p - 1} \left(\sum_{S_1} \varepsilon_i^2 + \tfrac{1}{2} \sum_{S_2} \varepsilon_i^2 \right)$$

as our error estimate.

CHAPTER 5
NUMERICAL RESULTS

The method described in this thesis has been implemented in a FORTRAN program to solve Eqn. 1.1. In this chapter we use this program to investigate numerically some of the questions which are too difficult to answer mathematically, and to verify some of the mathematical results.

These computations were performed on a Pyramid 90x with floating point accelerator operating under the Pyramid Technology OSx 3.1 Operating System which is a dual port of AT&T Bell Laboratories' System V Release 2.0 and the University of California, Berkeley's 4.2BSD. The Pyramid Technology Optimizing FORTRAN 77 compiler was used with single precision, which has about 7 decimal digits.

For these computations, we use the following problems:

Problem 1. Laplace's equation on the L-shaped domain of Fig. 5.1(a) with the Dirichlet boundary conditions chosen so that the true solution is $r^{2/3}\sin(2\theta/3)$. Both x and y range from -1 to 1 and the reentrant corner is located at the origin. Fig. 5.1(a) shows a sample adaptively refined grid with the initial 6 triangles in bold. The solution exhibits the leading term of the singularity due to the 270° reentrant corner.

Problem 2. Laplace's equation on the hexagonal domain of Fig. 5.1(b). The domain has a slit along the positive x axis. x ranges from -1 to 1 and y from $-\sqrt{3}/2$ to $\sqrt{3}/2$. The Dirichlet boundary conditions are chosen so that the true solution is $r^{1/2}\sin(\theta/2)$. The reentrant corner is located at the origin. Fig. 5.1(b) shows a sample adaptively refined grid with the initial 6 triangles in bold. The solution exhibits the leading term of the singularity due to the 360° reentrant corner.

Problem 3. This is Problem 54 in the elliptic PDE population of Rice, et al. [26, 27]. The differential equation is

$$((1+x^2)u_x)_x + ((1+A^2)u_y)_y - (1+(8y-x-4)^2)u = f$$

on the unit square, where $A = 4y^2 + .9$. The right hand side and Dirichlet boundary conditions are chosen so that the exact solution is

$$2.25x(x-A)^2(1-D)/A^3 + 1/(1+(8y-x-4)^2)$$

where

$$B = \max\{0,\ (3-x/A)^3\}$$

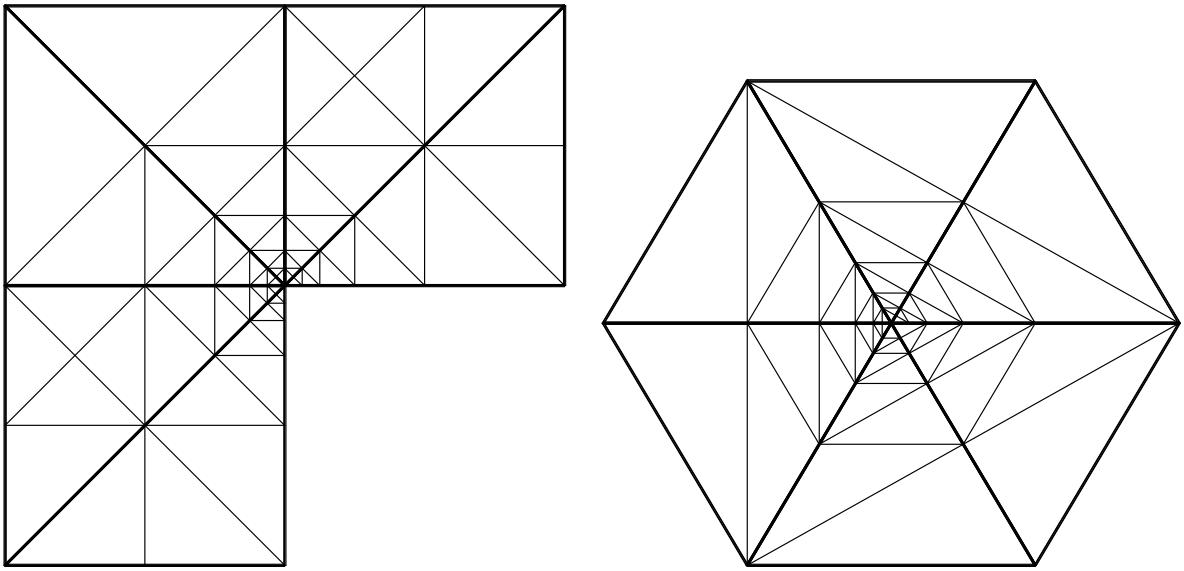
$$C = \max\{0,\ x-A\}$$

$$D = \begin{cases} 0 & \text{if } C < .02 \\ e^{-B/C} & \text{if } C \geq .02 \end{cases}$$

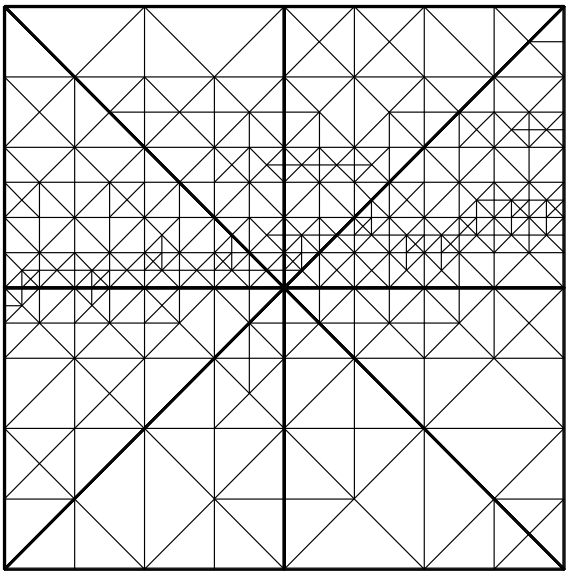
Fig. 5.1(c) shows a sample adaptively refined grid with the initial 8 triangles in bold. A contour plot of the solution can be found in [26] or [27]. The solution has a ridge in the vicinity of $y \approx .6 - .7$.

