COMPUTER IMPLEMENTATION OF THE FINITE ELEMENT METHOD

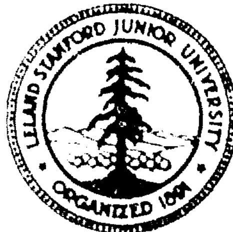
BY

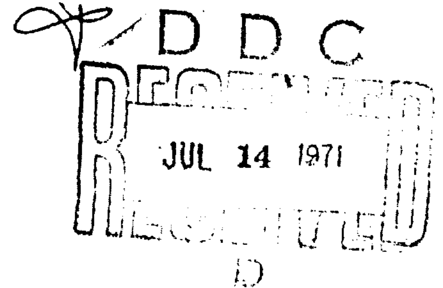
J. ALAN GEORGE

STAN-CS-71-208
FEBRUARY, 1971

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
Springfield, Va. 22151

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY





BEST AVAILABLE COPY

228

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Stanford University	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

COMPUTER IMPLEMENTATION OF THE FINITE ELEMENT METHOD

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name)

J. Alan George

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
February 1971	222	82

8a. CONTRACT OR GRANT NO. b. PROJECT NO. ONR - N-00014-67-A-0112-0029 NR 044-211 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) STAN-CS-71-208 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Office of Naval Research

13. ABSTRACT

A detailed study of the implementation of finite element methods for solving two-dimensional elliptic partial differential equations is presented. Generation and storage schemes for triangular meshes are considered, and the use of irregular meshes for finite element methods is shown to be relatively inexpensive in terms of storage. We demonstrate that much of the manipulation of the basis functions necessary in the derivation of the approximate equations can be done semi-symbolically rather than numerically as is usually done. Ordering algorithms, compact storage schemes, and efficient implementation of elimination methods are studied in connection with sparse systems of finite element equations. A Fortran code is included for the finite element solution of a class of elliptic boundary value problems, and numerical solutions of several problems are presented. Comparisons among different finite element methods, and between finite element methods and their competitors are included.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6801

Unclassified

Security Classification

Unclassified
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
partial differential equations						
sparse matrices						
automatic mesh generation						

DD FORM 1 NOV 65 1473 (BACK)
(PAGE 2)

Unclassified
Security Classification

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my thesis advisor, Professor George Forsythe, for suggesting the topic of this thesis and providing guidance and encouragement during its preparation. I also wish to thank the reading committee for their constructive criticism, and in particular Dr. Fred Dorr who supplied much sound advice and consultation during the last six months of this work.

I welcome this opportunity to thank Professor Gene Golub not only for his academic help but for his friendship, which, in numerous ways, helped make life at Stanford for my wife, Doris, and me generally more enjoyable. I also wish to thank Michael Saunders, my friend and fellow student, for many stimulating and helpful discussions.

Finally, I thank Phyllis Winkler and Laura Staggers for their typing.

This research was supported by the Office of Naval Research, the IBM Corporation, and the Atomic Energy Commission.

Table of Contents

page

CHAPTER 1: INTRODUCTION

1.1 Aims of the Thesis 1

1.2 The Variational Principle and a Brief Discussion of Ritz Methods 4

1.3 Essential Characteristics of Finite Element Methods 7

1.4 Tensor Product Spaces 14

1.5 Survey of the Thesis and Summary of Conclusions 16

CHAPTER 2: GENERATION AND STORAGE OF TWO-DIMENSIONAL TRIANGULAR MESHES

2.1 Introduction 21

2.2 Review of Previous Work on Mesh Generation 23

2.3 A Semi-Automatic Mesh Generation Scheme 30

2.4 An Automatic Two-Dimensional Domain Triangulator 34

2.5 A Storage Scheme for Finite Element Meshes and Associated Boundary Data 46

CHAPTER 3: GENERATION OF FINITE ELEMENT EQUATIONS

3.1 Introduction 52

3.2 Construction of Interpolating Polynomials 55

3.3 Generation of the Equations 66

3.4 Assembly of the Equations 77

3.5 Inclusion of Singular Functions in the Basis 81

CHAPTER 4: SOLUTION OF FINITE ELEMENT EQUATIONS

4.1 Introduction and Notation 83

4.2 Compact Storage Schemes for Sparse Matrices 92

4.3 Node Ordering for a Small Bandwidth 99

Page

4.4 Node Ordering to Reduce $|Pr(A)|$ 105

4.5 Some Experiments with Ordering Algorithms 108

4.6 The Value of N_A^Z for Arbitrary Elements and Triangular or Quadrilateral Meshes 116

4.7 Analysis of Storage and Computational Requirements for a Model Problem 124

4.8 Miscellaneous Topics and Concluding Remarks 130

CHAPTER 5: FINITE ELEMENT SOLUTIONS TO SOME SELECTED PROBLEMS

5.1 Introduction 133

5.2 The L-Shaped Membrane Eigenvalue Problem 134

5.3 Eigenvalues of Rhombical Domains 140

5.4 A Dirichlet Problem 150

REFERENCES 154

APPENDIX A: SOME REPRESENTATIVE TRIANGULAR ELEMENTS 162

APPENDIX B: O/S 360 FORTRAN CODE FOR FINITE ELEMENT METHODS 164

APPENDIX C: SAMPLE DECK SET-UPS AND RUNS 209

CHAPTER 1

INTRODUCTION

1. Aims of the Thesis

Our main goal in this thesis is a detailed study of the implementation of finite element methods for solving linear elliptic partial differential equations in two dimensions. Our study is restricted to problems which can be formulated as finding the stationary values of a quadratic integral over a given class of functions. Thus, we consider inhomogeneous second order elliptic boundary value problems in the plane which are either formulated as least squares problems or can be placed in variational form. In the text we consider equations with variable coefficients and problems involving boundary integrals, although the Fortran code we actually present can handle a less general class of integrals. However, the majority of the program would remain unchanged for more general problems.

Our viewpoint will not be that of a person who wishes to solve a specific problem. Instead, we will adopt the attitude of one who must provide a general program which is efficient, easy to use, and applicable to a reasonably large subclass of two dimensional linear elliptic boundary value problems. Thus, the capability of handling odd-shaped domains and general (non-Dirichlet) boundary conditions in a uniform manner will be important. Our study will include the problems of mesh generation and the solution of the sparse systems of finite element equations, as well as the actual generation of those equations.

We will also be interested in the performance of finite element methods (for our chosen class of problems). We will evaluate them by comparing numerical solutions to selected problems obtained by different numerical methods, including among others, finite difference methods. We will also compare different finite element methods; that is, finite element methods using different bases. Our results should offer some evidence as to which numerical technique is best, although the question of what we mean by "best" is indeed very complex. Obviously, if we choose our problems carefully, almost any method can be made to look best. If we have a specific problem that must be solved many times, then it may very well be worthwhile to find the best method for that particular problem (even though the method is applicable to a rather small class of problems, and therefore unsuitable for the purposes we have set down above). For our purposes, the following questions will be of more or less equal importance in evaluating and comparing numerical methods:

(a) What accuracy is achieved for a given amount of computation?

(b) What storage is required?

(c) Does the method rely on domain shape? (For example, does it only apply for square domains, or rectangular polygons?)

(d) Does the method utilize a special technique which requires some information known only to an expert in the field? If so, can the technique be integrated into the program so that the amateur user can use the technique unassisted?

(e) How generally applicable is the method? For example, must the coefficients of the differential operator be constants or be restricted in some other way? Are normal derivative or mixed boundary conditions easily handled?

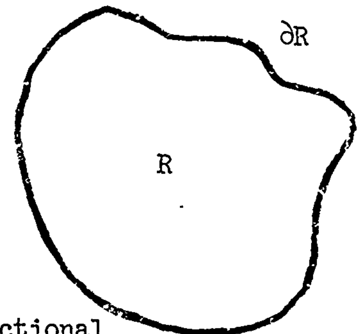
Obviously, whether some or all of these considerations are important depends upon individual needs and circumstances, but from our viewpoint of designing a general purpose program, we would like to use a method which yields a satisfactory response to all of them. Our aim is to show that finite element methods are very strong candidates.

Many of the comparisons of numerical methods which appear in the literature are made in the context of solving a specific problem, and the comparisons are often made on the basis of (a) and perhaps (b), with much less emphasis (perhaps only acknowledgement) of differences in (c), (d), and (e). Given the high cost of program development and the diminishing cost of computing power and hardware, we feel these latter considerations deserve more attention than they normally receive. Our emphasis in this thesis will be on a method's general utility rather than on its ability to solve any particular problem "better" than it has been solved before. Hence, many of our conclusions will be of a qualitative rather than quantitative nature. Nevertheless, we feel such results are important and useful. A review of the thesis and a summary of our results are found in Section 1.5.

Throughout the text "section $n_1.n_2$ " will mean section n_2 of chapter n_1 . Equations, figures and tables in section n_2 will be numbered $(n_2.1),(n_2.2),\ldots$, and references in chapter n_1 to figure $(n_2.n_3)$ also in chapter n_1 will just be $n_2.n_3$; references to figure $(n_2.n_3)$ appearing in another chapter would be written $(n_1.n_2.n_3)$.

2. The Variational Principle and a Brief Discussion of Ritz Methods

For many boundary value problems of even order it is possible to construct an integral $I[v]$ which can be formed for all functions lying in a certain class V and which takes on a minimum for precisely the function $u \in V$ which satisfies the boundary value problem. This is called the variational formulation of the problem, and usually corresponds to minimizing the energy of a physical system. The differential equation of the boundary value problem is the Euler-Lagrange equation obtained by imposing the condition that the first variation of $I[v]$ vanish [C2]. For example, let R be a two dimensional region bounded by a piecewise smooth curve ∂R . Consider the problem



$$u_{xx} + u_{yy} = f \quad \text{in } R , \tag{2.1}$$

$$u = g \quad \text{on } \partial R . \tag{2.2}$$

The solution of (2.1)-(2.2) minimizes the functional

$$I[v] = \iint_R (v_x^2 + v_y^2 + 2fv)\,dx\,dy , \tag{2.3}$$

where $v \in V$, the class of functions in $C(R \cup \partial R)$ with first derivatives in $L_2(R)$ and satisfying (2.2) [C2].

The Ritz procedure for finding an approximate solution to (2.1)-(2.2) is as follows: Let $V^N \subset V$ be a finite dimensional subspace of V spanned by the functions ψ_i , $i = 1,2,\ldots,N$. Our aim is to obtain

an approximation v^N to u by minimizing $I[v]$ for $v \in V^N$. Writing v^N in the form

$$(2.4) \qquad v^N = \sum_{k=1}^{N} \alpha_k \psi_k \quad ,$$

where α_k, $k = 1,2,\ldots,N$ are real numbers to be determined, we use (2.4) in (2.3) to obtain the quadratic function

$$(2.5) \qquad I[v^N] = \sum_{i,j=1}^{N} \left\{ \iint_R (\psi_{i,x}\psi_{j,x} + \psi_{i,y}\psi_{j,y}) dx\,dy \right\} \alpha_i \alpha_j + 2 \sum_{i=1}^{N} \left\{ \iint_R f\psi_i \,dx\,dy \right\} \alpha_i$$

$$= \alpha^T A \alpha + 2\alpha^T b \quad ,$$

where $\alpha^T = (\alpha_1, \alpha_2, \ldots, \alpha_N)$,

$$(2.6) \qquad A_{ij} = \iint_R (\psi_{i,x}\psi_{j,x} + \psi_{i,y}\psi_{j,y})\, dx\,dy \quad ,$$

and

$$(2.7) \qquad b_i = \iint_R f\psi_i \,dx\,dy \quad .$$

Using the important fact that A is symmetric, we obtain the system of equations $A\alpha = -b$ which determines the coefficients α in (2.4) yielding the minimizing $v^* \in V^N$. Under appropriate hypotheses, $v^* \to u$ as $N \to \infty$ [K3]. The importance of the finite element method is that it allows us to construct ψ_i's which satisfy these hypotheses and which also have attractive computational properties. This is taken up in the next section.

Note that for our chosen class of problems, it will always be possible to arrange that the coefficient matrix of the linear system we must solve is symmetric, since $\alpha^T A\alpha = \frac{1}{2}\alpha^T A\alpha + \frac{1}{2}\alpha^T A^T\alpha = \frac{1}{2}\alpha^T(A + A^T)\alpha = \alpha^T \tilde{A}\alpha$, where $\tilde{A}$ is obviously symmetric.

3. Essential Characteristics of Finite Element Methods

The term "finite element" appears to have originated in the early 1950's with structural engineers who regarded conventional structures as composed of a number of separate elements interconnected at node points. The concept was extended to continuous problems such as plate bending and steady-state temperature distribution, where the elements are merely subdivisions of the domain of the problem with adjacent elements having a common vertex or common side. The most common element shapes are triangles and rectangles. Our attention will be devoted almost exclusively to triangular elements in this thesis, primarily because odd shaped domains can be more easily divided into triangles than rectangles.

Finite element methods are Ritz methods which use basis functions having small support; that is, Ritz methods which make use of a so-called "local basis". In Chapter 3 we will discuss the actual procedure. At this point we simply observe that finite element methods make use of trial functions v^N (see Section 1.2) having the form

$$v^N = \sum_{k=1}^{N} \alpha_k \psi_k \quad , \tag{3.1}$$

where

(a) v^N is a piecewise polynomial on $R \cup \partial R$.

(b) v^N is a polynomial on each element.

(c) each basis function ψ_k is associated with a node point lying on a vertex, side, or interior of an element, and is non-zero only on elements containing the node. This property is depicted below:

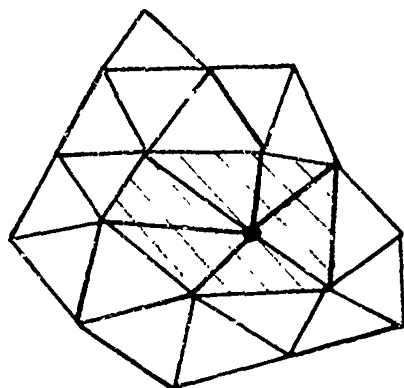


Figure 3.1-a

Support of ψ_k associated with a corner (vertex) node.

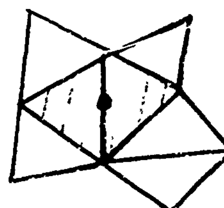


Figure 3.1-b

Support of ψ_k associated with a side node.

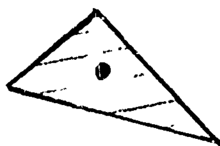


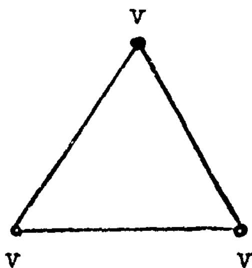
Figure 3.1-c

Support of ψ_k associated with an interior node.

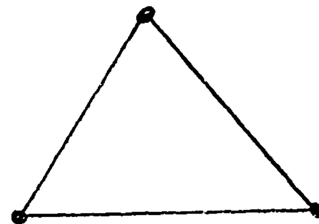
More than one basis function may be associated with a particular node, and because of the way the basis functions are chosen, the nodal parameters α_k associated with each ψ_k turn out to be the <u>value</u> or the <u>value of a derivative</u> of v^N at the corresponding node point. The choice of these nodal parameters is done on the basis of (1) the number of degrees of freedom v^N has on each element and (2) the continuity

requirements of v^N. Indeed, a common practice is not to consider the basis functions, but instead, to choose the parameters so as to uniquely characterize the polynomial on each element and at the same time to attain a desired degree of continuity across interelement boundaries.

For example, consider piecewise linear polynomials, for which v^N is a linear function on each triangle. The trial solution v^N can be uniquely characterized by its value at any three non-collinear points. By choosing these three parameters at the vertices, we can guarantee continuity along interelement boundaries. We would indicate this subspace by the <u>element stencil</u>



, or just



It is fairly easy to see that this amounts to using a "pyramid function" at each vertex node, as depicted below:

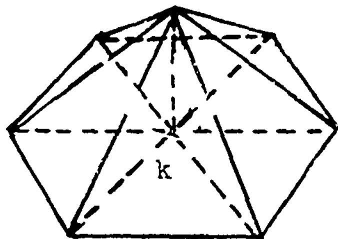
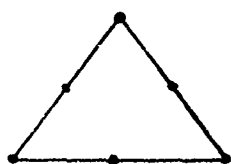
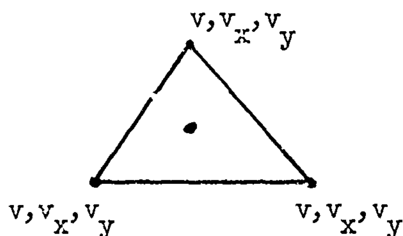


Figure 3.2 Pyramid Function ψ_k associated with node k .

Some other common stencils associated respectively with quadratic and cubic polynomials are



Quadratic



Cubic

Note that in the last example, three basis functions will be associated with each corner node, and the function associated with the interior node will be non-zero only on the triangle containing the node. A (non-exhaustive) list of stencils can be found in Appendix A.

Piecewise polynomials derived in this way are sometimes referred to as _interpolation polynomials_, since they are characterized by the values (and perhaps derivative values) that they assume at the node values. Note, however, that in our application the piecewise polynomial will not (usually) interpolate the solution of our boundary value problem.

We will not consider the important mathematical question of when (and how fast) $v^N \to u$ as $N \to \infty$. We will simply make some observations and refer to relevant sources in the literature:

(i) Because each basis function vanishes over most of the domain, the linear system that is generated is sparse. Strang [S5] emphasizes this by stating that "by a suitable choice of the trial functions ... the Galerkin equations... turn out to be difference equations". Whether we call them _finite element_ or _finite difference_ equations is largely a matter of taste; we prefer the former, and reserve the term "finite difference" for those methods based on divided difference approximations. For polynomial basis functions of low degree the two approaches sometimes yield the same equations. Our distinction is made on the _method of derivation_ rather than the end result.

(ii) The value of finite element methods will obviously depend upon how well the trial functions can approximate the true solution of our boundary value problem. This problem has been studied for general elliptic operators and tensor product approximating spaces in [B8,S6,S1] and in references contained therein. We will briefly discuss the practical advantages and disadvantages of these spaces in Section 1.4. Bramble and Zlámal [B12], Zlámal [Z4,Z5], Ženíšek [Z1], Goël [G2],

and others have proved convergence of the method and presented bounds for various elliptic operators and piecewise polynomials on triangles. Qualitatively, their results say that if the approximating subspace is admissible, and the true solution u is "smooth enough", then an increase in d (the degree of the piecewise polynomial) induces an equivalent decrease in the error bound. That is, the error bounds are of the form

$$\|u - v^N\|_q \leq \mathcal{C}\, h^{d+1-q} \|u\|_{d+1} \quad ,$$

where $\|u\|_\ell^2 = \sum_{|i| \leq \ell} \|D^i u\|_{L^2}^2$, $i = (i_1, i_2)$, $|i| = i_1 + i_2$, and

$D^i u = \dfrac{\partial^{|i|} u}{\partial x^{i_1} \partial y^{i_2}}$. Here h is the maximum length of any triangle side in the mesh, and $\mathcal{C}$ is a constant which depends upon the sharpest angle in the mesh and the polynomial basis (element) being employed. For specific details, the reader is referred to the papers mentioned above.

(iii) The condition of the finite element linear system which we obtain will obviously depend upon our choice of ψ's . Indeed, one of the problems of using the Ritz technique has been the numerical instability of the discrete problem, caused by choosing almost linearly dependent trial functions. Intuitively, we would anticipate that such problems would be much less troublesome for the finite element method because the majority of the ψ's will be orthogonal (by virtue of having disjoint support). Strang and Fix [S6] study this problem in depth for

uniform meshes by examining the condition number $H(G) = \|G^h\| \, \|G^{h^{-1}}\|$ of the Gram matrix, whose entries are the inner products of the basis elements ψ_i . They conclude that all the usual piecewise polynomial trial functions yield a stable basis, where stability means that $H(G)$ remains bounded as $h \to 0$. They show that the condition of the coefficient matrix A obtained from the application of the finite element method (using a stable basis) to a uniformly elliptic operator of order $2m$ is of the form $\mathcal{C}h^{-2m}$, where h is the mesh width and $\mathcal{C}$ depends on the choice of the basis. This result is of practical significance; for a given problem it says that as long as we use a stable basis, the condition of the coefficient matrix does not deteriorate as we increase the degree of our polynomials. Note that these results only apply for uniform meshes, and it is not known how detrimental severe grading of the mesh may be to the condition of the matrix A .

4. Tensor Product Spaces

Suppose $D_1 = [0,1]$ is divided up into a uniform mesh with grid points $(ih, i = 0,1,2,\ldots,n)$ and assume we have a basis $(\tilde{\psi}_i(x), i = 0,1,2,\ldots,n)$ on $[0,1]$, where each $\tilde{\psi}_i$ is non-zero on the interval $[i\text{-}ph, i\text{+}ph]$, with p small. Now consider the domain $D_2 = [0,1] \times [0,1]$, covered by a square mesh with grid points $((ih,jh), i = 0,1,2,\ldots,n, j = 0,1,2,\ldots,n)$. To construct a tensor product basis on D_2 we form the functions $\psi_{ij} = \tilde{\psi}_i(x)\tilde{\psi}_j(y)$, $i,j = 0,1,2,\ldots,n$. The trial function v^N, where $N = (n+1)^2$, is then given by

$$v^N = \sum_{i,j=0}^{n} \alpha_{ij}\psi_{ij} \quad . \tag{4.1}$$

The main advantage to this approach is that it is possible to obtain a relatively smooth approximation with only moderate N, since if $\tilde{\psi}_i \in \mathcal{C}^q[D_1]$, $i = 0,1,\ldots,n$, then $\psi_{ij} \in \mathcal{C}^q[D_2]$, $i,j = 0,1,\ldots,n$. This is often done by taking a spline basis for the $\tilde{\psi}_i$'s. For example, we can have $v^N \in \mathcal{C}^2(D_2)$ by using the cubic spline basis [S1]. For a specific degree of smoothness, the number of parameters (unknowns) in the problem increases as n^k for k dimensions. Of course, the reason for these properties is the fact that the interelement boundaries [which are h by h squares] are constrained to lie along coordinate axes, and this brings us to the major disadvantage of this method of basis construction.

Because our elements are squares [or perhaps rectangles -- it is easy to scale the basis functions], our domain must be restricted to be

the union of rectangles. Furthermore, it is virtually impossible to grade (i.e., subdivide) the net "locally". If a fine mesh is desired in a region of the domain, then it must be made fine in an interval in each coordinate, even though we only desire the fine mesh in the intersection of these intervals. It is fairly easy to conceive of realistic problems which would force the grid to be almost uniformly fine.

However, there is some reason for optimism regarding this geometrical problem; Bramble and Schatz [B11] and Babuska [B1] have analyzed some methods that do not require that the basis functions satisfy any boundary conditions. The basic idea is to imbed the given domain R with boundary ∂R in a larger domain $R' \supset R$, with the basis functions satisfying homogeneous boundary conditions on the boundary of R'. A boundary integral on ∂R scaled by $h^{-\gamma}$, $0 \leq \gamma < \infty$ (where h is the mesh width), is added to a least squares formulation of the problem. The boundary integral is designed so that its minimum occurs when the approximate solution satisfies the boundary conditions on ∂R. As would be expected, their error estimates depend upon the smoothness of the boundary data and the solution. They show that $\gamma = 3$ is optimal in some situations.

We have not pursued this avenue of investigation in this thesis because the approach we use to generate our basis functions allows us to fairly easily satisfy boundary conditions.

5. Review of the Thesis and Summary of Conclusions

As our title implies, the emphasis in this thesis is on implementation, and such a study leads to interesting practical problems which are seldom discussed in papers on finite element methods. Engineering articles on finite element methods are often devoted to discussing the virtues of particular elements for solving specific problems. Mathematical papers, on the other hand, are usually concerned primarily with rates of convergence of various finite element spaces. We feel our work lies between these two extremes; we are concerned with the actual implementation of finite element methods and how they compare in practice with other methods for solving elliptic boundary value problems.

In Chapter 2 we examine the problem of generation and storage of two-dimensional triangular meshes. We begin by reviewing previous work on automatic mesh generation. We then present a semi-automatic procedure for triangulation of a domain. The method requires the user to provide a gross triangulation of the domain, reflecting any desired grading. The mesh is then refined by any specified factor by the program. We feel this compromise solution, although not particularly elegant, is important for several reasons: (a) the required input for most domains is small, (b) the method can easily be adapted for use with graphical display equipment), (c) curved boundaries can be incorporated easily, (d) the net can be graded under control of the user, and (e) inter-element boundaries can be forced to lie in specific positions (along lines of material discontinuity, for example).

Chapter 2 also contains a description of a completely automatic domain triangulator. Although the algorithm cannot be considered a finished product, we have included it because we feel it represents a promising approach to automatic triangulation. It is applicable to

arbitrary simply connected domains and is designed to produce graded nets where appropriate. Some examples of meshes produced by the algorithm are presented and some further areas of research are suggested.

The final section of Chapter 2 contains an efficient storage scheme to represent arbitrary triangular meshes. Using this scheme along with some results obtained in Chapter 4, we compare the storage required for the mesh to the number of non-zero elements in the coefficient matrix. We show that except for piecewise linear polynomials, the storage required for the mesh is small compared to that required for just the non-zero elements in the coefficient matrix. We conclude that the mesh storage will seldom be an important factor in overall storage requirements in the application of finite element methods.

Chapter 3 deals in detail with the actual generation of the finite element equations. The process consists of two phases. The first is the computation of the stiffness matrices which express our integral over each element in terms of the nodal parameters used to characterize it. The second phase consists of assembling these matrices into a single large system and eliminating those parameters whose values are already specified by boundary conditions. For the first phase we describe one method for generating coefficients of the equations on each triangle. We justify our use of the approach over others by demonstrating where much of the computation and manipulation of the basis functions can be carried out symbolically, thus avoiding use of numerical (or hand) integration and/or differentiation. Section 3.4 deals with the assembly of the equations. Boundary conditions which involve derivative parameters cause annoying implementation problems if the boundary is not parallel to the x or y axis, since relations between several parameters

must sometimes be satisfied. We discuss two alternate methods of handling these problems and compare the implementation of each.

A study of sparse matrix methods is the subject of Chapter 4, with particular emphasis on the type of matrices arising from finite element methods. We introduce the concept of the profile of a matrix, and distinguish between graph methods, profile methods and band methods. We present arguments and experimental evidence supporting the use of profile methods.

In Section 4.5 we compare several ordering algorithms applied to matrices arising from different finite element bases. These experiments show the following: (a) profile methods can be significantly better than band methods, in terms of both storage requirements and operation counts; (in Sections 4.1 and 4.2 we show that they will never be worse than band methods.) (b) the "reverse Cuthill-McKee" ordering (our terminology), which we have discovered compares very favorably with other methods tested ; (c) comparison of times required to produce the reverse Cuthill-McKee ordering with some of the times required for the entire finite element solution (reported in Chapter 5) suggests that the use of the algorithm is relatively inexpensive. We feel that such information is extremely important. It is often contended by experienced users that automatic ordering is unnecessary because they can produce an ordering empirically that is close to optimal. This may very well be true, but not all users are experienced, and more important, one must still devise a way of communicating the desired ordering to the computer. We have shown that this largely clerical process can best be

left to the computer. The code for doing the ordering appears as part of phase 1 in Appendix B.

Also in Chapter 4, we derive formulas for the density of finite element matrices for general elements and arbitrary triangular and quadrilateral meshes with holes. Such results are important in managing storage, since we can allocate storage for the matrix as soon as the mesh and element to be used are known.

Chapter 5 contains results of several numerical experiments. The chapter contains numerical solutions to the L-shaped membrane eigenvalue problem, rhombical membrane eigenvalue problems, and a hollow square Dirichlet problem. Our comparisons are between different finite element methods as well as between finite element methods and their competitors. These experiments showed the following:

(a) Efficiency in general increased with increasing degree of piecewise polynomial. This was true in all three examples, and because the solutions ranged from very smooth ones to ones with singularities in their first derivatives, we feel this information is significant.

(b) Finite difference methods compared rather unfavorably with our finite element solutions. Even for the problem where special fast direct methods for solving the difference equations could be utilized [B15,G1], our finite element solutions appeared preferable.

(c) Several methods for finding eigenvalues yielded more accurate numbers than finite element methods (involving roughly the same cost), and also produced bounds. However, these methods use techniques which utilize a special feature of the equation or of the domain, and are

difficult to implement in a general code. Again we emphasize that we are not saying these methods are inferior; we are simply saying that they are less suitable than finite element methods as the core of a general boundary value problem solver.

Appendix A contains a list of some typical elements. Some of these are referred to throughout the text.

Appendix B contains a listing of the Fortran code we have developed for solving a class of linear elliptic boundary value problems. We have segmented the code into modules, each one designed to carry out a specific task or set of tasks. The modules execute in sequence, with information passing from one to the next via external storage media which can be disk, drum or tape. Our reasons for segmenting our code and attempting to keep each segment itself modular are (a) to ease maintenance and/or modification of the code, (b) to allow the program to be run on smaller machines than the one we used, and (c) to facilitate documentation and understanding of the code by localizing specific functions. Specific details of the functions of each segment are found in comments in the code itself.

CHAPTER 2

GENERATION AND STORAGE OF TWO-DIMENSIONAL TRIANGULAR MESHES

1. Introduction

The first step in most numerical methods for solving partial differential equation problems is that of discretizing the domain in question. In our case, the problem consists of dividing our given domain R into disjoint triangles whose union is $R \cup \partial R$, with adjacent tirangles having a common side. If R has curved boundaries, we will admit "curvilinear" triangles having one curved side in the triangulation near the boundary. Figure 1.1 is an example of such a triangulation.

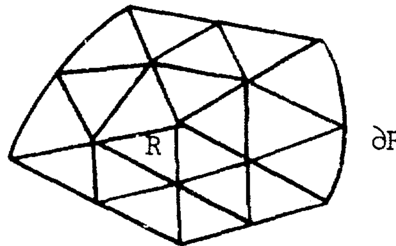


Figure 1.1

Manual generation of a triangulation of R is an extremely tedious job. A completely automatic triangulation procedure, on the other hand, while obviously desirable, is complicated and difficult to implement with any degree of flexibility. In Section 2 we review the literature on two-dimensional triangulation and in Sections 3 and 4 we present two new methods for triangulation of two dimensional domains. The method described in Section 3 is a semi-automatic scheme, while the one in

Section 4 is almost completely automatic. Section 5 contains a description of an efficient scheme for storing a representation of an arbitrary two-dimensional triangular mesh.

Once we have a suitable triangulation of the domain, we are faced with the problem of numbering the nodes (more precisely, the parameters associated with the nodes) in order to reduce the computation and/or storage requirements for the solution of the algebraic system. Although it is possible to defer any ordering (and then actually order the parameters rather than the nodes), the problem will be considerably larger if each node has more than one parameter associated with it. Since all the parameters associated with a particular node are connected in the same way to other parameters as well as all being connected to each other, little is lost by ordering the nodes. Many good ordering algorithms require work proportional to the number of nodes multiplied by the square of the number of neighbours each node has, so substantial savings can be achieved by ordering at this stage. We defer discussion of these algorithms and the criteria used to reduce storage and computational requirements until Chapter 4, although again we emphasize that they should be applied at this stage.

2. Review of Previous Work on Mesh Generation

Mesh generation is a difficult part of a boundary value problem-solver to automate, and even the most generally applicable programs require substantial human assistance, especially in describing the boundary ∂R. In most boundary value problems the solution is not uniform in character over the domain. Often it is fairly smooth over most of R, and varies rapidly only over a small part of the domain, perhaps near a corner. For this reason it should be convenient, if desired, to indicate areas of the domain R where the grid can be coarse and areas where it should be fine. This grading capability could provide substantial savings in storage requirements and computer time.

For practical reasons finite difference programs have tended not to provide for the grading of nets. This is due largely to the ease with which one can store a regular rectangular net in a conventional two-dimensional array and the severe storage management problems which immediately result when one departs from such a scheme. In the regular case, actual coordinates do not even have to be stored, which is a persuasive argument for using a regular net. Also, truncation error bounds for some difference operators are much better for regular nets, and the determination of the coefficients for the difference operator is usually much easier (a prime consideration if an iterative scheme is being used and the coefficients are being computed each time they are needed). Thus, finite difference programs usually make use of uniform meshes, or meshes which are uniform in various parts of the region. Boundary points that result when ∂R intersects the mesh at a point other than a node point are treated by using well-known interpolation formulas. (These special boundary points may cause storage problems even when the simple two-dimensional array storage method is used; see Forsythe and Wasow [F5, pp. 361-63] for a discussion.) If the boundary is curved, it may be

rather awkward to find the correct formula to preserve the order of accuracy. In this context, the actual mesh generation is not a difficult problem. The problems arise where the boundary (which can have more or less arbitrary shape) intersects the regular mesh. Cryer [C3] treats this entire problem in considerable detail, and we will not discuss it further here.

The mesh generation question with regard to finite elements has a somewhat different flavor. In this case, grading the mesh is essentially without cost provided we are going to store the node coordinates anyway. It is often stated that irregular nets are expensive to use because the coordinates of each net point must be stored, and for finite difference methods this objection is valid. [For example, suppose we are solving Laplace's equation on the unit square. Using a uniform n by n grid, the required storage is about n^2, assuming we are going to solve the equations using SOR. By comparison, if our mesh is irregular, we must remember the coordinates of each of the n^2 nodes. Then we would need a total of $3n^2$ words of storage, and if we want to avoid recalculation of the coefficients of the difference operator at each iteration (which will no longer all be the same), we will need $8n^2$ words of storage.] However, for finite element methods, the number of node points will ordinarily be considerably fewer than the number of parameters since each node will usually have derivative as well as function-value parameters associated with it. As the degree of the basis functions increases, the storage required for the nodes quickly becomes small compared with that required for the coefficient matrix. This point is taken up in Section 5 of this chapter.

We will now review some methods appearing in the literature which have dealt with this triangulation problem.

Cheung and Pedro [C4] have written a program that generates a triangulation using the following general scheme. The domain is divided by one family of straight lines (which do not intersect in the domain but are not necessarily parallel) or arcs of circles (not necessarily concentric) or both. Each line is further divided into a number of divisions to yield node points. The node points on adjacent lines are then joined in a zig-zag manner to form triangles. The number of divisions in adjacent lines can only differ by one -- a hindrance if pronounced grading of the net is desired. This restriction can also lead to triangles with very sharp angles.

No attempt appears to be made to automatically achieve a nodal numbering yielding a small bandwidth; instead manual "supervision" has to be exercised at various stages. The only attempt to avoid or remove small angles is done when forming two triangles from a quadrilateral; the lengths of the diagonals are computed and the shorter is used to form the triangles. (This can be disastrous; consider the quadrilateral $(-1,0)$, $(0,-2)$, $(1,0)$, $(0,\epsilon)$ where ϵ is positive but very small.)

Frederick, Wong, and Edge [F7] present a two-stage, semi-automatic method for triangulating a two-dimensional domain. The first stage consists of manually plotting the boundary of the domain and the node points (in the order designed to minimize or at least reduce the bandwidth of the resulting linear system) on an electromagnetic graph-tracing table. The coordinates

of the points are automatically punched on cards which then serve as input to a computer program that generates the triangles. There are a number of potential drawbacks to this approach. The first is that for odd-shaped domains it is surprisingly difficult to number the nodes empirically so as to achieve a small bandwidth, especially if the net is graded rather severely. As we shall see in Chapter 4, bandwidth is not necessarily a very good criterion anyway, and to number the nodes empirically to achieve other (more satisfactory) criteria can be even more difficult. Secondly, without actually drawing in the triangles as you go along it is hard to decide where the next node would be placed. If the triangles are to be drawn, very little more manual effort would be necessary to tabulate their respective nodes, thus eliminating the computer program completely. As the authors point out, however, the computer-based part of the procedure eliminates the clerical errors which would inevitably result from tabulation by humans. Although it is unfortunate that special-purpose equipment is required, the basic procedure is very appealing. It is easy to see how the same basic idea could be implemented in an interactive way by using a cathode ray display with a light pen. All the above objections could be eliminated if an automatic ordering scheme (such as one of those discussed in Chapter 4) were included in the implementation.

Barfield [B4] proposes a method based on a conformal mapping of the boundary of a closed two-dimensional region onto the perimeter of a rectangular polygon in which is inscribed an orthogonal rectilinear grid. The method consists essentially of finding the function which conformally maps the given domain R onto the polygon, and then using the inverse of the mapping so determined to find the image of the orthogonal grid in the

polygon. The method obviously generates rectangles rather than triangles, so that each rectangle would have to be subdivided to obtain a triangulation of R . While the method is indeed very elegant, considerable care appears to be necessary to avoid distortion, and "long, slender" squares yield very poor triangles. Also, the work involved in computing the mapping may be substantial.

Winslow [W5] proposes a method of mesh generation which consists essentially of solving an elliptic boundary-value problem using finite difference methods. The mesh lines are regarded as two intersecting sets of equipotentials, each set satisfying Laplace's equation in the interior of the given two-dimensional domain R . "Boundary conditions" are determined by where the lines are required to intersect the boundary S . Because of the well-known averaging feature of harmonic functions, the generated mesh varies smoothly over the entire domain, its relative grading being determined by the density of the points of intersection on S (i.e., the boundary conditions). Triangular and quadrilateral grids can be generated using the method. Although the examples reported are very nice, they are for an extremely simple domain, and Winslow does not discuss the problem of how to concisely describe a general domain to the program (assuming that the program has the facility for handling one), and how to easily input the boundary conditions (the ends of the potential lines). As with most partial differential equation problems, the above tasks and the associated data management problems are difficult to implement in general; once done, the generation of the equations and their solutions are relatively straightforward, even though they may require considerable computer time. He concedes that the method does not always work satisfactorily near re-entrant corners, with node points outside the domain sometimes being produced.

Reid and Turner [R1] use the following scheme to generate nearly regular meshes. A regular equilateral triangular mesh is placed over the domain R so that ∂R is inside the mesh boundary. Points where triangle sides intersect the boundary are called "boundary points", and node points of the mesh closer than $h/2$ to a boundary point are moved to the boundary point in such a way as to guarantee the monotonicity of the resulting finite element coefficient matrix. [A matrix is said to be <u>monotone</u> if it is non-singular and all elements of its inverse are non-negative.] They consider only piecewise-linear polynomials. The node points and their incident edges which remain outside ∂R after the relocation process is complete are then discarded, yielding a mesh on R which is regular except near the boundary. The authors' assumption appears to be that ∂R has <u>no corners</u>, and this restriction on ∂R simplifies the node relocation considerably. Corners in ∂R must necessarily end up as vertices in the triangulation, so the presence of corners imposes further restrictions on the relocation of nodes. It seems clear that we would want h to be of the same order of magnitude as (or smaller than) the shortest arc in ∂R in order to avoid generating triangles with sharp angles. Such a requirement could force the mesh to be finer than otherwise necessary. This scheme obviously assumes that the user desires a regular mesh, and this may not always be true.

Kamel and Eisenstein [K1] present a mesh generation scheme that is also based on a regular mesh. The user supplies the boundary ∂R as a sequence of arcs subdivided by nodes. First the authors find the "best" regular mesh having the same number of boundary nodes as the given boundary ∂R. Here "best" means "closest to circular shaped." Their program begins at a node of a regular mesh and successively annexes rings of triangles (the last

ring may only be partially annexed) until the number of boundary nodes in the mesh equals the number of nodes on ∂R . This determines the number and relative positions of the triangles for the mesh. The correct number of nodes are then placed inside R and the mesh is then smoothed by applying several passes on the interior nodes, using the formula

$$x_i = \Big(\sum_{y \in \eta(x_i)} y \Big) \Big/ |\eta(x_i)| \quad . \tag{2.1}$$

The authors caution that their procedure does not work well if the input boundary has nodes with abrupt changes in spacing, or if the domain shape is too complex. They imply that interaction with the algorithm using a graphics terminal is an advisable, if not necessary, part of using their method.

3. A Semi-Automatic Mesh Generation Scheme

Ideally, a mesh generation procedure should have the capability of grading the net (i.e., making the net finer in selected areas of the domain) on the basis of information supplied by the user. This immediately raises the question of _how_ a desired grading can be easily transmitted to the program. Also, sometimes the "material" in the domain varies abruptly from one region to another, and it may be desirable that triangle interfaces coincide with material interfaces to allow discontinuities in derivatives. This requirement would obviously complicate a completely automatic triangulation procedure by imposing constraints on some of the node positions.

With these considerations in mind we have arrived at the following compromise. The user must supply a very gross triangulation of the domain, reflecting the desired grading of the net, and with triangle boundaries lying in any desired position. This removes both of the problems raised above. The large triangles can then be subdivided by the computer in the obvious manner. If in addition the program has the capability of subdividing triangles having one _curved_ side, the amount of input for most domains can be kept small.

The algorithm used to subdivide each input triangle is very simple. For some integer k , depending on how fine a final mesh is required, each triangle side is evenly divided into k segments by $k-1$ nodes. Nodes of consecutive sides are joined by parallel lines yielding k^2 triangles, each congruent to the original large one. This has the advantage that no sharp angles are generated; the smallest angle in the original triangulation is the same as the smallest in the final triangulation.

For "curvilinear" triangles (having one curved side) the algorithm is similar. Suppose we have the following triangle (Fig. 3.1-a) which we must refine by a factor of eight (Fig. 3.1-b).

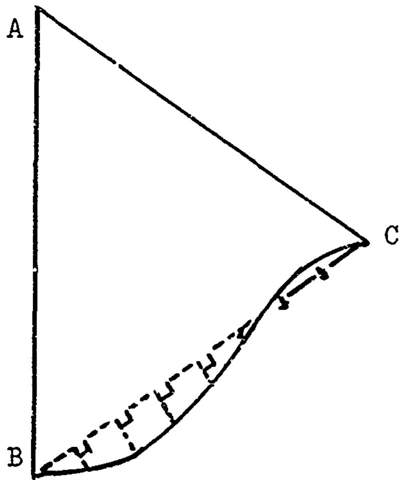


Fig. 3.1-a

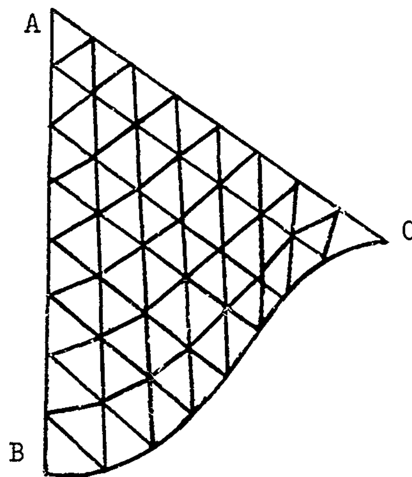


Fig. 3.1-b

Seven node points are generated on each of the straight lines AB , BC , and AC as described above. The seven node points on the curve BC are then obtained by finding (approximately) the points of intersection of the curve with lines perpendicular to the straight line BC and passing through the node points on it. The node points on AB and AC are each joined to the node points on the curve as in Fig. 3.1-b by straight lines, and their points of intersection are then used to form the triangles.

Below is an example of the procedure:

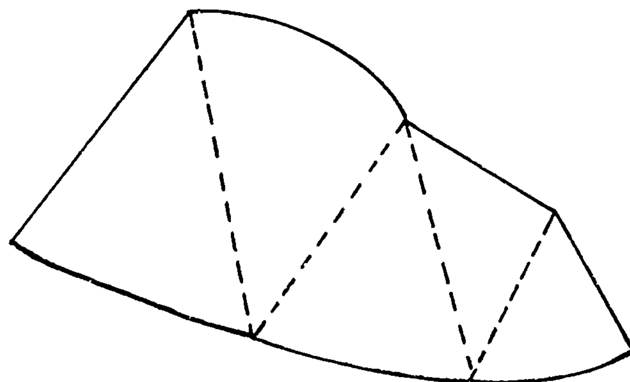


Figure 3.2-a. Input Domain. Gross triangulation indicated by dashed lines.

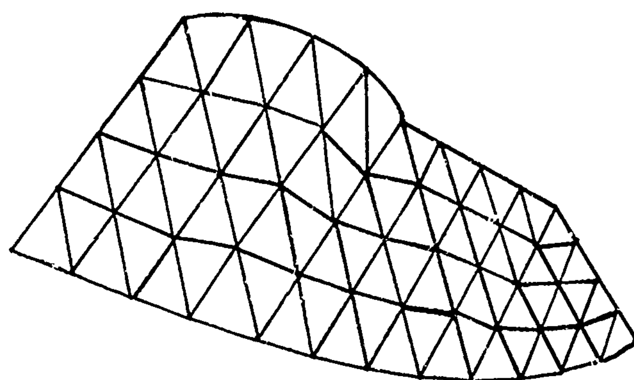


Figure 3.2-b. Domain subdivided by a factor of 4.

This approach to mesh generation could very conveniently be adapted to use with an interactive display system. The fact that the user has control of the mesh while not being obliged to provide large amounts of input is particularly attractive in this regard.