5.1 Convergence of the discretization error

With uniform refinement, the rate of convergence of the discretization error depends on the smoothness of the solution. For Problems 1 and 2, the best one can hope for is α=1/3 and 1/4,



(a) (b)



(c)

Fig. 5.1. Domains and sample grids for (a) Problem 1
(b) Problem 2 (c) Problem 3

respectively, no matter what degree polynomials are used. It is possible for adaptive refinement to

recover the optimal rate of convergence. We solve Problems 1 and 2 using linear, quadratic and cubic elements with both uniform and adaptively refined grids. For these solutions, we use one V-cycle for the solution phase, and increase the number of vertices by the factor f=4, 2.39 and 1.58 for linear, quadratic and cubic elements, respectively, except for Problem 2 where we use f=2 for quadratics. We use f=4 instead of 32.5 for linear elements to give us a sufficient number of data points for our graphs. The results are presented in Figs. 5.2, 5.3 and 5.4 for Problems 1, 2 and 3, respectively. The data points on the graphs are labeled with A, B and C for linear, quadratic and cubic elements, respectively, for the uniform grids, and 1, 2 and 3 for linear, quadratic and cubic elements, respectively, for the adaptive grids. The observed rate of convergence is given by the slope of a linear least squares fit of the data. When appropriate, we discard some of the first data points in determining the slope. These slopes are given in Tables 5.1 and 5.2.

Most of the results are as expected. For uniform grids, the rate of convergence is about 1/3 and 1/4 for Problems 1 and 2, respectively, for all three polynomial degrees. For adaptive grids, the rate of convergence is about 1/2, 1 and 3/2 for linear, quadratic and cubic elements, respectively, for both problems. For Problem 3 the rate of convergence is slightly larger than 1/2, 1 and 3/2 for linear, quadratic and cubic elements, respectively, for both uniform and adaptive grids. Although the uniform grids achieve the optimal order of convergence, the adaptive grids have a smaller constant of proportionality. We note that for the uniform grid and linear elements for Problem 3, the orientation of the grid affects the error, resulting in a bumpy graph.

In numerical experiments that compare low and high order methods with uniform grids for problems with well behaved solutions (e.g. [27]) it is usually observed that for very low accuracy it is more efficient to use linear elements, but for moderate and high accuracy the high order elements are more efficient. We observe the same result when using adaptively refined grids for Problems 1 and 2.

We also see that the convergence of the error with CPU time is of optimal order for the adaptive grids. As expected from the operation counts of §3.3, the relative placements of the graphs of linear, quadratic and cubic elements are shifted in the time vs. error graph from what their relative placements were in the nodes vs. error graph.

5.2 Effectivity index

The effectivity index of an error estimate is defined to be the ratio of the error estimate to the norm of the error. This is used as a measure of the accuracy of the error estimate. It is desirable to have an effectivity index near 1. When solving the problems in §5.1 we compute the error estimate of §4.2 after each solution phase and measure the effectivity index. We present the effectivity indices for the adaptive grids in Tables 5.3, 5.4 and 5.5. Each value corresponds to one of the data points in the graphs of Figs. 5.2, 5.3 and 5.4. We observe that the error estimate is very good, not only for linear elements, but for quadratics and cubics, too. The only deficiency is with quadratics and cubic for Problem 2, where the error is underestimated. This is easily explained. The error estimate is designed to estimate the *discretization* error, but not the *solution* error. For Problem 2 with quadratics and cubics, the solution error is almost as big as the discretization error, so the total error is underestimated. To verify this, we solved Problem 2 with quadratic and cubic elements again, this time using 2 V-cycles so that the solution error is much smaller than the discretization error. The effectivity indices for this are presented in Table 5.6. Here we see the more desirable situation of slightly overestimating the error. We also note that