4. An Automatic Two-Dimensional Domain Triangulator

In this section we present a scheme for automatically triangulating a two-dimensional domain. Unlike the fully automatic schemes discussed in Section 2.2, this method does not utilize a regular mesh; in fact, it specifically is designed to allow for the construction of graded nets. It can be used for general simply connected domains, as the examples appearing later will demonstrate.

The basic strategy of the method is as follows. The user is required to supply the initial boundary as a sequence of arcs, along with a simple rule indicating how each arc is to be subdivided. The sequence of arcs must form a closed loop, so for now we assume R has no holes. We then have an "initial boundary" consisting of a sequence of nodes connected by straight lines. We then proceed to annihilate R by successively removing triangles from R, as depicted in Figure 4.1. As each triangle is removed, we obtain a new "current boundary". This boundary, along with some associated information can be conveniently stored as a two-way linked list. Our goal is to cover (or annihilate) R with as few triangles as possible consistent with the requirements that the mesh vary smoothly and have no sharp angles or long sides. For example, for a unit square domain with each side divided into segments of length 0.01, we would like the generated mesh to be composed largely of triangles which are close to equilateral triangles having sides of length 0.01.

We will employ two methods of forming triangles. The first, which we will refer to as "trimming", is depicted by (i), (iii), (iv) and (vi) in Figure 4.1. The second method of generating triangles requires the generation of a node in R, as shown by (ii) and (v) in Figure 4.1. We

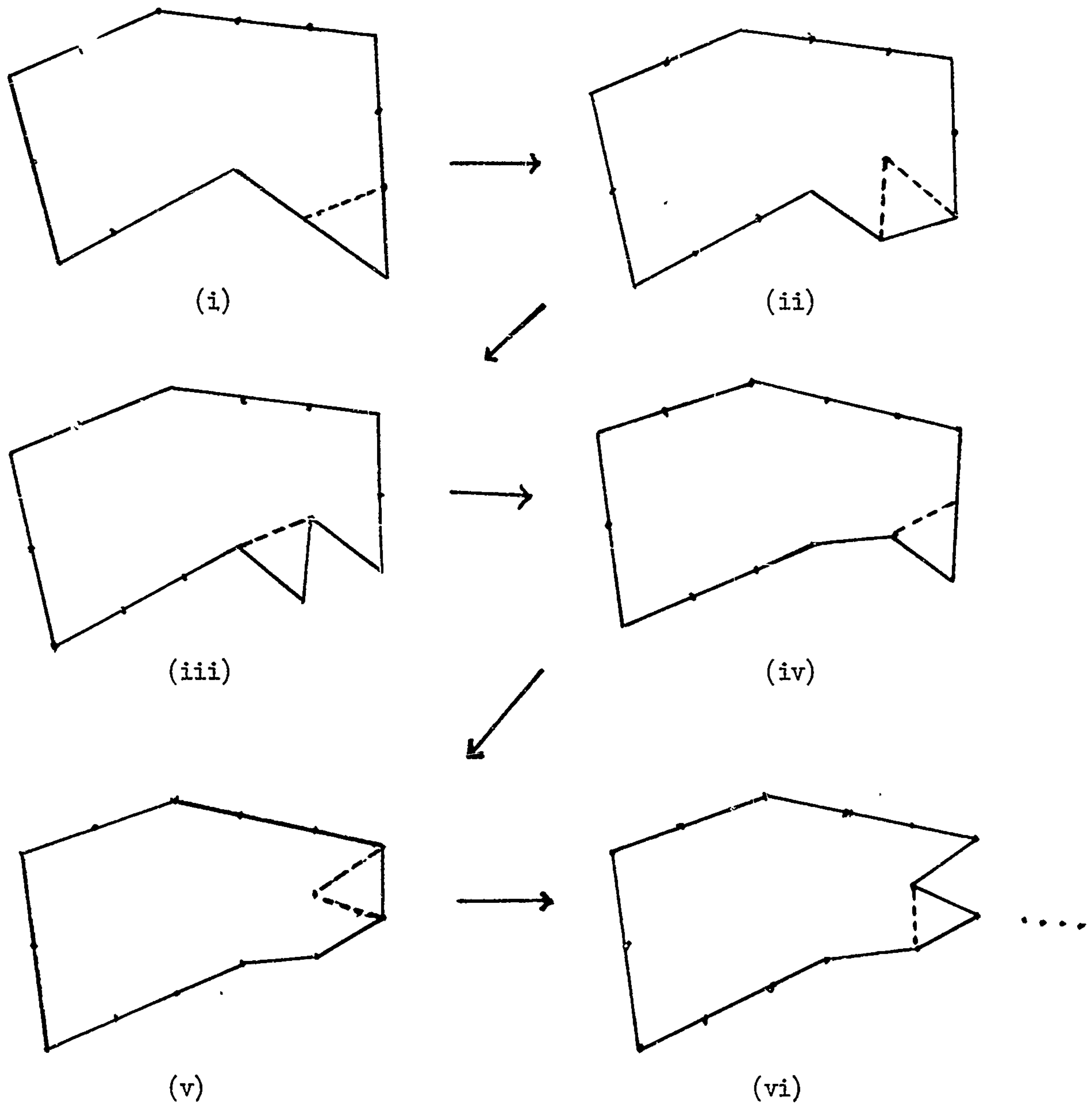


Figure 4.1

will refer to this method as "notching".

First we discuss the generation of nodes. Consider the diagram below

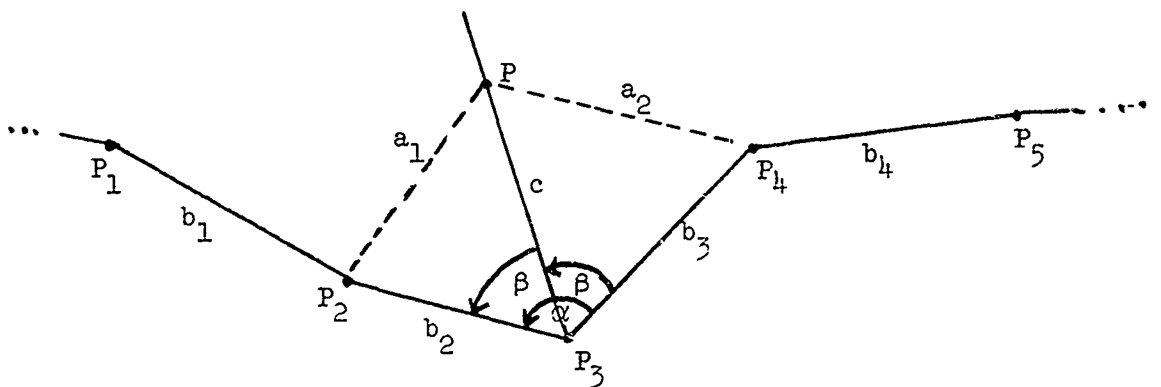


Figure 4.2

Let $\bar{\ell}$ be the average distance between consecutive nodes in the <u>initial boundary</u> ∂R . Then P is the point on the bisector of angle $P_2P_3P_4$ which approximately minimizes

(4.1) $$\mathcal{F}(P) = |c-b_2| + |c-b_3| + \gamma\{|a_1-b_1| + |a_2-b_4|\} + \frac{1}{1+\gamma}\{|a_1-\bar{\ell}| + |a_2-\bar{\ell}|\} ,$$

where

$$\gamma = \sum_{i=1}^{4} |b_i-\bar{b}| / 4\bar{b} ,$$

and

$$\bar{b} = \sum_{i=1}^{4} b_i / 4 .$$

The first two terms are designed to make the (potential) triangles close to equilateral. The third term has a smoothing influence on the

lengths of the arcs of the current boundary, and the last term attempts to make the lengths of the arcs of the current boundary converge to $\bar{l}$. If either or both of the neighboring vertices have angles less than $5\pi/6$, the same procedure is performed at these vertices, yielding two or three nodes. Their centroid is chosen as the trial node.

Now that we have a method for generating interior nodes, we can now describe the algorithm. In words it is as follows:

Step 1.

For each vertex on the current boundary having interior angle α less than or equal to $\pi/3$, form a triangle by trimming and remove it from R , as depicted below.

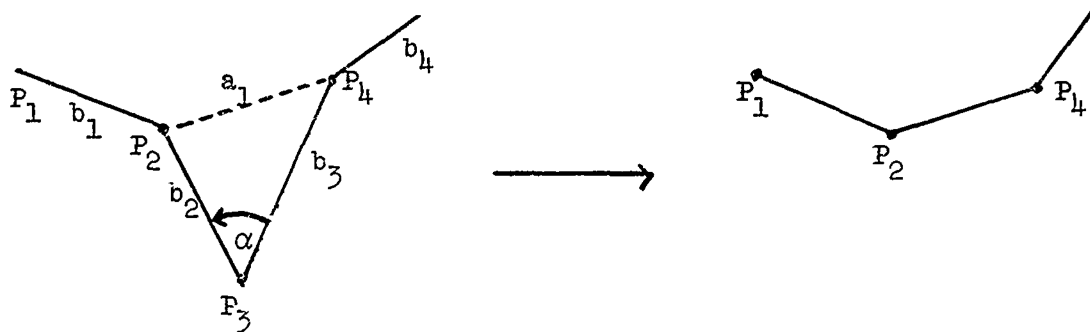


Figure 4.3

Step 2.

Find any consecutive vertices both having interior angles α_1 and α_2 less than $5\pi/6$. If none are found, proceed to step 3. Otherwise, choose the pair with the minimum value of $|\alpha_1 - 2\pi/3| + |\alpha_2 - 2\pi/3|$, and generate an interior node P as described above. We then have a

situation such as one of these below:

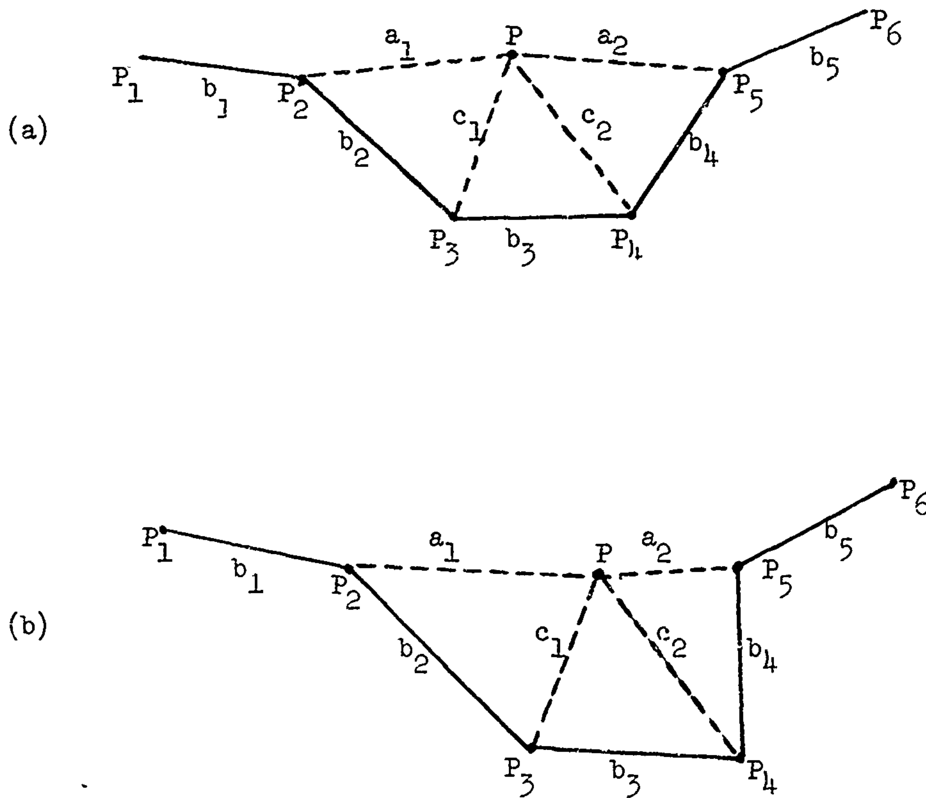


Figure 4.4

It is obvious what we should do in Figure (4.4-a), but in Figure(4.4-b), it is debatable whether we should trim triangle $P_3P_4P_5$, or notch triangle PP_3P_4 (followed presumably by two trims of triangles PP_2P_3 and PP_4P_5 .) Let $d_1 = |P_2-P_4|$ and $d_2 = |P_3-P_5|$, where $|P_i-P_j|$ is the distance between points P_i and P_j . Let $S_1 = \{b_1,d_1,b_4,b_5\}$, $S_2 = \{b_1,b_2,d_2,b_5\}$ and $S_3 = \{b_1,a_1,a_2,b_5\}$. Let v_1 , v_2 and v_3 be the average value of the members in S_1 , S_2 and S_3 respectively. Now define w_1 , w_2 and w_3 by

$$w_1 = (|b_1 - v_1| + |d_1 - v_1| + |b_4 - v_1| + |b_5 - v_1|) / 4v_1 \quad ,$$

$$w_2 = (|b_1 - v_2| + |b_2 - v_2| + |d_2 - v_2| + |b_5 - v_2|) / 4v_2 \quad ,$$

$$w_3 = (|b_1 - v_3| + |a_1 - v_3| + |a_2 - v_3| + |b_5 - v_3|) / 4v_3 \quad .$$

Let $w_k = \min\{w_1, w_2\}$. Then if $k = 1\{2\}$, $w_k > w_3$, and angle $P_1P_3P_4$ $\{P_3P_4P_5\}$ is less than or equal to $\pi/2$, then trim triangle $P_2P_3P_4$ $\{P_3P_4P_5\}$. Otherwise, notch triangle PP_3P_4 . Then go to step 1.

Step 3.

Find any vertex having interior angle $\alpha \leq \pi/2$. If there are none, go to step 4. Otherwise compute an interior node corresponding to the vertex as indicated below.

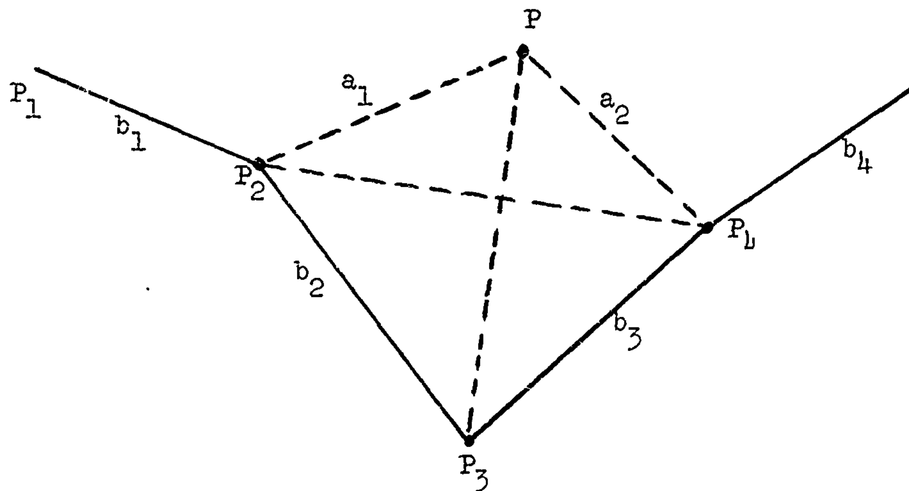


Figure 4.5

Let $d_3 = |P_2-P_4|$ and $d_4 = |P-P_3|$. Define v_4 , v_5 , w_4 and w_5 by

$$v_4 = (b_1 + d_3 + b_4) / 3 \quad ,$$

$$v_5 = (b_1 + a_1 + a_2 + b_4) / 4 \quad ,$$

$$w_4 = (|b_1-v_4| + |d_3-v_4| + |b_4-v_4|) / 3v_4 \quad ,$$

$$w_5 = (|b_1-v_5| + |a_1-v_5| + |a_2-b_5| + |b_4-v_5|) / 4v_5 \quad .$$

If $w_4 \leq w_5$ then trim triangle $P_2P_3P_5$ and go to step 1. Otherwise repeat step 3 until a successful trim is achieved or until all the vertices have been tested. If no trim can be made, proceed to step 4.

Step 4.

Let μ_1 and μ_2 be the smallest and largest distances between consecutive nodes in the current boundary. We then do the following

(4a) Set $\gamma = \mu_1 + \frac{\mu_1}{\mu_2} (\mu_2-\mu_1)$.

(4b) Of those nodes on the current boundary having at least one of its incident boundary arcs less than or equal to γ , choose the node having the smallest angle α

(4c) If $\alpha \leq \pi$, generate a node, as in Figure 4.2, notch a triangle (either PP_2P_3 or PP_3P_4) and go to step 1.

Otherwise, go to (4d).

(4d) If $\gamma = \mu_2$; stop (we have failed). Otherwise, set $\gamma = \gamma + \frac{\mu_1}{\mu_2} (\mu_2-\mu_1)$ and go to (4b).

Remarks:

(1) The parameter γ in step 4 was found to be necessary to force the program to consider first those areas of the domain to be covered by a relatively fine mesh. The averaging effect built into the node generator combined with this restriction on the lengths of the arcs considered first tends to fill in the domain near the short boundary arcs first; the size of the triangles increases with distance from the boundary.

(2) Steps 1, 2 and 3 are designed to remove any "protrusions" from the current domain. Their overall effect is to make the current boundary convex or near convex.

(3) An interesting and potentially better method for generating nodes might be to allow P (Figure 4.2) to lie anywhere in the current domain, rather than restrict it to lie on the bisector of the angle $P_2P_3P_4$. Minimizing $\mathcal{F}(P)$ would be considerably more complicated, but might be justified if triangulations with many fewer triangles resulted.

(4) In all cases where a node is generated, we check to see if it lies in the current domain by using an algorithm described in [N2], and before forming any triangle we check to make sure no nodes lie inside the triangle. Thus, our algorithm is "fail safe"; if it terminates successfully, it has generated a legal triangulation.

(5) As we mentioned above, the current boundary can best be stored as a linked list, so that deletions and insertions can be carried out with little data rearrangement. To reduce computation, the lengths of each boundary arc and the sine and cosine of each interior angle were also

retained in conjunction with the linked list. These quantities were computed once by the routines "trim" and "notch" which actually modify the current boundary, and were then available as needed by steps 1-4. Other quantities might also have been retained.

(6) Figure 5.10 demonstrates the use of the algorithm when the domain has a hole in it. We simply provide a "boundary arc" cutting through the domain, joining the outer boundary to the inner one. The fact that the closed loop forming the boundary overlaps itself and in some parts does not really correspond to a boundary at all does not effect the algorithm. The smoothing program (discussed below) does not move node points lying on these pseudo boundary arcs; hence, this device can be used to force some inter-element boundaries to lie in specified positions. In Section 2.3 we explained why this might sometimes be desirable.

Below are several examples of domain triangulations. The output of the algorithm described above has been smoothed by carrying out three or four sweeps of the interior (non-boundary) nodes using formula (2.1). The nodes on the curved portions of the boundary were obtained in the same manner as described in Section 2.3.

As we implied in remark 4 above, more sophisticated methods of node generation and trim/notch strategies might yield "better" triangulations, and such investigations are potentially fruitful topics of further research. It is even difficult to define precisely what we mean by a good graded mesh. It depends on the relative importance of (a) sharp angles (b) the total number of triangles (c) the smoothness of variation of the mesh, and perhaps other factors. It would be nice also to be able to a priori guarantee certain desirable characteristics of the generated mesh in terms of characteristics of the initial boundary.

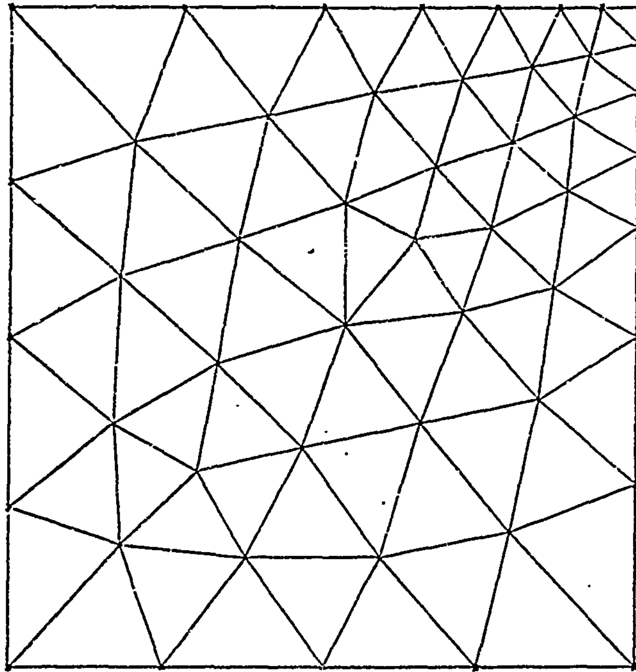


Figure 4.6

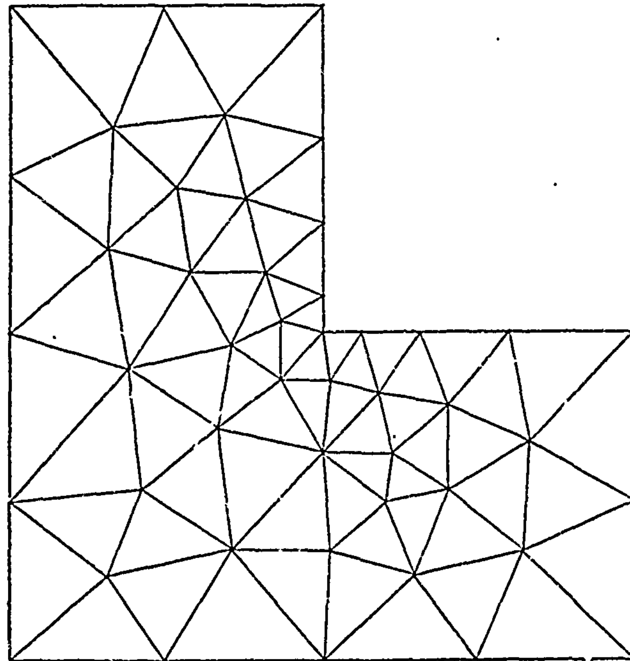


Figure 4.7

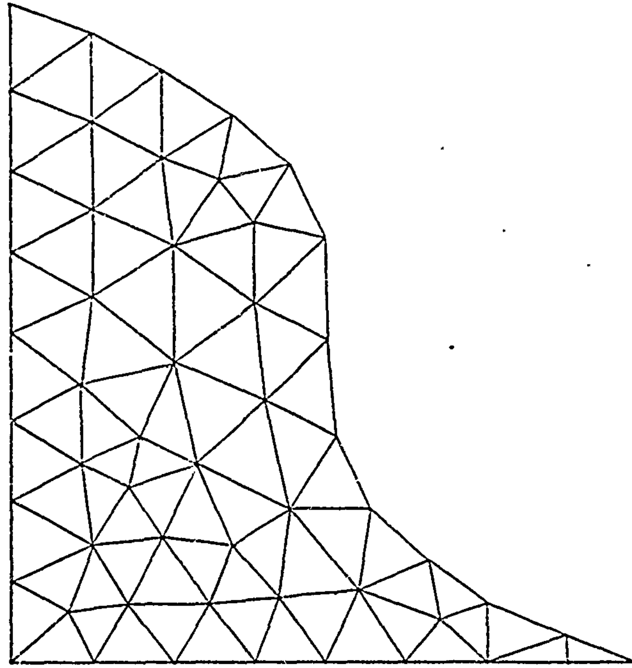


Figure 4.8

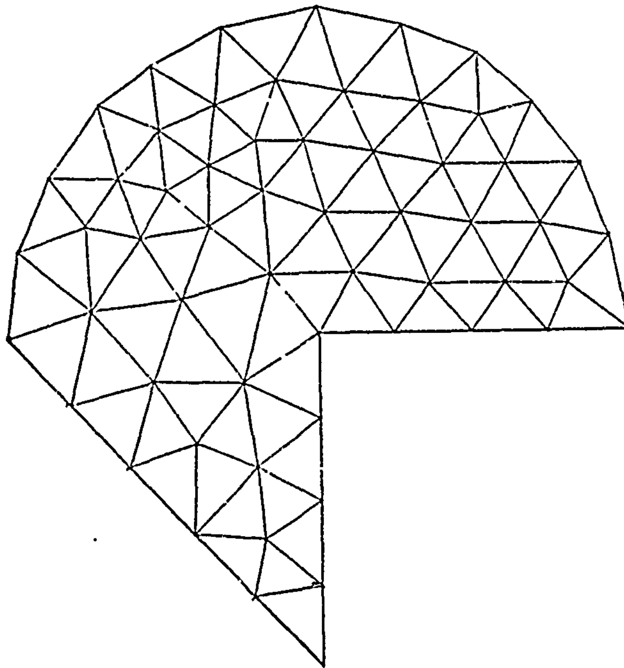


Figure 4.9

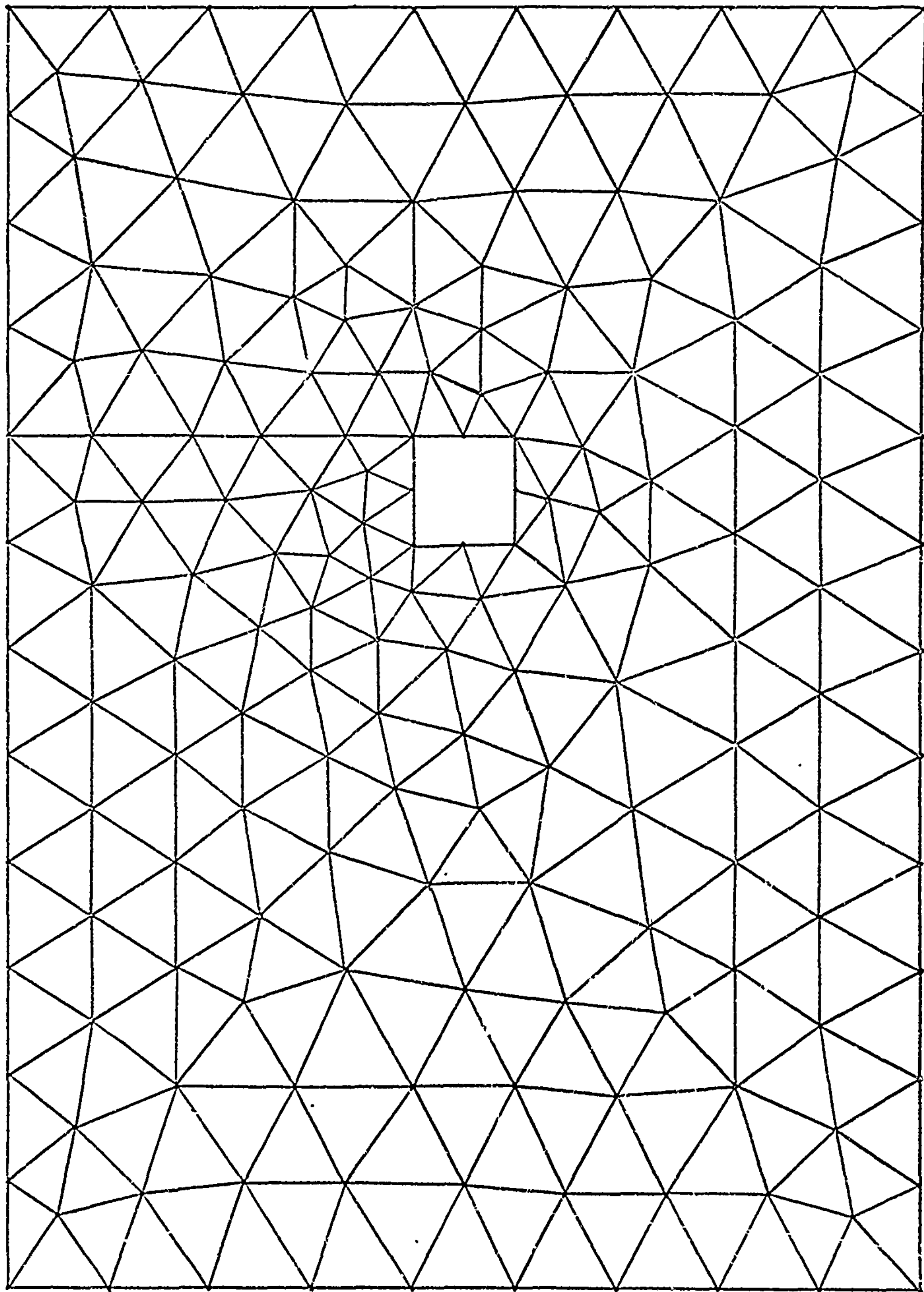


Figure 4.10

5. A Storage Scheme for Finite Element Meshes and Associated Boundary Data

As we have tried to emphasize in the preceding sections, the finite element mesh will probably not be regular; thus the storage scheme for it must be general. In this section we present a scheme for storing general finite element meshes, and show that for most elements, the required storage is small compared to the storage required to store the non-zero elements of the coefficient matrix.

We shall see in Chapter 3 that the procedure for generating the finite element equations is carried out element-by-element; therefore, it is beneficial to be able to retrieve the node coordinates for each triangle easily. On the other hand, we do not want to store copies of the node coordinates for each triangle, because many or all of the nodes belong to more than one triangle. Another point is that we really only need to remember the vertices of the triangles in the mesh; node coordinates on the sides and in the interior of the triangle can be generated as needed, provided we have a formula for generating them.

For definiteness, suppose our mesh has V vertices, S triangle sides, N_Δ triangles, and H holes in it. The number of interior sides {vertices} and boundary sides {vertices} will be denoted by $S_I\{V_I\}$ and $S_B\{V_B\}$ respectively. In [E1] the following relations between these mesh parameters are proved.

$$(5.1) \qquad N_\Delta = \frac{1}{3}(S_B + 2S_I) = V_B + 2V_I - 2(H-1) \quad .$$

For a typical mesh having $S_I \gg S_B$, $V_I \gg V_B$, and small H , the the relations (5.1) yield

$$(5.2) \qquad V \doteq \frac{1}{2} N_\Delta \quad ,$$

and

(5.3) $$S \doteq \frac{3}{2} N_\Delta \quad .$$

To aid in describing the scheme we are about to present, consider the figure below, where the domain has been covered by "3-10" elements (see Appendix A for details). The nodes are numbered sequentially, beginning with the vertex nodes, followed by the arc-midpoint nodes (see below), followed by the nodes on the sides and interiors of the triangles. A node with tag k is understood to have coordinates (x_k, y_k) . The circled numbers are <u>boundary reference numbers</u> which are associated with the corresponding triangle sides. Later, boundary conditions can be assigned with respect to these numbers. The <u>arc-midpoint</u> nodes tagged 6 and 7 are generated and allowed for in the storage scheme so that some form of interpolation along the boundary can be subsequently done. See Zlámal [Z6] for one such possibility, where quadratic interpolation is used.

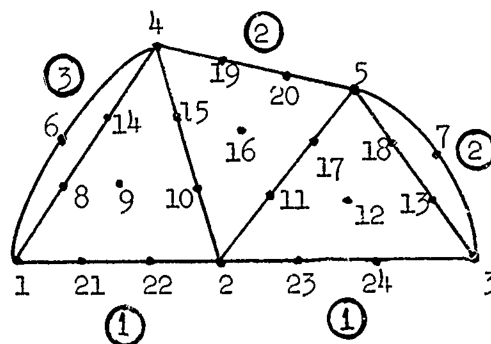


Figure 5.1

The storage scheme is depicted schematically below. Note that the pointers for each triangle are listed in a counterclockwise manner, in order of

vertices, sides, and interior. Suppose the vertices are numbered (x_1,y_1) , (x_2,y_2) and (x_3,y_3) . We adopt the convention that the i-th side of the triangle is the one with endpoints (x_i,y_i) , (x_k,y_k) , where $k = (i+1) \bmod 3$.

A pointer will ordinarily require fewer bits than a node coordinate or a coefficient of an equation. For example, on an IBM 360 computer, pointers may conveniently be stored in two bytes (a half-word) whereas a coordinate would require four or eight bytes. In general, we will denote this ratio by α $(\alpha \leq 1)$. Ignoring the storage required for the boundary table (since we assume $S_B \ll S_I$) , then the amount of storage required for the mesh is approximately

$$(5.4) \qquad V_M = \alpha m N_\Delta + 2V \doteq (\alpha m+1)N_\Delta \ ,$$

where m is the number of <u>nodes</u> associated with each element.

Let n_V , n_S and n_I be the number of parameters associated, respectively, with vertex nodes, the node(s) on each triangle side (not including the endpoints), and the interior of each triangle. For example, element 3-10 would yield $n_V = 1$, $n_S = 2$, and $n_I = 1$. We now want to show that V_M is usually small compared to the number N_Z^A of non-zero elements in the coefficient matrix A . In Section 4.6 we show that

$$(5.5) \qquad N_Z^A \doteq \sigma_1(V-3) + \sigma_2(S+3-2V)$$

$$\doteq \left(\frac{\sigma_1 + \sigma_2}{2}\right) N_\Delta \qquad , \quad \text{(using (5.2) and (5.3))}$$

where σ_1 and σ_2 depend on n_V , n_S , and n_I .

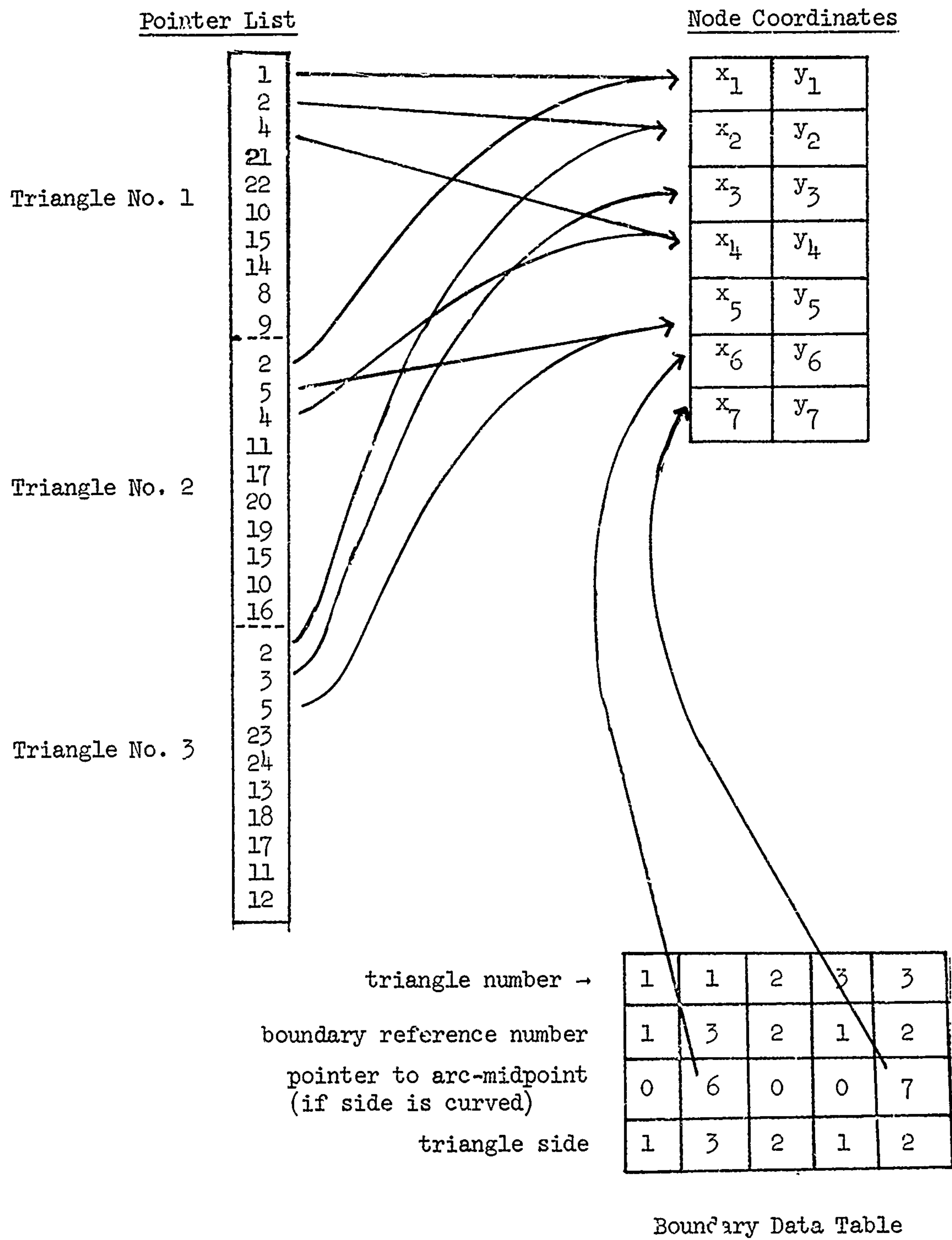


Figure 5.2

The following table serves to make our point.

Element[+]	σ_1	σ_2	$\frac{\sigma_1+\sigma_2}{2}$	$\alpha m+1$			
				$\alpha = \frac{1}{4}$	$\alpha = \frac{1}{3}$	$\alpha = \frac{1}{2}$	$\alpha = 1$
1-3	5	2	3.5	7/4	2	5/2	4
2-6	27	19	23	5/2	3	4	7
3-4	64	37	50.5	2	7/3	3	5
3-10	84	69	76.5	14/4	13/3	6	11
4-6	161	106	133.5	5/2	3	4	7
4-15	200	176	188	19/4	6	9/2	16
5-6	272	139	205.5	5/2	3	4	7
5-21	405	370	387.5	25/4	8	23/2	22

[+]See Appendix A for a description of the elements.

Table 5.1

Thus for all but piecewise linear polynomials, $\nu_M \ll N_Z^A$, even when $\alpha = 1$. If a direct method is being used to solve the generated system, the storage required for the decomposition will be much more than N_Z^A , so that ν_M becomes rather insignificant in comparison to overall storage requirements. Our conclusion is that the use of an irregular rather than regular mesh for finite element methods does not in general cause an important increase in storage requirements.

In future chapters we will often need the dimension of A , the number of parameters (unknowns) in our problem. Using (5.2), (5.3), and

the definition of V , S , n_V , n_S and n_I above, we have

$$N = n_V V + n_S S + n_I N_\Delta \tag{5.6}$$

$$\doteq \left(n_I + \frac{1}{2} n_V + \frac{3}{2} n_S\right) N_\Delta \quad .$$

CHAPTER 3

GENERATION OF FINITE ELEMENT EQUATIONS

1. Introduction

In this chapter we discuss in detail the computation involved in the generation of the finite element equations. The general procedure with minor variations appears rather frequently in engineering articles (usually with regard to a specific problem and element); Zlámal [Z5] has also described the procedure, again for a specific situation. Felippa and Clough [F1] give an excellent summary of the generation process although they give few details. Unfortunately, we feel that too little emphasis is devoted to carefully identifying which of its several sub-tasks are independent of others, and which ones are dependent only on particular components of the problem being solved. For example, is a specific computation dependent only on the characteristics of the piecewise polynomial, and independent of the differential operator and the boundary conditions? How much of the computation can be salvaged if only part of the problem is changed and how can that amount be maximized for a given change? Answers to questions such as these are important in the design and implementation of efficient programs. In this chapter we identify these various sub-tasks and indicate which parts of the generation procedure can be isolated as separate modules. The equation generation phase is itself inherently modular, even though in its entirety it is usually regarded as the second of three stages in the application of the finite element method. The first phase is the mesh generation, and the third is the solution of the generated algebraic system.

As we stated in Chapter 1, the finite element method is a Ritz-Galerkin method where the trial functions have small support. That is, the approximate solution is represented in terms of a local basis. Generation of such a basis for rectangular domains is fairly straightforward, as we described in Section 1.4. However, for domains of arbitrary shape, where it is not convenient or possible to restrict the support of the basis functions to rectangles, a different approach is necessary, and is provided by the use of so-called interpolation polynomials [F1, Z1]. The construction of such polynomials and their relationship to the local basis is the subject of Section 2.

Once we have the basis for our approximate solution $v(x,y)$, the next step is to carry out the integrations required to obtain the coefficients of the linear system, as described in Chapter 1, Section 2. We emphasize that the computational procedure is considerably different from the formal description appearing in Chapter 1. The integrations required to determine the coefficients are carried out element-by-element, and the actual basis functions are not (explicitly) generated at all. This computation, where the equations are actually generated, is the subject of Section 3.

The last part of the generation procedure is usually referred to as assembly of the equations, or just "assembly", and is the subject of Section 4. Suppose our (linear) elliptic boundary value problem is cast in a variational form, with a functional $I[v]$ that we wish to minimize with respect to the parameters of v . The result of the element-by-element process described in Section 3 is a set of small quadratic functions, each one representing a contribution to $I[v]$ of a particular subdomain (element) of the domain R . These small functions have some parameters in common,

and the process of combining these functions into a single large one is the task referred to as "assembly". The elimination of parameters whose values are determined by boundary conditions is also done at this stage.

2. Construction of Interpolating Polynomials

In this section we describe the construction of interpolating polynomials on triangles. However, the procedure and many of our remarks apply for a general polygon. Let R be a simply or multiply connected domain in the (x,y) plane with piecewise linear boundary ∂R. Zlámal [Z6] has described a method for removing this restriction on ∂R. We assume R has been triangulated into N_Δ triangles, with adjacent triangles having either a common vertex or a common side and with the union of the closed triangles equal to $R \cup \partial R$. An example of a domain triangulated in this way appears below.

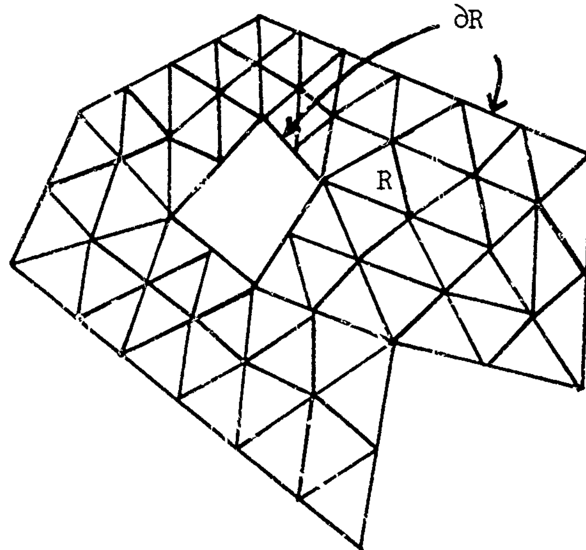


Figure 2.1

Our aim is to construct a piecewise polynomial of degree d on $R \cup \partial R$. To do this we assume that on each triangle $T^\nu \subset R \cup \partial R$, $v(x,y)$ is a polynomial $p^\nu(x,y)$ of degree d. We impose the conditions that p^ν and p^γ on neighboring triangles have common values and/or

derivatives at node points lying on their common boundary. We begin by studying the choice of parameters necessary to have $v(x,y)$ of class $C^{(\sigma)}$. This problem has also been considered in [H2] for general polygons, and we give a special case of their arguments below.

Consider the figure below, depicting two adjacent triangles T^{ν} and T^{γ} having common boundary L . Directions tangent and normal to L will be denoted respectively by s and n . Thus $\frac{\partial v}{\partial n}(Q_1)$ is the derivative of v normal to L evaluated at Q_1 . The notation $v(s)$ will mean the function v evaluated at the point $Q_1 + s(Q_2-Q_1)$.

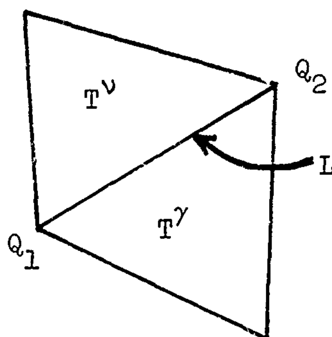


Figure 2.2

Suppose we require that

$$(2.1) \qquad D^{\tau}p^{\nu}(Q_i) = D^{\tau}p^{\gamma}(Q_i) \quad , \quad i = 1,2 \; , \quad |\tau| \le \beta \; ,$$

where $\tau = (\tau_1,\tau_2)$, $|\tau| = \tau_1+\tau_2$, and $D^{\tau}v = \frac{\partial^{|\tau|}v}{\partial^{\tau_1}x\,\partial^{\tau_2}y}$. Then

$$(2.2) \qquad \frac{\partial^k p^{\nu}(Q_i)}{\partial s^k} = \frac{\partial^k p^{\gamma}(Q_i)}{\partial s^k} \quad , \quad i = 1,2 \quad , \quad k \le \beta \; ,$$

which implies $v(s)$ will be continuous along L if

$$(2.3) \qquad d+1 = 2(\beta+1) \quad .$$

If (2.3) is satisfied, $\frac{\partial^k p^\nu}{\partial n^k}$ is a polynomial of degree $d-k$ in s, having $d-k+1$ coefficients. Thus we require $d-k+1$ conditions of agreement between $\frac{\partial^k p^\nu}{\partial n^k}$ and $\frac{\partial^k p^\gamma}{\partial n^k}$ along L if $\frac{\partial^k v}{\partial n^k}$ is to be continuous along L. The conditions (2.2) imply

$$(2.4) \qquad \frac{\partial^j p^\nu(Q_i)}{\partial n^j} = \frac{\partial^j p^\gamma(Q_i)}{\partial n^j} \quad , \quad i = 1,2 \quad , \quad j \leq \beta-k \; ,$$

imposing $2(\beta-k+1)$ conditions. Therefore, we need $d-k+1-2(\beta-k+1) = 2(\beta+1)-k-2(\beta-k+1) = k$ more conditions of agreement imposed on $\frac{\partial^k p^\nu}{\partial n^k}$ and $\frac{\partial^k p^\gamma}{\partial n^k}$. Carrying out the same arguments for $k = 1,2,\ldots,\sigma$ and summing implies we need $\sigma(\sigma+1)/2$ additional "normal derivative" parameters situated at nodes along L if v is to be of class $C^{(\sigma)}$ along L. Using the fact that the number of coefficients in a general d-th degree polynomial in two variables is $n_d = (d+1)(d+2)/2$ yields the inequality

$$(2.5) \qquad \frac{(d+1)(d+2)}{2} \geq 3\left\{\frac{(\beta+1)(\beta+2)}{2} + \frac{\sigma(\sigma+1)}{2}\right\} \quad ,$$

where the term $\frac{(\beta+1)(\beta+2)}{2}$ is the number of derivatives $D^\tau v$, $|\tau| \leq \beta$. The factor 3 appears because a triangle has 3 sides and 3 vertices. The inequality (2.5) yields the conditions

$$(2.6) \qquad \beta(\beta+1) \geq 3\sigma(\sigma+1) \quad \text{and} \quad d = 2\beta+1 \quad .$$

Surplus degrees of freedom in the polynomial can be associated with nodes in the interior of the triangle. For approximation properties of these piecewise polynomials see [Z1,Z4,Z5].

The conditions (2.6) imply, in particular, that we require d to be at least 5 {9} for v to be in $C^{(1)}$ $\{C^{(2)}\}$. Note that this applies only to the polynomials described above. A common technique used to reduce the number of parameters in the problem is to restrict the polynomial of degree d on each triangle to be of degree $d-k$, $k > 0$ in parts of the triangle. For example, Goël [G2] begins with the 3-4 element (Appendix A) and by a suitable modification forces the normal derivative to each side of the triangle to vary linearly along the boundary. Agreement in value and first derivatives at the vertices Q_1 and Q_2 guarantees continuity in the first derivatives along L. Zienkiewicz [Z3] and Clough and Tocher [C1] also present techniques for achieving the same goal. Irons [I1] describes a method for constructing a quartic element generating a piecewise polynomial subspace $v \in C^{(1)}$. Bell [B6] describes a method for eliminating the side parameters on the 5-6 element by imposing the condition that the derivative of the polynomial normal to each triangle side be a cubic rather than a quartic. Zlámal [Z6] uses a similar technique to eliminate the centroid parameter from element 3-4.

We will refer to elements of the type just described as <u>deficient elements</u>, to distinguish them from elements which are polynomials of a particular degree over the whole triangle. We have restricted our studies in this thesis to non-deficient elements. (An explanation appears at the end of this section.)

We now turn to the actual construction of interpolating polynomials. Let the number of nodes associated with each triangle be $m = 3(m_S+1)+m_I$, where $m_S \geq 0$ is the number of nodes on each triangle side (not including the endpoints), and let $m_I \geq 0$ be the number of nodes in the interior of each triangle. We denote the total number of nodes in the domain by M , and the coordinates of the nodes by $Q_i = (x_i,y_i)$, $i = 1,2,\ldots,M$. The indices of the nodes of triangle T^ν will be denoted by $\nu_1,\nu_2,\ldots,\nu_m$, with the vertex nodes coming first in counterclockwise order, followed by the side nodes also in counterclockwise order, followed by the interior nodes (in no specific order). When $m_S > 0$ we assume that the side nodes evenly sub-divide the triangle sides. Triangle T^ν is depicted in Figure 2.3 below.

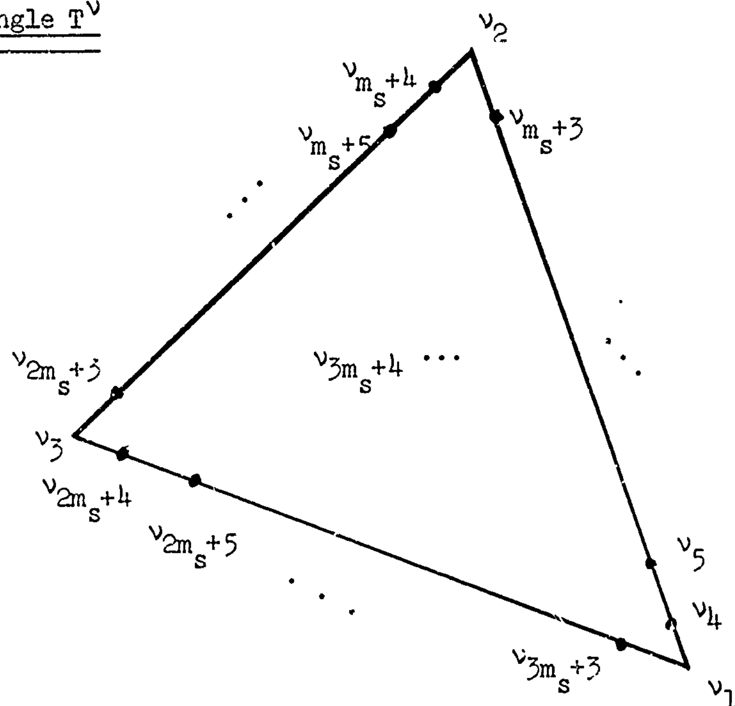


Figure 2.3

To simplify the notation in the sequel, we will assume that $\nu_i = i$. We begin by expressing $p^\nu(x,y)$ in the form

$$(2.7) \qquad p^\nu(x,y) = \sum_{k=1}^{n_d} \alpha_k^\nu \varphi_k^{(d)} = \alpha^{\nu^T} \varphi^{(d)} = \varphi^{(d)^T} \alpha^\nu \quad ,$$

where $\varphi^{(d)}$ is the n_d-dimensional column vector whose elements are the monomial terms of the general d-th degree polynomial in two variables. We assume that the terms appear in order of non-decreasing degree, and in increasing powers of y for consecutive terms of the same degree. For example,

$$(2.8) \qquad \varphi^{(3)^T} = (1,x,y,x^2,xy,y^2,x^3,x^2y,xy^2,y^3) \quad .$$

The superscript d will not usually be included explicitly. The vector α^ν contains the coefficients of p^ν , and α_k^ν and $\varphi_k^{(d)}$ refer to the k-th components of α^ν and $\varphi^{(d)}$ respectively.

Now our goal is to represent p^ν on T^ν in terms of its <u>nodal parameters</u>. For example, if $d = 1$ $(n_d = 3)$, p^ν can be uniquely characterized by its <u>values</u> at the vertices of T^ν . If p^ν is a cubic polynomial $(n_d = 10)$, one way to characterize it is by the parameters $D^\tau p^\nu(Q_i)$, $i = 1,2,3,$ $|\tau| \leq 1$, and $p^\nu(Q_4)$, where Q_4 is at the centroid of T^ν . Note that both of these characterizations assure continuity across interelement boundaries, as predicted by the theory presented in the first part of this section.

We denote the number of parameters associated with node i by μ_i , and the vector of those parameters by q_i . Its j-th element will be denoted by $q_{i,j}$. The parameters associated with $p^\nu(x,y)$, ordered as

indicated by Figure 2.3, are then given by

$$q^{\nu^T} = (q_1^T, q_2^T, \ldots, q_m^T) \quad . \tag{2.9}$$

Now suppose $\mathcal{L}_i$ is a column vector of length μ_i whose elements are linear functionals designed to produce the parameters associated with node i when it is applied to v. For example, the vertex nodes for the cubic polynomial discussed above would have associated vector functionals of the form

$$\mathcal{L}_i[f] = \begin{pmatrix} f(Q_i) \\ f_x(Q_i) \\ f_y(Q_i) \end{pmatrix} \quad . \tag{2.10}$$

Such an operator applied to a j-dimensional vector is understood to operate term by term; a column vector would yield a $\mu_i j$ -dimensional column vector, a row vector would produce a μ_i by j matrix. Defining $\mathcal{L}^\nu$ by

$$\mathcal{L}^{\nu^T} = (\mathcal{L}_1^T, \mathcal{L}_2^T, \mathcal{L}_3^T, \ldots, \mathcal{L}_m^T) \quad , \tag{2.11}$$

we have immediately the identity

$$\mathcal{L}^\nu[v] = q^\nu \quad . \tag{2.12}$$

Using (2.7) along with the fact that v is p^ν on T^ν, we can rewrite (2.12) as a matrix equation involving α^ν and q^ν:

$$\begin{aligned} q^\nu &= \mathcal{L}^\nu[p^\nu] \\ &= \mathcal{L}^\nu[\varphi^T\alpha^\nu] \\ &= \mathcal{L}^\nu[\varphi^T]\alpha^\nu \\ &= C^\nu\alpha^\nu \quad . \end{aligned} \tag{2.13}$$

As we stated in Section 3.2, we are restricting our basis to be polynomials of a specific degree on each element (non-deficient elements) so we assume that $n_d = \sum_{\ell=1}^{m} \mu_\ell$. C^ν will be non-singular provided our node points are distinct and our parameters associated with each node point are linearly independent.

Using (2.13) in (2.7) yields

$$\text{(2.14)} \qquad p^\nu(x,y) = \alpha^{\nu^T} \varphi = q^{\nu^T} C^{\nu^{-T}} \varphi \qquad ,$$

giving the polynomial on T^ν in terms of the parameters which we have chosen to characterize it. Here the notation $C^{\nu^{-T}}$ means $(C^{\nu^{-1}})^T$.

Consider again the cubic example discussed above on triangle T^ν having vertices $Q_i = (x_i, y_i)$, $i = 1,2,3$, and centroid $Q_4 = (x_4, y_4)$. Thus $\mu_i = 3$, $i = 1,2,3$, and $\mu_4 = 1$. Then q^ν is

$$\text{(2.15)} \qquad q^{\nu^T} = (v_1, v_{1,x}, v_{1,y}, v_2, v_{2,x}, v_{2,y}, v_3, v_{3,x}, v_{3,y}, v_4) \quad ,$$

where $v_{i,t}$ denotes the first partial derivative of v with respect to t at the point $Q_i = (x_i, y_i)$. The matrix C^ν is

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 & x_1^3 & x_1^2y & x_1y_1^2 & y_1^3 \\ 0 & 1 & 0 & 2x_1 & y_1 & 0 & 3x_1^2 & 2x_1y_1 & y_1^2 & 0 \\ 0 & 0 & 1 & 0 & x_1 & 2y_1 & 0 & x_1^2 & 2x_1y_1 & 3y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 & x_2^2 & x_2^2y_2 & x_2y_2^2 & y_2^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & 0 & x_3 & 2y_3 & 0 & x_3^2 & 2x_3y_3 & 3y_3^2 \\ 1 & x_4 & y_4 & x_4^2 & x_4y_4 & y_4^2 & x_4^3 & x_4^2y_4 & x_4y_4^2 & y_4^3 \end{bmatrix} .$$

We can write (2.14) in the form

$$(2.16) \qquad p^{\nu}(x,y) = \sum_{k=1}^{n_d} q_k^{\nu}\left(\sum_{j=1}^{n_d} C_{kj}^{\nu^{-T}} \varphi_j\right) = \sum_{k=1}^{n_d} q_k^{\nu} \psi_k^{\nu}$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{\mu_i} q_{i,j}\psi_{i,j}(x,y) \quad ,$$

where $\psi_{k,j}$ is associated with the <u>j-th parameter of node k</u>. The ψ's are often referred to as <u>coordinate functions</u> in the engineering literature, and are the members of the local basis in terms of which $v(x,y)$ is being represented. It is easy to verify that $\psi_{k,j}$ vanishes on the boundary of the union of the triangles to which node k belongs, provided the nodal parameters have been chosen to guarantee continuity across interelement boundaries. The function $\psi_{k,j}$ is defined to be zero outside the region. There will obviously be $N = \sum_{k=1}^{M} \mu_k$ parameters and corresponding basis functions in the representation of v on R.

The procedure we have described for generating the basis functions is in a sense quite general. The generation of the matrix C^{ν} can be isolated in a subroutine, and the only required input is

(i) the degree d of the polynomial,

(ii) the node coordinates (or a formula for generating them),

(iii) the nodal parameters.

Item (ii) is supplied by the mesh, while (i) and (iii) can be specified by the user. Each row of C^{ν} is obtained by evaluating the components of φ at the corresponding node coordinate (perhaps after differentiating them, if the corresponding parameter is a derivative parameter). Observe that the matrix terms are simple monomial terms of the form $x_i^{\ell_1} y_i^{\ell_2}$, $\ell_1, \ell_2 \geq 0$.

Differentiation of them can be easily done symbolically, with obvious computational advantages. Furthermore, entries in each row will have common factors of the form x_i^ℓ and y_i^ℓ, $\ell \geq 0$. Thus, the generation of C^ν can be implemented in an efficient as well as general way.

Provided we choose linearly independent parameters equal in number to $(d+1)(d+2)/2$, the matrix C^ν will be non-singular and we can obtain the basis functions on T^ν in the factored form $\psi_i^\nu = \{C^{\nu^{-T}} \varphi\}_i$. Having $p^\nu(x,y)$ in the form (2.14) is particularly convenient for our intended applications. Anticipating the next section, observe that on triangle T^ν the following equation holds

$$(2.17) \qquad D^\tau p^\nu(x,y) = q^\nu C^{\nu^{-T}} D^\tau \varphi \quad ,$$

where the differential operator D^τ operates term by term on φ. Thus, if the basis functions are derived in this factored form, derivatives of the trial function v can be easily obtained <u>symbolically</u>. Furthermore, for two differential operators D^τ and D^σ, expressions of the form $D^\tau p^\nu(x,y) D^\sigma p^\nu(x,y)$ become

$$(2.18) \qquad q^{\nu^T} C^{\nu^{-1}} [D^\tau \varphi \{D^\sigma \varphi\}^T] C^{\nu^{-T}} q^\nu \quad ,$$

and again the matrix in the square brackets can be obtained symbolically. Its terms are monomial terms of the form $x^{\ell_1} y^{\ell_2}$, $\ell_1, \ell_2 \geq 0$.

Note that the ease with which we can manipulate the basis functions depends upon being able to express each basis (coordinate) function as a linear combination of monomials. For some deficient elements this is not possible, and differentiation and integration of the coordinate functions must be done numerically [H1] and/or carried out by hand and

programmed explicitly. This would not be particularly disadvantageous for a special program designed to solve a specific class of problems. Also, in a production setting, many of the computations involving the functions can be done once and the results stored in a library. However, from our point of view of designing a general purpose program we have favored the use of non-deficient elements, which guarantee the invertibility of C^{ν} and the representation of the basis functions as linear combinations of monomial terms.

3. Generation of the Equations

This section describes the actual calculation of the finite element equations once we have an expression for our piecewise polynomial as discussed in the previous section. Suppose our problem is cast in a variational form, and we wish to minimize a functional $I[v] = I_R[v]+I_{\partial R}[v]$ with respect to the parameters of v, where

$$(3.1) \qquad I_R[v] = \iint_I (a_1 v_x^2 + a_2 v_x v_y + a_3 v_y^2 + a_4 v^2 + a_5 v)\, dx\, dy \quad ,$$

$$(3.2) \qquad I_{\partial R}[v] = \int_{\partial R} (a_6 v^2 + a_7 v)\, ds \quad ,$$

and v is restricted to satisfy a linear boundary condition of the form

$$(3.3) \qquad a_8 v + a_9 v_n + a_{10} v_s = a_{11} \quad \text{on} \quad \partial R \; .$$

Here a_i, $i = 1,2,\ldots,11$ are functions of x and y, and v_n and v_s are the (inward) normal derivative and (counter-clockwise) tangential derivative of v on ∂R.

Our interest here is in the implementation; consequently, we will not concern ourselves with the range of boundary value problems that can be covered by the above form, or relations and/or smoothness that the functions a_j, $j = 1,2,\ldots,11$ and v must possess in order for the problem to be correctly formulated. Also, we do not mean to imply that the procedure to be described applies only to the above functional. It will be clear that the construction applies to other quadratic integrands (involving derivatives of higher order, for example).

We begin by observing that $I[v]$ can be expressed as a sum of the

contributions from each triangle $T^\nu \subset R \cup \partial R$. Thus we can write

$$(3.4) \qquad I[v] = \sum_{\nu=1}^{N_\Delta} I^\nu[v] = \sum_{\nu=1}^{N_\Delta} (I_R^\nu[v] + I_{\partial R}^\nu[v])$$

where $I_R^\nu[v]$ has the form (3.1) with the domain of integration replaced by T^ν , and $I_{\partial R}^\nu[v]$ has the form (3.2) with the contour of integration ∂R replaced by ∂R^ν , the part of T^ν lying on ∂R . For T^ν with no side on ∂R , $I_{\partial R}^\nu[v]$ is obviously zero and does not have to be considered. The basic procedure is to obtain expressions for each term of the summation (3.4) as functions of the parameters of v .

Consider first the term $I^\nu[v]$ corresponding to triangle T^ν :

$$(3.5) \qquad I_R^\nu[v] = \iint_{T^\nu} (a_1 v_x^2 + a_2 v_x v_y + a_3 v_y^2 + a_4 v^2 + a_5 v)\,dx\,dy \quad .$$

Recall from Section 2 that our expression for $p^\nu(x,y)$ on T^ν could be written in the form

$$(3.6) \qquad p^\nu(x,y) = q^{\nu^T} C^{\nu^{-T}} \varphi \quad ,$$

and we observed in Section 3.2 that $D^\tau p^\nu = q^\nu C^{\nu^{-T}} D^\tau \varphi$, where the operator D^τ operates on the column vector φ term by term. Substituting (3.6) into (3.5), we obtain the following expression for the first four (quadratic) terms of (3.5):

$$(3.7) \qquad q^{\nu^T} C^{\nu^{-T}} \left\{ \iint_{T^\nu} a_1 \varphi_x \varphi_x^T + a_2 \varphi_x \varphi_y^T + a_3 \varphi_y \varphi_y^T + a_4 \varphi\varphi^T \,dx\,dy \right\} C^{-1} q^\nu \quad .$$

Deferring treatment of the last term in (3.5) until later, suppose T^ν has one or more sides lying on ∂R and denote that segment of ∂R

by ∂R^ν . Then we have

$$(3.8) \qquad I^\nu_{\partial R}[v] = \int_{\partial R^\nu} (a_6 v^2 + a_7 v) ds \quad ,$$

and again using (3.6), we obtain the following quadratic function from the first term in the integrand of (3.8):

$$(3.9) \qquad q^{\nu^T} C^{\nu^{-T}} \left\{ \int_{\partial R^\nu} a_6 \varphi\varphi^T ds \quad C^{\nu^{-1}} q^\nu \right\}.$$

We will denote the sum of the matrices in braces in (3.7) and (3.9) by H^ν . The so-called stiffness matrix is then given by

$$(3.10) \qquad A^\nu = C^{\nu^{-T}} H^\nu C^{\nu^{-1}} \quad ,$$

and the quadratic terms of $I^\nu_R[v]$ yield the function $q^{\nu^T} A^\nu q^\nu$.

Turning now to the linear terms in $I^\nu_R[v]$ and $I^\nu_{\partial R}[v]$ we obtain, using exactly the same procedure, the expression

$$(3.11) \qquad q^{\nu^T} C^{\nu^{-T}} \left\{ \iint_{T^\nu} a_5 \varphi \, dx\, dy + \int_{\partial R^\nu} a_7 \varphi \, ds \right\} \quad .$$

Denoting the vector in braces by w^ν , the linear terms in $I^\nu[v] = I^\nu_R[v] + I^\nu_{\partial R}[v]$ yield

$$(3.12) \qquad q^{\nu^T} C^{\nu^{-T}} w^\nu = q^{\nu^T} b^\nu \quad ,$$

where the vector b^ν is usually referred to as a load vector by engineers.

Repeating the above procedure for each triangle T^ν , $\nu = 1,2,\ldots,N_\Delta$, we obtain finally

$$(3.13) \qquad I[v] = \sum_{\nu=1}^{N_\Delta} (q^{\nu^T} A^\nu q^\nu + q^{\nu^T} b^\nu) \quad ,$$

where we note that there will be parameters q_j^ν common to more than one of the terms of the summation.

If we assume all our boundary conditions are natural (i.e., they are satisfied automatically because of the design of the functional being minimized), then (3.3) is null, and our approximate solution is obtained by minimizing (3.13) with respect to the q^ν's . That is, we satisfy

$$(3.14) \qquad \sum_{\nu=1}^{N_\Delta} \left\{ (A^\nu + A^{\nu^T}) q^\nu + b^\nu \right\} = 0 \quad .$$

If v must satisfy some boundary conditions of the form (3.3), then some of the q^ν's are constrained to assume certain values or satisfy certain relations. This entire assembly problem and incorporation of boundary conditions is examined in the next section.

We now examine the details of implementation of the procedure outlined in (3.6)-(3.12). To reduce the amount of computation that must be done for each triangle, it is convenient to confine as much of the computation as possible to a standard canonical triangle T^o for which part of the computation can be done once and for all. The savings that can be realized depend rather heavily on whether the coefficients of the functional a_i , $i = 1,2,\ldots,7$ are constants or variable. The following scheme has been described for particular problem-element combinations by Zlámal [Z5], Dupuis and Goël [D3] and others.

Let T^o have vertices (0,0), (1,0) and (0,1) . Then the linear transformation mapping T^o (ξ-η plane) onto T^ν (x-y plane) having

vertices (x_i, y_i) $i = 1,2,3$ is

$$(3.15) \quad \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} \Delta x_1 & -\Delta x_3 \\ \Delta y_1 & -\Delta y_3 \end{pmatrix} \begin{pmatrix} \xi \\ \eta \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + J^\nu \begin{pmatrix} \xi \\ \eta \end{pmatrix} ,$$

where $\Delta x_i = x_{i+1} - x_i$ with the subscripts interpreted modulo 3. The inverse mapping is then

$$(3.16) \quad \begin{pmatrix} \xi \\ \eta \end{pmatrix} = \frac{1}{|J^\nu|} \begin{pmatrix} -\Delta y_3 & \Delta x_3 \\ -\Delta y_1 & \Delta x_1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} - \frac{1}{|J^\nu|} \begin{pmatrix} \Delta x_3 y_1 - \Delta y_3 x_1 \\ \Delta x_1 y_1 - \Delta y_1 x_1 \end{pmatrix} ,$$

where $|J^\nu|$ is the determinant of J^ν. Note that

$$(3.17) \quad \iint_{T^\nu} w(x,y)\,dx\,dy = \iint_{T^0} w(x(\xi),y(\eta))\,|J^\nu|\,d\xi\,d\eta \quad .$$

Now define the quantities $\tilde{q}$, $\tilde{\varphi}$, and $\tilde{C}$ on T^0 (in the ξ-η plane) in exactly the same way as their counterparts were defined in the x-y plane. For example,

$$(3.18) \quad \tilde{\varphi}^{(3)} = (1,\xi,\eta,\xi^2,\xi\eta,\xi\eta^2,\xi^3,\xi^2\eta,\xi\eta^2,\eta^3) \quad .$$

Using (3.16) and (3.17), the integrals (3.5) and (3.8) can be expressed in the form

$$(3.19) \quad \iint_{T^0} (g_1^\nu w_\xi^2 + g_2^\nu w_\xi w_\eta + g_3^\nu w_\eta^2 + g_4^\nu w^2 + g_5^\nu w)\,|J^\nu|\,d\xi\,d\eta \quad ,$$

and

$$(3.20) \quad \int_{\partial T^0} (g_6^\nu w^2 + g_7^\nu w)\,|J^\nu|\,ds \quad ,$$

where $w(\xi,\eta) = v(x(\xi,\eta),y(\xi,\eta))$, and the g_i^ν 's will depend not only on their respective a_i 's, but also upon the mapping and the other terms in the functional. The contour integral (3.20) is understood to apply only to the part of T^o corresponding to ∂R^ν . Again it is convenient to collect the linear and quadratic terms together. Carrying out the above procedure for the quadratic terms in (3.19) and (3.20) we obtain

$$(3.21) \qquad \tilde{q}^T \tilde{C}^{-T} \left\{ \iint_{T^o} (g_1^\nu \tilde{\varphi}_\xi \tilde{\varphi}_\xi^T + g_2^\nu \tilde{\varphi}_\xi \tilde{\varphi}_\eta^T + g_3^\nu \tilde{\varphi}_\eta \tilde{\varphi}_\eta^T + g_4^\nu \tilde{\varphi} \tilde{\varphi}^T) |J^\nu| \, d\xi \, d\eta + \int_{\partial T^o} (g_6^\nu \tilde{\varphi} \tilde{\varphi}^T \; |J^\nu|) ds \right\} \tilde{C}^{-1} \tilde{q} \quad .$$

The expression in braces in (3.21) is the ξ-η counterpart of the matrix H^ν defined above, and we denote it by $\tilde{H}^\nu$. Then the ξ-η counterpart of A^ν is given by

$$(3.22) \qquad \tilde{A}^\nu = \tilde{C}^{-T} \tilde{H}^\nu \tilde{C}^{-1} \quad .$$

The linear terms of (3.19)-(3.20) yield

$$(3.23) \qquad \tilde{q}^T \tilde{C}^{-T} \left\{ \iint_{T^o} g_5^\nu \, \varphi |J^\nu| \, d\xi \, d\eta + \int_{\partial T^o} g_7^\nu \, \varphi \, |J^\nu| ds \right\} \quad ,$$

or

$$(3.24) \qquad \tilde{q}^T \tilde{C}^{-T} \tilde{w} = \tilde{q}^T \tilde{b} \quad .$$

Finally, since we wish $I^\nu[v]$ to be expressed in terms of the parameters in the x-y plane rather than the ξ-η plane, we must apply a transformation derived from (3.15) to $\tilde{A}^\nu$ and $\tilde{b}$. Specifically, using

(3.15), we can easily construct a block diagonal matrix K satisfying

$$\tilde{q} = K^{\nu} q^{\nu} \quad , \tag{3.25}$$

from which we can get, by substitution of (3.25) into (3.22) and (3.24), the following

$$A^{\nu} = K^{\nu^T} \tilde{A}^{\nu} K^{\nu} \quad , \quad b^{\nu} = K^{\nu^T} \tilde{b}^{\nu} \tag{3.26}$$

The following points are important in the implementation of the above.

(i) If the coefficients of the quadratic terms in the functional are constants (or at least constant over each triangle), then the corresponding g^{ν}'s will be constant over the triangles. Thus $\tilde{A}^{\nu}$ can be expressed as the sum of matrices of the form $\gamma_1 G_1 + \gamma_2 G_2 + \gamma_3 G_3 + \gamma_4 G_4$, where the G_i's are independent of ν, (and thus need to be computed once), and $\gamma_i = \gamma_i(\nu)$. For example, the first term would be

$$\underbrace{g_1^{\nu} \, |J^{\nu}|}_{\gamma_1(\nu)} \; \underbrace{\tilde{C}^{-T} \iint_{T^o} \tilde{\varphi}_{\xi} \, \tilde{\varphi}_{\xi}^T \, d\xi \, d\eta \quad \tilde{C}^{-1}}_{G_1} \tag{3.27}$$

The generation of the G_i's can be done very efficiently as follows. First we compute

$$\vartheta_{ij} = \iint_{T^o} \xi^i \eta^j \, d\xi \, d\eta = i!j!/(i+j+2)! \quad , \tag{3.28}$$

for all i and j less than μ, where μ depends upon d and the terms in the functional. The components of the integral are then $I_{r_1 r_2}$, where

r_1 and r_2 are simple integer functions. When g_i^ν is not constant over each triangle, numerical integration will probably be necessary to evaluate the expressions in the braces in (3.21) and (3.23). Even in this instance, having the basis functions in the form (2.14) is still very convenient, since it allows us to compute the integrand at the evaluation points very efficiently. For example, consider evaluating the i,j-th component of the integrand of the first bracketed integral in (3.21) at the point (ξ_μ, η_μ). The function to be evaluated will have the form

$$(3.29) \qquad g_1^\nu \xi_\mu^{\ell_1 - 2} \eta_\mu^{\ell_2} + g_2^\nu \xi_\mu^{\ell_1 - 1} \eta_\mu^{\ell_2 - 1} + g_3^\nu \xi_\mu^{\ell_1} \eta_\mu^{\ell_2 - 2} + g_4^\nu \xi_\mu^{\ell_1} \eta_\mu^{\ell_2} ,$$

$$\ell_1, \ell_2 \geq 0 .$$

Assuming we have the basis functions in a convenient symbolic form, the evaluation of the integrand can be optimized considerably by precomputing the common factor $\xi_\mu^{\ell_1 - 2} \eta_\mu^{\ell_2 - 2}$.

(ii) The matrix $\tilde{C}$ and its LU decomposition need only be computed once, since $\tilde{C}$ is independent of ν.

(iii) The computation done so far has been independent of the boundary conditions (3.13). Thus a change in them would not require re-computation of the A^ν and b^ν, $\nu = 1,2,\ldots,N_\Delta$. Also note that changes in a_5 and a_7 would not change A^ν, $\nu = 1,2,\ldots,N_\Delta$

(iv) Consider the calculation represented by (3.22), and denote n_d by n. Normally, one would expect the congruence transformation to require $2n^3 + O(n^2)$ multiplicative operations, since we need to perform $2n$ back-solves, each requiring $n^2 + O(n)$ operations. We will show how to reduce the computation to $\frac{7}{6} n^3 + O(n^2)$ under the assumption that $\tilde{H}^\nu$ is symmetric. [Equation (3.14) above implies that we only need $\tilde{A}^\nu + \tilde{A}^{\nu^T}$; therefore, if $\tilde{H}^\nu$ is not symmetric, we can compute $\tilde{C}^{-T}(\tilde{H}^\nu + \tilde{H}^{\nu^T})\tilde{C}^{-1}$]. The following technique has also been used in [M2] in connection with solving generalized eigenvalue problems.

Suppose we have the LU decomposition of $\tilde{C}^T$. Then the basic procedure is

(a) Solve $LU\, W = \tilde{H}^\nu$,

(b) Solve $LU\, \tilde{A}^\nu = W^T$.

Consider step (a). Suppose we compute only the <u>lower triangle</u> of W; i.e., we do not complete the U-solve, so that W has the form $\begin{pmatrix} * & 0 \\ * & * \end{pmatrix}$. It is easy to show that now the calculation of W requires the following number of multiplicative operations:

$$\mathcal{O}_a = n \cdot \frac{n^2}{2} + \sum_{i=1}^{n} \frac{i(i+1)}{2} = \frac{n^3}{2} + \frac{n^3}{6} + O(n^2) = \frac{2}{3} n^3 + O(n^2) .$$

Now consider step (b). We use the following notation to indicate partitions of L, U and C^T, where the upper left partition is k by k:

$$\tilde{C}^T = \left(\begin{array}{c|c} C_1^k & C_2^k \\ \hline C_3^k & C_4^k \end{array}\right), \quad L = \left(\begin{array}{c|c} L_1^k & \\ \hline L_3^k & L_4^k \end{array}\right), \quad U = \left(\begin{array}{c|c} U_1^k & U_2^k \\ \hline & U_4^k \end{array}\right).$$

We will denote the i-th column of $\tilde{A}^{\nu}$ by a_i, its first k elements by a_i^k, and its last $n-k$ elements by $a_i^{k'}$. The first i elements of the i-th row of W will be denoted by w_i. Then step (b) can be described as follows: For $k = n, n-1, \ldots, 1$ compute

$$L_1^k U_1^k a_k^k = w_k - C_2^k a_k^{k'} \quad .$$

The first step yields the <u>last row and column</u> of A; the next step yields the remaining unknown parts of the (n-1)-st row and column and so on. Note that at each stage the vector $a_k^{k'}$ has already been computed by previous steps. Here we use the fact that $L_1^k U_1^k = C_1^k$. The number of multiplicative operations $\mathcal{O}_b$ required for step (b) is given by

$$\mathcal{O}_b = \sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i(n-i) = n \sum_{i=1}^{n} i = \frac{n^3}{2} + O(n^2) \quad .$$

Thus, the total computation required for the congruence transformation has been reduced from $2n^3 + O(n^2)$ to $\mathcal{O}_a + \mathcal{O}_b = \frac{7}{6} n^3 + O(n^2)$.

When the coefficients of the quadratic terms are constants, this technique will not be too important since the number of such congruence transformations will be small. The computation of the G matrices discussed above is initialization, and for $N_\Delta \gg n_d$, the work required for equation generation is essentially proportional to $N_\Delta n_d^2$. However, if one or more of the quadratic coefficients is variable, a congruence transformation must be done for each triangle, and using this technique saves $\frac{5}{6} n_d^3 N_\Delta$ multiplicative operations.

The equation generation can be summarized as follows:

Step 1 (Initialization)

(i) Compute $\tilde{C}$ and its LU decomposition.

(ii) If all the quadratic terms have constant coefficients then compute the appropriate G matrices and store them.

Step 2

For each triangle T^{ν} do the following:

(iii) Compute the mapping from T^{0} to T^{ν}, and generate the quantities $|J^{\nu}|$ and g_j^{ν}.

(iv) Generate $\tilde{A}^{\nu}$ and $\tilde{b}^{\nu}$.

(v) Apply the transformation K^{ν} to $\tilde{A}^{\nu}$ and $\tilde{b}^{\nu}$ to obtain A^{ν} and b^{ν}.

4. Assembly of the Equations

Having completed the procedure described in Section 3.3 for each triangle, we have a system of the following form to solve:

$$\text{(4.1)} \qquad \sum_{\nu=1}^{N_\Delta} \left\{ (A^\nu + A^{\nu^T}) q^\nu + b^\nu \right\} = 0 \quad ,$$

or

$$\sum_{\nu=1}^{N_\Delta} \left\{ B^\nu q^\nu + b^\nu \right\} = 0 \quad .$$

Combining the terms in (4.1), and renumbering the q_i^ν's and b_i^ν's from 1 to N, we obtain the system

$$\text{(4.2)} \qquad Aq = b \quad .$$

As we pointed out in the previous section, if boundary conditions of the form (3.3) are imposed, then some of the elements of q will be required to assume specific values or satisfy specific relations.

Suppose first that the boundary conditions only impose constraints on single parameters, rather than specifying relations that must hold between several parameters. Partitioning q into q_1 and q_2, equation (4.2) can be written in the form:

$$\text{(4.3)} \qquad \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad .$$

Now if q_2 must satisfy $q_2 = \bar{q}_2$, we can solve

$$\text{(4.4)} \qquad A_{11} q_1 = b_1 - A_{12} q_2 \quad .$$

As Felippa and Clough [F1] point out, in order to avoid rearranging equations, we would actually solve the following system in some permuted form

$$(4.5)\qquad \begin{pmatrix} A_{11} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = \begin{pmatrix} b_1 - A_{12}\bar{q}_2 \\ \bar{q}_2 \end{pmatrix} \quad .$$

We shall see in Chapter 4 that if we use "profile" methods, this practice costs almost no storage or computation. We denote this system by

$$(4.6)\qquad A'q' = b' \quad .$$

Now suppose further that the boundary conditions impose some general linear constraints on the solution of (4.6). As an example, we appeal to our cubic element 3-4 and the diagram below:

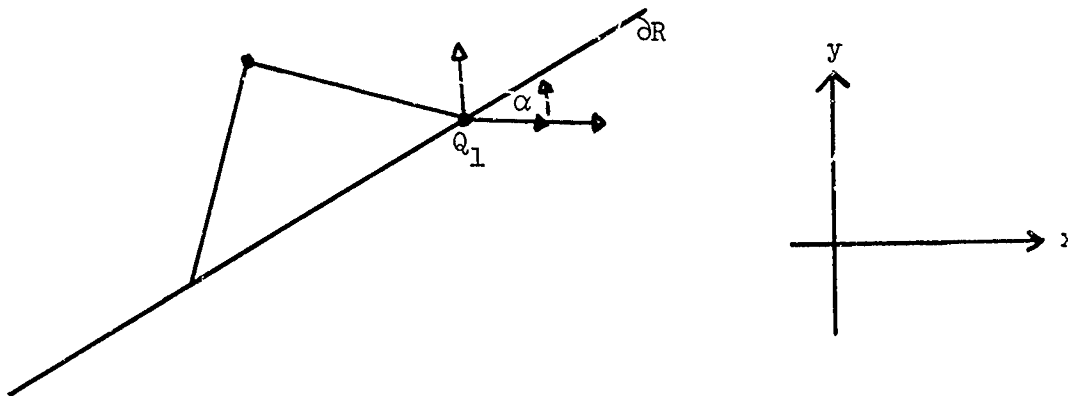


Figure 4.1

Suppose the boundary condition $\frac{\partial v}{\partial n} = g(x,y)$ is imposed along ∂R . Then at the point Q , we want to impose the condition (4.7) on the derivative parameters at the node Q_1 .

$$(4.7) \qquad -v_x(Q_1)\sin\alpha + v_y(Q_1)\cos\alpha = g(Q_1) \quad .$$

If our boundary conditions impose ℓ such constraints (where $\ell \ll N$ in general), we can write them as an $\ell \times N$ matrix equation

$$(4.8) \qquad Qq' = c \quad .$$

The solution of (4.6) can be viewed as the point which minimizes the quadratic function

$$(4.9) \qquad \Phi(q') = \frac{1}{2} q'^T A' q' - q'^T b' \quad .$$

Using the method of Lagrange multipliers to minimize (4.9) subject to the constraints (4.8), we obtain the following system to solve

$$(4.10) \qquad \begin{pmatrix} A' & Q^T \\ Q & 0 \end{pmatrix} \begin{pmatrix} q' \\ \lambda \end{pmatrix} = \begin{pmatrix} b' \\ c \end{pmatrix} \quad ,$$

where λ is a vector of ℓ Lagrange multipliers. The algorithm for solving (4.10) is

(4.11)

a) Solve $A'W = Q^T$ and compute $Y = QW$ (and its LU decomposition)

b) Solve $A'y = b'$

c) Solve $Y\lambda = Qy-c$

d) Solve $A'q' = b'-Q^T\lambda$.

At first sight this algorithm appears expensive, since $\ell+2$ solutions of systems of the form (4.6) are involved. However, if the coefficients in (3.3) are constants, Y remains constant for different boundary conditions. Thus, in such circumstances, our problem may be solved using

steps (b), (c) and (d) once the LU decomposition of Y is available.

Another possibility for handling boundary conditions of the form (4.7) is to modify the parameters of our problem. Applying the proper orthogonal transformation to the stiffness matrices, we rotate the derivative parameters to get v_n and v_s. The boundary condition then imposes a condition only on v_n, and the variable can be eliminated in the obvious way.

Which of the two approaches for handling derivative boundary conditions is better? It is fairly clear that the latter approach will generally require less computation, since the rotations which must be applied are relatively inexpensive and each one saves a solution of the system (4.6). For very large systems, the difference between the two computations will be great. Furthermore, the relative difference between the work required to decompose A' and that required for a back solution is not as large for band systems as for dense systems, since the factor is the bandwidth rather than N. Thus our remark above that A' need only be decomposed once is not as important as you would expect.

In support of the first method of treating derivative boundary conditions is its simple and uniform implementation. The computation can be isolated in one subroutine which generates the matrix Q. In contrast, the second approach is very complicated. Corners having interior angles which are not multiples of $\pi/2$ may force us to apply non-orthogonal transformations to the derivative parameters in order to handle boundary conditions imposed on both incident edges. The fact remains, however, that such complexity pays off. For typical problems (and a one shot computation) the first approach can require twice as much computation as the second.

5. Inclusion of Singular Functions in the Basis

For some elliptic boundary-value problems, particularly in domains with re-entrant corners, the solutions may have unbounded derivatives at some of the corners, so they are hard to approximate by polynomials. A successful approach due to Fix [F2] is to enrich the basis by adding appropriate "singular" functions that represent the solution accurately near the corners.

Fix employed tensor product spaces rather than the interpolation method for generating the finite-element equations (the distinction between the two approaches was made in Chapter 3). Thus, once he had designed the appropriate singular functions having small support, the inclusion of them in the basis was straightforward. The extra terms were simply added to the expansion for v^N.

The inclusion of such singular functions is still possible with the interpolation approach, but the procedure is not quite so obvious. Suppose we wish to include one singular function ψ^* in the basis, and assume that $\psi^* \neq 0$ on triangle T^ν. We will ignore the complication of the mapping of T^ν onto the canonical triangle T^0. Using the notation we developed in Section 1 of this chapter, we consider the computation involving the following term on triangle T^ν:

$$(5.1) \qquad \iint_{T^\nu} u_x^2 \, dx\, dy \quad .$$

We first note that the basis functions on T^ν under "normal" circumstances are given by ψ_i^ν, $i = 1,2,3,\ldots,m$, where

$$(5.2) \qquad \psi_i^\nu = \{C^{\nu^{-T}} \varphi\}_i \quad .$$

The approximation to u on T^ν is thus given by

$$(5.3) \qquad v(x,y) = \sum_{i=1}^{m} q_i^\nu \psi_i^\nu = q^{\nu^T} C^{\nu^{-T}} \varphi \quad .$$

In this form it is clear how to add the singular function. Including the singular function ψ^* in the sum of (5.3) and going in reverse we have:

$$(5.4) \qquad v(x,y) = \sum_{i=1}^{m} q_i^\nu \psi_i^\nu + q^* \psi^*$$

$$= (q^{\nu^T}, q^*) \left[\begin{array}{c|c} C^{\nu^{-T}} & \\ \hline & 1 \end{array}\right] \left[\begin{array}{c} \varphi \\ \hline \psi^* \end{array}\right] \quad .$$

The expression for (5.1) is therefore

$$(5.5) \qquad (q^{\nu^T}, q^*) \left[\begin{array}{c|c} C^{\nu^{-T}} & \\ \hline & 1 \end{array}\right] \iint_{T_r} \left[\begin{array}{c} \varphi_x \\ \hline \psi_x^* \end{array}\right] (\varphi_x^T, \psi_x^*) \, dx\, dy \left[\begin{array}{c|c} C^{\nu^{-1}} & \\ \hline & 1 \end{array}\right] \left[\begin{array}{c} q^\nu \\ \hline q^* \end{array}\right] \quad .$$

In this particular example, the stiffness matrix for T^ν will be $(n+1)$ by $(n+1)$ rather than n by n . The extension to more than one singular function is clear.

CHAPTER 4

SOLUTION OF FINITE ELEMENT EQUATIONS

1. Introduction and Notation

In this chapter we will study the storage and solution of finite element systems of equations. As we pointed out in Chapter I, the $N \times N$ finite element coefficient matrix A will in general be sparse; that is, many (perhaps most) of its elements will be zero. To say that a matrix is sparse, with no further qualification, is not of much practical significance. What is important is whether we can make use of its sparseness to reduce storage and/or computation requirements in its subsequent processing; that is, we are interested in whether the matrix has exploitable structure rather than just its sparseness. One of our aims in this chapter will be to study the structure of finite element equations and to show how such structure can be utilized. In this connection we present some experiments comparing several ordering algorithms (i.e., algorithms which order or reorder the rows and columns of A with the aim of reducing storage and computation requirements). We also present two efficient methods for storing sparse matrices.

We have confined our attention to direct methods for solving finite element equations for the following reasons:

(1) Storage is becoming increasingly abundant, and one of the prime reasons for using iterative methods is that they generally require much less storage than direct methods. Computer memories are steadily becoming larger, the capacity and performance of peripheral storage devices such as disks and drums is improving rapidly, and large bulk core storage [F8] (which can be viewed as a very fast peripheral storage device) is becoming common. The use of virtual memory [D1, M5] is another important development. Under ideal conditions, the user is allowed to address a very large memory ($\approx 2^{24}$ words

on the IBM 360/67) which need not exist physically but where addresses are automatically mapped onto actual physical addresses during execution. We do not mean to imply that storage is not an important consideration in the choice of methods; our contention is simply that the characteristics of today's computer systems allow the solution of large linear systems with direct methods.

(2) Finite element methods tend to yield denser systems of equations than usual finite difference methods. Suppose the parameter $q_{i,j}$ is associated with node i . Then there will usually be a non-zero entry in $q_{i,j}$'s equation for every parameter associated with every triangle containing node i . It is easy to see that higher degree polynomials must lead to denser systems, because more parameters will be associated with each triangle. We discuss this subject in detail in Section 6 of this chapter. Since the amount of computation per iteration for most iterative schemes is proportional to the number of non-zero elements in the matrix, this increased density increases the solution time for iterative methods. [However, for fixed N , higher degree polynomials yield systems which require more computation for their direct solution also, so it is difficult to make precise statements as to which methods require the least computation.] Fix and Larsen [F3] have compared Gaussian elimination and successive over-relaxation (SOR) for some special tensor-product spaces, and their analysis and numerical experiments suggest that SOR is more efficient for some problems, if N is large enough. Their conclusions are based on the assumption that the equations have only one right side, and in many practical situations, this is unlikely. Also, their analysis is based solely on operation counts. For tensor-product bases such an analysis is reasonable, since the structure of the grid and the coefficient matrix can conveniently

be stored in two-dimensional arrays. The data management is no more complex than that resulting from using a five point difference operator on a regular mesh. However, for an arbitrary triangular mesh, A will not have such regular structure, and the calculation of a single component of the residual vector may be relatively expensive. In general, A will be symmetric and only its upper or lower triangle will be stored; therefore, in order to compute a single component of the residual, we must be able to access lines of elements in both rows and columns of the upper (or lower) triangle of A . If the storage scheme is "row oriented", accessing elements in a specific column may require scanning several rows, and visa versa for column-oriented schemes. By contrast, elimination schemes can be conveniently implemented so that they operate only on rows or only on columns. We discuss this subject in detail later; our point is that data management can be important in comparing methods.