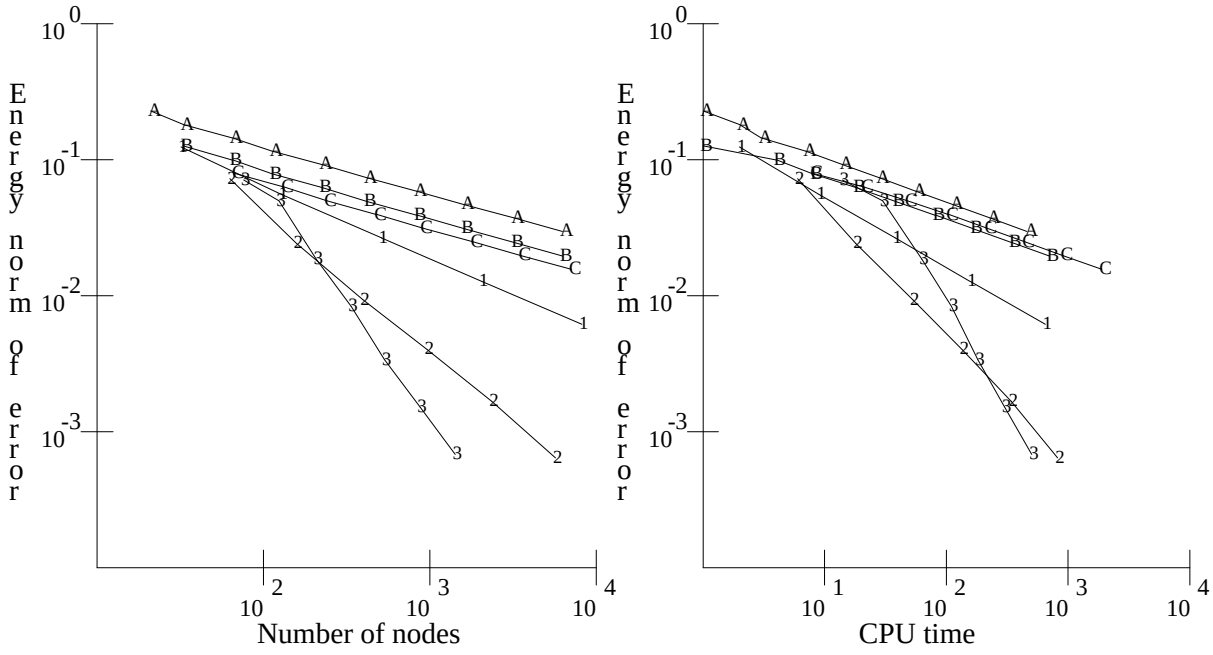


Fig. 5.2. Results for Problem 1

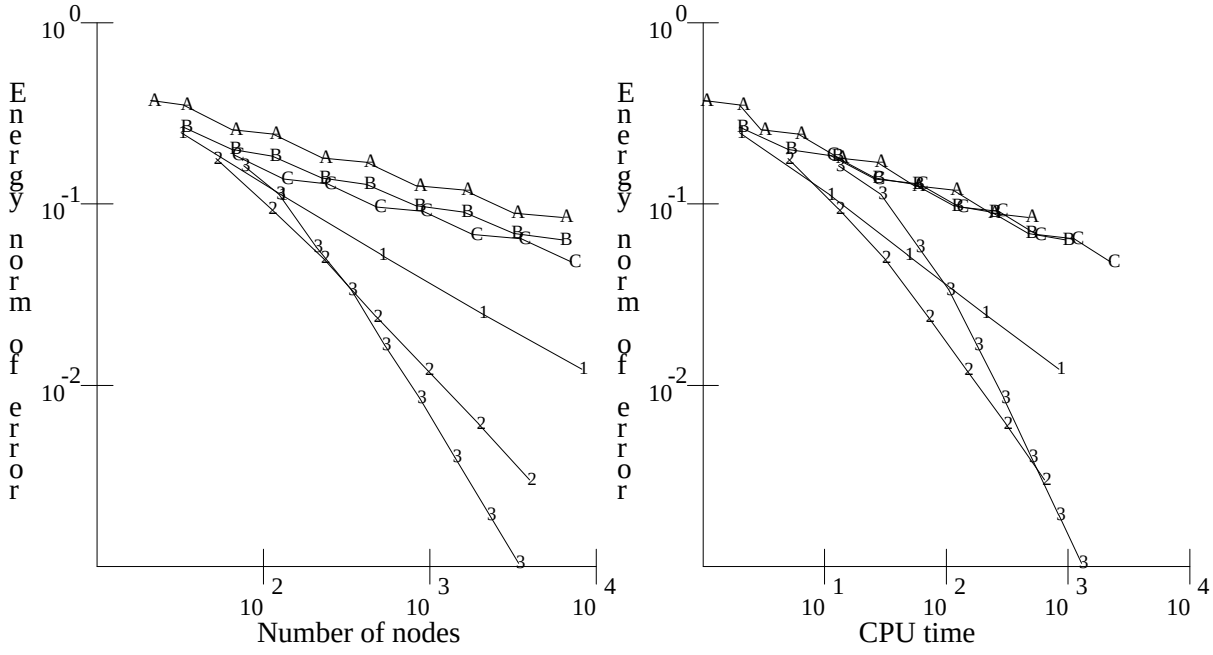


Fig. 5.3. Results for Problem 2

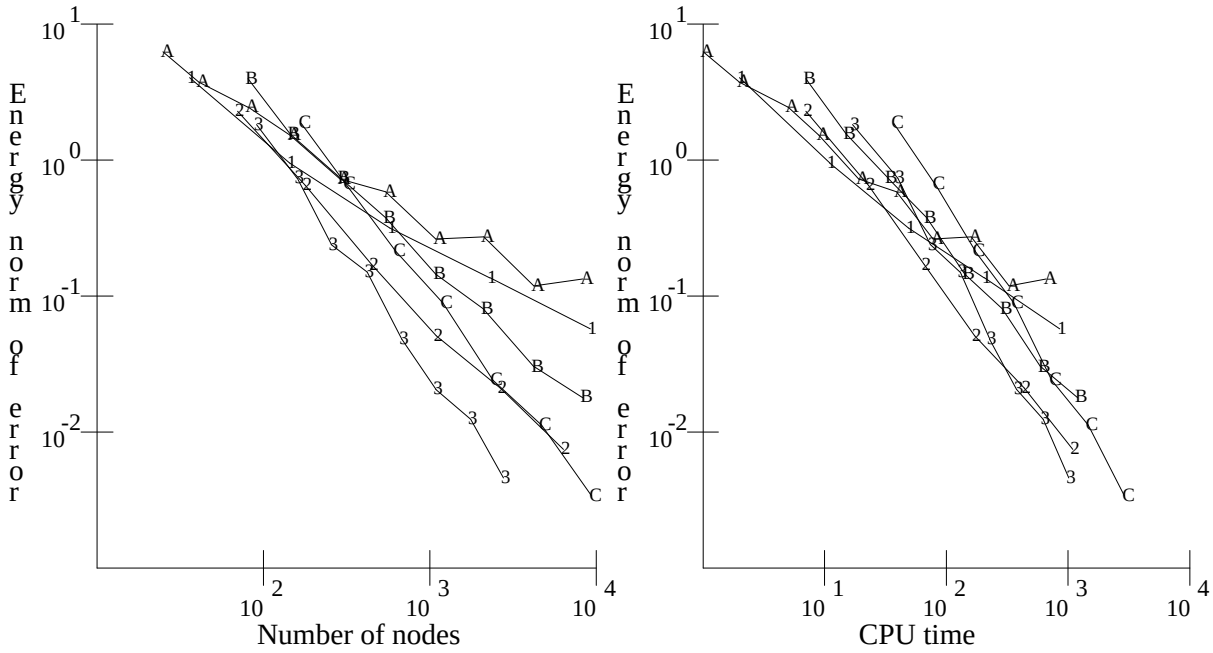


Fig. 5.4. Results for Problem 3

Table 5.1. Observed order of convergence with uniform grids			
problem	linear elements	quadratic elements	cubic elements
1	.355	.355	.349
2	.284	.273	.264
3	.697	1.131	1.555

Table 5.2. Observed order of convergence with adaptive grids			
problem	linear elements	quadratic elements	cubic elements
1	.540	1.011	1.633
2	.542	.967	1.496
3	.616	1.159	1.680