(3) Finally, and perhaps most important, a rather large amount of practical engineering experience indicates that direct methods are preferable to iterative ones. The reasons for this include:

(i) Finite element systems (designed to yield a prescribed accuracy) tend to have a considerably lower order N than systems resulting from usual finite difference methods. This is due in part to the ease with which we can grade the net (thus making efficient use of each degree of freedom). Also, as we shall see in Chapter 5, increasing the degree d of our piecewise polynomial allows us to decrease N and still obtain the prescribed accuracy.

(ii) Direct methods allow the use of iterative refinement [F4, W2], which provides an estimate of the condition of the discrete problem and the accuracy of the discrete solution. Such information is hard to

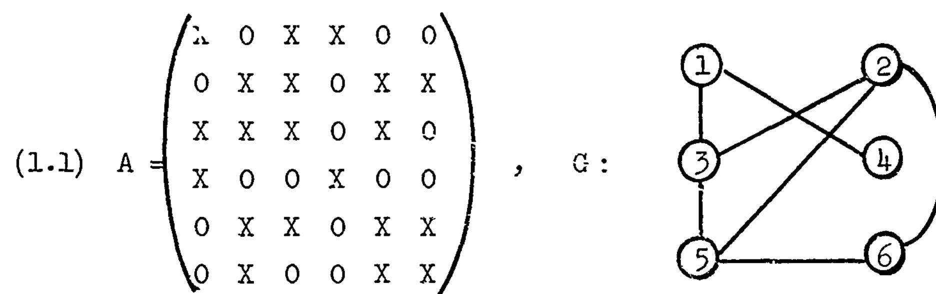
obtain using iterative methods. Since we do not know the true (discrete) solution, the error at each step of the iteration must be estimated on the basis of such measurable quantities as the size of the residuals or the size of the last correction vector. Unfortunately, small residuals or small changes in successive iterates do not guarantee small errors in the computed solution. By using direct methods, we also avoid the problem of finding a "good" over-relaxation parameter.

(iii) Usually, more than one right side must be processed. The initial cost of the decomposition, which represents the majority of the computation for the first solution, does not have to be repeated for succeeding right sides.

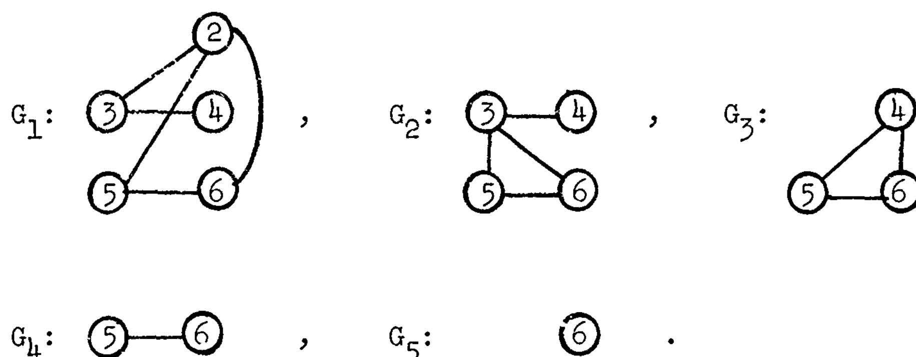
The study of sparse matrix problems is a rapidly expanding field. (See Willoughby [W3], and the extensive references therein.) In the sequel, we will assume A is a symmetric positive definite matrix. As we observed in Chapter I, finite element methods for elliptic problems cast in a variational form yield this type of matrix. Following Rose [R3] and Cuthill and McKee [C5], A will be said to have <u>bandwidth</u> m if and only if $a_{ij} \neq 0 \Rightarrow |i-j| \leq m$. Note that this differs from the usual definition of bandwidth, which is defined in terms of m to be $2m+1$. For any matrix W, we define the quantities $f_i^W = \min\{j | w_{ij} \neq 0\}$, $i = 1,2,\ldots,N$, and $\delta_i^W = i - f_i$. Thus, $m = \max_i\{\delta_i^A\}$. The number N_Z^W will denote the number of non-zero elements in W.

Rose [R3] has given a detailed graph theoretic analysis of the Cholesky decomposition algorithm. With Rose we define the <u>graph</u> $G = (X,E)$ associated with A, where X and E are sets of nodes and edges, respectively. <u>Vertices</u> correspond to rows of the matrix, and <u>edges</u> correspond to <u>non-zero</u>, <u>off-diagonal</u> elements of A. If $i > j$ and

$a_{ij} \neq 0$, then vertex i is joined to vertex j by an edge. (We then say that vertices i and j are adjacent.) The <u>degree</u> of a vertex is the number of edges incident to it. An example demonstrating this notation appears below. An "X" indicates a non-zero element, and a "0" indicates a zero element.



The ordering of the equations induces a corresponding ordering of the vertices of G . In general, we denote an ordering α on X ($\{1,2,3,\ldots,N\} \overset{\alpha}{\rightarrow} X$) by G^{α} . Denoting the set of vertices adjacent to vertex i by η_i ("neighbours" of vertex i) , we can describe the Cholesky decomposition of A into LL^T by a sequence of <u>elimination graphs</u> [R3] $G = G_0, G_1, G_2, \ldots, G_{N-1}$, where G_i is obtained from G_{i-1} by deleting vertex i and its incident edges and adding edges so that the vertices of η_i are <u>pairwise adjacent</u>. Using our example above, we have:



The zero/non-zero structure of L is thus given by

$$
(1.2) \qquad L = \begin{pmatrix} X & & & & & \\ & X & & & & \\ X & X & X & & & \\ X & & X & X & & \\ & X & X & X & X & \\ & X & X & X & X & X \end{pmatrix} .
$$

The number of edges added during elimination is usually referred to as the fill-in, and is simply the difference between N_Z^L and the number of non-zero elements in the lower triangle of A, including the diagonal. Rose [R3] points out that the fill-in will be zero iff for all $N \geq i > j > k > 0$, $(a_{ij} \neq 0 \wedge a_{ik} \neq 0) \Rightarrow a_{jk} \neq 0$. He shows that L must have this property (if we ignore the occurrence of accidental zeros), and calls matrices having this property perfect elimination matrices.

An element a_{ij}, $i \geq j$ will be said to lie in the profile of A $(a_{ij} \in Pr(A))$ if $f_i^A \leq j \leq i$. Hence $a_{ij} \neq 0 \Rightarrow a_{ij} \in Pr(A)$, but $a_{ij} \in Pr(A) \nRightarrow a_{ij} \neq 0$. This is a simple but important generalization of the concept of bandwidth. Observe that $Pr(A) = Pr(L)$. We will denote the number of elements in $Pr(A)$ by $|Pr(A)|$. Thus A is sparse if $|Pr(A)|$ is significantly less than N^2, even if $m = N$. Obviously, $|Pr(A)| = \sum_{i=1}^{N} (\delta_i^A + 1)$.

Now the decomposition of A into LL^T is unique; however, the amount of computation done to obtain L will depend on the structure of A, and how carefully we take advantage of it. Suppose A is $N \times N$ with bandwidth m. Then treating A as a dense band matrix, it is easy to show that the number

of multiplicative operations required to compute L is approximately

$\Theta_B = \frac{Nm(m+3)}{2} - \frac{m^3}{3}$. We will refer to the algorithm as the "band Cholesky (BC) decomposition algorithm".

Suppose now that $\delta_i^A < m$ for at least one i , and we take advantage of this fact. The following theorem gives the number Θ_P of multiplicative operations required to compute L , if we consider A and L as having <u>dense profiles</u>.

<u>Theorem 1.1</u>

Let f^A be as defined above. Then the number Θ_P of multiplicative operations required to compute L is given by

$$(1.3) \qquad \Theta_P = \sum_{i=2}^{N} \frac{\delta_i^A(\delta_i^A + 3)}{2} \quad .$$

In addition, N square root operations and Θ_P-N additions are required.

<u>Proof:</u>

Let us denote the elements of L by ℓ_{ij} and consider the computation of the i-th column of L . The element ℓ_{ii} is computed using the formula

$$(1.4) \qquad \ell_{ii} = \{a_{ii} - \sum_{j=f_i^A}^{i-1} \ell_{ij}^2\}^{1/2} \quad ,$$

which requires $\delta_i^A = i - f_i^A$ multiplications, δ_i^A additions and a square root operation. The elements ℓ_{ik} , $k = f_i^A, f_i^A+1, \ldots, i-1,$, are computed using

$$(1.5) \qquad \ell_{ki} = \{a_{ki} - \sum_{j=q_{ik}}^{i-1} \ell_{ij}\, \ell_{kj}\} / \ell_{ii} \quad ,$$

which requires $\delta_i(\delta_i+1)/2$ multiplicative operations and $\delta_i(\delta_i-1)/2$ additions. Summing over i yields (1.3). This method will be referred to as the "profile Cholesky (PC) decomposition algorithm".

The following is obvious:

Proposition 1.1

For any ordering of A, we have $\theta_P \leq \theta_B$.

Finally, suppose we are prepared to take advantage of every non-zero element in A and L; that is, we will operate only on those elements which are actually changed by the elimination process. Let d_i be the degree of the i-th vertex in the elimination graph G_{i-1}. Then we have

Theorem 1.2 (Rose [R3])

The number of multiplicative operations θ_G required to compute L is given by

$$\theta_G = \sum_{i=1}^{N-1} \frac{d_i(d_i+3)}{2} \quad . \tag{1.6}$$

An additional N square root operations and $\sum_{i=1}^{N-1} \frac{d_i(d_i+1)}{2}$ addition operations are required.

The reader is referred to [R3] for the proof of (1.6). This algorithm will be referred to as the "graph Cholesky (GC) decomposition algorithm".

Now we must consider the tradeoff between the amount of computation and storage required by the different algorithms and their relative complexity.

Note that the graph theoretic analysis of elimination implicitly assumes that we are prepared to take full advantage of the structure of A ; thus, for these results to be relevant, we must employ a very sophisticated program, such as that of Gustafson et al [G3]. [In our 6 by 6 example above, we must detect and make use of the fact that $\ell_{42} = 0$.] Hence, for the GC algorithm to be worthwhile, L must have a significant number of zero elements within its profile, and it has been our experience that the L's derived from finite element coefficient matrices do not have sparse profiles. (See Section 4.5 for some numerical experiments in support of this claim.) Therefore, we have confined our studies to the BC and PC algorithms. We should emphasize that our decision is based only on empirical evidence; just how dense $Pr(L)$ must be over all orderings appears to be an open question, even for piecewise linear polynomials on a square regular right triangular mesh.

So, in summary, we have chosen for various reasons to limit our attention to direct methods for solving finite element systems, and to look at no more of the structure of the matrix than its profile. Within this framework, our goals are to reduce storage, reduce computation, and to simplify data management. These goals compete with one another, and the characteristics of the particular computer system (hardware and software) will have considerable effect on which is most important.

Finally, in the sequel, the reader should keep in mind that f_i^A , δ_i^A , $Pr(A)$, θ_B , θ_P and θ_G are all functions of the ordering α of A . Thus comparisons between such quantities should be understood to mean for the same α , unless specifically stated otherwise.

2. Compact Storage Schemes for Sparse Matrices

As in the previous section, let us denote our sparse, symmetric, positive definite coefficient matrix by A , with Cholesky factorization LL^T . When piecewise polynomials of degree > 1 are used, the matrix A will be more dense than that resulting from usual finite difference schemes. Unfortunately, its profile is observed to become only slightly more dense with increasing degree. Hence it is advantageous to store the matrix in a compact manner to save storage. It is important to keep the organization simple to allow rapid row and/or column operations on the matrix. The prime consideration is not whether we can randomly access a particular element of the matrix efficiently but whether we can efficiently multiply the matrix by a vector or multiply one of its rows by a vector.

As we have mentioned before, finite element coefficient matrices tend to have a good deal less uniformity in structure than those arising from traditional finite difference methods. Because of the likelihood of graded nets and the possibility of associating more than one parameter with each grid point, it is not convenient to design a storage scheme based on the geometry of the mesh in question. This is in contrast with most storage schemes for difference equations.

Ideally, the number of storage units required to store the $N \times N$ symmetric coefficient matrix A should be equal to N_0 , the number of non-zero elements in the lower triangle of A (including the diagonal). While it is obviously possible to store A in N_0 storage locations, the problem is to find an efficient mapping function that allows us to easily locate element a_{ij} . In this section we describe two methods for efficiently storing a sparse symmetric matrix.

Method 1. Let ν be a vector defined by

$$\nu_i = \sum_{j=1}^{i} \{1 | a_{ij} \neq 0\} \quad , \quad i = 1,2,\ldots,N \ . \tag{3.1}$$

Obviously, $\sum_{i=1}^{N} \nu_i = N_0$. Let β_i be defined by

$$\beta_i = \sum_{j=1}^{i} \nu_j \quad , \quad i = 1,2,\ldots,N \ . \tag{3.2}$$

The non-zero elements of the i-th row of the lower triangle of A are then stored in contiguous locations of an array S of length N_0 beginning at $S_{\beta_{i-1}+1}$ and ending at S_{β_i} . In an array ω , also of length N_0 , the corresponding distances of the elements from the diagonal are placed. Hence, if $\beta_{i-1} < p \leq \beta_i$, then S_p contains element $a_{i,i-\omega_p}$. An example is useful in understanding the scheme. Consider the following 15×15 matrix.

$$\left[\begin{array}{ccccccccccccccc}
8 &&&&&&&&&&&&&& \\
6 & 4 &&&&&&&&&&&&& \\
& 8 & 9 &&&&&&&&&&&& \\
& 1 & 4 & 6 &&&&&&&&&&& \\
& 9 & & & 12 &&&& \multicolumn{3}{c}{\text{SYMMETRIC}} &&&& \\
4 & & 8 & & 1 & 2 &&&&&&&&& \\
&&&& 2 & & 4 &&&&&&&& \\
&&& 2 &&& 3 & 3 &&&&&&& \\
&&&&& 1 & 6 & & 8 &&&&&& \\
&&&& 8 &&&& 9 & 9 &&&&& \\
&&&&&&& 4 &&& 7 &&&& \\
&&&&&& 9 &&&& 6 & 2 &&& \\
&&&&&&&&& 6 &&& 8 && \\
&&&&&&&& 5 &&&& 2 & 2 & \\
&&&&&&&&&&& 7 && 4 & 4
\end{array}\right]
\quad
\begin{array}{c}
\beta \\ 1 \\ 3 \\ 5 \\ 8 \\ 10 \\ 14 \\ 16 \\ 19 \\ 22 \\ 25 \\ 27 \\ 30 \\ 32 \\ 35 \\ 38
\end{array} \tag{2.3}$$

Here $N_0 = 38$ and the vectors S and ω are given by

$$(2.4) \qquad S = \begin{vmatrix} 8 \\ 6 \\ 4 \\ 8 \\ 9 \\ 1 \\ 4 \\ 6 \\ 9 \\ 12 \\ 4 \\ 8 \\ 1 \\ 2 \\ \cdot \\ \cdot \\ \cdot \end{vmatrix} , \qquad \omega = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 2 \\ 1 \\ 0 \\ 3 \\ 0 \\ 5 \\ 3 \\ 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \end{vmatrix}$$

At first glance, the overhead for this method appears prohibitively high since each word stored requires an extra word to store its "offset" from the diagonal. However, note that the elements of ω will all be bounded by m, the bandwidth of A. On the IBM 360, for example, the array ω can be declared as short integer (2 bytes = 16 bits), whereas the elements of S may be 4 or 8 bytes long. If A is being stored in double precision, the overhead is only about 25 percent, and the total storage required is essentially proportional to N_0. To access a particular element a_{ij} of A will require scanning $\beta_i - \beta_{i-1}$ elements of the array ω. Since the elements ω_k for $\beta_{i-1} < k \leq \beta_i$ are ordered, a binary search can be used, so the amount of work required to access element a_{ij} would be proportional to $\log_2(\beta_i - \beta_{i-1})$. Even for rather dense bands (resulting from use of polynomials of high degree), this is very satisfactory. For example, using quintic polynomials on a typical mesh, we would need to access about 4 elements of ω before finding a_{ij}.

If storage is very scarce, a somewhat more efficient scheme is the following:

Method 2. Let A and β be as described above, and define the vector δ^A as in Section 4.1. Let A be stored in the array S as in Method 1, but instead of defining the array ω as in Method 1, let ω be a bit array of length $\sum_{i=1}^{N}(\delta_i^A+1)$. Define the vector μ by

$$(2.5) \qquad \mu_i = \sum_{j=1}^{i} (\delta_i^A + 1) \quad , \quad i = 1,2,\ldots,N \ .$$

Now define ω by

$$(2.6) \qquad \omega_{\mu_i-\ell} = \begin{Bmatrix} 1 & \text{if } a_{i,i-\ell} \neq 0 \\ 0 & \text{if } a_{i,i-\ell} = 0 \end{Bmatrix} \ , \quad \ell \leq \delta_i^A \ , \quad i = 1,2,\ldots,N \ .$$

We again use the example (2.3) to aid in understanding the scheme. The arrays μ and ω are given by

$\mu^T = (1, 3, 5, 8, 12, 18, 21, 26, 30, 36, 40, 46, 50, 56, 60)$, and

$\omega^T = (111111111100110101011101100111101100011100110001110011000111011)$.

row 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Thus, the zero/non-zero structure of the i-th row of the lower triangle of A is stored in the segment of ω beginning at $\omega_{\mu_{i-1}+1}$ and ending at ω_{μ_i} . The storage required to store A is thus

$$(2.7) \qquad (N_0 + 2N) \text{ words} + \mu_N \text{ bits.}$$

Note that the storage required for μ and β becomes less significant with increasing N_0 and fixed N . The use of a bit array may cause some program overhead (unless the machine is bit addressable), and since ω is not ordered, up to $\mu_i - \mu_{i-1}$ elements of ω will have to be examined to

retrieve element a_{ij} . (Note that $\mu_i - \mu_{i-1}$ may be $\gg \beta_i - \beta_{i-1}$.) Although this method will undoubtedly require considerably more program overhead than method 1 to use, it uses extremely little unnecessary storage. For example, using this method on an IBM 360 computer to store a dense 500×500 symmetric matrix in short precision requires less than 4 percent more storage than the usual method of storing a triangular matrix in a one-dimensional array. The percentage overhead would be halved if the array were being stored in double precision.

Thirdly, we present a method due to Jennings [J1] which is applicable when $|Pr(W)| \approx N_Z^W$; that is, when there are few zero elements within $Pr(W)$. As we mentioned before, it has been our experience that the L's derived from finite element coefficient matrices have this property.

Method 3 ("Profile Storage Scheme")

Let the lower triangle (including the diagonal) of A be stored row by row in contiguous locations of a one-dimensional array S . Defining the vector μ as in method 2 above, then element a_{ij} , $i \geq j$ is given by S_p , where $p = \mu_i - i + j$. [Note that S now has the same zero/non-zero configuration as ω in method 2.]

This method obviously applies equally well to storing lower triangular matrices, and it is primarily for this reason that we present it. The overhead for this storage scheme is only the storage required for μ . To store L , we need $|Pr(L)| + N$ words. If A is stored in this manner, the PC algorithm can be applied "in place" and no temporary storage is necessary.

Finally, we mention the most commonly used method for storing band matrices [M1], which we will refer to as the "diagonal storage scheme" or simply as method 4. The diagram below describes the storage layout:

$$
(2.8) \qquad \begin{pmatrix}
 & \bigcirc & & & a_{11} \\
 & & & a_{21} & a_{22} \\
 & & a_{31} & a_{32} & a_{33} \\
 & & \iddots & \vdots & \vdots \\
a_{m,1} & & & & a_{mm} \\
\vdots & & & & \vdots \\
a_{N,N-m} & \cdots & & a_{N,N-1} & a_{N,N}
\end{pmatrix}_{N \times (m+1)}
$$

The required storage is $(m+1)N$, and in order for the method to be efficient we should have $m \ll N$.

The following observation is of practical interest:

<u>Proposition 2.1</u>

Let $V_B = (m+1)N$ and $V_P = |Pr(L)| + N$. Then for any ordering of A, $V_P \leq V_B + N$, and if $\sum_{i=1}^{N}(m-\delta_i) \geq N$, then $V_P \leq V_B$.

<u>Proof:</u>

$$
\begin{aligned}
V_P = N + |Pr(L)| &= N + \sum_{i=1}^{N} (\delta_i+1) \\
&= N + \sum_{i=1}^{N} \delta_i + N \\
&= (N+1)m + N - \sum_{i=1}^{N} (m-\delta_i) \\
&= V_B + N - \sum_{i=1}^{N} (m-\delta_i) .
\end{aligned}
$$

Thus, Proposition 2.1 says that for any ordering, the storage required for method 3 cannot exceed that required for method 4 by more than N words. In practical situations we have found that V_P is always considerably less than V_B. See, for example, the experiments in section 4.5 and the analysis in section 4.7.

Note that there is only a very weak relationship between $|Pr(A)|$ and m. All we can show is

$$N + m \leq |Pr(A)| \leq (m+1)N - m(m+1) / 2 . \quad (2.9)$$

Essentially, (2.9) says that for a fixed m, $|Pr(A)|$ can vary by nearly a factor of N.

3. Node Ordering for a Small Bandwidth

The reasons most often presented for reducing the bandwidth of a matrix are to reduce the storage and computation required to solve the associated linear system or eigenproblem. However, these reasons are valid only if we plan to store and process the matrix as a dense band matrix. In view of Prop. 1.1, Prop. 2.1 and (2.9), the only justification for ordering to achieve a small bandwidth is to simplify data management. In this section we discuss the reasons for bandwidth reduction and present some algorithms for obtaining small bandwidth orderings. Note that the question here is not whether we should use the BC or PC algorithms for a fixed ordering α , but rather, when we should use m (instead of $|Pr(A)|$ or θ_P) as a criterion (objective function) to minimize over all orderings α of A .

To begin with, regardless of the ordering α of A , if A is symmetric and positive definite, there seems to be no reason to use the BC rather than the PC algorithm. We say this because $V_P \leq V_B$ (usually), $\theta_P \leq \theta_B$, and the computational overhead of the PC over the BC algorithm is negligible. However, the linear system we want to solve may not always be positive definite; although elliptic problems will yield positive definite matrices, many methods for solving the associated eigenproblem involve shifts of origin which destroy the positive definiteness of the system being solved [W2]. When A is indefinite, partial pivoting is required to maintain numerical stability, and the profile storage scheme is no longer applicable since we are now computing $PA = LU$ for some (a-priori) unknown N by N permutation matrix P . The only storage scheme which is well adapted to partial pivoting is method 4. For an Algol procedure for computing the LU decomposition of indefinite band matrices, see [M1]. Thus, in this

situation $m = m^A$ is important, since we can only guarantee that $m^{PA} \leq 2m^A$, and the combined storage requirement for L and U (using the diagonal storage scheme of Section 4.2) is therefore $(3m+1)N$.

The work of Bunch [B13] on stable decompositions of symmetric indefinite systems may be important in this regard since a shift of origin does not destroy symmetry. We compute $PAP^T = LDL^T$, where D is block diagonal with 1 by 1 and 2 by 2 blocks. Unfortunately, there does not appear to be any way to bound m^{PAP^T} a-priori. Thus, to be competitive (with the band LU algorithm) with respect to storage {computation} we should have $m > N/6$ $\{m > N/2\sqrt{3}\}$.

Another situation in which we might wish to have a small bandwidth is when auxiliary storage must be used. Overlay versions of band decomposition algorithms can be implemented most efficiently if km^2 $(1 \leq k \leq 3)$ storage units of main memory are available. Hence, it is important to have m small. Note that this does not preclude the possibility of using the FC algorithm, if applicable. Having m small simply limits the number of rows or columns we should have available at any given time.

If a matrix A can be stored in such a way that only its non-zero elements need to be stored and considered in a residual calculation, it is clear that bandwidth ordering makes no sense for iterative schemes that require only a residual calculation.

Obviously, a useful bandwidth reducer must consume less time than it saves the linear equation solver, or else significantly reduce the amount of storage required. It will be relatively unimportant in practice whether the minimum bandwidth is achieved, but we should get reasonably close to the minimum bandwidth in an economical amount of time. Note that an easily ascertained lower bound for the bandwidth (not necessarily attainable) can

be obtained by finding $\lceil k/2 \rceil$, where k is the largest number of non-zero elements in any row.

Two basic approaches to ordering for a small bandwidth are in current use. They can be classed as <u>direct</u> (or one-pass) and <u>iterative</u>. The direct schemes [R3 , C5] usually work closely with the associated graph, and proceed by successively removing (i.e., numbering) the nodes of the graph according to some strategy based on the (usually local) structure of the graph. The iterative schemes, on the other hand, assume a given ordering and attempt to improve the ordering (again according to some strategy) by finding appropriate row and/or column interchanges. Since the direct methods only need a single starting node to begin, while the iterative schemes need an initial <u>ordering</u>, a reasonable procedure is to use a direct method to obtain an initial ordering and then use an iterative scheme to improve it. The problem of finding an initial starting node is discussed in Section 5.

We now describe two popular direct methods for bandwidth ordering.

A. <u>Spanning Tree Method</u> (Cuthill and McKee [C5]).

1. Choose a starting node x_1 , and define $Q = \{x_1\}$.
2. For each node in Q (in the order in which they are numbered), number their unnumbered neighbors in order of <u>increasing</u> degree.
3. Set Q = {nodes assigned numbers in the last execution of Step 2}.
4. If $|Q| = 0$, then stop; otherwise go to Step 2.

The algorithm is equivalent to finding a spanning tree (rooted at the initial node) of the graph G, hence the name. [A tree is a connected graph with N nodes and $N-1$ edges. A spanning tree of the graph G is a subgraph of G which is a tree and contains all N nodes.]

The obvious advantage of this method is that it is very efficient. The required work is proportional to N times the average degree of the vertices, and thus only increases linearly with N. Very good results are obtained, provided a good starting node is selected.

The minimum degree algorithm [R3] is similar to method A above and is as follows:

B. Minimum Degree Algorithm

1. Set $i = 1$.
2. In the elimination graph G_{i-1} choose x_i to be any vertex satisfying

$$|\eta(x_i)| = \min_{y \in X_{i-1}} |\eta(y)|$$

where $G_{i-1} = (X_{i-1}, E_{i-1})$.

3. Set $i = i+1$.
4. If $i > N$, then stop; otherwise go to Step 2.

From a practical point of view this algorithm has little to offer over Method A, and is obviously inferior with respect to the amount of work that is required; $N(N+1)/2$ vertices must be tested. A practical modification that drastically reduces the amount of work required, and actually improves the results obtained as well, is to restrict the candidates considered in Step 2 to those having at least one numbered neighbor. Nevertheless, experience has shown that the Cuthill-McKee algorithm seldom

produces a larger bandwidth than the minimum degree algorithm, and even with the above modifications the latter requires substantially more work than the former.

We now turn to iterative methods for reducing the bandwidth of a matrix [R4, T3]. Here it is more convenient to speak in matrix, rather than graph-theoretic, terms. The differences among these iterative schemes are largely matters of programming techniques rather than fundamental ideas. The general idea follows: Assume we are given an initial ordering yielding a bandwidth of m. Non-zero elements satisfying $|i-j| = m$ will be referred to as edge elements. Since we are assuming that the matrix is zero/non-zero symmetric, we will preserve the symmetry by interchanging corresponding columns whenever rows are interchanged.

1. Set $max = m$.
2. Try to interchange rows containing edge elements with rows not containing edge elements so as to reduce the bandwidth, and simultaneously interchange columns.
3. Re-compute m. If $m < max$, then set $max = m$ and go to Step 2.
4. If max is greater than or equal to its value when Step 4 was last executed, then stop. Otherwise compute a vector v of N values as follows:

$$v_i = \sum_{j=1}^{N} \{j|a_{ij} \neq 0\} \Big/ \sum_{j=1}^{N} \{1|a_{ij} \neq 0\} .$$

Order the equations in increasing order of v, and order the columns correspondingly. The first time this is done the bandwidth may increase; after the first step repeat as long as the bandwidth decreases. Re-compute m, set $max = m$, and go to Step 2.

Step 4 has the effect of reordering the rows so that as nearly as possible each row has the same number of non-zero elements on each side of the diagonal element. It could be called the balancing stage. For matrices that have an inherent band structure (as ours have), Step 4 does not have much effect, but for randomly sparse matrices Step 4 can improve the performance of the reducer remarkably.

4. Node Ordering to Reduce $|Pr(A)|$

In the light of Prop. 1.1 and Prop. 2.1, it should be clear that if A is symmetric and positive definite, a potentially profitable strategy for ordering is to look for orderings which reduce θ_P or $|Pr(A)|$ $(= |Pr(L)|)$.

The term "near optimal" as it appears in the literature [R3,T3] usually means near-optimal with respect to fill-in. Under our assumption that $Pr(L) \approx N_Z^L$, a near optimal ordering should "nearly" minimize $|Pr(A)|$. [Since θ_P is a more difficult function to work with, we have not tried to look for orderings to reduce it. Tacitly, we have assumed that an α yielding a small $|Pr(A)|$ will also yield an acceptable θ_P .]

As with bandwidth ordering algorithms, there are direct and iterative schemes for near-optimal ordering. In order to explain the first (direct) method we define the deficiency $D(x_i)$ [R3] of a vertex x_i in a graph G by

(4.1) $$D(x_i) = |\{(x_j,x_k) | x_j \in \eta(x_i) \wedge x_k \in \eta(x_i) \wedge x_j \notin \eta(x_k)\}| .$$

Recall the construction of elimination graphs. It is easy to verify that if $D(x_i) = 0$, $G_i = (X_i,E_i)$ is obtained from $G_{i-1} = (X_{i-1},E_{i-1})$ by deletion of x_i and its incident edges; no edges are added. This provides the motivation for the

A. Minimum Deficiency Algorithm [R3,T3]: Let $G_0 = (X,E)$. Then

1. Set $i = 1$.
2. In the elimination graph G_{i-1} , choose x_i to be any vertex such that

$$|D(x_i)| = \min_{y \in X_{i-1}} |D(y)|$$

where

$$G_{i-1} = (X_{i-1}, E_{i-1}) \quad .$$

3. Set $i = i+1$.

4. If $i > N$, stop; otherwise go to Step 2.

In this direct algorithm the next node to be numbered is the one that will introduce the fewest non-zero elements when it is eliminated. It is obviously fairly expensive to find this node, since a deficiency test of a node y involves $|\eta(y)| \cdot |\eta(y)+1| / 2$ edge tests. Since the graph usually must be stored as a bit matrix, and few machines are bit-addressable, these tests may involve considerable overhead. As with the minimum degree algorithm (Section 4.3) we have found that restricting the candidates in Step 2 to those nodes that have at least one numbered neighbor does not hurt the ordering produced by the minimum deficiency method, and this restriction drastically reduces the amount of computation involved.

The following iterative scheme has been found to significantly reduce $|Pr(A)|$. Again we will revert to matrix notation. The vector f^A is as defined in Section 4.1.

B. Profile Reduction Algorithm.

1. Compute $Q^* = \sum_{i=1}^{N} (i - f_i^A)$.

2. Let the vector v be defined by

$$v_i = \sum_{j=i}^{N} \{1 | f_j^A = i\} \quad .$$

3. For each row i having $v_i \geq 0$, examine those rows $j = i+1, i+2, \ldots, i+k$ for some (small) $k > 0$, and determine the number of words s_{ij} of storage that can be saved by

interchanging rows (and corresponding columns) i and j.

If the maximum s_{ij} is positive, interchange rows i and j (and corresponding columns), adjusting the vector f^A accordingly.

4. Compute $Q = \sum_{i=1}^{N} (i - f_i^A)$. If $Q < Q^*$, then set $Q^* = Q$ and go to Step 2; otherwise stop.

The actual search for the best interchange (Step 3) is by far the most expensive part of the algorithm, and in a practical situation only those rows with v_i greater than some threshold should be tested since the maximum possible gain in storage resulting from interchanging row i and row j is $s_{ij} \leq v_i \times (j-1)$. A reasonable threshold seems to be 3 or 4. Good results have been obtained with the parameter k mentioned in Step 3 set to 5.

5. Some Experiments with Ordering Algorithms

The coefficient matrix A obtained from the finite element formulation of a problem tends to have considerably less uniformity in structure than the matrix arising from a finite difference method applied to the same problem. First, the node points of the finite element mesh may not all play the same role, and as a result have different connectivities. Whether a parameter is associated with a vertex, side or interior node and whether there is more than one parameter associated with the node will greatly affect the number of non-zero elements in its equation. Second, the finite element mesh will very likely be graded, which also causes disorder in the structure of A .

Our aims in this section are

(a) to report on the performance of several ordering algorithms and demonstrate the savings attainable by using profile instead of band methods for storage and computation;

(b) to report on an intriguing and agreeable property of the reverse Cuthill-McKee ordering (our terminology) which we have discovered. That is, if the Cuthill-McKee algorithm numbers the nodes $1,2,\ldots,N$, then the reverse Cuthill-McKee (RCM) ordering would be $N,N-1,\ldots,1$;

(c) to present some experimental evidence supporting our implicit assumption that the profile of L is usually quite dense; i.e., $|Pr(L)| \approx N_Z^L$.

We will make use of the following labels for the different algorithms and quantities in this section. Some of them are repeated in other sections.

CM	--	Cuthill-McKee	Initial Ordering Algorithms
RCM	--	Reverse Cuthill-McKee	
MDG	--	Minimum Degree	
MDF	--	Minimum Deficiency	
BR	--	Bandwidth Reduction	Improvement Algorithms
PR	--	Profile Reduction	
BC	--	Band Cholesky	Decomposition Algorithms
PC	--	Profile Cholesky	

$\begin{Bmatrix}\theta_B\\ \theta_P\end{Bmatrix}$ -- multiplicative operation count for the $\{^{BC}_{PC}\}$ decomposition algorithm

ν_B -- storage required to store a symmetric or lower triangular matrix using the band oriented method 4 (Section 4.2)

ν_P -- storage required to store a symmetric or lower triangular matrix using the (profile) storage method 3 (Section 4.2)

Pr(A) -- profile of the matrix A .

$\mathcal{D}$(Pr(A)) -- density of the profile of A .

In order to keep the number and size of our tables at a level where the information can be readily assimilated, we have eliminated the MDG algorithm from consideration because we found it to be much inferior to the CM algorithm. As we mentioned before, it is natural to use a direct ordering algorithm to obtain an initial ordering for the iterative improvement schemes (BR or PR). We have limited our studies to the orderings provided by CM, CM-PR, RCM and MDF. [The hyphen should be read as "followed by".] The application of the BR algorithm to the CM and RCM orderings reduced m by only one or two, and so the results are not included. The application of the PR algorithm to the RCM and MDF ordering resulted in only a small reduction in Pr(A) , and was also

not included. We have limited our studies to elements 1-3, 2-6 and 3-10 (see Appendix A), and to the three domains shown below:

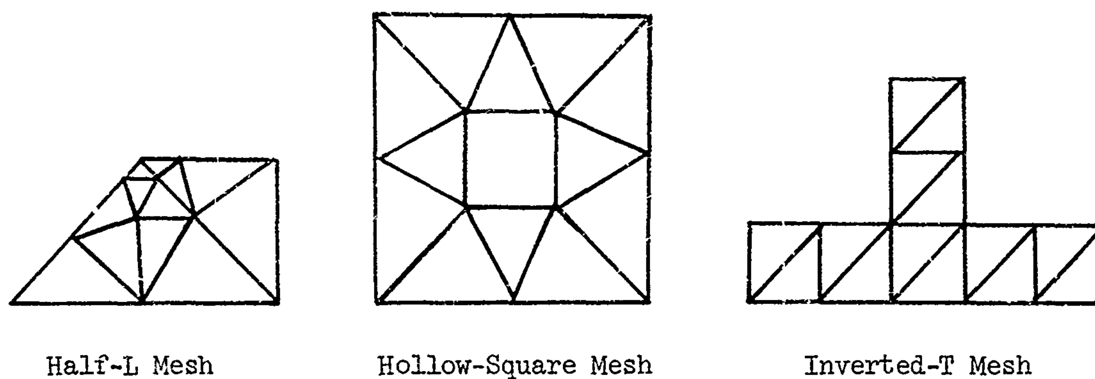


Figure 5.1

For the experiments, the meshes were subdivided by various factors as described in Section 2.3 in such a way that for a given domain each element yielded the same N. The reported times are in seconds for an IBM 360/91 computer. The values of θ and V for each algorithm have been scaled by the values for the CM ordering. The actual values for the CM ordering are reported in parentheses. As before, we indicate the bandwidth by m.

We made use of the geometry of the domain to choose an initial node for the "initial ordering" algorithms. We arbitrarily picked a node from one of the two most widely separated triangles in the domain. For "long, straight" domains this will obviously work well, but for U-shaped domains, for example, it could lead to a bad choice. One should have the capability of forcing the algorithms to begin at a particular node in cases where the above strategy could lead to an unfortunate choice. From a practical point of view, designing and executing a sophisticated algorithm in order to search for a good starting node would probably be more expensive than it's ultimate value would warrant.

The results of the experiments are contained in the following three tables.

Matrix and Elimination Statistics for Several Ordering Algorithms for the Half-L Domain

Element		CM	CM-PR	RCM	MDF
1-3					
	Time	.183	1.5	.183	16.124
	m	20	20	20	83
	θ_B	1(30053)	1	1	13
N = 241	θ_P	1(37545)	1	.96	.91
	ν_B	1(5061)	1	1	6
N_Z^A = 1585	ν_P	1(4197)	1	.97	.95
	$\vartheta(Pr(A))$	.231	.231	.236	.243
	$\vartheta(Pr(L))$	1.000	1.000	1.000	1.000
	Fill-in	3043	3043	2950	2844
2-6					
	Time	.26	10.3	.26	20.83
	m	42	48	42	52
	θ_B	1(196302)	1.27	1	1.47
N = 241	θ_P	1(123342)	.36	.36	.30
	ν_B	1(10363)	1.15	1	1.25
N_Z^A = 2581	ν_P	1(7413)	.61	.60	.54
	$\vartheta(Pr(A))$	.197	.328	.337	.361
	$\vartheta(Pr(L))$	.979	.979	.990	.996
	Fill-in	5608	2805	2734	2484
3-10					
	Time	.25	9.9	.25	27.24
	m	63	81	63	84
	θ_B	1(406188)	1.55	1	1.4
N = 241	θ_P	1(239873)	.30	.21	.19
	ν_B	1(15424)	1.28	1	1.3
N_Z^A = 3793	ν_P	1(10172)	.58	.49	.44
	$\vartheta(Pr(A))$	.203	.353	.428	.467
	$\vartheta(Pr(L))$	.979	.985	.990	.996
	Fill-in	7707	3606	2655	2277

Table 5.1

Matrix and Elimination Statistics for Several Ordering Algorithms for the Hollow Square Domain

Element		CM	CM-PR	RCM	MDF
1-3					
	Time	.17	1.79	.17	9.2
	m	16	16	16	20
	θ_B	1(34776)	1	1	1.5
N = 252	θ_P	1(26299)	1	.99	.95
	ν_B	1(4284)	1	1	1.2
N_Z^A = 1620	ν_P	1(3698)	1	.99	.97
	$\mathcal{B}$(Pr(A))	.272	.272	.273	.278
	$\mathcal{B}$(Pr(L))	1.000	1.000	1.000	1.000
	Fill-in	2510	2505	2489	2421
2-6					
	Time	.23	10.49	.23	18.3
	m	36	46	36	42
	θ_B	1(155609)	1.56	1	1.32
N = 252	θ_P	1(95520)	.35	.36	.33
	ν_B	1(9324)	1.27	1	1.16
N_Z^A = 2628	ν_P	1(6837)	.60	.62	.59
	$\mathcal{B}$(Pr(A))	.219	.372	.362	.381
	$\mathcal{B}$(Pr(L))	.982	.972	.991	.994
	Fill-in	5029	2328	2493	2316
3-10					
	Time	.32	10.6	.32	25.1
	m	71	78	71	63
	θ_B	1(528768)	1.17	1	.81
N = 252	θ_P	1(232516)	.31	.22	.19
	ν_B	1(18144)	1.10	1	.89
N_Z^A = 3852	ν_P	1(10325)	.57	.49	.44
	$\mathcal{B}$(Pr(A))	.204	.367	.426	.465
	$\mathcal{B}$(Pr(L))	.988	.988	.994	.996
	Fill-in	7898	3468	2738	2345

Table 5.2

Matrix and Elimination Statistics for Several

Ordering Algorithms for the Inverted T Domain

Element		CM	CM-PR	RCM	MDF
1-3					
	Time	.22	2.52	.22	9.32
	m	20	20	20	35
	θ_B	1(63283)	1	1	2.82
$N = 301$	θ_P	1(31496)	1	.88	.74
	ν_B	1(6321)	1	1	2.52
$N_Z^A = 1909$	ν_P	1(4258)	1	.95	.88
	$\mathcal{B}(Pr(A))$	.280	.280	.296	.321
	$\mathcal{B}(Pr(L))$	1.000	1.000	1.000	1.000
	Fill-in	2852	2847	1886	2334
2-6					
	Time	.30	12.36	.30	18.38
	m	40	46	40	42
	θ_B	1(230017)	1.3	1	1.1
$N = 301$	θ_P	1(103348)	.34	.34	.28
	ν_B	1(12341)	1.15	1	1.05
$N_Z^A = 2628$	ν_P	1(7422)	.60	.59	.56
	$\mathcal{B}(Pr(A))$	.238	.406	.415	.441
	$\mathcal{B}(Pr(L))$	.981	.983	.985	.994
	Fill-in	5292	2414	2368	2130
3-10					
	Time	.30	11.5	.30	22.2
	m	57	72	57	65
	θ_B	1(441244)	1.52	1	1.26
$N = 301$	θ_P	1(188422)	.30	.23	.19
	ν_B	1(17458)	1.26	1	1.14
$N_Z^A = 4525$	ν_P	1(9887)	.57	.51	.47
	$\mathcal{B}(Pr(A))$	.252	.455	.515	.560
	$\mathcal{B}(Pr(L))$	.981	.972	.821	.995
	Fill-in	6993	2737	1428	1881

Table 5.3

The information in the above tables leads us to the following conclusions:

(1) We appear to be fully justified in assuming that L is almost dense. [We have computed the fill-in for some random orderings as well, and although $\beta(\Pr(L))$ was smaller for some other orderings, we observed that $|\Pr(A_i)| \leq |\Pr(A_j)| \Rightarrow N_Z^{L_i} \leq N_Z^{L_j}$, where A_i is the matrix A with some ordering α_i , and $A_i = L_i L_i^T$. In other words, reducing the profile appears to reduce the fill-in.]

(2) The RCM algorithm seems to be easily the best algorithm. The ordering not only supplies a near optimal bandwidth, but also yields a profile almost as good as the MDF algorithm, which is prohibitively expensive. [There are several reasons why methods based on elimination graphs are expensive to use. First, even if we restrict the candidates to be ordered first to those having at least one numbered neighbor, the number of candidates tends to be quite large, particularly for elements with relatively many nodes. Secondly, we not only must test edges of the graph, but we also must usually add edges as new elimination graphs are formed. This addition of edges requires computer time, and also increases the degree of the nodes which are candidates or potential candidates for subsequent ordering. Since the required work for each step of the MDF algorithm is proportional to the <u>sum</u> of the squares of the degrees of the nodes being tested, these added edges can dramatically increase the amount of work involved.]

The reason that the RCM ordering is superior to the CM order (profile-wise) can be explained as follows. The CM algorithm tends to order the neighbors of each node <u>consecutively</u>, and the non-zero elements of A thus tend to be arranged in sequences in successive rows (columns) of the lower (upper) triangle of A . This is just the reverse of what we want for a

small profile; hence the discovery of the RCM ordering.

(3) It is very beneficial to use profile methods rather than band methods. The following table, which can be obtained from the tables above, brings out this point dramatically.

Element \ Domain	Half-I	Hollow Square	Inverted T
1-3	.72 .82	.75 .86	.44 .64
2-6	.21 .43	.22 .45	.14 .35
3-10	.13 .32	.098 .28	.097 .28

Table 5.4: $\theta_P(\mathrm{RCM}) / \theta_B(\mathrm{RCM})$ and $\nu_P(\mathrm{RCM}) / \nu_B(\mathrm{RCM})$ for Each Element-Domain Combination.

(4) Although we make no claims about the programming of the ordering algorithms (they could be improved by programming some of the bit-pushing in machine language), the reported times are an accurate reflection of relative numbers of edge tests (zero/non-zero tests) required by each algorithm. Hence, although the magnitudes of the times might be improved by a more careful implementation, we would not expect their relative size to change much.

6. The Value of N_Z^A for Arbitrary Elements and Triangular or Quadrilateral Meshes

Suppose we have an arbitrary triangular mesh with N_Δ triangles, V_B boundary vertices, and V_I interior vertices. Let S_B be the number of triangle sides lying on the boundary and S_I be the number of sides lying in the interior of the mesh. Let H be the number of holes in the mesh (domain).

In order to characterize the stencil, let n_V, n_S and n_I be the number of parameters associated respectively with vertex nodes, the node(s) on each side, and the interior of each triangle. For example, element 3-10 (Appendix A) would yield $n_V = 1$, $n_S = 2$, and $n_I = 1$. As in Chapter 3, we let $n = 3(n_V + n_S) + n_I$.

Our aim in this section is to obtain N_Z^A in terms of N_Δ, V_B, S_B, n_C, n_S and n_I. Our method of proof is similar to that in [El], where the following relations between mesh parameters are proved.

$$N_\Delta = \frac{1}{3}(S_B + 2S_I) = V_B + 2V_I + 2H - 2 \quad . \tag{6.1}$$

Consider the following typical mesh:

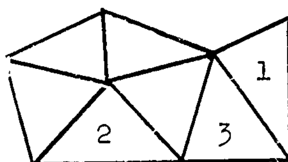


Figure 6.1

Our strategy is to successively remove triangles from the mesh in such a way as to leave all remaining triangles with at least one side inside the mesh. (Thus, triangles 1 or 2 in Figure 6.1 can be removed, but 3 cannot.) As we remove triangles, we will count the number of non-zero elements removed from A. We ignore those cases where elements are accidentally zero because of the regular properties of the mesh and/or the coefficients of the differential operator. We need the following

Lemma 6.1

Let a triangle of type 1 (having two external sides) be removed from the mesh. Then N_Z^A is reduced by

$$\sigma_1 = n^2 - (n_S + 2n_V)^2 \quad . \tag{6.2}$$

Proof:

The total number of elements in A due to the interaction of parameters associated with a triangle is n^2. However, not all the connections are removed by the deletion of triangle 1; those corresponding to parameters lying on the remaining side of triangle 1 (including its end nodes) are not removed, and there are $(n_S + 2n_V)^2$ such non-zero elements. This proves the lemma.

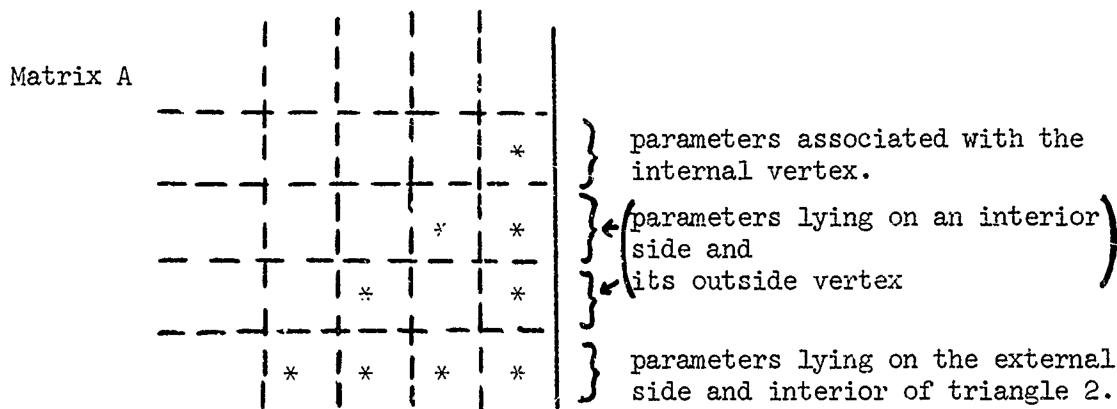
Lemma 6.2

Let a triangle of type 2 be removed from the mesh. Then N_Z^A is reduced by

$$\sigma_2 = 2n(n_I + n_S) - (n_I + n_S)^2 + 2(n_S + n_V)^2 \quad . \tag{6.3}$$

Proof:

As in Lemma 6.1, we first note that the total contribution to N_Z^A from the connections of parameters in triangle 2 is n^2 . However, two of the triangle sides and their incident vertex remain in the mesh, so the connections of their parameters must not be counted unless they correspond to different remaining sides. The truth of (6.3) can be demonstrated by assuming the equations in question are all grouped last in A and examining Figure 6.2. The submatrices marked with an asterisk are the parts removed from A (in the diagram below)



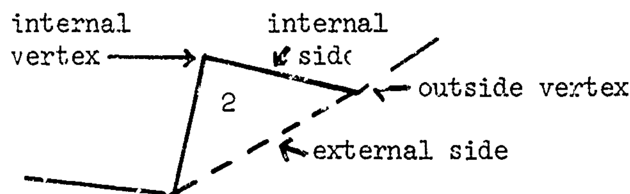


Figure 6.2

Summing the elements in the marked submatrices yields (6.3).

Now suppose the mesh has a hole in it. Eventually we will reach a situation where the hole is bounded at one place by a single side such as depicted below.

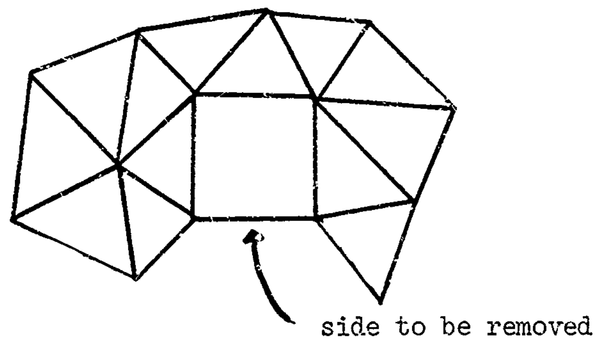


Figure 6.3

Such a side will be referred to as a <u>connecting side</u>. We present the following

<u>Lemma 6.3</u>

Let n_V , n_S and n_I be as above. Then the removal of a connecting side from the mesh reduces N_Z^A by

$$(6.4) \qquad \sigma_3 = n_S^2 + 4n_S n_V + 2n_V^2 \quad .$$

The proof is similar to that employed in Lemma 6.2 and we omit it.

We can now prove the following

Theorem 6.4

Let V , S and H be the number of vertices, sides and holes respectively in a two dimensional triangular mesh. Let n , σ_1 , σ_2 and σ_3 be defined as above. Then N_Z^A is given by

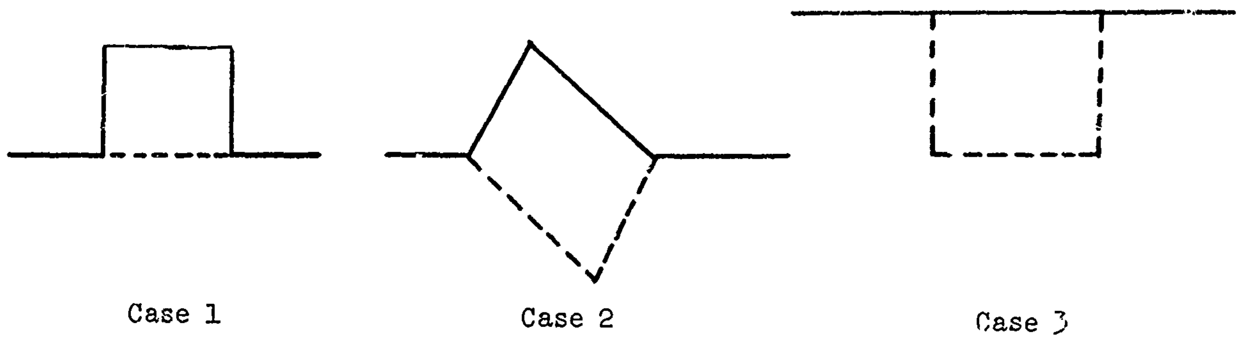
$$(6.5) \quad n^2 + (V-3)\sigma_1 + (S+3-H-2V)\sigma_2 + H\sigma_3 \ .$$

Proof:

Suppose we reduce our mesh to a single triangle by successively removing triangles of type 1 and type 2, (and connecting sides if any), leaving A with n^2 non-zero elements. In order to reduce the mesh to one triangle we must remove $V-3$ triangles of type 1, since removal of a type 2 triangle or a connecting side removes no vertex. Thus the removal of type 1 triangles results in the reduction of N_Z^A by $\sigma_1(V-3)$. Now each hole will result in the occurrence of one connecting side being removed during the reduction of the mesh, and this will reduce N_Z^A by $\sigma_3 H$. These two forms of demolition account for the removal of $2(V-3)+H$ triangle sides, and three sides remain in our final triangle. Hence, we must have removed $S-3-2(V-3)-H$ triangles of type 1, accounting for $\sigma_2(S+3-2V-H)$ non-zero elements. Summing the above expressions yields (6.5).

Using (6.1), N_Z^A can be expressed in terms of other (perhaps more commonly available or easily obtainable) mesh parameters.

A similar analysis can be carried out for quadrilateral elements. If the mesh has no holes, there are three cases:



If $N_{\square}$ is the number of quadrilaterals in the mesh, we can obtain the following expressions using the same techniques as we did for the triangular mesh:

$$N_Z^A = \sigma_1\gamma_1 + \sigma_2\gamma_2 + \sigma_3\gamma_3 + n^2 \quad , \tag{6.6}$$

where

$$\sigma_1 = n^2 - (2n_V + n_S)^2$$

$$\sigma_2 = n(n_I + n_V + 2n_S) - (n_I + n_V + 2n_S)^2 + 2(n_V + n_S)^2 \quad ,$$

$$\sigma_3 = n(n_I + n_S) - (n_I + n_S)^2 + 6(n_S + n_V)^2,$$

$$n = 4(n_V + n_S) + n_I \quad ,$$

and γ_1 , γ_2 and γ_3 are non-negative integers satisfying

$$\begin{aligned} 2\gamma_1 + \gamma_2 \qquad &= V-4 \\ 3\gamma_1 + 2\gamma_2 + \gamma_3 &= S-4 \\ \gamma_1 + \gamma_2 + \gamma_3 &= N_{\square}-1 \quad . \end{aligned} \tag{6.7}$$

The numbers γ_1, γ_2 and γ_3 are, respectively, the number of instances of case 1, case 2 and case 3 encountered during the reduction of the mesh. The coefficient matrix of (6.7) is singular, reflecting the fact that there are alternate ways to demolish the mesh, resulting in different values of γ_1, γ_2 and γ_3. We can resolve the problem as follows. First we observe that $-\sigma_3 = \sigma_1 - 2\sigma_2$. Using (6.7) in (6.6), we have

$$(6.8) \qquad N_Z^A = \sigma_1\gamma_1 + \sigma_2(V-4-2\gamma_1) + \sigma_3(2N_\square - S + 2 + \gamma_1) + n^2$$

$$= \gamma_1(\sigma_1 - 2\sigma_2 + \sigma_3) + \sigma_2(V-4) + \sigma_3(2N_\square - S + 2) + n^2$$

$$= \sigma_2(V-4) + \sigma_3(2N_\square - S + 2) + n^2.$$

If our mesh has H holes in it, and we rename the σ_3 of Lemma 6.3 as σ_4, equation (6.8) becomes

$$(6.9) \qquad N_Z^A = \sigma_2(V-4) + \sigma_3(2N_\square - S + 2 + H) + \sigma_4 H + n^2.$$

This information is important because it allows us to allocate the exact amount of storage for the non-zero elements of A as soon as we know the mesh and the characterization of the polynomial on each element. It is also useful in checking that our mesh is consistent and our program is working correctly.

The expressions we have derived allow us to obtain an estimate for the <u>density</u> $\mathcal{D}(A) = N_Z^A / N^2$ for finite element coefficient matrices. Using (6.4) and (2.5.6), along with (2.5.2) and (2.5.3), we have

(6.10) $$\mathcal{B}(A) \doteq \frac{\left(\frac{\sigma_1 + \sigma_2}{2}\right) N_\Delta}{\left[\left(n_I + \frac{1}{2} n_V + \frac{3}{2} n_S\right) N_\Delta\right]^2} = \frac{\Gamma(n_V, n_S, n_I)}{N_\Delta} ,$$

where

(6.11) $$\Gamma(n_V, n_S, n_I) = \frac{\sigma_1 + \sigma_2}{2\left(n_I + \frac{1}{2} n_V + \frac{3}{2} n_S\right)^2} .$$

The average number of non-zero elements per row of the coefficient matrix is obviously given by $\frac{1}{2}(\sigma_1 + \sigma_2)/(n_I + \frac{1}{2} n_V + \frac{3}{2} n_S)$.

Some typical values of Γ and average number of non-zero elements per row are tabulated below.

Element	$\Gamma(n_V, n_S, n_I)$	Average number of non-zero elements per row
1-3	14.00	7.00
2-6	5.75	11.50
3-4	8.08	20.20
3-10	3.78	17.00
4-6	6.59	29.67
4-15	2.94	23.5
5-6	10.15	45.67
5-21	2.48	31.00

Table 6.1

7. Analysis of Storage and Computational Requirements for a Model Problem

In this section we obtain estimates of θ_B , θ_P , γ_B and γ_P for a particular mesh, in order to demonstrate the savings attainable by using profile methods rather than band methods. The mesh we consider is obtained by subdividing a unit square into p^2 small squares of side $1/p$, and then subdividing each small square into two right triangles. An example with $p = 6$ is given below.

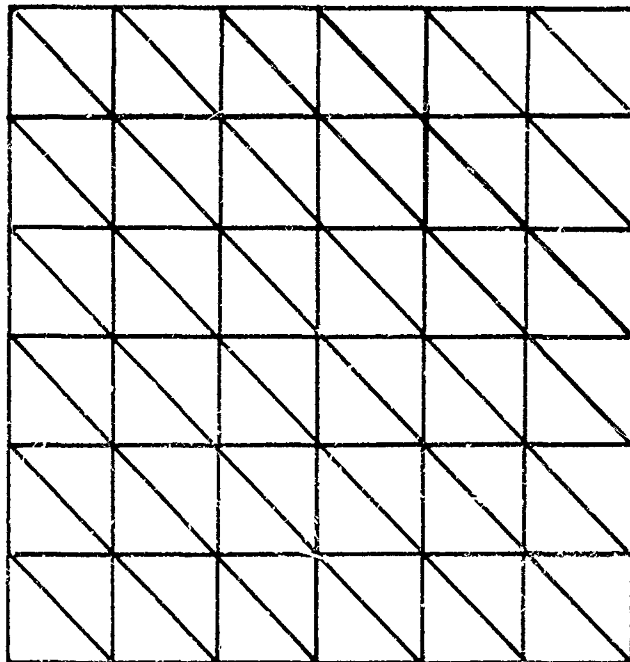


Figure 7.1. Six by Six Regular Right Triangular Mesh

As in Section 4.6, let n_V , n_S and n_I denote the number of parameters associated respectively with vertex nodes, the node(s) on each side, and the interior of each triangle. We number the nodes diagonal by

diagonal, beginning at the lower left hand corner, and considering nodes lying between consecutive diagonals as a row. For example, stencil 3-10 (Appendix A) would yield the numbering shown below:

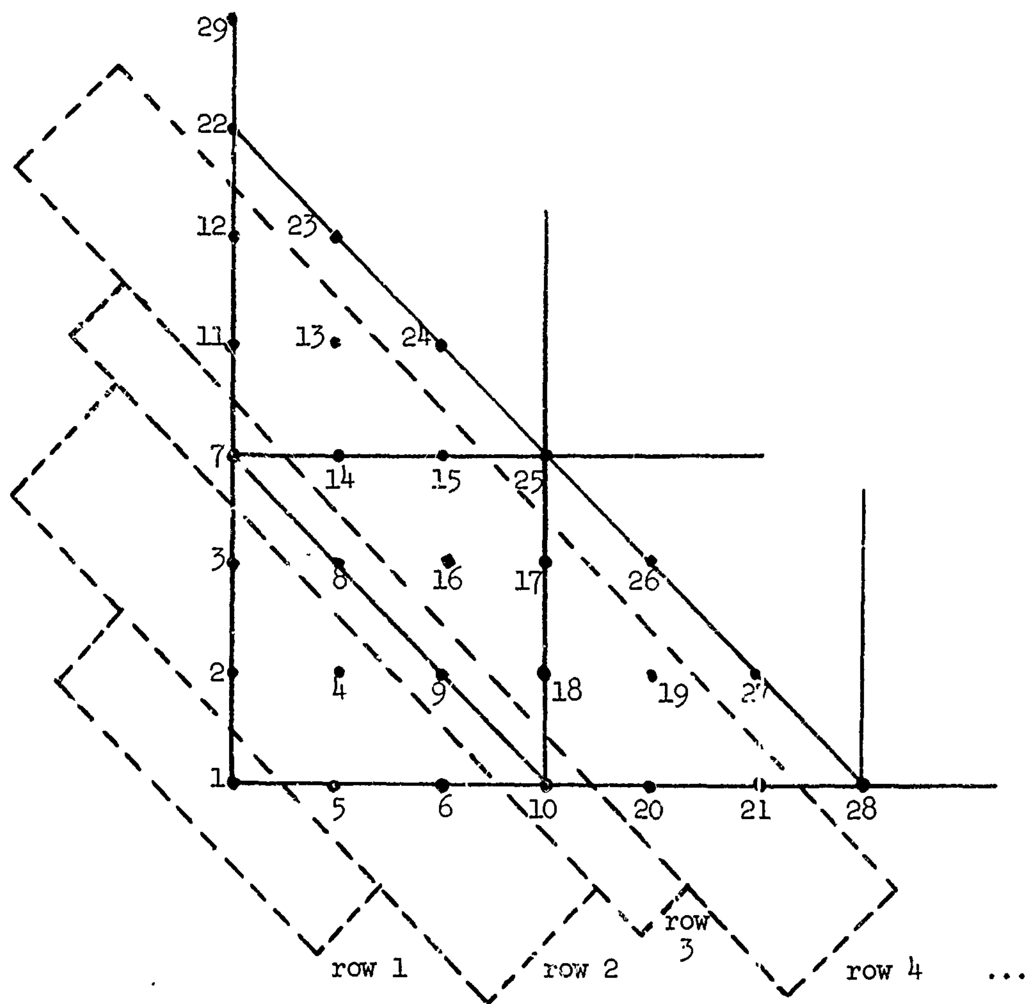


Figure 7.2

As usual, we denote our symmetric positive definite coefficient matrix by A, with Cholesky factorization LL^T. Making use of (6.1), and recalling the meaning of N_Δ, V_B, S_B and H, we can write

$$(7.1) \qquad N = n_V\left(\frac{N_\Delta + V_B}{2} - H + 1\right) + n_S\left(\frac{2N_\Delta + S_B}{2}\right) + n_I N_\Delta ,$$

where N is the dimension of A. For a p by p mesh such as Figure 7.1, $N_\Delta = 2p^2$ and $S_B = V_B = 4p$, yielding

$$(7.2) \qquad N = (n_V + 3n_S + 2n_I)p^2 + O(p) .$$

For this ordering, the bandwidth of A is given approximately by

$$(7.3) \qquad m \doteq (p+2)n_V + 3pn_S + (2p-1)n_I \doteq (n_V + 3n_S + 2pn_I)p$$
$$= \beta_B p .$$

Thus, $\theta_B \doteq \frac{\beta_B^3}{2} p^4$ and $\nu_B \doteq \beta_B^2 p^3$.

We now wish to obtain estimates for θ_P and ν_P. To simplify the algebra, we assume $n_V = 1$, $n_S = n_I = 0$ (piecewise linear polynomials). We can then prove the following

Theorem 7.1

For a $p \times p$ regular rectangular grid, the coefficient matrix A obtained using piecewise linear polynomials satisfies

$$(7.4) \qquad |Pr(A)| = \frac{2p^3}{3} + \frac{7}{2}p^2 + \frac{23}{6}p .$$

Proof:

Considering the first node point as diagonal 1, and recalling the definition of δ_i^A (Section 4.1), we see that

$$\delta_1^A = 0$$

$$\delta_2^A = 1$$

$$\delta_3^A = 2 \;, \quad \delta_4^A = 2$$

$$\delta_5^A = 3 \;, \quad \delta_6^A = 3 \;, \quad \delta_7^A = 3$$

$$\vdots$$

$$(7.5) \qquad \delta_i^A = r \;, \quad \frac{r(r-1)}{2} + 2 \le i \le \frac{r(r+1)}{2} + 1 \quad , \quad 1 \le r \le p \;.$$

Now for the main diagonal, $\delta_i^A = p+1$, and for the diagonals above the main diagonal we can show that

$$(7.6) \qquad \delta_{N-i}^A = r \;, \quad \frac{(r-1)(r-2)}{2} \le i \le \frac{r(r-1)}{2} - 1 \quad , \quad 3 \le r \le p+1 \;.$$

Using the formula $|Pr(A)| = \sum_{i=1}^{N} (\delta_i^A + 1)$ along with (7.5) and (7.6), we have

$$|Pr(A)| = \sum_{i=1}^{p} i(i+2) + p(p+2) + 1 + \sum_{i=1}^{p} i(i+1)$$

$$= \sum_{i=1}^{p} 2i^2 + 3i + p(p+2) + 1$$

$$= \frac{p(p+1)(2p+1)}{3} + p^2 + 2p + 1 = \frac{2p^3}{3} + \frac{7}{2}p^2 + \frac{23p}{6} + 1 \;.$$

Thus, using the profile storage scheme rather than the band storage scheme, we can save about one third of the storage for A or L . It is

straightforward (but tedious) to show that for a general stencil,

(7.7) $$|Pr(A)| \leq \frac{2}{3} \beta_P^2 p^3 ,$$

with $\beta_P \leq \beta_B$.

Recall that in Section 4.1 we showed that

(7.8) $$\Theta_P = \sum_{i=1}^{N} \frac{\delta_i^A(\delta_i^A + 3)}{2} .$$

Again assuming $n_V = 1$ and $n_S = n_I = 0$, we can prove the following

Theorem 7.2

Let the PC algorithm be applied to A . Then the number of multiplicative operations required to compute L is given by

(7.9) $$\Theta_P = \frac{1}{4} p^4 + 3p^3 + \frac{41}{4} p^2 + \frac{19}{2} p .$$

Proof:

Using (7.8) along with (7.5) and (7.6), we have

$$\Theta_p = \sum_{i=1}^{p} \frac{i(i+1)(i+4)}{2} + \frac{p(p+2)(p+5)}{2} + \sum_{i=1}^{p} \frac{i(i+2)(i+5)}{2}$$

$$= \sum_{i=1}^{p} (i^3 + 6i^2 + 7i) + \frac{p^3 + 7p^2 + 10p}{2}$$

$$= \frac{1}{4} p^4 + 3p^3 + \frac{41}{4} p^2 + \frac{19}{2} p \quad .$$

Again, with some tedious algebra, we can show that

$$\Theta_p \leq \frac{1}{4} \beta_p^3 p^4 \quad . \tag{7.10}$$

It is, therefore, possible to halve the computation required to compute L by using profile instead of band methods. Note that we did not prejudice our comparison by ordering diagonally, since the bandwidth would be the same if we numbered our nodes in the usual row by row fashion.

8. Miscellaneous Topics and Concluding Remarks

In this section we discuss several modifications of elimination methods which are useful in various circumstances.

We begin by discussing a technique often referred to by engineers as "static condensation" (SC), which can be employed to eliminate some of the unknowns in (3.4.4) at the element level [F1]. As we described in Chapter 1, a basis function corresponding to an internal node of T^ν is non-zero only on T^ν. Hence, the corresponding parameter is connected only to parameters associated with T^ν. Suppose we partition q^ν into q_1^ν and q_2^ν, where q_2^ν corresponds to interior node parameters. We can write (3.4.1) in the form

$$\begin{pmatrix} B_{11}^\nu & B_{12}^\nu \\ B_{21}^\nu & B_{22}^\nu \end{pmatrix} \begin{pmatrix} q_1^\nu \\ q_2^\nu \end{pmatrix} = \begin{pmatrix} b_1^\nu \\ b_2^\nu \end{pmatrix} , \tag{8.1}$$

where q_2^ν is independent of T^ν, $\gamma \neq \nu$. Then (8.1) can be replaced by (8.2) by eliminating q_2^ν:

$$[B_{11}^\nu - B_{12}^\nu B_{22}^\nu B_{21}^\nu] q_1^\nu = b_1^\nu - B_{12}^\nu B_{22}^{\nu^{-1}} b_2^\nu . \tag{8.2}$$

In this way, the dimension of A_{11} (see Section 3.4) can be reduced by $N_\Delta n_I$. We can carry out a somewhat superficial analysis of the model problem discussed in Section 4.7 to show the savings possible by using this technique. To simplify the analysis we will consider the use of the band Cholesky algorithm, and consider element 3-4. It is easy to show that using the ordering of Section 4.7, the band width m is about $5p$ and the number of equations N is about $5p^2$, yielding

$$\theta_B \doteq 125p^4 . \tag{8.3}$$

Now consider the corresponding quantities if we apply static condensation and eliminate $2p^2$ variables before assembly. The bandwidth m is now only about $3p$, and $N \doteq 3p^2$. Thus

$$(8.4) \qquad \hat{\theta}_B \doteq 27p^4 .$$

It is fairly easy to show that the number of multiplicative operations required to eliminate the variables is

$$(8.5) \qquad \theta_{SC} \doteq 32p^2 ,$$

which means the technique pays (in terms of multiplicative operations) for this particular element, problem, and solution method for all p. Of course, its use might be justified for storage reasons alone, even if it did not reduce the computation.

In general, θ_{SC} is given by

$$(8.6) \qquad \left[\frac{n_I^3}{6} + 3n_I(n_I + 1)(n_S + n_V) + 9n_I(n_V + n_S)^2 \right] N_\Delta .$$

Another technique sometimes used in connection with solving finite element equations is the so-called frontal-solution method [I2, K2]. The basic strategy is to combine the assembly and decomposition of A by alternating between the accumulation of coefficients of the equations (most of the coefficients depend on more than one element) and the elimination. A square submatrix of A (in some stage of reduction) is the only main storage required. The matrix corresponds to "active" variables; that is, variables which have not been eliminated and for which there are non-zero coefficients in the equations so far encountered. The subset of active variables continuously changes as new elements are processed. The main point that is usually made in favor of these schemes is that variables are

eliminated as soon as possible, rather than in a predetermined order. However, this flexibility is obtained at a rather high cost in programming complexity, and the question of ordering has really only been moved a level higher. The problem of optimal equation ordering has been replaced by the problem of optimal order of element assembly. Our general impression is that these methods will be most valuable when main storage is at a high premium.

CHAPTER 5

FINITE ELEMENT SOLUTIONS TO SOME SELECTED PROBLEMS

1. Introduction

In this chapter we will present finite-element solutions to some much-studied problems for which numerical solutions have been presented in the literature. Our purpose is not necessarily to present more accurate solutions than have been presented before, but rather to demonstrate that the finite element method enables us to obtain comparatively good results efficiently and without resorting to special methods. We will provide evidence suggesting that the finite-element method is not only desirable because of its flexibility regarding irregular domains but is competitive or superior to common alternate methods with respect to efficiency.

The term efficiency is somewhat difficult to define quantitatively since storage requirements, computer time, and manpower have different relative costs in different situations. Loosely, efficiency will mean "number of correct digits per dollar".

We would like to emphasize that the finite-element solutions presented in this chapter have been produced by a general program. No use was made of any special characteristics of the problems other than those an engineer would reasonably expect. For example, we graded the net small near the re-entrant corner of the L-shaped membrane eigenvalue problems (Section 2, this chapter), but we did not attempt to incorporate "singular functions" into the basis [F2,F6].

Since we are using a Ritz procedure, our computed eigenvalues for the problems below are upper bounds for the true eigenvalues.

2. The L-Shaped Membrane Eigenvalue Problem

The L-shaped membrane eigenvalue problem has been studied by many authors. For background material, see Forsythe and Wasow [F5] and Moler [M3], and for various special computational methods, see Reid and Walsh [R2], Fix [F2], Schwartz [S7], and Fox, Henrici, and Moler [F6]. The domain R consists of the union of three unit squares, and we wish to find the stationary values λ_i $(0 < \lambda_1 < \lambda_2 \leq \lambda_3 \leq \cdots)$ of the functional:

$$I[u] = \iint_R [u_x^2 + u_y^2]\,dx\,dy \Big/ \iint_R u^2\,dx\,dy \quad , \tag{2.1}$$

where $u = 0$ on the boundary S.

The interesting aspect of this problem is provided by the re-entrant corner, which leads to unbounded derivatives of the fundamental eigenfunction in the neighborhood of the corner. Thus, the eigenfunction is difficult to approximate by functions which do not exhibit a similar behavior. The value $\lambda_1 = 9.63972$ reported in [F6] is accurate to the last digit, and we will use it for comparison.

Our first experiments make use of the following triangular mesh:

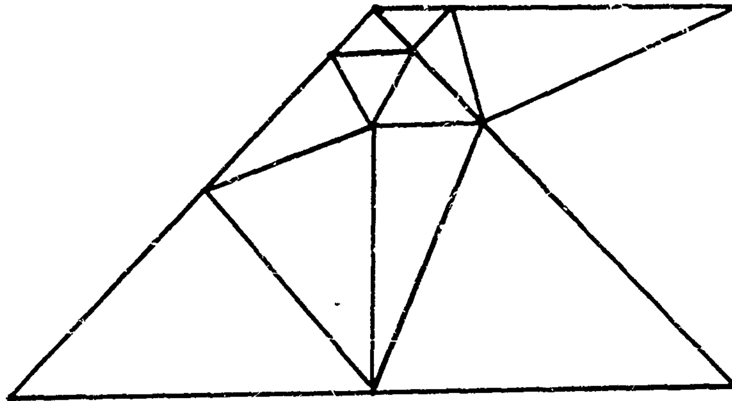


Figure 5.1

We are obviously making use of the symmetry of the first eigenfunction here, and have graded the net appropriately near the corner. In the tables below k is the factor by which the mesh of Figure 5.1 was sub-divided. The eigenvalues we found using inverse iteration [W2] with an initial guess of 9.6 . The computed λ_1^* is believed to be correct to the last digit. Set-up time includes the time required to generate the mesh and order the nodes as well as the time required to generate and assemble the equations. The missing times in the table were so small that they were meaningless. All times are in seconds on an IBM 360/91.

$\lambda_1^*-\lambda_1$ for piecewise linear functions

k	Number of Equations	Bandwidth	Set-up Time	Solution Time	$\lambda_1^*-\lambda_1$
1	5	3	.33	--	3.4003
2	22	6	.57	.05	1.0089
3	51	12	.62	.10	.4605
4	92	12	.93	.17	.2640
5	145	15	1.33	.30	.1718
6	210	18	2.34	.55	.1210
7	287	21	2.73	.90	.0901
8	376	24	3.53	1.28	.0699

Table 2.1

The rate of convergence of the computed λ_1^* to λ_1 as $k \to \infty$ is obviously exceedingly slow.

The systems of linear equations involved in the inverse iteration routine we solved using a band Gaussian elimination routine. The code appears in Appendix B. It is interesting to note that the set-up time dominates the solution time in all cases. This is due in part to the characteristics of the IBM 360/91, which has a very fast floating-point arithmetic unit and a look-ahead instruction stack. Both features tend to make "number crunching" tasks, such as Gaussian elimination, proceed rapidly and efficiently. The set-up procedure, on the other hand, requires considerable bookkeeping and branching. Programs of this type do not make effective use of the powerful machine features mentioned above. Another reason for the relatively large set-up time is that we are using low degree polynomials. The number of triangles to be processed (and the associated overhead) is larger with respect to N than it would be for quadratics, for example. Note, however, that the ratio

(Set-up time)/(Solution time) is steadily (if slowly) decreasing.

Our second experiment again makes use of the mesh of Figure 5.1, but we now use polynomials of higher degree to demonstrate how efficient they can be. Table 2.2 contains results for polynomials of degrees 1 through 6 ; in all cases the <u>original mesh</u> was used. Our inverse iteration routine for these experiments used a symmetric indefinite equation solver using the pivoting algorithm of J. R. Bunch [B13]. The code for this computation appears in Appendix B.

$\lambda_1^* - \lambda_1$ for Piecewise Polynomials of Degrees 1 to 6

Degree	Number of Equations	Bandwidth	Set-up Time	Solution Time	$\lambda_1^* - \lambda_1$
1	5	3	.33	--	3.4003
2	22	11	.43	.1	.3720
3	51	24	.70	.25	.0160
4	92	42	.87	.95	.0063
5	145	65	1.7	3.02	.0034
6	210	101	2.68	6.02	.0021

Table 2..2

It is obvious that for this problem the use of polynomials of degree > 1 are considerably more effective than linear ones.

It is interesting to note that the λ_1^* obtained using quintic polynomials (Table 5.2) yielding 145 equations is comparable to the λ_1^* obtained by Moler [M3] using finite difference methods on a uniform mesh with h = 1/100 (yielding 15,000 equations). Our storage requirements

were virtually the same; we required 15385 words (including the storage of A and B of the generalized eigenvalue problem $Ax = \lambda Bx$). Moler's Fortran program, written specifically for this problem, took about 12 minutes to execute on an IBM 7090. Thus there is a factor of roughly 150 in execution times. The ratio of speeds of the arithmetic units is about 100, while the effective memory speed ratio is about 10. The ratio of times for other operations lie somewhere between these two extremes. We feel we can safely say that the finite element method is at least fully competitive with finite difference methods for this problem.

It was, of course, not necessary to use inverse iteration. We could have used a method due to Peters and Wilkinson [P1] which essentially finds the zeros of $\det(A-\lambda B)$. Although the running times would be considerably larger than for inverse iteration, the required storage for our quintic problem would be a total of 10,536 words (storage for A and B and an additional $((m+1) \times (2m+1))$ words for the determinant evaluation). Both this method and inverse iteration can be used to find subdominant eigenvalues, whereas the method used in Moler [M3] is applicable only for an end eigenvalue. To find subdominant eigenvalues using his technique would require some form of deflation to render the dominant eigenvalues equal to zero. To avoid making the coefficient matrix dense, the deflation would have to be done implicitly which implies that the eigenvectors corresponding to dominant eigenvalues would have to be available. We feel that the ability of the high order finite element methods to obtain respectable results using only moderate numbers of parameters is particularly important for eigenvalue problems because it enables us to

apply well known, dependable methods for finding the eigenvalues of the discrete problem.

We again emphasize that we are not implying that finite element methods are the best ones to use for solving this particular problem. Indeed, the method proposed by Fox, Henrici and Moler [F6] is probably the best known method for finding the eigenvalues of the L-shaped membrane. However, the use of such techniques requires information which may only be known to an expert in the field, and the utilization of them in a general code is complicated.

3. Eigenvalues of Rhombical Domains

Bounds for the eigenvalues of rhombical domains have been obtained by Moler [M4], Birkhoff and Fix [B7], and Stadter [S4]. Moler obtains his bounds using a method of particular solutions, and Stadter obtains bounds using the method of intermediate problems [S4]. In this section we will show that with finite element formulations having relatively few parameters we can get close to or within the bounds produced by the methods described in the above references.

The problem we considered is the equation (2.1) of Section 2 with a rhombical domain of side π and skew angle θ as indicated below:

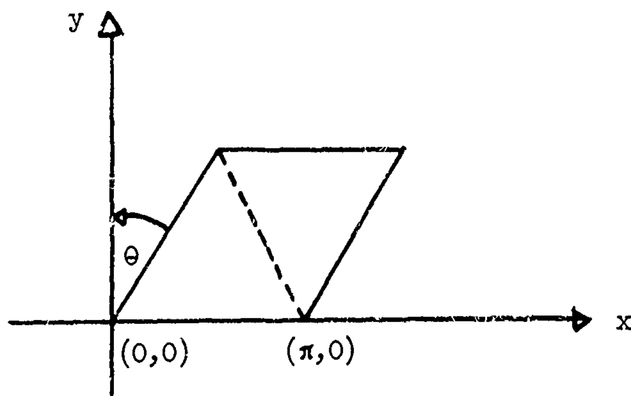


Figure 5.2

Our first experiment takes no account of symmetry, and the results are compared with some of the bounds presented by Moler [M4]. These results are summarized in the table below. As before, k indicates the factor by which the input mesh (indicated by the dashed line in Figure 5.2) has been subdivided. N is the number of equations, m is the bandwidth and d is the degree of the piecewise polynomials.

The method employed by Moler utilizes particular solutions to the Laplacian operator, and the 20 particular solutions used were carefully chosen to agree with symmetries of the eigenfunction being approximated. Each bound calculation required about 20 seconds on an IBM 360/67. Our calculations were done on an IBM 360/91. Our set-up times (for each example) and solution times (for each eigenvalue) have been included in Table 3.1 for comparison purposes. The 360/67 and 360/91 have radically different design features and a comparison between the two machines is difficult. The largest ratio of execution times this author has encountered between identical programs run on the two machines is 15, and that was for a very special program. Usually the ratio is from three to six and is almost always less than ten.

Case 1: Rhombical Membrane Eigenvalues: $\theta = 30^\circ$.

	λ_1^*	λ_2^*	λ_3^*	λ_4^*	Set-up and Solution times
Moler's Bounds	2.51921 2.52606	5.33333 5.33334	7.24150 7.29028	8.47510 8.50997	≈ 20 sec per eigenvalue on an IBM 360/67
k = 2, N = 49, m = 22, d = 4	2.52302	5.33341	7.26942	8.5047	.41 .13
k = 3, N = 121, m = 43, d = 4	2.52284	5.33339	7.26653	8.49424	.69 .95
k = 4, N = 225, m = 65, d = 4	2.52279	5.33334	7.26611	8.49374	1.36 3.7
k = 2, N = 81 m = 35, d = 5	2.52284	5.33340	7.26651	8.49420	1.2 .4

Table 3.1

Case 2: Rhombical Membrane Eigenvalues: $\theta = 5^{\circ}$

	λ_1^*	λ_2^*	λ_3^*	λ_4^*
Moler's Bounds	2.01218 2.01248	4.90375 4.90403	5.15659 5.15750	7.99206 7.99394
N = 49, m = 22, d = 4, k = 2	2.01232	4.90567	5.16407	8.00979
N = 121, m = 43, d = 4, k = 3	2.01226	4.90405	5.15735	7.99516
N = 81, m = 35, d = 5, k = 2	2.01226	4.90408	5.15730	7.99851
N = 196, m = 69, d = 5, k = 3	2.01225	4.90389	5.15705	7.99308

Table 3.2

Our first observation is that again the higher degree polynomials appear to be more efficient. For example, in Case 1, using quintics with $N = 81$ and $m = 35$ yields results as good as the quartic example having $N = 121$ and $m = 43$. For Case 2, the singularities in the derivatives of the eigenfunctions near the corners are less troublesome, and the value of the higher degree polynomials is less pronounced, although still apparent. We point out that our numbers are upper bounds to the true eigenvalues.

Moler's method is clearly superior if accurate upper <u>and</u> lower bounds are required, or if approximations to many eigenvalues are desired. However, his method may be expensive and/or difficult to apply to problems whose operators do not have simple or easily generated families of particular solutions.

Moler's results are for moderate values of θ, and only for the fixed membrane problem. We now wish to make some comparisons with the results of Stadter [S4] and Birkhoff and Fix [B7]. They report bounds for $\theta = 30^\circ\ (15^\circ)\ 75^\circ$ for the rhombus fixed at all edges, and Stadter reports bounds for the rhombus fixed at two opposite edges and free on the remaining two edges.

We begin with the fixed membrane problem. The bounds reported are for eigenvalues corresponding to eigenfunctions which are symmetric with respect to both diagonals. For purposes of comparison, we restricted our first experiment correspondingly. Our domain is the hatched area shown below:

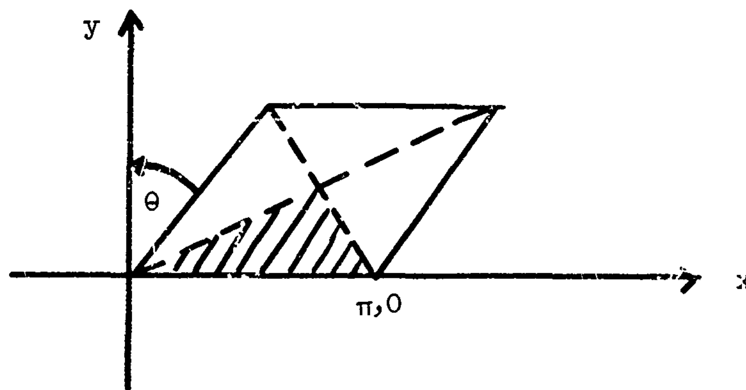


Figure 3.1

The boundary condition $u_n = 0$ is imposed along the boundary of the hatched region interior to the rhombus. This is easy to do since it is a natural boundary condition.

In the discussion below λ_n^A and λ_n^B are computed eigenvalues reported by Birkhoff and Fix [B7]. The values λ_n^A were obtained using the Rayleigh-Ritz method with the approximating space spanned by the affine transforms of the first 30 eigenfunctions of the square membrane. The values λ_n^B were obtained using a second space of dimension 30 which included special singular functions having the appropriate behaviour at the corners. In both cases, only the even-even symmetry class was sought.

The eigenvalues $\lambda_n^{U,k}$ and $\lambda_n^{L,k}$ below are upper and lower bounds supplied by the method of intermediate problems as applied to the rhombical membrane problem by Stadter [S4]. The superscript k indicates the number of intermediate problems used, and is the size of the two (dense) matrix eigenvalue problems which must be solved to obtain the bounds. In [S4] Stadter reports bounds for $k = 15$ and $\theta = 30^\circ$ (15°) 75°, and in [W1] he reports bounds for $k = 30$ and $\theta = 45^\circ$.

The eigenvalues $\lambda_n^{\ell,k}$ below are our finite element results for piecewise polynomials of degree ℓ yielding k by k (band) eigenvalue problems.

In Table 6 we compare λ_n^B , $\lambda_n^{5,55}$, $\lambda_n^{L,50}$ and $\lambda_n^{U,50}$. For the following reasons, we feel it is fair to compare λ_n^B with $\lambda_n^{5,55}$ even though the space which produced λ_n^B is only of dimension 30. First, Birkhoff and Fix report that $2\frac{1}{2}$ times as much computer time was needed to find the λ_n^B's as the λ_n^A's for a given angle θ . (We assume this increase was caused by complications introduced by the incorporation of the singular functions.) Second, our eigensystems have band structure, whereas theirs are dense. A third somewhat qualitative reason might be termed the "nuisance factor". All our finite element computations were done with a general purpose program; no special modifications with regard to special basis functions or geometry were necessary.

The time required to generate the finite element eigenproblem for each angle on the IBM 360/91 was about 0.6 seconds. About 0.2 seconds were required to find each eigenvalue using inverse iteration. By comparison, 2 minutes were required on an IBM 7094 to produce the λ_n^B's for a given θ . [Since it appears that the major portion of the time used was for the generation of the eigenproblem rather than its solution, the fact that Birkhoff and Fix used a method yielding <u>all</u> the eigenvalues of the discrete problem is relatively unimportant.] Roughly 4 seconds of IBM 360/91 time was required to produce the upper and lower bounds ($\lambda^{U,50}$ and $\lambda_n^{L,50}$) for each angle using the method of intermediate problems. The results are tabulated below for $n = 1,2,\ldots,6$ and $\theta = 30^\circ$, 45° , 60° and 75° .

Symmetric Eigenvalues for the Fixed Rhombical Domain

n	$\lambda_n^{5,55}$	λ_n^{B}	$\lambda_n^{L,50}$	$\lambda_n^{U,50}$
Case 1: $\theta = 30^\circ$				
1	2.5228	2.5238	2.5224	2.5241
2	8.4939	8.5060	8.4916	8.5008
3	14.233	14.256	14.224	14.261
4	17.156	17.183	17.139	17.167
5	27.173	27.110	26.983	27.096
6	29.606	29.620	29.433	29.537
Case 2: $\theta = 45^\circ$				
1	3.5210	3.5210	3.5201	3.5263
2	10.158	10.190	10.154	10.173
3	18.785	18.864	18.737	18.802
4	22.115	22.135	22.095	22.214
5	30.153	30.289	29.785	29.942
6	39.663	39.582	39.493	39.777
Case 3: $\theta = 60^\circ$				
1	6.3238	6.3598	6.3217	6.3485
2	14.968	15.088	14.958	15.005
3	25.333	25.571	25.202	25.338
4	38.064	38.981	37.436	37.774
5	43.581	43.717	43.480	44.013
6	54.267	56.379	51.883	52.575
Case 4: $\theta = 75^\circ$				
1	20.194	20.283	20.185	20.407
2	36.373	36.452	36.301	36.617
3	53.596	53.562	52.794	53.499
4	76.746	80.125	70.951	72.660
5	110.20	111.52	90.964	94.982
6	154.89	144.38	112.87	121.75

Table 3.3

We offer the following observations:

(1) The remarks of Birkhoff and Fix suggesting that their Rayleigh-Ritz methods yield much more accurate upper bounds than the method of intermediate problems seems to be barely justifiable. In [B7] their comparisons of λ_n^A and λ_n^B are against $\lambda^{U,15}$ for $\theta = 30°$, $60°$ and $75°$. For $\theta = 45°$ the comparison is against $\lambda_n^{U,30}$, and for this case λ_n^A was a sharper upper bound in only half of the cases, and although λ_n^B was better in all cases, it was only marginally better in most of them.