Table 5.3. Effectivity index for Problem 1

linear elements	quadratic elements	cubic elements
1.178	.800	.748
1.129	.919	.729
1.161	.968	.783
1.122	1.000	.801
1.115	1.061	1.040
	1.069	1.091
		1.198

Table 5.4. Effectivity index for Problem 2

linear elements	quadratic elements	cubic elements
.836	.723	.704
1.030	.748	.629
.999	.944	.672
.997	.853	.685
1.000	.852	.749
	.856	.822
	.863	.790
		.818
		.856

Table 5.5. Effectivity index for Problem 3

linear elements	quadratic elements	cubic elements
.711	.732	.704
1.069	.922	.760
1.263	1.065	.716
1.355	1.197	.833
1.501	1.400	1.037
	1.586	.975
		.896
		.929

Table 5.6. Effectivity index for Problem 2 with v=2	
quadratic elements	cubic elements
.737	.710
.793	.636
1.083	.697
1.078	.744
1.106	.881
1.194	.966
1.182	1.054
	1.123
	1.055

the error is overestimated in Problem 3. This is probably due to the order of convergence being slightly larger than optimal.

5.3 Convergence of the multigrid iteration

In this section we consider the effect of several factors on the rate of convergence of the multigrid iteration. We use the techniques of §3.4 to determine the rate of convergence for the L-shaped domain of Problem 1 with linear elements. This is computed using a uniform grid and also with an adaptively refined grid. For the adaptive grid we consider three forms of relaxation:

(i) full black, in which we relax at all the black nodes after solving the coarse grid problem ($\nu_1 = ½$, $\nu_2 = 1$)

(ii) local black, in which we relax at the black nodes that are neighbors of red nodes as discussed in §3.2

(iii) no black relaxation ($\nu_1 = ½$, $\nu_2 = ½$)

The results of these computations are presented in Table 5.7.

We first note that the reentrant corner has a pronounced effect on the rate of convergence. The only difference between the uniform grid here and the V-cycle in Table 3.2 is the shape of the domain, yet the rate of convergence has slowed from .125 to about .2. The use of a nonuniform grid has almost no affect on the asymptotic rate of convergence. Using local black relaxation slows the rate of convergence very slightly. The difference is small enough to ignore, especially when one considers that the full black relaxation requires more than O(N) operations for a highly nonuniform grid. When no black relaxation is performed, the rate of convergence deteriorates rapidly. We see from the data that the rate of convergence behaves like 1-O(1/log N), so it is not bounded away from 1, and, in fact, O(log N) iterations are required to reduce the error by a given constant.

5.4 Effect of f on the error

In this section we consider how f, the factor by which the number of nodes is increased during the refinement phase, affects the accuracy of the solution. We examine this for each part of the error -- the solution error and the discretization error.

Table 5.7. Rate of convergence of the multigrid iteration with the L shaped domain and linear elements.

levels	uniform grid		adaptive grid			
	nodes	σ	nodes	full black	local black	no black
3	21	.081	13	.081	.081	.188
4	33	.127	18	.070	.075	.217
5	65	.137	20	.081	.093	.256
6	113	.154	25	.084	.097	.273
7	225	.165	27	.090	.103	.289
8	417	.177	40	.086	.109	.347
9	833	.187	53	.105	.129	.419
10	1601	.196	78	.118	.143	.498
11	3201	.205	82	.120	.146	.498
12			114	.141	.162	.544
13			116	.135	.163	.544
14			155	.139	.168	.580
15			159	.136	.169	.580
16			312	.178	.200	.645
17			316	.173	.201	.645
18			443	.192	.212	.679
19			459	.186	.212	.679
20			606	.194	.217	.705

The selection of f must be such that the solution error is of the same order as the discretization error, given the error-reducing power of the solution phase. Our approach is to choose an f such that the solution error is no larger than the discretization error. Under this condition, a formula for an upper bound on f was derived in §4.1. One may choose a different bound on how large the solution error can be, such as half the discretization error, but in any case the crucial quantity is the ratio of the solution error to the discretization error. We examine how this ratio behaves as a function of f for Problem 1 using quadratic elements, v=1, and an adaptive grid with at most 2000 nodes. In Fig. 5.5 we show this ratio for values of f between 1.5 and 4. We are attempting to examine $\lim_{N \to \infty} \frac{|| \text{ soln. err. } ||}{|| \text{ disc. err. } ||}$ as a function of f. For f sufficiently large, this limit may be infinite, but we believe it is finite for $f \leq 4$. In order to have N be a power of f times the initial number of nodes, we must use a different value of N for each value of f. This is probably the cause of the irregularities in the graph, and indicates that we have not quite reached the limit. However, we expect that the correct relationship is close to this curve. We see that the relationship is slightly concave upward and that the ratio is 1 at approximately 2.8. This is reasonably close to the value 2.4 determined in §4.1. In fact, it is remarkable that it is this close since we do not have a uniform grid or the unit square, and the error reduction of the solution phase does not include the reduction made by the local relaxations during the refinement phase.

The second consideration is the effect on the discretization error. We may expect the choice of f to affect the discretization error because if f is large we are not improving the solution very often. This may affect the error indicators enough that the adaptively refined grid is no longer

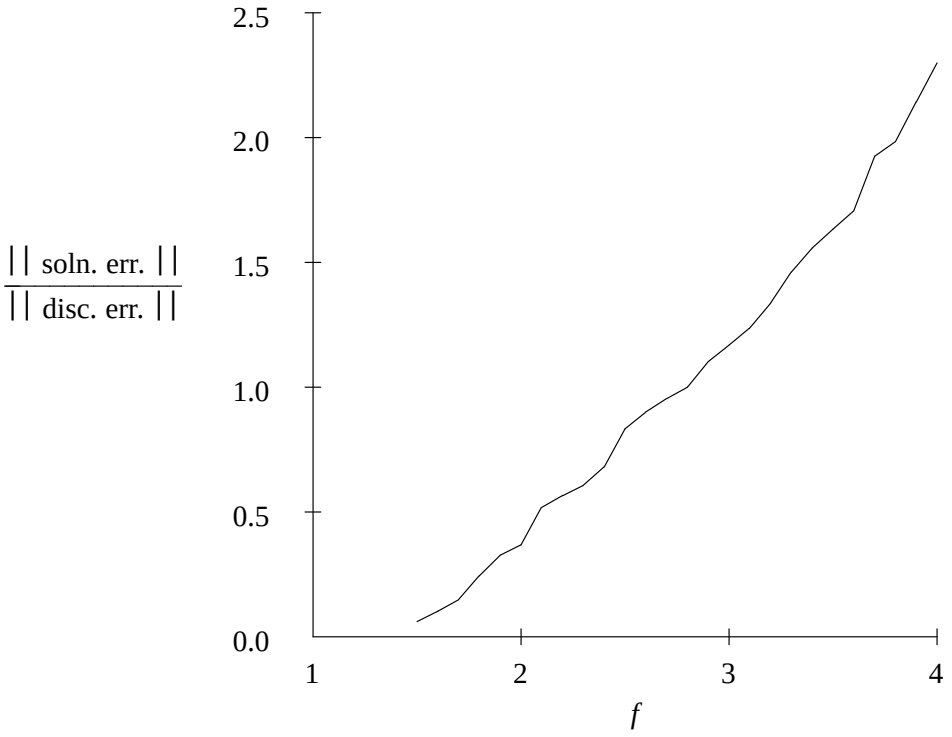


Fig. 5.5. Solution error as a function of f

optimal. We again solve Problem 1 with quadratic elements and values of f between 1.5 and 4, this time with exactly 2000 nodes. Extra V-cycles are employed at the end to determine the discretization error. We present these results in Table 5.8. We see that there are small fluctuations in the discretization error, but there is no discernible pattern. We conclude that the solution error must be much larger than the discretization error before the adaptivity is seriously affected.

Table 5.8. Discretization error as a function of f

f	error	f	error
1.5	1.46e-3	2.8	1.47e-3
1.6	1.46e-3	2.9	1.48e-3
1.7	1.49e-3	3.0	1.50e-3
1.8	1.49e-3	3.1	1.64e-3
1.9	1.51e-3	3.2	1.72e-3
2.0	1.50e-3	3.3	1.72e-3
2.1	1.58e-3	3.4	1.65e-3
2.2	1.50e-3	3.5	1.67e-3
2.3	1.50e-3	3.6	1.59e-3
2.4	1.51e-3	3.7	1.51e-3
2.5	1.47e-3	3.8	1.49e-3
2.6	1.66e-3	3.9	1.52e-3
2.7	1.49e-3	4.0	1.52e-3

CHAPTER 6
POSSIBLE FUTURE DIRECTIONS

To present our high order finite element, adaptive refinement, multigrid method we have concentrated on the solution of self-adjoint second order elliptic partial differential equations with Dirichlet boundary conditions on polygonal domains in two dimensions using C^0 p^{th} degree piecewise polynomials over triangles. Clearly the method also applies to curved domains, more general boundary conditions, and probably approximation spaces with more continuity, however the definition of the hierarchical basis for curved domains is not obvious (here we are not considering the use of isoparametric elements). When a triangle with a curved side is divided, the union of the two new triangles is not necessarily the old triangle. In this case, the domain of the old basis functions must be extended to cover the new triangles. The polynomial does not change, only the domain of definition. Now the value of the old basis function can be determined at the new nodes to define the matrix S used for basis changes, and the extension of the algorithm is clear.

The principles on which our method is based can also be used similarly for many more classes of problems and types of underlying methods. In this chapter we present some preliminary thoughts on possible directions in which future research on this approach may go.

6.1 Three dimensional problems

The first possible extension we consider is to three dimensional elliptic problems. Very little seems to have been done in either of the areas of adaptive refinement or multigrid solution for three dimensional problems. All of the aspects of our method are easily extended to more dimensions, providing the possibility of high order methods for three dimensional elliptic problems using adaptive refinement and a multigrid solution with O(N) operations. Of course, we do not actually know that the multigrid iteration will reduce the error by a factor which is independent of the number of nodes, but there is no indication that it will not.

Since the relaxation, restriction and prolongation operators are defined in terms of the hierarchical basis only, it is obvious how to define a multigrid iteration for these problems once we understand what the hierarchical basis is. Moreover, once we know the hierarchical basis, it is easy to define an error indicator for the adaptive refinement, which leaves us only with the question of how to divide tetrahedra. Once we know how to divide tetrahedra, the definition of the hierarchical basis is obvious.

The only difficulty in bisecting tetrahedra is in determining what is meant by "newest vertex" bisection. For bisection a new vertex is added at the midpoint of one of the edges, but one vertex does not uniquely determine the edge as it does with triangles. Instead, we need *two* newest vertices. For this we use the newest and second newest vertices. Then we bisect the tetrahedra by passing a plane through the newest vertex, second newest vertex and midpoint of the edge that is opposite *both* of these vertices. The newest vertices propagate in a natural manner. This is illustrated in Fig. 6.1 where we label the newest and second newest vertices with "1" and "2", respectively. In this figure (and Fig. 6.2) we use dashes for hidden lines and dots for the lines added by the bisection.

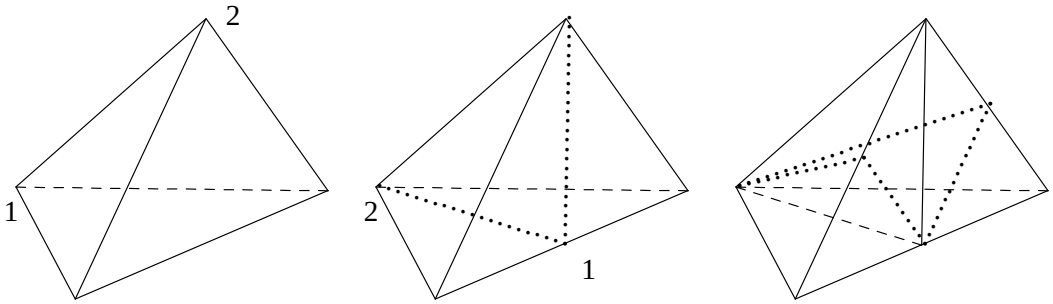


Fig. 6.1. Newest vertex bisection of tetrahedra

The edge on which the new vertex is placed is common to either four or eight tetrahedra. Thus the addition of one vertex (or basis function(s) associated with one vertex) involves simultaneously dividing *four or eight* tetrahedra, analogous to dividing *two* triangles. The division of four tetrahedra is illustrated in Fig. 6.2. We believe the other processes and properties of triangles presented in Chapter 2 have their analogues in three dimensions. One obvious property is that the number of tetrahedra shapes is finite. We see that the faces are bisected by newest vertex bisection, hence there is a finite number of face shapes. This implies that there is a finite number of tetrahedra shapes. One property that is not so obvious is the bound on the length of the recursion