(2) The upper bounds produced by the finite element method appear to be fully competitive with the λ_n^B's, and are appreciably better for the lower eigenvalues.

(3) Experiments with polynomials of various degrees again indicate that efficiency increases with increasing polynomial degree.

(4) Our finite element solutions made no use of

(a) information about the behavior of the solution near the corners of the domain

(b) the fact that the domain is affinely equivalent to one in which the eigenproblem can be solved exactly.

We feel that these points are important because the utilization of (a) appears to be awkward in a general implementation, and (b) places a rather severe restriction on the application of the method of intermediate problems.

We now turn briefly to the fixed-free rhombical membrane eigenvalue problem. Stadter [S4] restricted his attention to eigenvalues corresponding to eigenfunctions symmetric with respect to the center of the rhombus. It

was not convenient for us to restrict our problem correspondingly, so we solved the "full" problem. We report results for $\lambda_1^{4,15}$ and $\lambda_3^{4,15}$.

$\lambda_1^{4,15}$ and $\lambda_3^{4,15}$ for the Fixed-Free Rhombical Membrane

θ	$\lambda_1^{4,15}$	$\lambda_1^{L,15}$	$\lambda_1^{U,15}$	$\lambda_3^{4,15}$	$\lambda_3^{L,15}$	$\lambda_3^{U,15}$
30	1.2343	1.1820	2.8550	4.9105	4.6585	5.1547
60	2.8550	2.5046	3.6533	7.6453	6.9881	9.5382
75	8.3400	6.8038	13.043	19.177	14.233	27.438

Again, with a moderate number of parameters we can easily improve on the upper bounds produced by the method of intermediate problems.

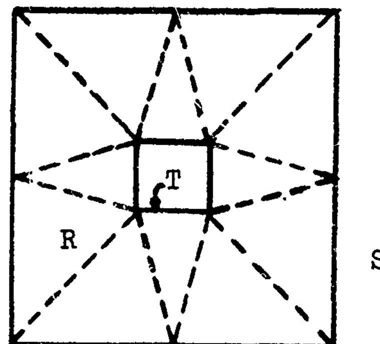
4. A Dirichlet Problem

We now consider finite element solutions to the following problem:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad \text{on } R ,$$

(4.1)

$$u = e^x \cos y \quad \text{on } S \cup T .$$



The boundary S is a 1.28×1.28 square and T is a $.25 \times .25$ square with lower left corners at $(0,0)$ and $(.5,.5)$ respectively. The input mesh is indicated by the dashed lines in the diagram above. Experiments were run on an IBM 360/91.

Details of the various polynomials can be found in Appendix A. As before, N indicates the number of finite element equations and k is the factor by which the input mesh was subdivided. The profile Cholesky algorithm and the RCM ordering (see Chapter 4) were employed in all cases.

We begin by comparing different element/mesh combinations which yielded roughly the same accuracy.

Element	N	k	Set-up Time (Seconds)	Solution Time (Seconds)	Operations for Solution $\times (10^{+3})$	Error $\times (10^{-5})$	Storage L	Storage A
2-6	540	5	3.18	.53	154	1.86	10,544	3000
3-4	216	3	1.56	.22	37	1.28	3,099	1620
3-10	180	2	1.13	.13	28	1.63	2,425	1188
4-15	72	1	.65	.04	7	1.14	684	468

Table 4.1

Table 4.1 demonstrates dramatically the value of using high degree polynomials for solving this problem. Set-up times, solution times and storage requirements decrease as the degree d of the polynomial increases. Observe the striking decrease in the operations required to solve the generated linear system.

We now present some experiments using the <u>initial</u> mesh and varying the degree.

Element	N	Set-up Time	Solution Time	Operations for Solution $(\times 10^3)$	Error	Storage L	Storage A
3-10	36	.482	--	2	2.35(-4)	209	144
4-15	72	.65	.04	6	1.14(-5)	684	468
5-21	120	1.10	.13	21	4.06(-7)	1692	1140
6-28	180	1.86	.20	53	1.16(-8)	3465	2340

Table 4.2

Again the case for higher degree polynomials is apparent. Compare, for example, the third entries in Tables 4.1 and 4.2. Their demands on system resources are about the same, but the error for the quintic is more than an order of magnitude less.

To compare the above results with what could be expected using finite difference methods we solved the problem using the standard five-point difference operator on a uniform square mesh with mesh width of $1/100$. The solution was obtained using an imbedding approach [G1,B15] which makes use of very fast direct methods for solving the discrete Laplacian equations on a rectangular domain. The set-up time for this procedure is large ($\approx$ 25 seconds for our problem on the IBM 360/91) and consists of computing a $q \times q$ "capacitance matrix". In our problem $q = 100$ and the computation of the capacitance matrix involves solving q 127×127 rectangular problems. However, once this initialization is done, we can obtain a solution to our given problem by solving 2 rectangular problems and a dense q by q system of linear equations. Assuming that we have computed and decomposed the capacitance matrix beforehand, we can solve our problem in about .7 seconds. This latter "solution time" has been found to be superior to SOR or ADI solution times (by factors of 5 to 8) for a number of typical problems [B15].

Thus, a (conservative) entry in Table 4.2 for finite differences would be

N	Solution Time	Operations	Error	Storage
15,504	.7	10^6	$7 \cdot 10^{-6}$	22000

Each solution of the 127×127 rectangular problem requires about .3 seconds on the IBM 360/91. Thus, even using the iterative scheme (based on fast direct methods) proposed by George [G1] which avoids the calculation of the capacitance matrix is unlikely to compare favorably in overall time (solution and set-up time) with the last entry in Table 4.2. Anyway, an equally important consideration is storage requirements, and the last entry in Table 4.2 requires only 5805 words. The observed error for the sixth degree polynomial was 1.16×10^{-8} compared to 7×10^{-6} for the difference equations.

Again we should point out that there are still better ways to solve this problem if we are prepared to take advantage of its particular characteristics. Moler (private communication) solved the problem by using a linear combination of particular solutions as a trial solution and determining the coefficients of the expansion by minimizing the two-norm of the error at a discrete set of points on the boundary $S \cup T$. The least squares solution of a 26×15 problem was all that was required and the program was only a few pages long; the error, however, was around 10^{-10}.

References

A1 B. Allen, "Procedures for the numerical solution of elliptic partial differential equations," Culham Operating System note no. 1/67, Culham Laboratory, England, 1967.

B1 Ivo Babuska, "Numerical solution of boundary value problems by the perturbed variational principle," Technical Note BN-624, Inst. for Fluid Dynamics and Appl. Math., University of Maryland, College Park, Maryland, 1969.

B2 Ivo Babuska, "Error bounds for finite element method," Numer. Math., 16 (1971), pp. 322-333.

B3 Ivo Babuska, "Finite element method for domains with corners," Technical Note BN-636, Inst. for Fluid Dynamics and Appl. Math., University of Maryland, College Park, Maryland, 1970.

B4 W. D. Barfield, "Numerical method for generating orthogonal curvilinear meshes," J. Computational Phys., 1 (1970), pp. 23-33.

B5 G. P. Bazeley, Y. K. Cheung, B. M. Irons, and O. C. Zienkiewicz, "Triangular elements in plate bending - conforming and non- conforming solutions," Proc. Conf. Matrix Meth. Struct. Mech., Wright Patterson Air Force Base, Dayton, Ohio, 1965

B6 Koplein Bell, "A refined triangular plate bending element," Internat. J. Numer. Meth. Engrg., 1 (1969), pp. 101-122.

B7 Garrett Birkhoff and George Fix, "Accurate eigenvalue computations for elliptic problems," pp. 111-151 of B9.

B8 G. Birkhoff, M. H. Schultz and R. S. Varga, "Piecewise Hermite interpolation in one and two variables with applications to partial differential equations," Numer. Math., 11 (1968), pp. 232-256.

B9 Garrett Birkhoff and Richard S. Varga, editors, Numerical Solution of Field Problems in Continuum Mechanics, SIAM-AMS Proceedings, American Mathematical Society, Providence, Rhode Island, 1970.

B10 James H. Bramble, editor, Numerical Solution of Partial Differential Equations, Academic Press, New York, 1966.

B11 J. H. Bramble and Alfred Schatz, "Rayleigh-Ritz-Galerkin methods for Dirichlet's problem using subspaces without boundary conditions," Comm. Pure Appl. Math., 23 (1970), pp. 653-675.

B12 James H. Bramble and Milos Zlámal, "Triangular elements in the finite element method," Math. Comp., 24 (1970), pp. xxx-xxx.

B13 J. R. Bunch, "On direct methods for solving symmetric systems of linear equations," Technical Report No. 33, Computer Center, University of California at Berkeley, 1969.

B14 Robert G. Busacker and Thomas L. Saaty, Finite Graphs and Networks: An Introduction with Applications, McGraw-Hill Book Company, New York, 1965.

B15 B. L. Buzbee, F. W. Dorr, J. A. George, and G. H. Golub, "The direct solution of the discrete Poisson equation on irregular regions," Computer Science Dept. Technical Report CS195, Stanford University, Stanford, California, 1970.

C1 R. W. Clough and J. L. Tocher, "Finite element stiffness matrices for analysis of plate bending," Proc. Conf. Matrix Meth. Struct. Mech., Wright Patterson Air Force Base, Dayton, Ohio, 1965.

C2 Lothar Collatz, The Numerical Treatment of Differential Equations, Springer Verlag, Berlin, 1960.

C3 Colin W. Cryer, "Topological problems arising when solving boundary value problems for elliptic partial differential equations by the method of finite differences," J. Assoc. Comput. Mach., 18 (1971), pp. 63-74.

C4 Y. K. Cheung and J. O. Pedro, "Automatic preparation of data cards for the finite element method," Research Report, Civil Engineering Division, School of Engineering, University College of Swansea, England, 1966.

C5 E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," Proc. 24th National Conf., Assoc. Comput. Mach., ACM Publication P-69, 1122 Ave. of the Americas, New York, N. Y., 1969.

C6 Alfonso F. Cárdenas and Walter J. Karplus, "PDEL - A language for partial differential equations," Comm. Assoc. Comput. Mach., 13 (1970), pp. 184-190.

D1 Robert C. Daley and Jack B. Dennis, "Virtual memory, processes, and sharing in Multics," Comm. Assoc. Comput. Mach., 11 (1969), pp. 306-312.

D2 F. W. Dorr, "The direct solution of the discrete Poisson equation on a rectangle," SIAM Rev., 12 (1970), pp. 248-263.

D3 George Dupuis and Jean-Jacques Goël, "Elements finis raffinés en elasticite bidimensionelle," Z. Angew. Math. Phys., 20 (1970), pp. 858-881.

E1 D. J. F. Ewing, A. J. Fawkes, and J. R. Griffiths, "Rules governing the number of nodes and elements in a finite element mesh," Internat. J. Numer. Meth. Engrg., 2 (1970), pp. 597-600.

E2 M. Engeli, "Design and implementation of an algebraic processor," Habilitationsschrift, Institut für Angewandte Mathematik der ETH, Zurich, 1966.

F1 Carlos Felippa and Ray W. Clough, "The finite element method in solid mechanics," pp. 210-252 of B9.

F2 George Fix, "High order Rayleigh-Ritz approximations," J. Math. Mech., 18 (1969), pp. 645-657.

F3 George Fix and Kathren Larsen, "Iterative methods for finite element approximations to elliptic boundary value problems," to appear.

F4 George E. Forsythe and Cleve B. Moler, Computer Solution of Linear Algebraic Systems, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1967.

F5 George E. Forsythe and W. R. Wasow, Finite-Difference Methods for Partial Differential Equations, John Wiley and Sons Inc., New York, 1960.

F6 L. Fox, P. Henrici, and C. Moler, "Approximations and bounds for eigenvalues of elliptic operators," Siam. J. Numer. Anal., 4 (1967), pp. 89-102.

F7 C. O. Frederick, Y. C. Wong and F. W. Edge, "Two dimensional automatic mesh generation for structural analysis," Internat. J. Numer. Meth. Engrg., 2 (1970), pp. 133-144.

F8 Kurt Fuchel and Sidney Heller, "Considerations in the design of a multiple computer system with an extended core storage," Comm. Assoc. Comput. Mach., 11 (1969), pp. 334-340.

G1 J. Alan George, "The use of direct methods for the solution of the discrete Poisson equation on non-rectangular regions," Computer Science Dept. Technical Report STAN-CS-70-159, Stanford University, Stanford, California, 1970.

G2 J.-J. Goël, "Construction of basic functions for numerical utilization of Ritz's method," Numer. Math., 12 (1968), pp. 435-447.

G3 F. Gustavson, W. Liniger, and R. Willoughby, "Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations," J. Assoc. Comput. Mach., 17 (1970), pp. 87-109.

H1 R. J. Herbold, M. H. Schultz, and R. S. Varga, "The effect of quadrature errors in the numerical solution of boundary value problems by variational techniques," Aequationes Math., 3 (1969), pp. 96-119.

H2 M. J. L. Hussey, R. W. Thatcher, and M. J. M. Bernal, "Construction and use of finite elements," J. Inst. Math. Appl., 6 (1970), pp. 262-283.

H3 John K. Hayes, "Four computer programs using Green's third formula to numerically solve Laplace's equation in inhomogeneous media," Report LA-4423, Los Alamos Scientific Laboratory of the University of California, Los Alamos, New Mexico, 1970.

I1 Bruce M. Irons, "A conforming quartic triangular element for plate bending," Internat. J. Numer. Meth. Engrg., 1 (1969), pp. 29-45.

I2 Bruce M. Irons, "A frontal solution program for finite element analysis," Internat. J. Numer. Meth. Engrg., 2 (1970), pp. 5-32.

J1 A. Jennings, "A compact storage scheme for the solution of symmetric linear simultaneous equations," Comput. J., 9 (1966), pp. 281-285.

K1 H. A. Kamel and H. K. Eisenstein, "Automatic mesh generation in two and three dimensional interconnected domains," Symposium on High Speed Computing of Elastic Structures, Liege, Belgium, 1970.

K2 Ian P. King, "An automatic reordering scheme for simultaneous equations derived from network systems," Internat. J. Numer. Meth. Engrg., 2 (1970), pp. 523-533.

K3 L. V. Kantorovich and V. I. Krylov, Approximate Methods of Higher Analysis, P. Noordhoff Ltd., The Netherlands, 1958.

M1 R. S. Martin and J. H. Wilkinson, "Solution of symmetric and unsymmetric band equations and calculation of eigenvectors of band matrices," Numer. Math., 9 (1967), pp. 279-301.

M2 R. S. Martin and J. H. Wilkinson, "Reduction of the symmetric eigenproblem $Ax = \lambda Bx$ and related problems to standard form," Numer. Math., 11 (1968), pp. 99-110.

M3 Cleve B. Moler, "Finite difference methods for the eigenvalues of Laplace's operator," Computer Science Dept. Technical Report CS22, Stanford University, Stanford, California, 1965.

M4 Cleve B. Moler, "Accurate bounds for the eigenvalues of the Laplacian and applications to rhombical domains," Computer Science Dept. Technical Report CS121, Stanford University, Stanford, California, 1969.

M5 Cleve B. Moler, "Matrix computations with Fortran and paging," Computer Science Dept. Technical Report CS196, Stanford University, Stanford, California, 1970.

M6 Stanley M. Morris and William E. Schiesser, "SALEM - A programming system for the simulation of systems described by partial differential equations," Proc. AFIPS Fall Joint Comput. Conf., The Thompson Book Co., National Press Bldg., Washington, D. C., 1968, pp. 353-357.

N1 B. Nivilet, L. Schmidt, G. Terrine and J. Cea, "Techniques numériques de l'approximation variationelle des problèmes elliptiques," Publication MMC/10.12.5/AI, Centre National de la Recherche Scientifique, Institut Blaise Pascal, Paris, 1966.

N2 S. Nordbeck and B. Rystedt, "Computer cartography point-in-polygon programs," BIT, 7 (1967), pp. 39-64.

P1 G. Peters and J. H. Wilkinson, "Eigenvalues of $Ax = \lambda Bx$ with band symmetric A and B," Comput. J., 12 (1969), pp. 398-404.

R1 J. K. Reid and A. B. Turner, "Fortran subroutines for the solution of Laplace's equation over a general region in two dimensions," Report T.P. 422, Theoretical Physics Division, Atomic Energy Research Establishment, Harwell, England, 1970.

R2 J. K. Reid and J. E. Walsh, "An elliptic eigenvalue problem for a reentrant corner," SIAM J. Appl. Math., 13 (1965), pp. 837-850.

R3 Donald J. Rose, "Symmetric elimination on sparse positive definite systems and the potential flow problem," Ph. D. Thesis, Div. of Engrg. and Appl. Phys., Harvard University, Cambridge, Massachusetts, 1970.

R4 Richard Rosen, "Matrix bandwidth minimization," Proc. 23rd National Conf., Assoc. Comput. Mach., ACM publication P-68, Brandon/Systems Press Inc., Princeton, New Jersey, 1968.

S1 Martin H. Schultz, "Elliptic spline functions and the Rayleigh-Ritz-Galerkin method," Math. Comp., 24 (1970), pp. 65-80.

S2 Cesar Simoes Salim and Helene K. Salim, "The sparse system," Computer Science Dept. Report, Rio Datacenter, Rio de Janeiro, Brazil, 1970.

S3 Jitka Segethova, "Elimination procedures for sparse symmetric linear algebraic systems of a special structure," Technical Report 70-121, Computer Science Center, University of Maryland, College Park, Maryland, 1970.

S4 James H. Stadter, "Bounds to eigenvalues of rhombical membranes," SIAM J. Appl. Math., 14 (1966), pp. 324-341.

S5 Gilbert Strang, "The finite element method and approximation theory," Symposium on the Numerical Solution of Partial Differential Equations, University of Maryland, College Park, Maryland, 1970.

S6 G. Strang and G. Fix, "A Fourier analysis of the finite element method," to appear.

S7 Blair K. Swartz, "Explicit $O(h^2)$ bounds on the eigenvalues of the half-L," Math. Comp., 22 (1968), pp. 40-59.

T1 R. P. Tewarson, "Computations with sparse matrices," SIAM Rev., 12 (1970), pp. 527-543.

T2 Coyt C. Tillman, "EPS: An interactive system for solving elliptic boundary value problems with facilities for data manipulation and general purpose computation," User's Guide, Project MAC Report MAC-TR-62, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1969.

T3 William F. Tinney, "Comments on using sparsity techniques for power system problems," pp. 25-34 of W3.

T4 James L. Tocher and Carlos A. Felippa, "Computer graphics applied to production structural analysis," Internat. Union of Theor. and Appl. Mech. Conf., Liege, Belgium, 1970.

W1 A. Weinstein, "Some numerical results in intermediate problems for eigenvalues," pp. 167-192 of B10.

W2 J. H. Wilkinson, The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, England, 1965.

W3 R. A. Willoughby, editor, IBM Sparse Matrix Proceedings, IBM Report RA1 (No. 11707), Thomas J. Watson Research Center, Yorktown Heights, New York, 1969.

W4 R. A. Willoughby, "Sparse matrix algorithms and their relation to problem classes and computer architecture," IBM Report RC 2833 (No. 13318), Thomas J. Watson Research Center, Yorktown Heights, New York, 1970.

W5 A. M. Winslow, "Equi-potential zoning for two dimensional meshes," Report No. UCRL-7312, Lawrence Radiation Laboratory, University of California, Livermore, Calif., 1964.

Z1 Alexander Ženíšek, "Interpolation polynomials on the triangle," Numer. Math., 15 (1970), pp. 283-296.

Z2 Alexander Ženíšek and Milos Zlámal, "Convergence of a finite element procedure for solving boundary value problems of the fourth order," Internat. J. Numer. Meth. Engrg., 2 (1970), pp. 307-310.

Z3 O. C. Zienkiewicz, The Finite Element Method in Structural and Continuum Mechanics, McGraw-Hill Book Company, London, 1967.

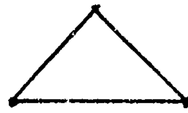
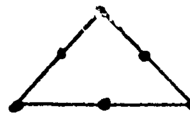
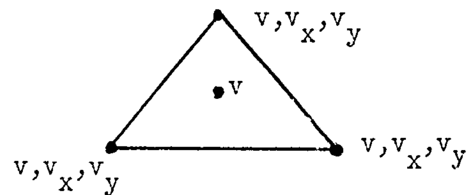
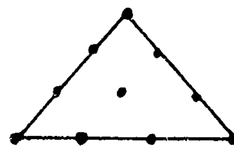
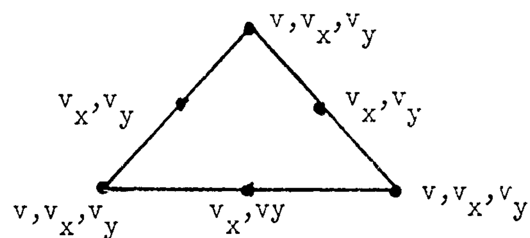
Z4 M. Zlámal, "On the finite element method," Numer. Math., 12 (1968), pp. 394-409.

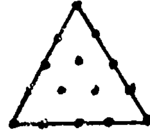
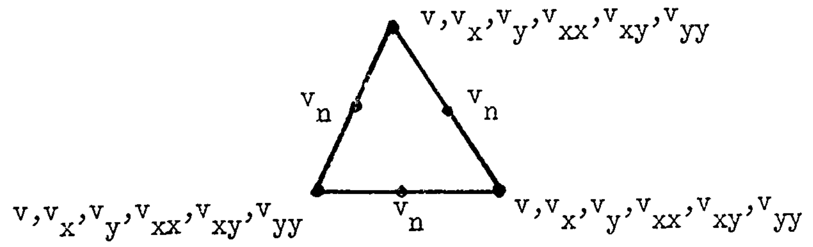
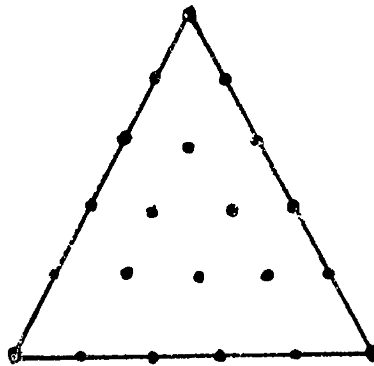
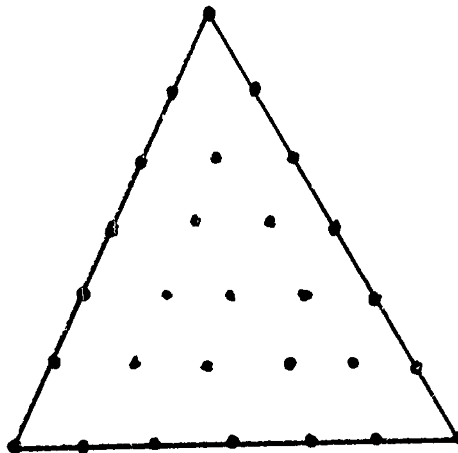
Z5 M. Zlámal, "On some finite element procedures for solving second order boundary value problems," Numer. Math., 14 (1969), pp. 42-48.

Z6 M. Zlámal, "A finite element procedure for the second order of accuracy," Numer. Math., 14 (1970), pp. 394-402.

Appendix A: Some Representative Triangular Elements

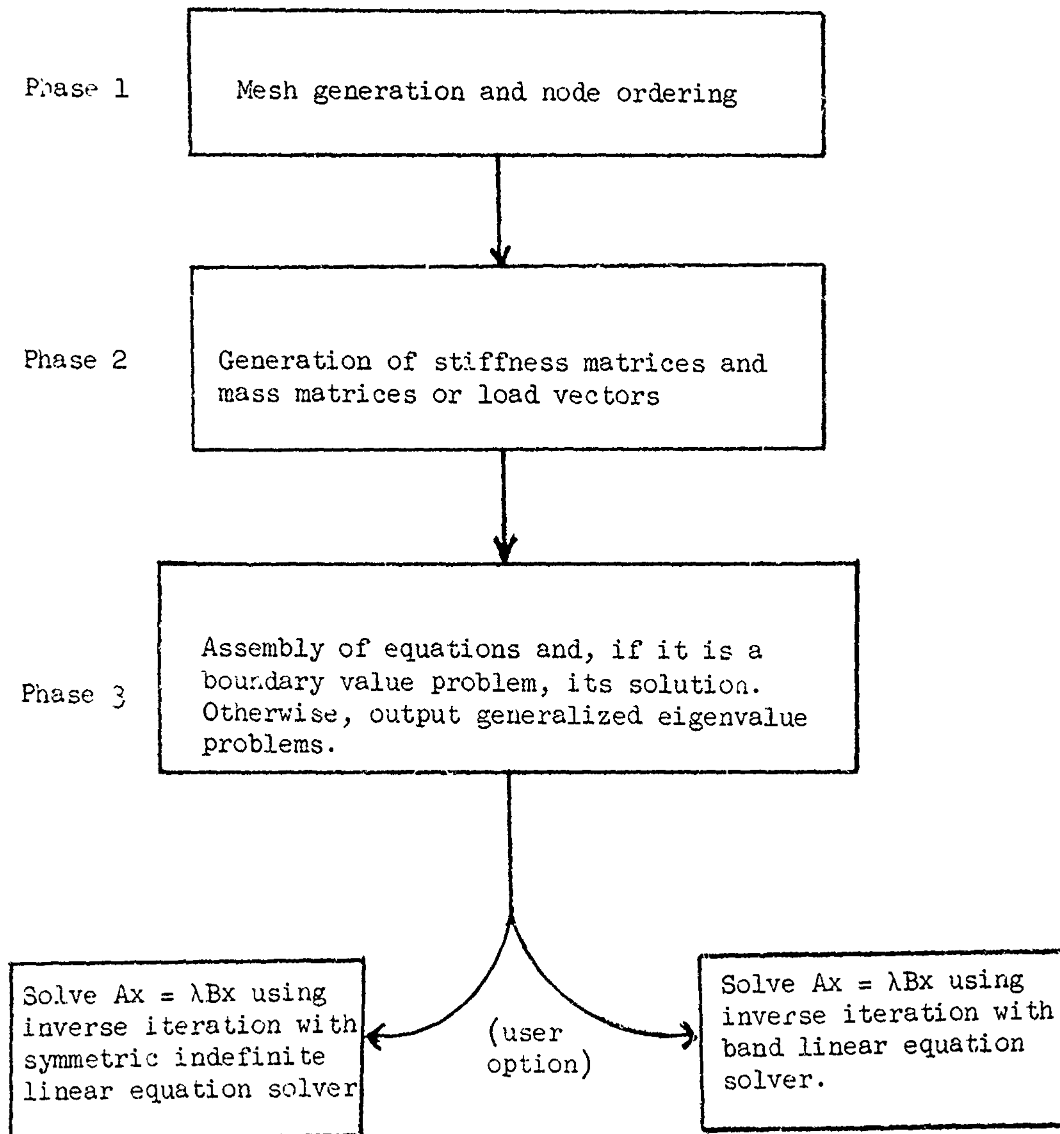
The labels on the stencils below indicate the parameters associated with each node. When no label appears, the function value v is to be assumed. The two-part hyphenated name refers respectively to the degree of the polynomial and the number of nodes associated with the element.

Element Name	Stencil
1-3	
2-6	
3-4	
3-10	
4-6	

Element Name	Stencil
4-15	
5-6	
5-21	
6-28	

Appendix B: O/S 360 Fortran Code for Finite Element Methods

The codes in this appendix are all written in O/S 360 Fortran. There are five separate programs whose relation is depicted in the following chart:



The transmission of data from one program to the next is done via Fortran units 1, 2 (and 3 if an eigenvalue problem must be solved). All data sets read and written are sequential, so the program would work without alteration whether the storage devices are disks, drums, or tape units. Only changes in the job control language would be necessary.

The program is set up to find the stationary values of $I_1[v] + I_2[v]$ and $I_1[v]/J[v]$, where

$$I_1[v] = \iint_R c_1 v_x^2 + c_2 v_y^2 + c_3 v^2 \, dxdy,$$

$$I_2[v] = \iint_R c_4 v \, dx \, dy,$$

and

$$J[v] = \iint_R v^2 dxdy$$

Here c_1, c_2, and c_3 are constants, and c_4 is a function supplied by the user in the subroutine FUNC. For further details and sample input see the comments in the code of PHASE 2 and in Appendix C.

With minor changes in the mainline of PHASE 2, other terms can be included in I_1 and I_2, and with somewhat more substantial changes variable coefficient quadratic terms could be handled. Note that phases 1 and 3 would not need to be altered.

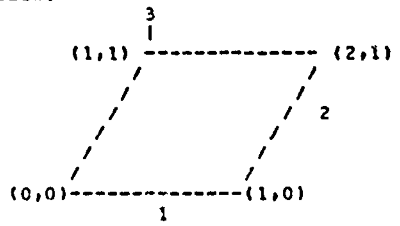
Piecewise polynomials of degree d $(1 \leq d \leq 9)$ utilizing $\binom{d+2}{2}$ value and first-derivative parameters can be selected by the user and are automatically generated by the program.

The choice of method for solving the generalized eigenvalue problem depends on the relative size of the number of equations and the bandwidth, as discussed in section 4.3. Both programs assume that the initial shift (SHIFT) supplied by the user is a good one; the decomposition of A-SHIFT*B is done only once at the beginning of the iteration.