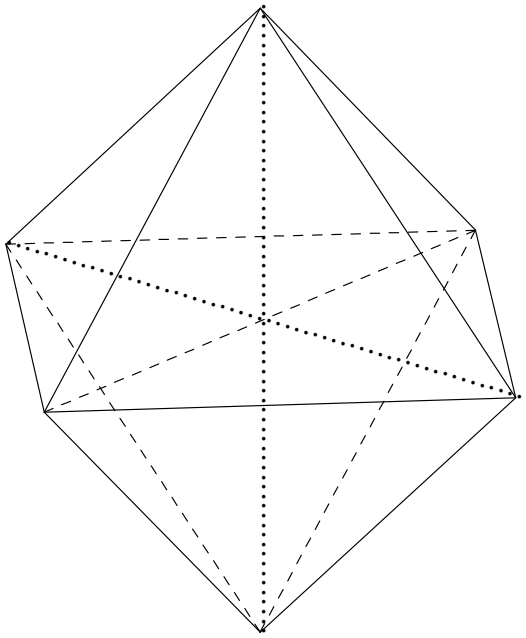


Fig. 6.2. Four tetrahedra bisected simultaneously

for maintaining compatibility. In three dimensions one must check *all* of the other tetrahedra that share the edge to be divided. Also, it may require *two* divisions of neighboring tetrahedra before the tetrahedra are compatibly divisible. This complicates the recursion somewhat in a nonrecursive language like FORTRAN. A bound on the number of tetrahedra that must be divided for compatibility remains an open question.

6.2 Time dependent problems

It should be obvious how our approach to solving elliptic problems can be used to solve parabolic problems of the form

$$u_t = Lu$$

and hyperbolic problems of the form

$$u_{tt} = Lu$$

where L is an elliptic operator.

For either of these problems, one can use any of a number of standard methods to handle the time direction. Such methods discretize the problem in time and require the solution of an elliptic PDE at each time step. One could use our multigrid method to solve the discrete systems that arise. However, we expect that due to the high accuracy of the "initial guess" provided by the solution at the previous time step, it will probably be more efficient to use a simpler solution method like ADI, SOR or conjugate gradient. The more interesting question is how to get the adaptive refinement to follow the behavior of the solution with time. What we propose as a possibility is similar to an approach suggested by Gannon [19].

The nature of our refinement is very local in the sense that the basic step is one of dividing a pair of triangles by connecting their opposing vertices through the midpoint of their common side. It is just as easy to reverse this process and "de-refine" four triangles into two triangles. In fact, we essentially do this during the multigrid iteration. Moreover, since our error indicator for potential divisions is an approximation of the coefficient of the 2-level hierarchical basis function at the new node, we can use the actual coefficients of the 2-level hierarchical basis functions of existing nodes as an indication of which nodes are not needed. Then, the process for one time step may be

```
set equations for this time step
solution phase
compute error indicators and determine coefficients for
    existing 2-level basis functions
refine grid where error indicators are large and de-refine grid where
    2-level basis coefficients are small
solution phase
```

It may be necessary to repeat this process more than once, and of course we need to maintain compatibility during the de-refinement just as we do during refinement. This just means that we would not remove refinements that are needed for compatibility. Initially, we should adapt a grid to the given initial conditions of the problem using the hierarchical coefficients of the interpolation of the initial condition as the error indicator. This process should result in grids which evolve with the solution so that the fine areas of the grid move with the "difficult" areas of the solution.

6.3 Rectangular elements

Throughout this thesis, we have concentrated on triangular elements. In principle, all aspects of our method apply to rectangular elements as well, once we have defined the hierarchical basis. The definition we gave of hierarchical bases in §3.1 applies not only to triangular elements but also to any space of functions over rectangular elements such as bilinear functions or C^1 Hermite bicubics. Gannon [19] considered the adaptive refinement of rectangular elements using a hierarchy of grids, but did not consider the hierarchical basis. Except for special situations, a locally refined rectangular grid necessarily contains incompatibilities, as in Fig. 6.3. The vertices at the incompatibilities are called *inactive* by Gannon. The inactive vertices do *not* have basis functions associated with them, whereas the active vertices do. Under the interpretation of refinement as the addition of hierarchical basis functions to the approximation space, it is clear how to perform adaptive refinement for rectangles. We simply add to the approximation space those basis functions (or groups of basis functions associated with a vertex) which have the largest hierarchical coefficients, but three points must be kept in mind to make sure that the resulting grid makes sense.

(i) You can add the basis function(s) associated with a vertex at the *center* of a rectangle *without* adding the basis functions associated with the vertices at the center of the *sides* of the rectangle. This creates inactive vertices.

(ii) If you add the basis functions associated with the vertices at the center of two *adjacent* rectangles, then you *must* also add the basis function(s) associated with the vertex at the center of the common side, since this vertex now becomes active.

(iii) If you add the basis function(s) associated with the vertex at the center of a *side* of a rectangle, then you *must* also add the basis functions associated with vertices at the centers of *both* neighboring rectangles, since both rectangles must be refined to create this active vertex.

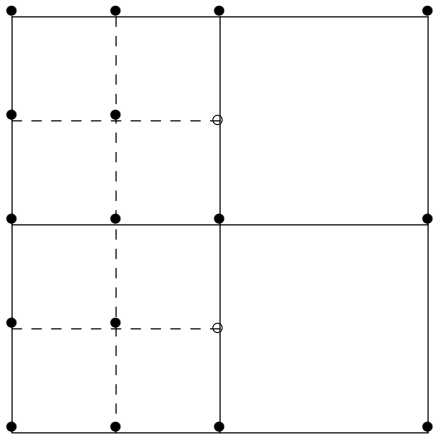


Fig. 6.3. Active (•) and inactive (○) vertices

The rest of the extension of our method to these spaces is straightforward since our method is defined in terms of the hierarchical basis only.

6.4 Collocation

Another method for determining a finite element approximation is the method of *collocation* (see, e.g., [8], [21], or [35]). For second order elliptic partial differential equations, the space of C^1 Hermite bicubics over rectangular elements is usually used. The method of collocation requires fewer operations than the Galerkin method to generate the discretization matrix and can be applied to elliptic problems not having a minimum principle. Other advantages and disadvantages of the collocation method are discussed by Dyksen, et al. [17]. One of the main disadvantages of collocation is that the resulting linear system is not symmetric and positive definite. As a result, the only reliable method of solution is banded Gauss elimination which cannot compete with the iterative solvers used for the linear system that the Galerkin method produces. Additionally, it has not been known how to do adaptive refinement for collocation with hermite bicubics because of the tensor product nature of the nodal basis. It is possible that our approach can be used for collocation to overcome these problems.

The discussion of rectangular elements in §6.3 should make it clear how one can do adaptive refinement for collocation with C^1 Hermite bicubics, except for the placement of collocation points and the evaluation of the error indicator. With the distinction between active and inactive vertices, the placement of collocation points becomes clear. With a uniform grid, there are four collocation points surrounding each vertex, placed at the "Gauss points" in the four rectangles surrounding the vertex (see, e.g. [8]). These points correspond to the four basis functions associated with each vertex. With an adaptive grid, we have basis functions associated with the active vertices only, *so we surround each active vertex with four collocation points, but do not place collocation points around the inactive vertices*. We illustrate this through an example in Fig. 6.4. How to compute the error indicator is an open question, but some approximation of the

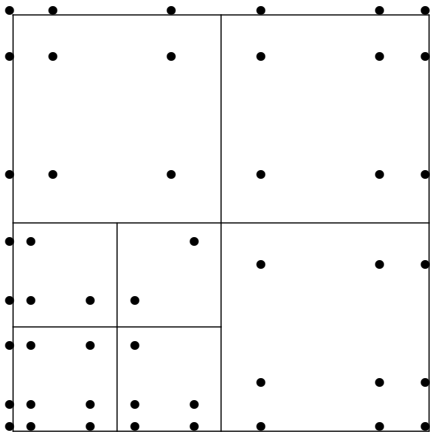


Fig. 6.4. Placement of collocation points for a nonuniform grid

hierarchical coefficient could be used.

It is also possible that the multigrid iteration can be modified for collocation. With the Galerkin method, the i^{th} equation in the coarse grid problem is basically

$$(Lu_{\text{low}}, \phi_i) = (f, \phi_i) - (Lu_{\text{high}}, \phi_i)$$

where u_{low} is the part of u that comes from the low level hierarchical basis functions, u_{high} is the part of u that comes from the high level hierarchical basis functions and $(\cdot, \cdot)$ is the L_2 inner product. By analogy, we suggest that the i^{th} equation of the coarse grid problem for collocation be

$$Lu_{\text{low}}(x_i, y_i) = f(x_i, y_i) - Lu_{\text{high}}(x_i, y_i)$$

where (x_i, y_i) is the i^{th} collocation point of the *coarse* grid. This corresponds to the interpretation of collocation as a finite element method in which the test functions are Dirac delta functions. Unfortunately, the coarse grid collocation points are not a subset of the fine grid collocation points and we would have to evaluate the hierarchical basis functions of every higher level at the coarse grid collocation points. But each collocation point lies in only one rectangle of each level; so for each collocation point there is at most 16 basis functions that must be evaluated on each higher level. Moreover, on the l^{th} level there are $O(4^l)$ collocation points and $L - l$ higher levels. Consequently, the number of basis function evaluations for one cycle on L levels is $O(\sum_{l=1}^{L} 4^l(L - l)) = O(4^L) = O(N)$. Thus it is possible to define a collocation multigrid that is analogous to our Galerkin multigrid and requires $O(N)$ operations for one V-cycle. Whether or not this multigrid iteration has an N-independent error reduction factor is an open question.

6.5 Parallelism

With the new architectures that have been developed for computing machinery, it has become more important to consider the parallelism present in numerical methods. The method we have developed has a high degree of natural parallelism, at least in principle. Perhaps the most obvious is in the relaxation. Since the submatrix for any level is diagonal (or block diagonal) the red nodes can all be relaxed in parallel (or the blocks can be done in parallel). We also note that the transfer operations take the form vector = vector + matrix*vector, which is an operation with a high degree of parallelism. The same is true of basis changes. The error indicators are completely independent of each other, so they can be computed in parallel. Finally, one can perform the division of several pairs of triangles in parallel provided that one is careful to avoid pairs that are too close together. However, it is not known how easily this algorithm maps onto any given architecture.

6.6 Other possibilities

We have presented some preliminary thoughts on how the approach we have taken to adaptive refinement and multigrid solution can be extended to other situations. There are many other worthy problems and solution methods to which the underlying principles may also apply. These include, but are not limited to, nonlinear problems, systems of PDE's, fourth order differential equations, the p version of the finite element method, and the combined h-p version of the finite element method.

REFERENCES

[1] Babuška, I. and Aziz, A. K. On the angle condition in the finite element method. *SIAM J. Num. Anal. 13* (1976), 214-226.

[2] Babuška, I. and Rheinboldt, W. Error estimates for adaptive finite element computations. *SIAM J. Num. Anal. 15* (1978), 736-754.

[3] Bank, R. E. and Sherman, A. H. The use of adaptive grid refinement for badly behaved elliptic partial differential equations. In *Advances in Computer Methods for Partial Differential Equations III*, Vichnevetsky, R. and Stepleman, R.S., Eds., IMACS, 1979, 33-39.

[4] Bank, R. E. and Sherman, A. H. An adaptive multilevel method for elliptic boundary value problems. *Computing 26* (1981), 91-105.