Phase 1

Generation of the Mesh and Ordering of the Nodes

```
C*************************************************************************|C
C                                                                         |C      NCR - NUMBER OF ISOLATED NODE POINTS WHICH MAY HAVE
C       FINITE ELEMENT SOLUTION OF PDE-S                                  |C           BOUNDARY CONDITIONS PRESCRIBED FOR THEM.
C                                                                         |C
C       PHASE 1: TRIANGULATION OF THE DOMAIN.                             |C      ICRNR(I,*)- BOUNDARY REFERENCE INFORMATION FOR THE I-TH
C                                                                         |C           ISOLATED GRID POINT.
C       PROGRAMMED BY ALAN GEORGE                                         |C         (I,1)- POINTER TO THE COORDINATES OF THE I-TH POINT.
C       COMPUTER SCIENCE DEPARTMENT,                                      |C         (I,2)- BOUNDARY REFERENCE NUMBER.
C       STANFORD UNIVERSITY                                               |C
C       JUNE 2, 1970                                                      |C      TNEW, TOLD, TSTRT - VARIABLES USED BY  THE TIMING
C-------------------------------------------------------------------------|C                       ROUTINE 'TIMER'.
C                                                                         |C
C       VARIABLES AND THEIR FUNCTION .........                            |C      MODE - A FLAG WHICH IS SET TO ONE WHENEVER THE CANONICAL
C                                                                         |C           TRIANGLE IS BEING PROCESSED.  IT IS USED BY THE
C       STAR INDICATES INPUT VARIABLE.                                    |C           ROUTINES 'FIND' AND 'INSERT'.
C                                                                         |C
C *     NDIVS - NUMBER OF SUBDIVISIONS TO BE DONE TO THE                  |C
C              INPUT TRIANGLES.  0 < NDIVS < 21. (DEFAULT: 1)             |C *    IBUG = 1 => DEBUGGING INFO PRINTED.(DEFAULT: 0)
C                                                                         |C
C *     NPS - NODES-PER-SIDE OF EACH TRIANGLE, NOT INCLUDING THE          |C *    LAST = 0 => ANOTHER TRIANGULATION DATA SET FOLLOWS,
C              ENDPOINTS. (DEFAULT: 0)                                    |C                 OTHERWISE = 1.(DEFAULT: 0)
C                                                                         |C
C *     NCEN -  NUMBER-OF-CENTROIDS(NUMBER OF NODES IN INTERIOR           |C-------------------------------------------------------------
C              OF EACH TRIANGLE).(DEFAULT: 0)                             |C//////    SAMPLE INPUT DATA SET ////////////////////////////////
C              NCEN MUST BE K(K+1)/2, K AN INTEGER.                       |C      & SIGN MUST BE IN COLUMN 2.
C                                                                         |C
C       INDEX - THE POSITIONS M * K + I, I = 1, 2, ..., M CONTAIN         |C      &PARMS NDIVS=4, LAST=1, IBUG=1 &END
C              THE INDICES OF THE M NODES OF THE K-TH TRIANGLE.           |C      &POINTS PT(1) = (0,0), PT(2) = (1,0), PT(3) = (2,1),
C                                                                         |C              PT(4) = (1,1)&END
C       Z -  COORDINATES OF THE NODES(STORED AS COMPLEX NUMBERS)          |C      &TR NODES=1,2,4, BND(1)=1, CORNER(3)=3, &END
C                                                                         |C      &TR NODES=2,3,4, BND(1)=2, ENDTR=T &END
C       NTRS - NUMBER OF TRIANGLES(INCLUDING CANONICAL ONE)               |C
C                                                                         |C
C       M - NUMBER OF NODES PER TRIANGLE                                  |C//////    END SAMPLE INPUT /////////////////////////////////////
C                                                                         |C
C       NPTS - NUMBER OF NODE POINTS, INCLUDING THOSE WHICH HAVE          |C         THE ABOVE DATA SET PRESCRIBES A RHOMBICAL  DOMAIN WITH
C              BEEN GENERATED FOR USE BY PHASE 2 FOR QUADRATIC            |C         VERTICES AT THE POINTS PRESCRIBED BY THE ARRAY PT.  THE
C              INTERPOLATION OF CURVED BOUNDARIES.                        |C         TWO INPUT TRIANGLES ARE TO BE SUBDIVIDED BY A FACTOR OF
C                                                                         |C         FOUR.  FOLLOWING THE CONVENTION THAT SIDE I HAS
C       MAXPTS - DECLARED DIMENSION OF THE ARRAY Z.                       |C         ENDPOINTS PT(NODES(I))AND PT(NODES((I+1)MOD 3)), THE
C                                                                         |C         SIDES OF GENERATED TRIANGLES LYING ON SIDE I OF THE
C       MXS - DECLARED COLUMN DIMENSION OF THE ARRAY ISIDE.               |C         INPUT TRIANGLE WILL HAVE ASSOCIATED WITH THEM THE
C                                                                         |C         REFERENCE NUMBER BND(I). THE POINT (1,1) WILL
C       MXC - DECLARED COLUMN DIMENSION OF THE ARRAY ICRNR.               |C         HAVE THE BOUNDARY REFERENCE NUMBER 3 ASSOCIATED
C              (SET IN THE PROGRAM)                                       |C         WITH IT.
C                                                                         |C         THE DOMAIN AND THE ASSOCIATED REFERENCE NUMBERS ARE
C       MXMEM2 - DECLARED DIMENSION OF THE ARRAY MEM2.                    |C         DEPICTED BELOW.
C                                                                         |C
C       NSIDE - TOTAL NUMBER OF TRIANGLE SIDES INVOLVED IN                |C
C              BOUNDARY CONDITIONS.                                       |C
C                                                                         |C
C       ISIDE(*,K)- CODED INFORMATION ABOUT THE K-TH TRIANGLE             |C
C                   SIDE WHICH IS ON A BOUNDARY.                          |C
C           (1,K)- THE ASSOCIATED TRIANGLE                                |C
C           (2,K)- BOUNDARY REFERENCE NUMBER, USED BY PHASE 3             |C
C                  TO ASSIGN BOUNDARY VALUES.                             |C
C           (3,K) = 0 IF SIDE IS STRAIGHT, OTHERWISE POINTS TO            |C
C                   MIDPOINT OF ARC PASSING THROUGH THE                   |C
C                   CORNER NODES.                                         |C
C           (4,K)- SIDE NUMBER (1-ST, 2-ND OR 3-RD)                       |C
```



```
C-----------------------------------------------------------------
C       MAINLINE PROGRAM FOR TRIANGULATION PROGRAM.
C-----------------------------------------------------------------
        IMPLICIT INTEGER*2(I-N)
        COMMON/SIXFO/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *          KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
        COMMON/ DP/NSIDE, NCR, MXS, MXC, ISIDE(4,200), ICRNR(2,30)
        COMMON /TIME/ TSTRT, TNEW, TOLD
        COMPLEX*16 NODE(3), Z(3000), PT(200)
C-----------------------------------------------------------------
        INTEGER*2 BND(3), CRV, NODES(3), CORNER(3)
        INTEGER*2 MEM2(80000)
        REAL*8 S1, S2
        LOGICAL ENDTR
C-----------------------------------------------------------------
        NAMELIST /TR/NODES, BND, CRV, S1, S2, ENDTR, NODE, CORNER
     *          /PARMS/NDIVS, IBUG, NPS, NCEN, M, LAST
     *          /POINTS/ PT
C-----------------------------------------------------------------
C
        TSTRT = TIMER1(0)
        MAXPTS = 3000
        MXS = 200
        MXC = 30
        MXMEM2 = 80000
        REWIND 1
   10   CONTINUE
C
C       DATA-SET-INPUT LOOP ...
C
C       INITIALIZE PARAMETERS...
C
        TSTRT = TIMER1(1)
        TNEW = TSTRT
        LAST = 0
        MODE = 0
        IBUG = 0
        NPS = 0
        NCEN = 0
        NDIVS = 1
        NPTS = 0
        NTRS = 0
        NSIDE = 0
        NCR = 0
        KLAST = 0
C
C       READ INPUT PARAMETERS
C
        CALL PAGE('   INPUT ...         ')
        READ(5, PARMS, END=900)
C
        NDV1 = NDIVS + 1
        M = 3 + NCEN + 3 * NPS
        KCEN = 0
        IF (NCEN .EQ. 0) GO TO 5
        DO 4 I = 1, 6
          IF (NCEN .NE. I*(I+1)/2) GO TO 4
          KCEN = I
          GO TO 5
    4   CONTINUE
        WRITE(6, 99) NCEN
   99   FORMAT(' INVALID NUMBER OF INTERIOR POINTS REQUESTED(',
     *          I4, ')    ERROR EXIT TAKEN'/)
        GO TO 900
C
    5   DO 3 I = 1, M
          MEM2(I) = 0
    3   CONTINUE
C
        WRITE(6, PARMS)
        READ(5, POINTS)
        WRITE(6, 100)
  100   FORMAT(///,T5, ' INDICES', T43, ' NODE COORDINATES',
     *          T72, '          BND          CORNER')
C
C       TRIANGLE-DESCRIPTION-INPUT-AND-PROCESS LOOP ....
C
    1   DO 2 I = 1, 3
          BND(I) = 0
          CORNER(I) = 0
    2   CONTINUE
C
        CRV = 0
        S1 = 0.0D0
        S2 = 0.0D0
        ENDTR = .FALSE.
C
        READ(5, TR)
        NODE(1) = PT(NODES(1))
        NODE(2) = PT(NODES(2))
        NODE(3) = PT(NODES(3))
        WRITE(6, 101) NODES, NODE, BND, CORNER
  101   FORMAT(1X, 3I4, 2F10.6, ',', 2F10.6, ',', 2F10.6, ',',
     *          3I4, ',', 3I4)
C
        CALL TRIANG(NODE, MEM2, BND, CORNER, CRV, S1, S2, Z, &900)
C
C       END TRIANGLE-DESCRIPTION-INPUT LOOP.
C
        IF (.NOT. ENDTR) GO TO 1
C
C       GENERATE NODES ETC. FOR 'CANONICAL' TRIANGLE.
C
        NODE(1) = (0.0, 0.0)
        NODE(2) = (1.0, 0.0)
        NODE(3) = (0.0, 1.0)
        BND(1) = 0
        BND(2) = 0
        BND(3) = 0
        CORNER(1) = 0
        CORNER(2) = 0
        CORNER(3) = 0
        WRITE(6, 101) NODES, NODE, BND, CORNER
        CRV = 0
        NDIVS = 1
        NDV1 = 2
        MODE = 1
        CALL TRIANG(NODE, MEM2, BND, CORNER, CRV, S1, S2, Z, &900)
C
        CALL TIMER('CORNERS GENERATED   ')
```

```
C
      CALL FILL(MEM2, Z)
      CALL TIMER('FILL-IN DONE          ')
C
C     COMPUTE CONSTANTS FOR STORAGE ALLOCATION.
C
      KIP = M *(NTRS+1)+ 1
      KID = KIP + NPTS
      KNMBRD = KID + NPTS
      KMSK = KNMBRD + NPTS
      KMAT = KMSK + NPTS
      KLAST = KMAT + NPTS * 8
      WRITE(5, 103)KLAST
  103 FORMAT(' STORAGE IN MEM2 USED:', I8/)
      IF (KLAST .GT. MXMEM2) GO TO 900
C
C     GENERATE AN ORDERING OF THE NODES WHICH WILL YIELD
C     A SMALL BANDWIDTH AND SMALL STORAGE AND
C     COMPUTATIONAL REQUIREMENTS.
C
      CALL CUTHIL(NPTS*1, MEM2, MEM2(KIP), MEM2(KID),
     *            MEM2(KNMBRD), MEM2(KMSK), MEM2(KMAT))
C
   20 CALL OUTPUT(MEM2, MEM2(KIP), Z)
C
C     END DATA-SET-INPUT LOOP .
C
C
      IF (LAST .EQ. 0) GO TO 10
C
  900 ENDFILE 1
      REWIND 1
      STOP
      END
C*****************************************************************
      SUBROUTINE BNDF(K, S, Z)
C-----------------------------------------------------------------
C     PARAMETRIC EQUATIONS DEFINING CURVED BOUNDARIES.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMPLEX*16 Z, Z1
      REAL*8 S, R(2), DSQRT
      EQUIVALENCE(Z1, R(1))
C-----------------------------------------------------------------
C
      GO TO(1, 2, 3), K
C
    1 R(1) = S
      R(2) = S**2
      GO TO 20
C
    2 R(1) = S
      R(2) = S**3
      GO TO 20
C
    3 R(1) = S
      R(2) = DSQRT(1.0D0 - S**2)
C
   20 Z = Z1
      RETURN
```

```
      END
C*****************************************************************
      SUBROUTINE TRIANG(NODE, INDEX, BND, CORNER, CRV, S
     *                  S1, S2, Z, *)
C-----------------------------------------------------------------
C     THIS ROUTINE DOES THE SUBDIVISION OF THE INPUT TRIANGLE
C     INTO THE NUMBER OF SUB-TRIANGLES DESIGNATED BY THE
C     VARIABLE NDIVS.
C-----------------------------------------------------------------
C
C     THE COORDINATES OF THE SUB-DIVIDING NODES OF EACH SIDE
C     OF THE TRIANGLE ARE STORED IN THE ARRAYS A, B, AND C
C     AS SHOWN BELOW.  ASSUME NDIVS=4.
C     Z1, Z2, AND Z3 ARE THE CORNER NODES(ANTI-CLOCKWISE ORDER).
C
C           C(5)
C
C           C(4)  B(4)
C
C  SIDE#3   C(3)        B(3)       SIDE#2(MAY BE CURVED).
C
C           C(2)               B(2)
C
C           A(1)  A(2)  A(3)  A(4)  A(5)
C
C                     SIDE#1
C
C     A(1)=C(1)=Z1.            D12=A(K+1)-A(K)
C     A(5)=B(1)=Z2             D13=C(K+1)-C(K)
C     C(5)=B(5)=Z3             D23=B(K+1)-B(K)(IF CRV=0).
C
C     THE INTERIOR NODES WHICH ARE TO BE GENERATED ARE
C     OBTAINED BY SIMPLE LINEAR INTERPOLATION FROM THE
C     ELEMENTS OF A, B, AND C.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMMON/FIXED/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *       KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
      COMMON/DR/NSIDE, NCR, MXS, MXC, ISIDE(4,200), ICRNR(2,30)
      COMPLEX*16 NODE(3), A(21), B(21), C(21),  Z(1),
     *      D12, D13, D23, Z1, Z2, Z3, T, TB, V, VB, Q, X
      INTEGER*2 CRV, I, J, K, P(21,21), D(21), IBUG,
     *      BND(3), CORNER(3), INDEX(1), IONE/1/, IZERO/0/
      REAL*8 S, S1, S2, SL, SS, RT(2), TEMP, DT, CDABS,
     *      DABS, RV(2), DS
      LOGICAL THERE
      EQUIVALENCE(RT(1), T),(RV(1), V)
C-----------------------------------------------------------------
C     BEGIN BODY OF PROGRAM
C
      Z1 = NODE(1)
      Z2 = NODE(2)
      Z3 = NODE(3)
      D12 = (Z2 - Z1)/ NDIVS
      D13 = (Z3 - Z1)/ NDIVS
      D23 = (Z3 - Z2)/ NDIVS
C
C     GENERATE VALUES FOR THE ARRAYS A, B, AND C...
C     THE ELEMENTS OF B WILL NOT BE THE CORRECT ONES IF SIDE 2
C     IS CURVED.  THEY WILL BE PROJECTED ONTO THE CURVE LATER.
```

```
C
      DO 1 I = 1, NOV1
         IM1 = I - 1
         D(I) = 0
         B(I) = Z2 + IM1 * D23
         C(I) = Z1 + IM1 * C13
         THERE = .FALSE.
         IF (I.EQ. 1 .OR. I.EQ.NOV1 .OR. BND(3).EQ.0)
     *      CALL FIND(P(I,1), Z, C(I), THERE)
C
         IF (.NOT. THERE) CALL INSERT(P(I,1), Z, C(I), &11)
C
         A(I) = Z1 + IM1 * C12
         IF (I .EQ. 1) GO TO 1
         THERE = .FALSE.
         IF (I.EQ.1 .OR. I.EQ.NOV1 .OR. BND(1).EQ.0)
     *      CALL FIND(P(1,I), Z, A(I), THERE)
C
         IF (.NOT. THERE) CALL INSERT(P(1,I), Z, A(I), &11)
C
    1 CONTINUE
C
      IF (CRV .EQ. 0) GO TO 10
C
C     WE HAVE A CURVED BOUNDARY, PROJECT ELEMENTS OF B ONTO IT.
C
      CALL NODE(CRV, S1, T)
      CALL NODE(CRV, S2, V)
C
      IF ((CDABS(T-B(1))+ CDABS(V-B(NOV1))).LT. 1D-10) GO TO 12
C
C     ERROR - THE GIVEN CURVE WHICH IS SUPPOSED TO FORM SIDE 2
C     OF THE GIVEN CURVILINEAR TRIANGLE DOES NOT PASS THRU THE
C     PRESCRIBED VERTICES OF THE TRIANGLE. TAKE ERROR EXIT.
C
      WRITE(6, 100)CRV, NODE, S1, S2, T, V
  100 FORMAT(' ERROR: CURVE ', I3, ' DOES NOT PASS THRU NODES',
     *       /, ' NODES:', 6F12.6, /, ' S1, S2:', 2F12.6, /,
     *       ' CURVE(S1), CURVE(S2):', 4F12.6, //,
     *       ' JOB ABANDONED'/)
C
   11 RETURN 1
C
   12 T = D23
      RT(2) = - RT(2)
      V = T
      K = 2
      SL = S1
      DS = (S2 - S1)/(20 * NDIVS)
C
C     B CONTAINS THE NODE POINTS ALONG THE STRAIGHT LINE JOINING
C     Z2 AND Z3. OUR AIM IS TO OTHOGONALLY PROJECT THEM ONTO
C     THE CONTOUR PRESCRIBED BY THE VARIABLE CRV.
C
   15 IF (K .GT. NOV1) GO TO 10
C
C     REPLACE B(K) BY ITS OTHOGONAL PROJECTION ONTO THE CONTOUR.
C
      IF (K .LT. NOV1)
     *      CALL PROJECT(V, B(K), B(K), CRV, SS, SL, DS, &11)
C
C     B(K-1) AND B(K) ARE NOW CONSECUTIVE NODES ON THE ARC.
C     THEY FORM VERTICES 2 AND 3 OF A TRIANGLE.  NOW WE
C     FIND (APPROXIMATELY) THE MIDPOINT OF THE ARC THEY
C     SUBTEND SO THAT WE CAN INTERPOLATE THE BOUNDARY LATER.
C     THE INDEX OF THE POINT GENERATED IS STORED IN D(K),
C     AND IS EVENTUALLY STORED IN THE TABLE ISIDE.
C
      CALL PROJECT(V,(B(K)+B(K-1))/2D0, T, CRV, S, SL, DS, &11)
      SL = SS
      CALL INSERT(D(K), Z, T, &11)
      K = K + 1
      GO TO 15
C
C     END OF ARC PROCESSING ------------------------------------
C
   10 CONTINUE
C
C     PROCESS ISOLATED BOUNDARY POINT REFERENCE DATA.
C
      DO 5 I = 1, 3
        IF (CORNER(I).EQ. 0) GO TO 5
        NCR = NCR + 1
        IF (NCR .LE. MXC) GO TO 9
        WRITE(6, 101)
  101   FORMAT(' STORAGE FOR ICRNR EXCEEDED, JOB ABORTED'/)
        GO TO 11
C
    9   ICRNR(1, NCR) = CORNER(I)
        GO TO(6, 7, 8), I
    6   ICRNR(2, NCR) = P(1,1)
        GO TO 5
    7   ICRNR(2, NCR) = P(1, NOV1)
        GO TO 5
    8   ICRNR(2, NCR) = P(NOV1, 1)
    5 CONTINUE
C
C     GENERATE NODES AND TRIANGLES.
C
      DO 2 K = 2, NOV1
         L1 = NOV1 - K
         THERE = .FALSE.
         IF (L1 .EQ. 0) GO TO 20
         IF (BND(2).EQ. 0 .OR. K .EQ. NOV1)
     *     CALL FIND(P(K, L1+1), Z, B(K), THERE)
         IF (.NOT. THERE) CALL INSERT(P(K,L1+1), Z, B(K), &11)
         IF (L1 .LT. 2) GO TO 20
         V = B(K)- C(K)
         T = V
         RT(2) = -RT(2)
         VB = T
C
C        GENERATE A 'LINE' OF NODES 'PARALLEL' TO SIDE 1 AND
C        STORE THEIR RESPECTIVE INDICES (IN Z) IN THE ARRAY P.
C
         DO 3 J = 2, L1
            X = B(NOV1-J+1)- A(J)
            Q = C(K)- A(J)
            T = VB * X
            DT = RT(2)
```

```
            T = VR * Q
            TEMP = 1.0D0
            IF (DABS(DT).GT. 1.0D-10)TEMP = RT(2)/ DT
            T = A(J)+ TEMP * X
            CALL INSERT(P(K, J), Z, T, &11)
  3      CONTINUE
C
C        GENERATE ANOTHER ROW OF TRIANGLES  'PARALLEL' TO
C        SIDE 1 USING THE NODES JUST GENERATED.
C
   20    L1 = L1 + 1
         DO 4 J = 1, L1
            CALL TRGEN(INDEX, K, J, L1, BNC, P(K-1,J),
     *            P(K-1,J+1), P(K,J), C(K), ICNE, &11)
C
            IF (J .NE. L1)
     *      CALL TRGEN(INDEX, K, J, L1, BNC, P(K,J),
     *            P(K-1,J+1), P(K, J+1), IZERO, IZERO, &11)
    4    CONTINUE
    2 CONTINUE
C
      RETURN
      END
C*********************************************************************
      SUBROUTINE PRJECT(V, B, T, CRV, S, S1, DS, *)
C---------------------------------------------------------------------
C     THIS ROUTINE STEPS ALONG CURVE 'CRV' UNTIL THAT
C     POINT Q SATISFIES (Q-B).   > 0 .
C---------------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMPLEX*16 V, B, T, Q, W
      REAL*8 S, S1, DS, RW(2)
      INTEGER*2 CRV
      EQUIVALENCE(RW(1), W)
C---------------------------------------------------------------------
      L = 0
    1 L = L + 1
      IF (L .GT. 50) GO TO 2
      S = S1 + L * DS
C
      CALL BNDF(CRV, S, Q)
C
      W = Q - B
      W = W * V
      IF (RW(1).LT. 0) GO TO 1
      T = Q
      RETURN
C
C     ERROR ROUTINE
C
    2 WRITE(6, 100)CRV, S, S1, DS, V, B, T
  100 FORMAT(' FAILURE IN PRJECT: CRV,S,S1,DS:', I4, 3F12.6, /,
     *   ' V, B, T:', 6F12.6, //, ' JOB ABANDONED'/)
C
      RETURN 1
      END
C*********************************************************************
      SUBROUTINE INSERT(K, Z, ZZ, *)
C---------------------------------------------------------------------
C     INSERT A NEW NODE POINT INTO Z.
C---------------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMMON/FIXED/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *        KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
      COMPLEX*16 Z(1), ZZ
C---------------------------------------------------------------------
      NPTS = NPTS + 1
      IF (NPTS .GT. MAXPTS) GO TO 1
C
      K = NPTS
      Z(K) = ZZ
      RETURN
C
    1 WRITE(6, 100)
  100 FORMAT(' STORAGE FOR NODES EXCEEDED, ERROR EXIT TAKEN'/)
      RETURN 1
      END
C*********************************************************************
      SUBROUTINE FIND(K, Z, ZZ, THERE)
C---------------------------------------------------------------------
C     SEE IF ZZ IS IN THE LIST Z OF NODE POINTS.
C---------------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMMON/FIXED/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *        KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
      COMPLEX*16 Z(1), ZZ, T
      LOGICAL THERE
      REAL*8 W(2)
      EQUIVALENCE(W(1), T)
C---------------------------------------------------------------------
      THERE = .FALSE.
      IF (NPTS .EQ. 0 .OR. MODE .EQ. 1) RETURN
C
      DO 1 I = 1, NPTS
         K = NPTS + 1 - I
         T = Z(K)- ZZ
         IF (W(1)*W(1)+ W(2)*W(2).LT. 2.0D-16) GO TO 10
    1 CONTINUE
C
      RETURN
C
   10 THERE = .TRUE.
      RETURN
      END
C*********************************************************************
      SUBROUTINE TRGEN(INDEX, K, J, L1, BNC, K1, K2,
     *                 K3, K4, IFLG, *)
C---------------------------------------------------------------------
C     PUTS POINTERS TO THE CORNER NODES INTO THE ARRAY INDEX
C     AND PUTS THE RELEVANT INFORMATION INTO THE ARRAY ISIDE.
C---------------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMMON/FIXED/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *        KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
      COMMON/DR/NSIDE, NCR, MXS, MXC, ISIDE(4,200), ICRNR(2,30)
      INTEGER*2 INDEX(1), BNC(3)
C---------------------------------------------------------------------
C
      NTRSM1 = NTRS
      NTRS = NTRS + 1
```

```
      L = M * NTRS
      INDEX(L+1) = K1
      INDEX(L+2) = K2
      INDEX(L+3) = K3
C
      IF (IFLG .EQ. 0) RETURN
C
      DO 1 I = 1, 3
         IF (BND(I).EQ. 0) GO TO 1
         IF (I .EQ. 1 .AND. K .NE. 2) GO TO 1
         IF (I .EQ. 2 .AND. J .NE. L1) GO TO 1
         IF (I .EQ. 3 .AND. J .NE. 1) GO TO 1
         NSIDE = NSIDE + 1
         IF (NSIDE .GT. MXS) GO TO 10
         ISIDE(1, NSIDE) = NTRS
         ISIDE(2, NSIDE) = BND(I)
         ISIDE(3, NSIDE) = 0
         IF (K4 .GT. 0 .AND. I .EQ. 2) ISIDE(3, NSIDE) = K4
         ISIDE(4, NSIDE) = I
    1 CONTINUE
C
      RETURN
C
C     ERROR ROUTINES
C
   10 WRITE(6, 100)
  100 FORMAT(' STORAGE FOR BOUNDRY DATA EXCEEDED, JOB ABORTED'/)
      RETURN 1
C
      END
C*******************************************************************
      SUBROUTINE OUTPUT(INDEX, IP, Z)
C-------------------------------------------------------------------
C     OUTPUT ROUTINE.
C-------------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMMON/FIXED/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *        KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
      COMMON/DB/NSIDE, NCR, MXS, MXC, ISIDE(4,200), ICRNR(2,30)
      COMPLEX*16 Z(1)
      INTEGER*2 INDEX(1), IP(1)
C-------------------------------------------------------------------
      NAMELIST / PARMS1/ NPTS, NTRS, NSIDE
C-------------------------------------------------------------------
C
C     LIST GENERATED DATA ON THE PRINTER ...
C
      WRITE(6, PARMS1)
      IF (IBUG .EQ. 0) GO TO 50
C
      CALL PAGE(' OUTPUT INFORMATION ')
C
      WRITE(6, 100)
  100 FORMAT(/, T40, ' POINTS AND COORDINATES.....'/)
      WRITE(6, 101)((I, Z(I)), I = 1, NPTS)
  101 FORMAT(1X, I4, 2F12.8, I4, 2F12.8, I4, 2F12.8, I4, 2F12.8)
C
      WRITE(6, 99)
   99 FORMAT('1 TRIANGLE       INDICES '/)
      DO 2 I = 1, NTRS
```

```
         K1 = I * M - 1
         K2 = K1 + M - 1
         WRITE(6, 102)I,(INDEX(K), K = K1, K2)
  102    FORMAT(1X, 2015)
    2 CONTINUE
C
      WRITE(6, 98)
   98 FORMAT('1BOUNDARY INFORMATION(ISIDE)'/)
      WRITE(6, 105)(I, I = 1, NSIDE)
  105 FORMAT(' SIDE#', 60I2)
      DO 3 I = 1, 4
         WRITE(6, 106)(ISIDE(I,K), K = 1, NSIDE)
  106    FORMAT(' #####', 60I2)
    3 CONTINUE
C
      WRITE(6, 107)
  107 FORMAT(//, ' CORNER INFO:',//,' I     REFERENCE, INDEX')
      DO 4 I = 1, NCR
         WRITE(6, 102)I,(ICRNR(J, I), J = 1, 2)
    4 CONTINUE
C
      WRITE(6, 103)
  103 FORMAT(/, ' ORDER VECTOR:'/)
      WRITE(6, 102)(IP(I),  I=1, NPTS)
C
C     PRINTING DONE .
C
C     WRITE DATA ON UNIT 1 FOR INPUT TO PHASE 2 AND 3.
C
   50 CALL TIMER('BEGIN OUTPUT ....   ')
C
      LINDEX = M * NTRS + M
      WRITE(1)  NPTS, KPTS, NTRS, NSIDE, NCR,
     *          NPS, NCEN, M, LAST
      WRITE(1) (Z(I), I = 1, NPTS)
      WRITE(1) LINDEX, (INDEX(K), K = 1, LINDEX)
      WRITE(1) ((ISIDE(K,I), K = 1, 4), I = 1, NSIDE)
      WRITE(1) NCR, ((ICRNR(J,I), J = 1, 2), I = 1, NCR)
      WRITE(1) (IP(I), I = 1, NPTS)
C
      CALL TIMER('OUTPUT COMPLETE.    ')
C
      RETURN
      END
C*******************************************************************
      SUBROUTINE FILL(INDEX, Z)
C-------------------------------------------------------------------
C     THIS  ROUTINE FILLS IN THE NODES ALONG THE TRIANGLE
C     SIDES AND INTERIOR BY CALLING APPROPRIATE ROUTINES.
C-------------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMMON/FIXED/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *        KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
      INTEGER*2 INDEX(1), I3/3/, I2/2/
      COMPLEX*16 Z(1)
C-------------------------------------------------------------------
C
      MODE = 0
      DO 1 I = 1, NTRS
         IF (NTRS .EQ. I)MODE = 1
```

```
      IF (NPS .EQ. 0) GO TO 10
C
      L = I * M
      CALL NOSIDE(Z, INDEX, INDEX(L+1), INDEX(L+2),
     *          I3, I, &2)
      CALL NOSIDE(Z, INDEX, INDEX(L+2), INDEX(L+3),
     *          I3 + NPS, I, &2)
      CALL NOSIDE(Z, INDEX, INDEX(L+3), INDEX(L+1),
     *          I3 + I2*NPS, I, &2)
C
   10 IF (NCEN .GT. 0)
     *  CALL CENGEN(Z, INDEX, I3 + I3*NPS, I, &2)
C
    1 CONTINUE
      RETURN
    2 RETURN 1
      END
C**********************************************************************
      SUBROUTINE NOSIDE(Z, INDEX, K1, K2, IBASE, ITR, *)
C----------------------------------------------------------------------
C     GENERATES NODES ALONG A TRIANGLE SIDE (BETWEEN POINTS
C     Z(K1)AND Z(K2)).
C----------------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMMON/FIXED/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *      KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
      COMPLEX*16 Z(1), DZ, Z1, TZ
      INTEGER*2 INDEX(1)
      LOGICAL THERE
C----------------------------------------------------------------------
C
      Z1 = Z(K1)
      DZ = (Z(K2)- Z1)/(NPS + 1)
C
      DO 1 I = 1, NPS
        TZ = Z1 + I * DZ
        CALL FIND(INDEX(IBASE+I+M*ITR), Z, TZ, THERE)
        IF (.NOT. THERE)
     *     CALL INSERT(INDEX(IBASE+I+M*ITR), Z, TZ,&2)
    1 CONTINUE
C
      RETURN
    2 RETURN 1
      END
C**********************************************************************
      SUBROUTINE CENGEN(Z, INDEX, IBASE, ITR, *)
C----------------------------------------------------------------------
C     GENERATE NODES IN THE INTERIOR OF THE TRIANGLE.
C----------------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMMON/FIXED/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *      KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
      INTEGER*2 INDEX(1)
      COMPLEX*16 Z(1), CENT, TZ, Z1, Z2, Z3, D12, D13
C----------------------------------------------------------------------
C
C     GET THE CORNERS OF THE TRIANGLE.
C
      L = ITR * M
      Z1 = Z(INDEX(L + 1))
```

```
      Z2 = Z(INDEX(L + 2))
      Z3 = Z(INDEX(L + 3))
      D12 = (Z2 - Z1)/(KCEN + 2)
      D13 = (Z3 - Z1)/(KCEN + 2)
      L1 = 0
C
C     NOW GENERATE INTERIOR NODES...
C
      DO 1 I = 1, KCEN
        KM1 = KCEN - I + 1
        DO 2 J = 1, KM1
          L1 = L1 + 1
          CALL INSERT(INDEX(IBASE+L+L1), Z, Z1+I*D12+J*D13, &3)
    2   CONTINUE
    1 CONTINUE
C
      RETURN
C
    3 RETURN 1
      END
C**********************************************************************
      SUBROUTINE CUTHIL(IDIM, INDEX, IP, ID, NMBRD, MSK, MAT)
C----------------------------------------------------------------------
C     GENERATE A PERMUTATION VECTOR IP WHICH YIELDS AN ORDERING
C     OF THE NODES SO THAT THE LINEAR SYSTEM GENERATED LATER HAS
C     DESIRABLE PROPERTIES.
C     IP IS THE ORDER VECTOR.
C     ID IS A VECTOR INDICATING WHETHER A NODE HAS ANY
C        UNNUMBERED NEIGHBORS LEFT. IF ID(K)=1, THEN THE
C       K-TH NODE HAS SOME;  OTHERWISE ID(K) IS ZERO.
C     NMBRD IS A VECTOR INDICATING WHICH NODES ARE NUMBERED. IF
C        NMBRD(K)=1, THEN NODE K IS; OTHERWISE NMBRD(K)=0.
C     MSK IS AN INTEGER VECTOR.  MSK(K) IS THE NUMBER OF
C        TRIANGLES HAVING NODE K AS A MEMBER.
C     MAT(K,J), J = 1, ...., MSK(K) ARE THE NUMBERS OF THE
C        TRIANGLES HAVING NODE K AS A MEMBER. MSK AND MAT ARE
C        GENERATED IN GENMAT AND USED BY THE ROUTINE NABORS.
C----------------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMMON/FIXED/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *      KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
      INTEGER IDIM
      INTEGER*2 MAT(IDIM,1), MSK(1), ADJ(200), INDEX(1), IP(1),
     *          ID(1), NMBRD(1)
C----------------------------------------------------------------------
C
      CALL GENMAT(IDIM, INDEX, IP, ID, NMBRD, MSK, MAT)
C
      I1 = INDEX(M + 1)
      KPTS = 1
      LAST1 = KPTS
      NMBRD(I1) = 1
      IP(KPTS) = I1
C
C     BEGIN MAIN ORDERING LOOP ...
C
   15 I2 = 1
      LAST2 = LAST1
      LAST1 = KPTS
      DO 3 I = LAST2, LAST1
```

```
C
C        CHECK IF NODE I HAS ANY UNNUMBERED NEIGHBORS LEFT.
C
         IF (ID(I).EQ. 0) GO TO 3
         CALL NABORS(IDIM,IP(I), ADJ, NA, NMBRD, INDEX, MSK, MAT)
         IF (NA .EQ. 0) GO TO 4
C
         I2 = 0
         CALL ORDER1(IDIM, NA, ADJ, NMBRD, INDEX, MSK, MAT)
C
         DO 13 J = 1, NA
           KPTS = KPTS + 1
           IP(KPTS) = ADJ(J)
C
C          MARK NODE ADJ(J) AS NUMBERED.
C
           NMBRD(ADJ(J)) = 1
   13    CONTINUE
         GO TO 3
C
C        MARK NODE I AS HAVING NO UNNUMBERED NEIGHBORS.
C
    4    ID(I) = 0
C
    3 CONTINUE
C
      IF (I2 .EQ. 0) GO TO 15
C
C     REVERSE THE ORDERING ...
C
      K = KPTS / 2
      DO 1 I = 1, K
        L = IP(I)
        IP(I) = IP(KPTS+1-I)
        IP(KPTS+1-I) = L
    1 CONTINUE
C
      CALL TIMER('ORDERING DONE        ')
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE ORDER1(IDIM, NA, ADJ, NMBRD, INDEX, MSK, MAT)
C-----------------------------------------------------------------
C     ORDER THE NA NODES IN THE ARRAY ADJ IN  ORDER OF
C     INCREASING DEGREE.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      INTEGER IDIM
      INTEGER*2 NMBRD(1), ADJ1(200), ADJ2(200), ADJ(200),
     *          MAT(IDIM,1), MSK(1), INDEX(1)
C-----------------------------------------------------------------
C
      DO 1 I = 1, NA
        CALL NABORS(IDIM, ADJ(I), ADJ2, ADJ1(I),
     *              NMBRD, INDEX, MSK, MAT)
    1 CONTINUE
C
      IF (NA .EQ. 1) RETURN
C
      NAM1 = NA - 1
      DO 2 I = 1, NAM1
        IP1 = I + 1
        DO 3 J = IP1, NA
          IF (ADJ1(I).LE. ADJ1(J)) GO TO 3
          K = ADJ1(I)
          ADJ1(I) = ADJ1(J)
          ADJ1(J) = K
          K = ADJ(I)
          ADJ(I) = ADJ(J)
          ADJ(J) = K
    3   CONTINUE
    2 CONTINUE
      RETURN
      END
C*****************************************************************
      SUBROUTINE NABORS(IDIM, ND, ADJ, NA, NMBRD,
     *                  INDEX, MSK, MAT)
C-----------------------------------------------------------------
C     FIND THE (NA) UNNUMBERED NEIGHBORS OF THE NODE ND.
C     PLACE THEIR (Z) INDICES IN THE ARRAY ADJ.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMMON/FIXED/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *       KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
      INTEGER IDIM
      INTEGER*2 NMBRD(1), ADJ(1), INDEX(1), MAT(IDIM,1), MSK(1)
C-----------------------------------------------------------------
C
      NA = 0
      L = MSK(ND)
      DO 1 I = 1, L
        ITR = M * MAT(ND, I)
        DO 2 J = 1, M
          K = INDEX(ITR+J)
          IF (K .EQ. ND .OR. NMBRD(K).EQ. 1) GO TO 2
          IF (NA .EQ. 0) GO TO 4
          DO 3 I1 = 1, NA
            IF (K .EQ. ADJ(I1)) GO TO 2
    3     CONTINUE
    4     NA = NA + 1
          ADJ(NA) = K
    2   CONTINUE
    1 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE GENMAT(IDIM, INDEX, IP, ID, NMBRD, MSK, MAT)
C-----------------------------------------------------------------
C     THIS ROUTINE INITIALIZES ID, IP, MSK, MAT, AND NMBRD.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2(I-N)
      COMMON/FIXED/NPTS, NTRS, NDIVS, NDV1, KCEN, NTRSM1, MODE,
     *       KPTS, MAXPTS, LAST, IBUG, NPS, NCEN, M
      INTEGER IDIM
      INTEGER*2 MAT(IDIM,1), MSK(1), IP(1), ID(1),
     *          NMBRD(1), INDEX(1)
C-----------------------------------------------------------------
C
```

```
      DO 1 I = 1, NPTS
        ID(I) = 1
        IP(I) = 0
        NMBRD(I) = 0
        MSK(I) = 0
    1 CONTINUE
C
      DO 2 I = 1, NTRSM1
        ITR = M * I
        DO 3 J = 1, M
          K = INDEX(ITR + J)
          MSK(K) = MSK(K)+ 1
          MAT(K, MSK(K)) = I
    3   CONTINUE
    2 CONTINUE
C
      CALL TIMER('MAT GENERATED.      ')
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE TIMER(HD)
C-----------------------------------------------------------------
C     PRINT THE TOTAL ACCUMULATED TIME AND THE TIME SINCE THIS
C     ROUTINE WAS LAST CALLED.
C     TNEW - (AT ENTRY)  TIME ROUTINE WAS LAST CALLED.
C     TSTRT - INITIAL TIME PROGRAM STARTED.
C-----------------------------------------------------------------
      COMMON /TIME/ TSTRT, TNEW, TOLD
      INTEGER*4 TIMER1
      REAL*4 HD(5)
C-----------------------------------------------------------------
      TOLD = TNEW
      TNEW = TIMER1(1)
      SEC1 = (TNEW - TOLD)* 0.001
      SEC2 = (TNEW - TSTRT)* 0.001
C
      WRITE(6, 100)HD, SEC1, SEC2
  100 FORMAT(/, 1X, 5A4, '  TIME USED:', F8.3, ' SECONDS.',
     *   '             ACCUMULATED TIME:', F8.3, ' SECONDS'/)
      RETURN
      END
C*****************************************************************
      SUBROUTINE PAGE(HD)
C-----------------------------------------------------------------
C     SKIP TO A NEW PAGE AND PRINT OUT A HEADING.
C-----------------------------------------------------------------
      REAL*4 HD(5)
C-----------------------------------------------------------------
      WRITE(6, 100)
  100 FORMAT('1 FINITE ELEMENT SOLUTION OF PDE-S .............',
     *    '  PHASE1:(TRIANGULATION OF THE DOMAIN) '/)
C
      WRITE(6, 101)HD
  101 FORMAT(T40, 5A4)
C
      RETURN
      END
```