[5] Bank, R. E. and Weiser, A. Some a posteriori error estimators for elliptic partial differential equations. *Math. Comp. 44* (1985), 283-301.

[6] Bank, R. E., Dupont, T. F. and Yserentant, H. The hierarchical basis multigrid method. Preprint SC-87-1, Konrad-Zuse-Zentrum für Informationstechnik (1987).

[7] Becker, E. B. *Finite Elements*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[8] Birkhoff, G. and Lynch, R. E. *Numerical Solution of Elliptic Problems*, SIAM, Philadelphia, 1984.

[9] Bondy, J. A., and Murty, U. S. R. *Graph Theory with Applications*, American Elsevier Publishing Co., Inc., New York, 1976.

[10] Braess, D. The contraction number of a multigrid method for solving the Poisson equation. *Numer. Math. 37* (1981), 387-404.

[11] Braess, D. and Hackbusch, W. A new convergence proof for the multigrid method including the V-cycle. *SIAM J. Num. Anal. 20* (1983), 967-975.

[12] Braess, D. The convergence rate of a multigrid method with Gauss-Seidel relaxation for the Poisson equation. *Math. Comp. 42* (1984), 505-519.

[13] Brandt, A. Multi-level adaptive solutions to boundary-value problems. *Math. Comp. 31* (1977), 333-390.

[14] Brandt, A. Multi-level adaptive techniques (MLAT) for partial differential equations: ideas and software. In *Mathematical Software III*, Rice, J.R., Ed., Academic Press, New York, 1977, 277-318.

[15] Craig, A. W. and Zienkiewicz, O. C. A multigrid algorithm using a hierarchical finite element basis. In *Multigrid Methods for Integral and Differential Equations*, Paddon, D.J. and Holstein, H., Eds., Clarendon Press, Oxford, 1985, 301-312.

[16] De, J. P., Gago, S. R., Kelly, D. W., Zienkiewicz, O. C. and Babuška, I. A posteriori error analysis and adaptive processes in the finite element method: Part II - Adaptive mesh refinement. *Int. J. Num. Meth. Eng. 19* (1983), 1621-1656.

[17] Dyksen, W. R., Houstis, E. N., Lynch R. E. and Rice, J. R. The performance of the collocation and Galerkin methods with Hermite bicubics. *SIAM J. Num. Anal. 21* (1984), 695-715.

[18] Fried, I. Condition of finite element matrices generated from nonuniform meshes. *AIAA J. 10* (1972), 219-221.

[19] Gannon, D. B. Self adaptive methods for parabolic partial differential equations. Ph.D. Thesis, University of Illinois, Urbana, IL, 1980.

[20] Garey, M. R. and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[21] Houstis, E. N., Mitchell, W. F. and Rice, J. R. Collocation software for second-order elliptic partial differential equations. *ACM Trans. Math. Soft. 11* (1985), 379-412.

[22] Kamowitz, D. and Parter, S. On MGR[*v*] multigrid methods. *SIAM J. Num. Anal. 24* (1987), 366-381.

[23] Kelly, D. W., De, J. P., Gago, S. R., Zienkiewicz, O. C. and Babuška, I. A posteriori error analysis and adaptive processes in the finite element method: Part I - Error analysis. *Int. J. Num. Meth. Eng. 19* (1983), 1593-1619.

[24] Mitchell, W. F. A comparison of adaptive refinement techniques for elliptic problems. Report no. UIUCDCS-R-1375, Department of Computer Science, University of Illinois, Urbana, IL (1987) (to appear in *ACM Trans. Math. Soft.*).

[25] Rheinboldt, W. C. On a theory of mesh-refinement processes. *SIAM J. Num. Anal. 17* (1980), 766-778.

[26] Rice, J. R., Houstis, E. N. and Dyksen, W. R. A population of linear, second order elliptic partial differential equations on rectangular domains. *Math. Comp. 36* (1981), 475-484.

[27] Rice, J. R. and Boisvert, R. F. *Solving Elliptic Problems Using ELLPACK*, Springer-Verlag, New York, 1985.

[28] Ries, M., Trottenberg, U. and Winter, G. A note on MGR methods. *Lin. Alg. App. 49* (1983), 1-26.

[29] Rivara, M. C. Mesh refinement processes based on the generalized bisection of simplices. *SIAM J. Num. Anal. 21* (1984), 604-613.

[30] Rivara, M. C. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *Int. J. Num. Meth. Eng. 20* (1984),

745-756.

[31] Rivara, M. C. Design and data structure of fully adaptive, multigrid, finite-element software. *ACM Trans. Math. Soft. 10* (1984), 242-264.

[32] Sewell, E. G. Automatic generation of triangulations for piecewise polynomial approximation. Ph.D. Thesis, Purdue University, West Lafayette, IN, 1972.

[33] Sewell, E. G. A finite element program with automatic user-controlled mesh grading. In *Advances in Computer Methods for Partial Differential Equations III*, Vichnevetsky, R. and Stepleman, R.S., Eds., IMACS, 1979, 8-10.

[34] Stewart, G. W. *Introduction to Matrix Computations*, Academic Press, New York, 1973.

[35] Strang, G. and Fix, G. S. *An analysis of the finite element method*, Prentice-Hall, Englewood Heights, New Jersey, 1973.

[36] Stüben, K. and Trottenberg, U. Multigrid methods: Fundamental algorithms, model problem analysis and applications. In *Multigrid Methods*, Hackbusch, W. and Trottenberg, U., Eds., Springer-Verlag, Berlin, 1982, 1-176.

[37] Yserentant, H. Hierarchical bases give conjugate gradient type methods a multigrid speed of convergence. *Appl. Math. and Comp. 19* (1986), 347-358.

[38] Yserentant, H. On the multi-level splitting of finite element spaces. *Numer. Math. 49* (1986), 379-412.

[39] Zienkiewicz, O. C., Kelly, D. W., Gago, J. and Babuška, I. Hierarchical finite element approaches, error estimates and adaptive refinement. In *The Mathematics of Finite Elements and Applications IV*, Whiteman, J.R. Ed., Academic Press, New York, 1982, 313-346.