Phase 2

Generation of the Equations

```
C*********************************************************************|C
      BLOCK DATA                                                      |C
C---------------------------------------------------------------------|C
C  FINITE ELEMENT SOLUTION OF PDE-S .                                 |C
C                                                                     |C
C  PHASE 2: GENERATION OF STIFFNESS MATRICES ETC.                     |C
C                                                                     |C
C  PROGRAMMED BY ALAN GEORGE                                          |C
C  COMPUTER SCIENCE DEPARTMENT,                                       |C
C  STANFORD UNIVERSITY,                                               |C
C  JUNE 2, 1973                                                       |C
C---------------------------------------------------------------------|C
C  VARIABLES AND THEIR FUNCTION ......                                |C
C                                                                     |C
C  * INDICATES INPUT VARIABLE.                                        |C
C                                                                     |C
C                                                                     |
C  ITBL1 -  TABLE WHOSE I-TH ROW HAS PARAMETER CODES ASSOCIATED       |
C      WITH THE I-TH NODE OF THE GENERAL TRIANGLE.  (M X NPND(I))     |
C                                                                     |
C  NPND - NPND(K) - NUMBER OF PARAMETERS ASSOCIATED WITH K-TH         |
C    NODE OF CURRENT TRIANGLE, LENGTH = M, GENERATED IN INIT1.        |
C                                                                     |
C* NPS -  NODES-PER-SIDE (NUMBER OF NODES ON EACH TRIANGLE            |
C         SIDE, NOT INCLUDING THE CORNER NODES).                      |
C                                                                     |
C* NCP - NUMBER OF CORNER PARAMETERS.                                 |
C                                                                     |
C* NSP(I) - NUMBER OF PARAMETERS ASSOCIATED  WITH THE I-TH            |
C         NODE ON EACH SIDE.                                          |
C                                                                     |
C* ICP(I) - PARAMETER CODE ASSOCIATED WITH THE I-TH                   |C
C        PARAMETER OF A CORNER NODE.                                  |C
C                                                                     |C
C* ISP(I,*) - PARAMETER CODES ASSOCIATED WITH THE I-TH                |C
C       NODE ON THE TRIANGLE SIDE.                                    |C
C                                                                     |C
C  PARAMETER CODES:                                                   |C
C         1 - U                                                       |C
C         2 - UX                                                      |C
C         3 - UY                                                      |C
C                                                                     |C
C  ITYPE - VECTOR OF TYPES FOR THE PARAMETERS OF THE                  |C
C          CURRENT TRIANGLE. (LENGTH IS M)                            |C
C                                                                     |C
C  N - THE NUMBER OF TERMS IN THE GENERAL POLYNOMIAL OF DEGREE        |C
C       'IDEG' IN 2 VARIABLES.                                        |C
C                                                                     |C
C* IBUG - IF ZERO, NO DEBUGGING INFORMATION IS PRINTED.               |C
C                                                                     |C
C  MAXZ - DECLARED DIMENSION FO THE ARRAY Z.                          |C
C                                                                     |C
C---------------------------------------------------------------------|C
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)      |C
      COMMON /MISC/ CFT(5), CFS(5), TPTS(5), SPTS(5), KQUAD           |C
C---------------------------------------------------------------------|C
C     THESE ARE COEFFICIENTS FOR 9-TH DEGREE QUADRATURE RULES         |C
C     ON (-1,1).  SEE STROUD AND SECREST (1966) FOR DETAILS ON        |C
C     CONICAL PRODUCT FORMULAS.                                       |C
C     CFS AND CFT CONTAIN THE COEFFICIENTS OF THE 9-TH                |C
C     DEGREE QUADRATURE RULES, RESPECTIVELY, FOR THE
C     INTEGRALS:
C           _ 1              _1
C          /                /
C         / (1-S) F(S) DS  / F(T) DT
C       _/               _/
C       0                0
C
C     SPTS AND TPTS ARE THE CORRESPONDING EVALUATION POINTS.
C     IN INIT1, THE COEFFICIENTS AND EVALUATION POINTS OF
C     A NINTH DEGREE CONICAL PRODUCT FORMULA FOR THE
C     (CANONICAL) TRIANGLE (0,0),(1,0),(0,1) ARE GENERATED.
C     TO CHANGE THE FORMULA, WE NEED ONLY CHANGE THE
C     DATA STATEMENTS BELOW AND THE TWO INSTANCES OF THE
C     CARD DECLARING THE LABELED COMMON AREA MISC.
C----------------------------------------------------------------
      DATA   CFT    / 0.2369268850561900, 0.4786286704993700,
     *                0.5688888888888900, 0.4786286704993700,
     *                0.2369268850561900/
      DATA   CFS    / 0.0629916580867700, 0.2956354802905OD0,
     *                0.5855479483387D0, 0.6686985523775D0,
     *                0.3871263609067D0/
      DATA   TPTS  / -0.9061798459386700, -0.5384693101056800,
     *             0D0,0.5384693101056800, 0.9061798459386700/
      DATA   SPTS  / -0.8029298284023500, -0.3909285467072700,
     *                0.1240503795052300, 0.6039731642527800,
     *                0.9203802858970600/
      DATA KQUAD/5/
      END
C*****************************************************************
C     MAINLINE PROGRAM - THIS PART WILL HAVE TO ALTERED SLIGHTLY
C     FOR EACH DIFFERENT PROBLEM THAT IS TO BE SOLVED. ALL
C     OTHER ROUTINES WITH THE EXCEPTION OF FUNC ARE GENERAL
C     SHOULD NOT HAVE TO BE MODIFIED FROM PROBLEM TO PROBLEM.
C
C     THIS MAINLINE IS SET UP NOW TO HANDLE FUNCTIONALS OF THE
C     FOLLOWING FORM:
C
C           _ _
C          / /              2         2         2
C     1/2 / /  UX2 * U + UY2 * U + U2 * U + 2 * H(X,Y) * U DX DY,
C       _/_/             X         Y
C     <DOMAIN>
C
C     WHERE UX2, UY2, AND U2 ARE CONSTANTS, AND H(X,Y) IS
C     SUPPLIED BY THE IRHS-TH FUNCTION IN THE SUBROUTINE FUNC.
C
C       SINCE THE COEFFICIENTS FOR THE QUADRATIC TERMS ARE
C     RESTRICTED TO BE CONSTANTS, THE G-MATRICES REFERRED TO IN
C     CHAPTER 3, SECTION 3 CAN BE COMPUTED ONCE AND FOR ALL BY
C     THE ROUTINE MGEN. THE STIFFNESS MATRICES FOR EACH TRIANGLE
C     ARE THEN LINEAR COMBINATIONS OF THESE MATRICES.   THE
C     COEFFICIENTS DEPEND UPON UX2, UY2, AND THE INVERSE OF THE
C     MAPPING FROM THE CANONICAL TRIANGLE TO THE ONE CURRENTLY
C     BEING PROCESSED.  THIS INFORMATION IS SUPPLIED BY THE
C     ROUTINE GENMAP, WHICH PLACES IT IN THE COMMON BLOCK MAP.
C     THE COEFFICIENTS ARE STORED IN GAM.
C
C         IF IEIG > 0, IT IS ASSUMED THAT THE GIVEN FUNCTIONAL
```

```
C       IS TO BE MINIMIZED SUBJECT TO THE CONSTAINT
C
C                --
C              / /       2
C             / /       U       DX DY = 1  .
C            _/_/
C         <DOMAIN>
C
C       SO-CALLED MASS MATRICES AS WELL AS STIFFNESS MATRICES
C       ARE THEN GENERATED.
C
C-----------------------------------------------------------------
C       BELOW IS A TYPICAL SET OF INPUT DATA FOR PHASE 2.  IT
C       SPECIFIES THE SOLUTION OF DEL U - U = 0  USING THE
C       3-4 ELEMENT (SEE APPENDIX A).
C
C       &PARMS IBUG=0,IDEG=3,NCP=3,ICP=1,2,3,
C              IRHS=0,UX2=1,UY2=1,FU2=1,IEIG=0 &END
C-----------------------------------------------------------------
        IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
        COMMON /MAP/ X12, Y12, X13, Y13, FJAC, P, Q, R, S
        COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, IBUG, NEVPTS,
     *         LPS, LA, LC, IRHS, IEIG, ISP(10,2),
     *         NPS, NCEN, M, N, IDEG, NCP, ICP(3), NSP(10)
        COMMON /TIME/ TSTRT, TOLD, TNEW
C-----------------------------------------------------------------
        INTEGER*2 IO/0/, I  /1/, LG(5)
        COMPLEX*16 Z(120   )
        INTEGER*4 IDIM, TIMER1
        REAL*8 GAM(4)
C-----------------------------------------------------------------
        NAMELIST /PARMS/ NCP, ICP, NSP, ISP, IBUG, IDEG, U2,
     *                  UX2, UY2, IRHS, IEIG
C-----------------------------------------------------------------
C
C       INITIALIZATION OF PARAMETERS TO DEFAULT VALUES ....
C
        MAXZ = 12000
C
C       MAXZ SHOULD BE ALTERED IF THE DECLARATION FOR Z IS CHANGED
C
        TSTRT = TIMER1(0)
        REWIND 1
        REWIND 2
C
  800   IEIG = 0
        TSTRT = TIMER1(1)
        TNEW = TSTRT
        IBUG = 0
        U2 = 0.0D0
        UX2 = 0.0D0
        UY2 = 0.0D0
C
C       READ INPUT
C
        CALL PAGE  (' INPUT INFORMATION   ')
        READ(5, PARMS, END=900)
        READ(1) NPTS, KPTS, NTRS, NSIDE, NCRNR, NPS,
     *          NCEN, M, LAST
C
```

```
C       COMPUTE SOME CONSTANTS FOR STORAGE MAPS....
C
        N = (IDEG + 1) * (IDEG + 2) / 2
        IDIM = N
        N2 = N * N / 2 + 1
        N8 = N / 8 + 1
        LPS = NPTS + 1
        LNPND = LPS + N8
        LTBL1 = LNPND + N8
        LTYPE = LTBL1 + N8 * 3
        LINDEX = LTYPE + N8
        LC = LINDEX + M * (NTRS+1) / 8 + 1
        LA = LC + N2
        LB = LA + N2
        LRHS = LB + N2
        IF (IEIG .EQ. 0) LRHS = LB + 1
        LRHS1 = LRHS + N/2 + 1
        LG(1) = LRHS1 + N/2 + 1
        IF (IEIG .GT. 0) LG(1) = LRHS + 1
        LG(2) = LG(1) + N2
        LG(3) = LG(2) + N2
        LG(4) = LG(3) + N2
        LG(5) = LG(4) + N2
        LASTZ = LG(5) + N2
        FT = 16.0 * LASTZ
        WRITE(6, 104) FT
  104   FORMAT(' BYTES USED IN Z:', F12.2/)
        IF (LASTZ .GT. MAXZ) GO TO 900
C
        CALL INIT1(IDIM, Z, Z(LINDEX), Z(LNPND), Z(LTBL1),
     *             Z(LTYPE), Z(LRHS), Z(LA))
C
        CALL CGEN(IDIM, Z(LINDEX), Z(LPS), Z(LTYPE),
     *             Z(LC), Z(LRHS), Z)
C
C---------------------------------------------------------------------
C--------------- BEGIN PROBLEM DEPENDENT SECTION OF THE PROGRAM
C---------------------------------------------------------------------
C
        CALL MGEN(IDIM, I1, IO, I1, IO, Z(LG(1)), 100, Z)
        CALL MGEN(IDIM, I1, IO, IO, I1, Z(LG(2)), 100, Z)
        CALL MGEN(IDIM, IO, I1, IO, I1, Z(LG(3)), 100, Z)
C
        IF (IEIG .NE. 0)
     *  CALL MGEN(IDIM, IO, IO, IO, IO, Z(LG(5)), 100, Z)
C
        IF (U2 .NE. 0.0D0)
     *  CALL MGEN(IDIM, IO, IO, IO, IO, Z(LG(4)), 100, Z)
C
        CALL SYMTRZ(IDIM, N, Z(LG(2)))
C
        CALL TIMER('END INITIALIZATION   ')
C
C       MAIN PART OF PROGRAM
C
        DO 6 I = 1, NTRSM1
          IF (I .GT. IBUG) IBUG = 0
          CALL GENMAP(I, Z(LINDEX), Z)
          CALL ZMAT(IDIM, N, Z(LA))
C
```

```
      GAM(1) = P*P*UX2 + Q*Q*UY2
      GAM(2) = 2*P*R*UX2 + 2*Q*S*UY2
      GAM(3) = P*R*UX2 + S*S*UY2
      GAM(4) = U2
C
      DO 17 K = 1, 4
        IF (GAM(K) .NE. 0.0D0)
     *   CALL MATADD(IDIM, N, FJAC * GAM(K), Z(LA), Z(LG(K)))
   17 CONTINUE
C
      IF (IRHS .GT. 0)
     *CALL RHSIDE(IDIM, Z(LPS), Z(LRHS), Z(LRHS1), Z(LC))
C
      IF (IEIG .GT. 0) CALL ZMAT(IDIM, N, Z(LB))
      IF (IEIG .GT. 0)
     *   CALL MATADD(IDIM, N, FJAC, Z(LB), Z(LG(5)))
C
      CALL MAPBAK(IDIM, Z(LTYPE), Z(LA), Z(LB), Z(LRHS))
    6 CONTINUE
C
      CALL TIMER ('GENERATION COMPLETE ')
C
      IF (LAST .EQ. 0) GO TO 800
C
  900 ENDFILE 1
      REWIND 1
      REWIND 2
C
      STOP
      END
C*****************************************************************
      DOUBLE PRECISION FUNCTION FUNC (K, X, Y)
C-----------------------------------------------------------------
C     DEFINITIONS OF FUNCTIONS FOR H(X,Y) IN THE INTEGRAL.
C-----------------------------------------------------------------
      REAL*8 X, Y, DCOS
      INTEGER*2 K
C-----------------------------------------------------------------
C
      GO TO (1, 2, 3, 4, 5, 6, 7), K
C
    1 FUNC = 0.0
      RETURN
C
    2 FUNC = 4.0D0
      RETURN
C
    3 FUNC = (2.0D0 - 100.0D0*Y*Y) * DCOS(10.0D0 * X)
      RETURN
C
    4 FUNC = (X*X + Y*Y) * 12.0D0
      RETURN
C
    5 CONTINUE
      FUNC = 2*X*X*(2*X-3) + 6*Y*(Y-2)*(2*X-1)
      RETURN
C
    6 FUNC = -1.0D0
      RETURN
C
    7 FUNC = 2*(1-6*X+6*X*X)*(Y-Y*Y)**3 +
     *      6 * (1-5*Y+5*Y*Y)*(X-X*X)**2*(Y-Y*Y) -
     *    (X-X*X)**2*(Y-Y*Y)**3
      RETURN
      END
C*****************************************************************
      INTEGER FUNCTION IXP*2(I)
C-----------------------------------------------------------------
C     DETERMINE THE EXPONENT OF X IN THE I-TH TERM OF THE
C                                      2
C     SEQUENCE 1, X, Y, X , XY, .....
C-----------------------------------------------------------------
      INTEGER*2 I
C-----------------------------------------------------------------
      I2 = I * 2
      K = 0
    1 K = K + 1
      IF (K * (K+1) .LT. I2) GO TO 1
      IXP = K - I + K * (K-1) / 2
      RETURN
      END
C*****************************************************************
      INTEGER FUNCTION IYP*2(I)
C-----------------------------------------------------------------
C     DETERMINE THE EXPONENT OF Y IN THE I-TH TERM OF THE
C                                      2
C     SEQUENCE 1, X, Y, X , XY, .....
C-----------------------------------------------------------------
      INTEGER*2 I
C-----------------------------------------------------------------
      I2 = I * 2
      K = 0
    1 K = X + 1
      IF (K * (K+1) .LT. I2) GO TO 1
      IYP = I - K * (K-1) / 2 - 1
      RETURN
      END
C*****************************************************************
      DOUBLE PRECISION FUNCTION FACT(I)
C-----------------------------------------------------------------
C     COMPUTE I FACTORIAL.
C-----------------------------------------------------------------
      INTEGER*2 I
C-----------------------------------------------------------------
      FACT = 1D0
      IF (I .LE. 1) RETURN
      DO 1 K = 2, I
        FACT = FACT * K
    1 CONTINUE
      RETURN
      END
C*****************************************************************
      SUBROUTINE MAPBAK(IDIM, ITYPE, A, B, RHS)
C-----------------------------------------------------------------
C     MODIFY THE STIFFNESS MATRIX SO THAT IT APPLIES TO THE
C     TRIANGLE BEING PROCESSED RATHER THAN THE CANONICAL ONE
C     (I. E. , APPLY THE INVERSE MAP.)
C     DITTO FOR THE LOAD VECTOR.
C     THEN OUTPUT THEM ON UNIT 1.
C-----------------------------------------------------------------
```

```
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /MAP/ X12, Y12, X13, Y13, FJAC, P, Q, R, S
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, IBUG, NEVPTS,
     *       LPS, LA, LC, IRHS, IEIG, ISP(10,3),
     *       NPS, NCEN, M, N, IDEG, NCP, ICP(3), NSP(10)
C-----------------------------------------------------------------
      INTEGER*4 IDIM
      REAL*8 RHS(1), A(IDIM, 1), B(IDIM, 1)
      INTEGER*2 ITYPE(1), I1/1/
C-----------------------------------------------------------------
      DO 1 I = 1, N
        K = ITYPE(I)
        GO TO (1, 2, 1), K
C
C       TRANSFORM FIRST DERIVATIVE PARAMETERS
C
    2   CALL TRANS2(IDIM, I, I+I1, N, X12, Y12, X13, Y13, A)
        IF (IEIG .GT. 0)
     *  CALL TRANS2(IDIM, I, I+I1, N, X12, Y12, X13, Y13, B)
        IF (IRHS .EQ. 0) GO TO 1
        T = RHS(I)
        RHS(I) = T * X12 + X13 * RHS(I+1)
        RHS(I+1) = T * Y12 + Y13 * RHS(I+1)
    1 CONTINUE
C
      WRITE(2) ((A(J, K), K = 1, N), J = 1, N)
      IF (IRHS .GT. 0) WRITE(2) (RHS(J), J = 1, N)
      IF (IEIG .GT. 0)
     * WRITE(2) ((B(J, K), K = 1, N), J = 1, N)
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE TIMER (HD)
C-----------------------------------------------------------------
C     PRINT THE TOTAL ACCUMULATED TIME AND THE TIME SINCE THIS
C     ROUTINE WAS LAST CALLED.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /TIME/ TSTRT, TOLD, TNEW
C-----------------------------------------------------------------
      INTEGER*4 TIMER1
      REAL*4 HD(5)
C-----------------------------------------------------------------
      TOLD = TNEW
      TNEW = TIMER1(1)
      SEC1 = (TNEW - TOLD) * 0.001
      SEC2 = (TNEW - TSTRT) * 0.001
C
      WRITE(6, 100) HD, SEC1, SEC2
  100 FORMAT(/, 1X, 5A4, '  TIME USED:', F8.3, ' SECONDS.',
     *       '          ACCUMULATED TIME:', F8.3, ' SECONDS'/)
      RETURN
      END
C*****************************************************************
      SUBROUTINE PAGE(HD)
C-----------------------------------------------------------------
C     SKIP TO A NEW PAGE AND PRINT A HEADING.
C-----------------------------------------------------------------
      REAL*4 HD(5)
```

```
C-----------------------------------------------------------------
      WRITE(6, 100)
  100 FORMAT('1 FINITE ELEMENT SOLUTION OF PDE-S .............',
     *       ' PHASE2: (GENERATION OF STIFFNESS MATRICES ETC.)'/)
C
      WRITE(6, 101) HD
  101 FORMAT(T40, 5A4)
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE RHSIDE(IDIM, IPS, RHS, RHS1, C)
C-----------------------------------------------------------------
C     THIS ROUTINE GENERATES THE RIGHT SIDE. IT USES A CONICAL
C     PRODUCT INTEGRATION FORMULA ACCURATE FOR  9-TH DEGREE
C     POLYNOMIALS. THE EVALUATION POINTS ARE STORED IN THE
C     ARRAY ZEVPTS AND ARE PLACED THERE BY THE ROUTINE GENMAP.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /INI/ ZSTPTS(50), ZEVPTS(50), COEFF(50)
      COMMON /MAP/ X12, Y12, X13, Y13, FJAC, P, Q, R, S
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, IBUG, NEVPTS,
     *       LPS, LA, LC, IRHS, IEIG, ISP(10,3),
     *       NPS, NCEN, M, N, IDEG, NCP, ICP(3), NSP(10)
C-----------------------------------------------------------------
      INTEGER*4 IDIM
      COMPLEX*16 TZ
      INTEGER*2 IPS(1)
      REAL*8 FVALS(50), V(2), C(IDIM, 1), RHS(1), RHS1(1)
      EQUIVALENCE (TZ, V(1))
C-----------------------------------------------------------------
C
C     EVALUATE THE RHS FUNCTION ONCE AND FOR ALL AT
C     THE EVALUATION POINTS.
C
      DO 1 I = 1, NEVPTS
        TZ = ZEVPTS(I)
        FVALS(I) = FUNC(IRHS, V(1), V(2))
    1 CONTINUE
C
C       NOW INTEGRATE THE FUNCTION TIMES X**IXP(I) * Y**IYP(I)
C       FOR I = 1, 2, ..., N OVER THE CURRENT TRIANGLE.
C
      DO 2 I = 1, N
        TEMP = 0.0D0
        ITX = IXP(I)
        ITY = IYP(I)
        DO 3 J = 1, NEVPTS
          TZ = ZSTPTS(J)
          TEMP = TEMP + COEFF(J)*FVALS(J)*V(1)**ITX*V(2)**ITY
    3   CONTINUE
        RHS1(I) = TEMP* FJAC
    2 CONTINUE
C
      CALL SOLVE(IDIM, N, IPS, C, RHS1, RHS)
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE INIT1(IDIM, Z, INDEX, NPND, ITBL1,
```

```
     *                    ITYPE, RHS, RHS1)
C-----------------------------------------------------------------
C     THIS ROUTINE DOES ALL THE INITIALIZATION.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /INT/ ZSTPTS(50), ZEVPTS(50), COEFF(50)
      COMMON /MISC/ CFT(5), CFS(5), TPTS(5), SPTS(5), KQUAD
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, IBUG, NEVPTS,
     *       LPS, LA, LC, IRHS, IEIG, ISP(10,3),
     *       NPS, NCEN, M, N, IDEG, NCP, ICP(3), NSP(10)
C-----------------------------------------------------------------
      INTEGER*4 IDIM
      INTEGER*2 INDEX(1), NPND(1), ITYPE(1), ITBL1(IDIM,3)
      COMPLEX*16 Z(1), TZ
      REAL*8 V(2), RHS(1), RHS1(1)
      EQUIVALENCE (TZ, V(1))
C-----------------------------------------------------------------
      NAMELIST   /PARMS1/ NPTS, NTRS, NSIDE, NPS, NCEN, M, N,
     *              IDEG, IRHS, IEIG
C-----------------------------------------------------------------
C
C     READ INPUT FROM PHASE 1....
C
      WRITE(6, PARMS1)
      READ(1) (Z(I), I = 1, NPTS)
      READ(1) I, (INDEX(K), K = 1, I)
      READ(1) ((IDUMM, K = 1, 4), I = 1, NSIDE)
      READ(1) NCRNR, ((IDUMM, J = 1, 2), I = 1, NCRNR)
      READ(1) (IDUMM, I = 1, NPTS)
C
C     END PHASE 1 INPUT.
C
      NTRSM1 = NTRS - 1
      IF (IBUG .EQ. 0) GO TO 18
C
C     LIST INPUT FROM PHASE 1 ON THE PRINTER ....
C
      WRITE(6, 100)
  100 FORMAT('   ', T35, ' POINT NUMBERS AND COORDINATES:'/)
      WRITE(6, 101) ((I, Z(I)), I = 1, NPTS)
  101 FORMAT(1X, I4, 2F12.8, I4, 2F12.8, I4, 2F12.8, I4, 2F12.8)
      WRITE(6, 99)
   99 FORMAT('1 TRIANGLE      INDICES '/)
      DO 11 I = 1, NTRS
        K1 = I * M + 1
        K2 = K1 + M - 1
        WRITE(6, 102) I, (INDEX(K), K = K1, K2)
  102   FORMAT(1X, 26I5)
   11 CONTINUE
C
   18 CALL TIMER ('INPUT COMPLETE      ')
C
C     COMPUTE THE COEFFICIENTS FOR THE INTEGRATION FORMULA
C     AND THE EVALUATION POINTS ZSTPTS.
C
      DO 1 I = 1, KQUAD
         RHS(I) = (1.0D0 - SPTS(I)) / 2.0D0
         RHS1(I) = (1.0D0 + TPTS(I)) / 2.0D0
    1 CONTINUE
C
      NEVPTS = 0
      DO 2 I = 1, KQUAD
         DO 3 J = 1, KQUAD
            NEVPTS = NEVPTS + 1
            COEFF(NEVPTS) = CFS(I) * CFT(J) / 8.0D0
            V(1) = RHS(I)
            V(2) = RHS1(J) * (1.0D0 - RHS(I))
            ZSTPTS(NEVPTS) = TZ
    3    CONTINUE
    2 CONTINUE
C
      DO 14 I = 1, N
        NPND(I) = 0
        DO 15 J = 1, 3
          ITBL1(I,J) = 0
   15   CONTINUE
   14 CONTINUE
C
C     GENERATE TABLES USED BY OTHER ROUTINES.
C     SEE COMMENTS IN BLOCK DATA SUBPROGRAM FOR DETAILS.
C
      L1 = 3
      L = 0
      DO 12 I = 1, 3
         NPND(I) = NCP
         DO 13 J = 1, NCP
            ITBL1(I, J) = ICP(J)
            L = L + 1
            ITYPE(L) = ICP(J)
   13    CONTINUE
   12 CONTINUE
C
      IF (NPS .EQ. 0) GO TO 20
C
      DO 8 I = 1, 3
         DO 9 J = 1, NPS
            L1 = L1 + 1
            NPND(L1) = NSP(J)
            K1 = NSP(J)
            DO 10 K = 1, K1
               L = L + 1
               ITBL1(L1, K) = ISP(J, K)
               ITYPE(L) = ISP(J, K)
   10       CONTINUE
    9    CONTINUE
    8 CONTINUE
C
   20 IF (NCEN .EQ. 0) GO TO 30
C
      DO 16 I = 1, NCEN
         L1 = L1 + 1
         L = L + 1
         ITBL1(L1, 1) = 1
         NPND(L1) = 1
         ITYPE(L) = 1
   16 CONTINUE
C
   30 N = L
      WRITE(6, 103) IDEG, L
  103 FORMAT(' DEGREE OF PIECEWISE POLYNOMIAL:', I4,/,
```

```
     *         ' NUMBER OF PARAMETERS CHARACTERIZING IT:', I4/)
C
C     OUTPUT PHASE 2 PARAMETERS FOR PHASE 3 INPUT...
C
      WRITE(2) N, IRHS, IEIG
      WRITE(2) (NPNO(K), K = 1, M),
     *        ((ITBL1(I,J), J=1,3), I = 1, M)
      RETURN
      END
C*****************************************************************
      SUBROUTINE SYMTRZ (IDIM, N, A)
C-----------------------------------------------------------------
C     REPLACE A BY (A + A(TRANSPOSE)) / 2
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N)
      INTEGER*4 IDIM
      REAL*8 A(IDIM, 1), TEMP
C-----------------------------------------------------------------
      N1 = N - 1
      DO 1 I = 1, N1
       IP1 = I + 1
       DO 2 J = IP1, N
        TEMP = (A(I,J) + A(J,I)) / 2D0
        A(I,J) = TEMP
        A(J,I) = TEMP
    2  CONTINUE
    1 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE TRANSP (IDIM, N, A)
C-----------------------------------------------------------------
C     TRANSPOSE THE MATRIX A.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N)
      INTEGER*4 IDIM
      REAL*8 A(IDIM, 1), TEMP
C-----------------------------------------------------------------
      N1 = N - 1
      DO 1 I = 1, N1
       IP1 = I + 1
       DO 2 J = IP1, N
        TEMP = A(I,J)
        A(I,J) = A(J,I)
        A(J,I) = TEMP
    2  CONTINUE
    1 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE GENMAP(ITR, INDEX, Z)
C-----------------------------------------------------------------
C     GENERATE THE MAPPING FROM THE CANONICAL TRIANGLE TO THE
C     ONE CURRENTLY BEING PROCESSED.
C     LET THE VERTICES BE (X1,Y1), (X2,Y2), AND (X3,Y3).
C     THIS ROUTINE COMPUTES X12 = X2-X1, Y12 = Y2-Y1,
C     X13 = X3-X1, AND Y13 = Y3-Y1.  THE JACOBIAN OF THE TRANS.
C     IS ASSUMED BY FJAC, AND THE INVERSE OF THE JACOBIAN IS
C     THE MATRIX
C                 | P  Q |
C                 | R  S |
C     NOTE THAT THESE QUANTITIES ARE STORED IN COMMON SO THAT
C     THE USER CAN ACCESS THEM IN THE MAINLINE PROGRAM.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /INT/ ZSTPTS(50), ZEVPTS(50), COEFF(50)
      COMMON /MAP/ X12, Y12, X13, Y13, FJAC, P, Q, R, S
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, IBUG, NEVPTS,
     *        LPS, LA, LC, IRHS, IEIG, ISP(10,3),
     *        NPS, NCEN, M, N, IDEG, NCP, ICP(3), NSP(10)
C-----------------------------------------------------------------
      COMPLEX*16 Z1, Z(1), D12, D13, TZ
      INTEGER*2 INDEX(1)
      REAL*8 U(2)
      EQUIVALENCE (D12, X12), (D13, X13), (TZ, U(1))
C-----------------------------------------------------------------
C
      ITR1 = ITR * M
      Z1 = Z(INDEX(ITR1 + 1))
      D12 = Z(INDEX(ITR1+2)) - Z1
      D13 = Z(INDEX(ITR1+3)) - Z1
      IF (IRHS .EQ. 0) GO TO 2
C
C     ZEVPTS ARE THE IMAGES OF ZSTPTS UNDER THE MAP.
C
      DO 1 I = 1, NEVPTS
        TZ = ZSTPTS(I)
        ZEVPTS(I) = Z1 + U(1) * D12 + U(2) * D13
    1 CONTINUE
C
    2 FJAC = X12 * Y13 - X13 * Y12
      P = Y13 / FJAC
      Q = -X13 / FJAC
      R = -Y12 / FJAC
      S = X12 / FJAC
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE DERIV1 (DX, DY, KX, KY, CF)
C-----------------------------------------------------------------
C     CONSIDER THE POLYNOMIAL:
C
C                            2            2     3     2
C     . 1  +  X  +  Y  +  X   +  XY  +  Y   +  X   +X Y .....
C
C     DERIV1 PLACES THE COEFFICIENT, POWER OF X, AND POWER OF Y
C     OF THE DX-DY-TH DERIVATIVE OF THE I-TH TERM IN CF(I),
C     KX(I), AND KY(I) RESPECTIVELY.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, IBUG, NEVPTS,
     *        LPS, LA, LC, IRHS, IEIG, ISP(10,3),
     *        NPS, NCEN, M, N, IDEG, NCP, ICP(3), NSP(10)
C-----------------------------------------------------------------
      INTEGER*2 KX(1), KY(1), TX, TY, QX, QY, DX, DY
      REAL*8 CF(1)
C-----------------------------------------------------------------
```

```
C
      DO 1 I = 1, N
        TX = IXP(I)
        TY = IYP(I)
        QX = TX - DX
        QY = TY - DY
C
        IF (QX.LT.0 .OR. QY.LT.0)GO TO 2
C
        CF(I) = FACT(TX)/FACT(QX) * FACT(TY)/FACT(QY)
        KX(I) = QX
        KY(I) = QY
        GO TO 1
C
    2   CF(I) = 0.0D0
        KX(I) = 0
        KY(I) = 0
    1 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE DERIV2 (DX, DY, X, Y, AR)
C-----------------------------------------------------------------
C     DERIV2 IS SIMILAR TO DERIV1 EXCEPT IT  GENERATES
C     THE I-TH TERMS PRODUCED BY DERIV1 EVALUATED AT (X,Y).
C     THE EVALUATED TERMS ARE PLACED IN THE ARRAY AR.
C -----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, IBUG, NEVPTS,
     *       LPS, LA, LC, IRHS, IEIG, ISP(10,3),
     *       NPS, NCEN, M, N, IDEG, NCP, ICP(3), NSP(10)
C-----------------------------------------------------------------
      INTEGER*2 DX, DY, TX, TY, QX, QY
      REAL*8  AR(1), WX(55), WY(55), TEMP, X, Y
C-----------------------------------------------------------------
C
      WX(1) = 1.0D0
      WY(1) = 1.0D0
      K =  IDEG - DX - DY + 1
      K = K * (K+1) / 2
C
      DO 1 I = 1, K
        WX(I+1) = X * WX(I)
        WY(I+1) = Y * WY(I)
    1 CONTINUE
C
      DO 3 I = 1, N
        TX = IXP(I)
        TY = IYP(I)
        QX = TX - DX
        QY = TY - DY
C
        IF(QX.LT.0 .OR. QY.LT.0) GO TO 3
C
        TEMP = FACT(TX)/FACT(QX) * FACT(TY)/FACT(QY)
        AR(I) = TEMP * WX(QX+1)* WY(QY+1)
    3 CONTINUE
C
      RETURN
```

```
      END
C*****************************************************************
      SUBROUTINE CTRAN (IDIM, N, IPS, LU, A, W)
C-----------------------------------------------------------------
C     THIS ROUTINE APPLIES THE CONGRUENCE TRANSFORMATION
C     LU TO A,  I.E. A IS REPLACED BY LU(-T) A LU(-1).
C     IF VARIABLE QUADRATIC COEFFICIENTS ARE PRESENT, AND WE
C     ARE CARRYING OUT THIS TRANSFORMATION FOR EACH TRIANGLE,
C     THEN THIS ROUTINE SHOULD BE MODIFIED TO USE THE
C     TECHNIQUE DESCRIBED IN SECTION 3.3.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N)
      INTEGER*4 IDIM
      REAL*8 LU(IDIM, 1), A(IDIM, 1), W(IDIM, 1)
      INTEGER*2 IPS(1)
C-----------------------------------------------------------------
C
      DO 1 K = 1, N
        CALL SOLVE(IDIM, N, IPS, LU, A(1, K), W(1, K))
    1 CONTINUE
C
      CALL TRANSP(IDIM, N, W)
C
      DO 2 K = 1, N
        CALL SOLVE(IDIM, N, IPS, LU, W(1, K), A(1, K))
    2 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE MATADD (IDIM, N, CNST, A, B)
C-----------------------------------------------------------------
C     ADD CNST TIMES THE MATRIX B TO THE MATRIX A.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N)
      INTEGER*4 IDIM
      REAL*8 CNST, A(IDIM, 1), B(IDIM, 1), DABS
C-----------------------------------------------------------------
C
      IF (DABS(CNST) .LT. 1.0D-14) RETURN
C
      DO 1 I = 1, N
       DO 2 J = 1, N
         A(I,J) = A(I,J) + CNST * B(I,J)
    2  CONTINUE
    1 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE ZMAT(IDIM, N, A)
C-----------------------------------------------------------------
C     ZERO OUT THE N BY N MATRIX A.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N)
      INTEGER*4 IDIM
      REAL*8 A(IDIM, 1)
C-----------------------------------------------------------------
      DO 1 I = 1, N
        DO 2 J = 1, N
```

```
        A(I,J) = C...
    2   CONTINUE
    1 CONTINUE
C
      RETURN
      END
C***************************************************************
      SUBROUTINE TRANS2 (IDIM, K1, K2, N, B11, B12, B21, B22, A)
C---------------------------------------------------------------
C     APPLY THE 2 X 2 TRANSFORMATION B TO THE COLUMNS
C     AND ROWS K1 AND K2 OF THE MATRIX A.
C---------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N)
      INTEGER*4 IDIM
      REAL*8 B11, B12, B21, B22, A(IDIM, 1), T1, T2
C---------------------------------------------------------------
C
      DO 1 K = 1, N
         T1 = A(K,K1)
         T2 = A(K,K2)
         A(K,K1) = B11 * T1 + B21 * T2
         A(K,K2) = B12 * T1 + B22 * T2
    1 CONTINUE
C
      DO 2 K = 1, N
         T1 = A(K1,K)
         T2 = A(K2,K)
         A(K1,K) = B11 * T1 + B21 * T2
         A(K2,K) = B12 * T1 + B22 * T2
    2 CONTINUE
C
      RETURN
      END
C***************************************************************
      SUBROUTINE CGEN(IDIM, INDEX, IPS, ITYPE, C, ZPTS, Z)
C---------------------------------------------------------------
C     GENERATE THE MATRIX C WHICH RELATES THE MONOMIAL TERMS TO
C     THE PARAMETERS WE ARE USING TO CHARACTERIZE THE POLYNOMIAL
C     ON THE TRIANGLE.
C---------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, IBUG, NEVPTS,
     *       LPS, LA, LC, IRHS, IEIG, ISP(10,3),
     *       NPS, NCEN, M, N, IDEG, NCP, ICP(3), NSP(10)
C---------------------------------------------------------------
      INTEGER*4 IDIM
      COMPLEX*16 TZ, ZPTS(1), Z(1)
      REAL*8 X(2), C(IDIM, 1)
      EQUIVALENCE (TZ, X(1))
      INTEGER*2 IPS(1), INDEX(1), ITYPE(1)
C---------------------------------------------------------------
C
      CALL ZMAT(IDIM, N, C)
C
      ITR1 = NTRS * 4
      L = C
      L1 = 3
      DO 1 I = 1, 3
        DO 2 J = 1, NCP
          L = L + 1
```

```
          ZPTS(L) = Z(INDEX(I - ITR1))
    2   CONTINUE
    1 CONTINUE
C
      IF (NPS .EQ. 0) GO TO 10
C
      DO 3 I = 1, 3
        DO 4 J = 1, NPS
          L1 = L1 + 1
          K1 = NSP(J)
          K2 = INDEX(L1 + ITR1)
          DO 5 K = 1, K1
            L = L + 1
            ZPTS(L) = Z(K2)
    5     CONTINUE
    4   CONTINUE
    3 CONTINUE
C
   10 IF (NCEN .EQ. 0)  GO TO 20
C
      DO 6 I = 1, NCEN
        L1 = L1 + 1
        L = L + 1
        ZPTS(L) = Z(INDEX(L1 + ITR1))
    6 CONTINUE
C
   20 DO 7 I = 1, N
        TZ = ZPTS(I)
        K1 = IXP(ITYPE(I))
        K2 = IYP(ITYPE(I))
        CALL DERIV2(K1, K2, X(1), X(2), C(1,I))
    7 CONTINUE
C
      CALL DECOMP(IDIM, N, IPS, C, C)
C
      RETURN
      END
C***************************************************************
      SUBROUTINE MGEN(IDIM, IDX, IDY, JDX, JDY, H, CNST, Z)
C---------------------------------------------------------------
C                          2       2
C     LET Q = 1, X, Y, X , XY, Y , ...       (TO THE N-TH TERM)
C     THEN DEFINE R AS THE IDX-IDY-TH DERIVATIVE OF Q AND
C     DEFINE S AS THE JDX-JDY-TH DERIVATIVE OF Q.
C
C     THEN MGEN GENERATES CNST * H, WHERE H IS THE INTEGRAL OF
C                    T
C     THE MATRIX R * S  OVER THE CANONICAL TRIANGLE.
C     IF THE COEFFICIENTS OF ANY OF THE QUADRATIC TERMS IN THE
C     INTEGRAND ARE VARIABLE, THE STATEMENT H(I,J) = .....
C     MUST BE REPLACED BY A SEQUENCE OF STATEMENTS TO CARRY OUT
C     NUMERICAL INTEGRATION.  ANOTHER ARGUMENT WOULD HAVE TO BE
C     AVAILABLE TO INDICATE WHICH FUNCTION IN FUNC IS TO BE
C     USED.
C ---------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, IBUG, NEVPTS,
     *       LPS, LA, LC, IRHS, IEIG, ISP(10,3),
     *       NPS, NCEN, M, N, IDEG, NCP, ICP(3), NSP(10)
C---------------------------------------------------------------
```

```
      INTEGER*4 IDIM
      COMPLEX*16 Z(1)
      INTEGER*2 TXI, TYI, XPW, YPW
      INTEGER*2 IX(55), IY(55), JX(55), JY(55)
      REAL*8 H(IDIM, 1), CI(55), CJ(55)
C-----------------------------------------------------------------------
C
      CALL ZMAT(IDIM, N, H)
      CALL DERIV1(IDX, IDY, IX, IY, CI)
      CALL DERIV1(JDX, JDY, JX, JY, CJ)
C
      DO 1 I = 1, N
        TXI = IX(I)
        TYI = IY(I)
        TCI = CI(I) * CNST
        IF (TCI .EQ. 0.0D0) GO TO 1
        DO 2 J = 1, N
          TCJ = TCI * CJ(J)
          IF (TCJ .EQ. 0.0D0) GO TO 2
          XPW = TXI + JX(J)
          YPW = TYI + JY(J)
          K = XPW + YPW + 2
          H(I,J) =  TCJ * FACT(XPW) * FACT(YPW) / FACT(K)
    2   CONTINUE
    1 CONTINUE
C
      CALL CTRAN(IDIM, N, Z(LPS), Z(LC), H, Z(LA))
      RETURN
      END
C***********************************************************************
      SUBROUTINE DECOMP (IDIM, N, IPS, A, LU)
C-----------------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H, O-Z)
      INTEGER*4 IDIM
      REAL*8 LU(IDIM, 1)
      DIMENSION IPS(1), A(IDIM, 1)
C-----------------------------------------------------------------------
C       DECOMPOSE THE N*N MATRIX A INTO TRIANGULAR L & U SO THAT
C       L*U = A. IPS IS THE ROW PIVOT VECTOR.
C       MATRIX A WILL BE OVERWRITTEN BY LU IF A AND LU ARE
C       DECLARED TO BE THE SAME MATRIX IN THE CALL OF DECMP2.
C-----------------------------------------------------------------------
        INTEGER I, J, K, IP, KP, KP1, NM1, IDXPIV
        REAL*8 SCALES(55), EM, BIG, SIZE, PIVOT, ROWNRM, ZERO
C-----------------------------------------------------------------------
C       INITIALIZE IPS, LU, AND SCALES.
C
        ZERO = 0.0D0
        DO 5 I = 1, N
          IPS(I) = I
          ROWNRM = 0.0D0
          DO 2 J = 1, N
            LU(I, J) = A(I, J)
            ROWNRM = DMAX1(ROWNRM, DABS(LU(I, J)))
    2     CONTINUE
C
C         TEST FOR MATRIX WITH ZERO ROW.
C
          IF (ROWNRM .EQ. ZERO) GO TO 20
          SCALES(I) = 1.0/ROWNRM
```

```
    5   CONTINUE
C
C       GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING.
C
        NM1 = N - 1
        DO 17 K = 1, NM1
          BIG = 0.0D0
          DO 11 I = K, N
            IP = IPS(I)
            SIZE = DABS(LU(IP, K))*SCALES(IP)
            IF (SIZE .LE. BIG) GO TO 11
            BIG = SIZE
            IDXPIV = I
   11     CONTINUE
          IF (BIG .EQ. ZERO) GO TO 21
C         TEST FOR ZERO PIVOT.
C
C         NOW INTERCHANGE ROWS IF NECESSARY.
C
          IF (IDXPIV .EQ. K) GO TO 15
          J = IPS(K)
          IPS(K) = IPS(IDXPIV)
          IPS(IDXPIV) = J
   15     KP = IPS(K)
          PIVOT = LU(KP, K)
          KP1 = K + 1
          DO 16 I = KP1, N
            IP = IPS(I)
            EM = LU(IP, K)/PIVOT
            LU(IP, K) = EM
            DO 16 J = KP1, N
              LU(IP, J) = LU(IP, J) - EM*LU(KP, J)
   16     CONTINUE
   17   CONTINUE
C
        IF (LU(IPS(N), N) .EQ. ZERO) GO TO 21
C
        RETURN
C
C     ERROR MESSAGES....
C
   20 WRITE(6, 100)
  100 FORMAT(' ERROR IN DECOMP - ZERO ROWNORM ')
      RETURN
C
   21 WRITE(6, 101)
  101 FORMAT(' ERROR IN DECOMP - ZERO PIVOT ')
      RETURN
      END
C***********************************************************************
      SUBROUTINE SOLVE (IDIM, N, IPS, LU, B, X)
C-----------------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H, O-Z)
      INTEGER*4 IDIM
      REAL*8 LU(IDIM, 1)
      DIMENSION IPS(1), B(1), X(1)
C-----------------------------------------------------------------------
C       SOLVES A*X = B USING LU FROM SUBROUTINE DECOMP.
C       FIRST SOLVES THE TRIANGULAR LINEAR SYSTEM LY = B
C       AND THEN SOLVES THE SYSTEM U*X = Y.
```

```
C        IPS IS THE ROW INTERCHANGE VECTOR FROM DECOMP.
C-----------------------------------------------------------------------
         INTEGER I, J, IP, IP1, IM1, NP1, IBACK
         REAL*8 SUM
C-----------------------------------------------------------------------
         NP1 = N + 1
C
C        FIND U*X = L**(- 1)*B.
C
         X(1) = B(IPS(1))
         DO 2 I = 2, N
            IP = IPS(I)
            IM1 = I - 1
            SUM = 0.0D0
            DO 1 J = 1, IM1
               SUM = SUM + LU(IP, J)*X(J)
    1       CONTINUE
            X(I) = B(IP) - SUM
    2    CONTINUE
         X(N) = X(N)/LU(IPS(N), N)
C
C        FIND X = U**(- 1)*L**(- 1)*B.
C
         DO 4 IBACK = 2, N
            I = NP1 - IBACK
C
C           I GOES FROM (N - 1) TO 1.
C
            IP = IPS(I)
            IP1 = I + 1
            SUM = 0.0D0
            DO 3 J = IP1, N
               SUM = SUM + LU(IP, J)*X(J)
    3       CONTINUE
    4    X(I) = (X(I) - SUM)/LU(IP, I)
         RETURN
C      LAST CARD OF SUBROUTINE SOLVE2.
       END
```

Phase 3

Assembly of the Equations and Solution of Dirichlet Problems

```
C***********************************************************************
C          FINITE ELEMENT SOLUTION OF PDE-S.
C
C       PHASE 3:      ASSEMBLY OF THE EQUATIONS.
C
C PROGRAMMED BY ALAN GEORGE
C COMPUTER SCIENCE DEPARTMENT,
C STANFORD UNIVERSITY
C JUNE 2, 1970
C-----------------------------------------------------------------------
C VARIABLES AND THEIR FUNCTION...........
C
C * INDICATES INPUT VALUE.
C
C REFERENCE TO TRIANGLE WILL USUALLY MEAN 'TRIANGLE CURRENTLY
C          BEING PROCESSED'.
C
C IDOPE - VECTOR OF CODES FOR ALL PARAMETERS IN THE PROBLEM.
C       THE CODE IS:
C
C          PARAMETER      CODE  (+ OR -)
C
C             U             1     A POSITIVE CODE MEANS THE
C             UX            2   PARAMETERS'S VALUE IS SPECIFIED
C             UY            3   BY BOUNDARY CONDITIONS.
C
C          THE CODES FOR PARAMETERS ASSOCIATED WITH EACH NODE
C          APPEAR IN CONTIGUOUS POSITIONS IN IDOPE.
C
C IBASE -  IBASE(K)+1 POINTS TO THE POSITION OF THE FIRST
C          PARAMETER IN IDOPE ASSOCIATED WITH THE K-TH NODE.
C
C ISB2  -  IF IDOPE(K) > 0  THEN THE ELEMENT ISB2(K) POINTS
C   TO A POSITION IN THE ARRAY VALS WHERE THE CORRESP. VALUE
C   AND ANGLE (IF RELEVANT) ARE FOUND. IF IDOPE(K) < 0
C   THEN ISB2(K) MAY BE 0 IN WHICH CASE THE PARAMETER IS
C   THE VALUE, OR IT MAY POINT INTO VALS SPECIFYING THAT
C   THE PARAMETER (UN FOR EXAMPLE) MUST BE TAKEN IN A SPECIFIC
C   DIRECTION.
C
C VALS(*,K) - CONTAINS BOUNDARY INFORMATION ABOUT A SPECIFIC
C   PARAMETER.
C   (1,K) - VALUE
C   (2,K) - COSINE OF ASSOCIATED ANGLE IF RELEVANT.
C   (3,K) - SINE OF ASSOCIATED ANGLE IF RELEVANT.
C
C ISB1 -  ISB1(K) = 0  IF IDOPE(K) > 0,  OTHERWISE IT IS THE
C   POSITION OF THE K-TH PARAMETER IN THE LINEAR SYSTEM.
C
C NPND - NPND(K) - NUMBER OF PARAMETERS ASSOCIATED WITH K-TH NODE
C        OF CURRENT TRIANGLE, LENGTH = M, GENERATED IN THE
C        SUBROUTINE INITI IN PHASE2.
C
C ISUB1 -  POINTERS TO IDOPE CORRESPONDING TO PARAMETERS OF
C   CURRENT TRIANGLE  (FILLED BY OPTRS SUBROUTINE)
C          1-DIMENSIONAL, LENGTH = N.
C ISUB2 -  SAME INFORMATION AS ISUB1 EXCEPT I-TH ROW HAS THE
C    POINTERS CORRESPONDING TO THE I-TH NODE OF THE TRIANGLE.
C
C ITBL1 -  TABLE WHOSE I-TH ROW HAS PARAMETER CODES ASSOCIATED
C   WITH THE I-TH NODE OF THE GENERAL TRIANGLE.  (M X NPND(I))
C
C ITBL2 -  M X 3 MATRIX  WITH  ITBL2(I,J) = 1 IF  THE I-TH NODE
C   OF THE GENERAL TRIANGLE IS ON THE J-TH SIDE (AND HENCE IS
C   INVOLVED IF THE J-TH SIDE OF THE TRIANGLE IS PART OF A
C          BOUNDARY), OTHERWISE IT IS ZERO.
C
C  MU(I) - POSITION OF THE DIAGONAL ELEMENT OF THE I-TH
C          ROW OF THE LOWER TRIANGULAR CHOLESKY FACTOR.
C          CORRESPONDS TO MU IN METHOD 3, SEC. 2, CHAPTER 4.
C
C  IBETA - VECTOR USE FOR STORAGE MAPPING. CORRESPOND TO
C        THE VECTOR BETA OF METHOD 1, SEC. 2, CHAPTER 4.
C
C  MAP - VECTOR USE FOR COMPACT STORAGE MAPPING. CORRESPONDS
C      TO THE VECTOR OMEGA IN METHOD 1, SEC. 2 CHAPTER 4.
C
C  A (B) - STIFFNESS (MASS) MATRIX FOR CURRENT TRIANGLE
C
C  RHS1 - LOAD VECTOR FOR CURRENT TRAINGLE.
C
C  AL (BL) - OVERALL STIFFNESS (MASS) MATRIX.
C
C  LENA - NUMBER OF NON-ZERO ELEMENTS IN THE OVERALL STIFFNESS
C         (MASS) MATRIX.
C
C  LENL - NUMBER OF WORDS NEEDED TO STORE THE CHOLESKY FACTOR
C         OF THE STIFFNESS MATRIX.
C
C  RHS - OVERALL LOAD VECTOR.
C
C*IBND(I,*) - INFO. ABOUT THE BOUNDARY HAVING THE REFERENCE
C        NUMBER I (I WAS ASSIGNED BY INPUT TO PHASE 1) .
C
C    IBND(I,1) - INDEX INTO FBND FOR U VALUE.
C    IBND(I,2) - INDEX INTO FBND FOR UN VALUE
C    IBND(I,3) - INDEX INTO FBND FOR US VALUE
C
C*ISOL(*) - INDICES INTO FBND FOR TRUE SOLUTION, IF KNOWN.
C     (1) - INDEX INTO FBND FOR U VALUE.
C     (2) - INDEX INTO FBND FOR UX VALUE.
C     (3) - INDEX INTO FBND FOR UY VALUE.
C
C*NBNDS = NUMBER OF ROWS OF IBND TO BE INPUT.
C
C*IPRINT - THIS VARIABLE WILL NORMALLY BE SET TO ONE, WHICH
C   SIGNALS THAT VALUES OF THE PARAMETERS (UNKNOWNS AND THOSE
C   SPECIFIED BY BOUNDARY CONDITIONS) WILL BE LISTED.  IF IT
C   IS SET TO ZERO, ONLY THE MAXIMUM ERROR IN THE PARAMETERS
C   (ASSUMING THE SOLUTION IS SUPPLIED BY ISOL) WILL BE LISTED.
C
C-----------------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /TIME/ TSTRT, TOLD, TNEW
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, NBNDS, IBUG, N,
     *    IPRINT, NCEN, NPS, M, NEQNS, NVALS, NDOPE, IBW, NCR,
     *    LAST, LENL, LENA, KPTS, IEIG, IRHS, ISOL(3), IBND(20,3)
C-----------------------------------------------------------------------
      NAMELIST /PARMS/ NBNDS, IBUG, ISOL, IPRINT
     *        /PARMS2/ NEQNS, IBW, IRHS, IEIG, LENA
```

```
C-----------------------------------------------------------------
      INTEGER*4 TIMER1, IDIM
      COMPLEX*16 Z(15000)
C-----------------------------------------------------------------
C
      MAXZ = 15000
C     NOTE - MAXZ MUST BE CHANGED IF THE DECLARED SIZE OF
C     Z IS CHANGED.
      TSTRT = TIMER1(0)
      REWIND 1
      REWIND 2
      REWIND 3
C
   10 NBNDS = 0
      TSTRT = TIMER1(1)
      TNEW = TSTRT
      IBUG = 0
      IPRINT = 1
      IEIG = 0
C
C     ZERO MEMORY BEFORE BEGINNING ASSEMBLY .....
C
      DO 6 I = 1, MAXZ
        Z(I) = (0D0,0D0)
    6 CONTINUE
C
      DO 2 I = 1, 3
        ISOL(I) = 0
        DO 3 J = 1, 20
          IBND(J,I) = 0
    3   CONTINUE
    2 CONTINUE
C
      CALL PAGE('  INPUT INFORMATION ')
      READ(5, PARMS, END=900)
      WRITE(6, PARMS)
C
C     READ BOUNDARY REFERENCE INFORMATION.
C     IREF IS THE REFERENCE NUMBER.
C
      DO 1 I = 1, NBNDS
        READ(5,100) IREF, (IBND(IREF,J), J = 1, 3)
        WRITE(6,100) IREF, (IBND(IREF,J), J = 1, 3)
    1 CONTINUE
  100 FORMAT(10I3)
C
C     READ PARAMETERS OF PHASES ONE AND TWO .........
C
      READ (1)  NPTS, KPTS, NTRS, NSIDE, NCR,
     *          NPS, NCEN, M, LAST
      READ (2)  N, IRHS, IEIG
C
C     COMPUTE SOME CONSTANTS FOR STORAGE ALLOCATION.
C
      IDIM = N
      N2 = N * N / 2 + 1
      N8 = N / 8 + 1
      LINDEX = NPTS + 1
      LTBL1 = LINDEX + M * (NTRS+1) / 8 + 1
      LTBL2 = LTBL1 + 3 * N / 8 + 1
      LNPND = LTBL2 + 3 * N / 8 + 1
      LSUB1 = LNPND + N8
      LSUB2 = LSUB1 + N8
      LSIDE = LSUB2 + 3 * N / 8 + 1
      LCRNR = LSIDE + 4 * NSIDE / 8 + 1
      LP = LCRNR + 2 * NCR/8 + 1
      LBASE = LP + NPTS / 8 + 1
      LDOPE = LBASE + NPTS/8 + 1
C
C     GENERATE TABLES: IDOPE, ITBL2, IBASE
C
      CALL GENDOP(IDIM, Z(LINDEX), Z(LBASE), Z(LNPND), Z(LTBL1),
     *  Z(LTBL2), Z(LDOPE), Z(LSIDE), Z(LCRNR), Z(LP), Z)
      CALL TIMER('GENDOP DONE.          ')
C
C     COMPUTE MORE STORAGE CONSTANTS FOR STORAGE ALLOCATION.
C
      LSB1 = LDOPE + NDOPE/8 + 1
      LSB2 = LSB1 + NDOPE/8 + 1
      LMU = LSB2 + NDOPE/8 + 1
      LBETA = LMU + NDOPE/8 + 1
      LVALS = LBETA + NDOPE/8 + 1
C
      CALL BNDRY(IDIM, Z(LSUB1), Z(LSUB2), Z(LNPND), Z(LSB2),
     *  Z(LDOPE), Z(LINDEX), Z(LBASE), Z(LTBL2), Z(LSIDE),
     *  Z(LCRNR), Z(LVALS), Z)
C
      CALL TIMER('BNDRY DONE.           ')
C
      LMAP = LVALS + 3 * NVALS / 2 + 1
C
      CALL TABLES(IDIM, Z(LBASE), Z(LP), Z(LDOPE), Z(LSB1),
     *     Z(LSB2), Z(LNPND), Z(LSUB1), Z(LSUB2), Z(LINDEX),
     *     Z(LMU), Z(LBETA), Z(LMAP), Z(LVALS))
C
      WRITE(6, PARMS2)
C
C     MORE STORAGE CONSTANTS ....
C
      LAL = LMAP + LENA/8 + 1
      LBL = LAL + LENA/2 + 1
      LRHS = LBL + LENA/2 + 1
      IF (IFIG .EQ. 0) LRHS = LBL + 1
      LRHS1 = LRHS + NFONS/2 + 1
      LA = LRHS1 + N/2 + 1
      LB = LA + N2
      LX = LB + N2
      IF (IFIG .EQ. 0) LX = LA + 1
      LH = LX + NFONS/2 + 1
      LR = LH + NFONS/2 + 1
      LUL = LR + NFONS + 2
      LASTZ = LUL + LENL / 2 + 1
      IF (IFIG .NE. 0) LASTZ = LX
      FZ = LASTZ * 1600
      WRITE(6, 105) FZ
  105 FORMAT(' NUMBER OF BYTES USED IN Z:', F12.2/)
      IF (MAXZ .LT. LASTZ) GO TO 900
C
      CALL TIMER ('BEGIN ASSEMBLY...    ')
C
```

```
      CALL ASSEMB(IDIM, Z(LSUB1), Z(LSUB2), Z(LNPND), Z(LSB1),
     *  Z(LSB2), Z(LDOPE), Z(LP), Z(LMU), Z(LBETA), Z(LINDEX),
     *  Z(LBASE), Z(LMAP), Z(LVALS), Z(LRHS), Z(LRHS1), Z(LA),
     *  Z(LB), Z(LAL), Z(LBL), Z(LX), Z(LUL), Z(LR), Z(LH), Z)
C
      IF (LAST .NE. 1) GO TO 10
C
  990 END FILE 3
      REWIND 1
      REWIND 2
      REWIND 3
      STOP
      END
C*********************************************************************
      DOUBLE PRECISION FUNCTION FBND(K, X, Y, SN, CS)
C---------------------------------------------------------------------
C        FUNCTIONS DEFINING BOUNDARY CONDITIONS AND
C        THE TRUE SOLUTION IF IT IS KNOWN.
C---------------------------------------------------------------------
      INTEGER*2 K
      REAL*8 T1, T2, X, Y, CS, SN, TX(2)
      COMPLEX*16 TZ
      EQUIVALENCE (TZ, TX(1))
C---------------------------------------------------------------------
C
      GO TO (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11), K
C
    1 FBND = (X-X*X)*(X-X*X) * (Y-Y*Y)*(Y-Y*Y)*(Y-Y*Y)
      RETURN
C
    2 FBND = CS*(2D0*(X-X*X)*(1D0-2*X)*(Y-Y*Y)*(Y-Y*Y)*(Y-Y*Y))
     *     + SN*(3D0*(Y-Y*Y)*(Y-Y*Y)*(1D0-2D0*Y)*(X-X*X)*(X-X*X))
      RETURN
C
    3 FBND = Y * Y * DCOS (1C * X)
      RETURN
C
    4 FBND =  -10.0D0 * Y * Y * DSIN(10.0D0 * X) * CS
     *  +     2.0D0 * Y * DCOS(10.0D0 * X) * SN
      RETURN
C
    5 FBND = 0.0D0
      RETURN
C
    6 FBND = DEXP(X) * DCOS(Y)
      RETURN
C
    7 FBND = DEXP(X) * (CS * DCOS(Y) - SN * DSIN(Y))
      RETURN
C
    8 FBND = (2*X*X*X-3*X*X) * (Y*Y - 2*Y)
      RETURN
C
    9 FBND = CS*6*X*Y*(X-1)*(Y-2) + SN*2*(Y-1)*(2*X*X*X-3*X*X)
      RETURN
C
   10 FBND = X*X + Y*Y
      RETURN
C
   11 FBND = CS * 2D0 * X + SN * 2D0 * Y
```

```
      RETURN
C
      END
C*********************************************************************
      SUBROUTINE ASSEMB(IDIM, ISUB1, ISUB2, NPND, ISB1, ISB2,
     *  IDOPE, IP, MU, IBETA, INDEX, IBASE, MAP,
     *  VALS, RHS, RHS1, A, B, AL, BL, U, UL, R, H, Z)
C---------------------------------------------------------------------
C
C        THIS ROUTINE IS THE DRIVING PROGRAM FOR THE ASSEMBLY.
C
C---------------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, NBNDS, IBUG, N,
     *    IPRINT, NCEN, NPS, M, NBNDS, NVALS, NDOPE, IBW, NCR,
     *    LAST, LENL, LENA, KPTS, IEIG, IRHS, ISOL(3), IBND(20,3)
C---------------------------------------------------------------------
      INTEGER*4 IDIM
      INTEGER*2 ISUB1(1), ISUB2(IDIM,3), ISB1(1), ISB2(1),
     *       IDOPE(1), NPND(1), MU(1), IBETA(1), INDEX(1),
     *       IP(1), IBASE(1), MAP(1)
      REAL*8 RHS(1), RHS1(1), A(IDIM,1), B(IDIM,1), AL(1),
     *       BL(1), VALS(3,1), U(1), UL(1), R(1), H(1)
      COMPLEX*16 Z(1)
C---------------------------------------------------------------------
C
      DO 5 K = 1, NTRSM1
C
C        READ IN STIFFNESS MATRICES ETC.
C
        READ(2) ((A(I,J), J = 1, N), I = 1, N)
        IF (IRHS .GT. 0) READ(2) (RHS1(I), I = 1, N)
        IF (IEIG .GT. 0) READ(2)
     *      ((B(I,J), J = 1, N), I = 1, N)
C
        CALL OPTRS(IDIM, K, IBASE, INDEX, ISUB1, ISUB2, NPND)
C
C        IF TRIANGLE HAS SOME BOUNDARY SIDES, MODIFY IT TO
C        TAKE CARE OF DERIVATIVE PARAMETERS WHICH MIGHT BE
C        SPECIFIED  IN DIRECTIONS NOT ALONG A COORDINATE AXIS.
C
        CALL MODIFY(IDIM, ISUB1, IDOPE, ISB2, A, B, RHS1, VALS)
C
C       UPDATE THE OVER-ALL STIFFNESS (MASS) MATRIX
C       AND LOAD VECTORS IF PRESENT.
C
        CALL UPDATE(IDIM, ISUB1, ISB1, ISB2, IDOPE, IBETA,
     *              MAP, AL, BL, A, B, RHS, RHS1, VALS)
C
    5 CONTINUE
C
      CALL TIMER('ASSEMBLY COMPLETE.  ')
C
C     GENERATION OF EQUATIONS COMPLETE... NOW TO SOLVE
C     THE GENERATED LINEAR SYSTEM OR OUTPUT THE GENERALIZED
C     EIGENVALUE PROBLEM.
C
      IF (IEIG .EQ. 0) GO TO 6
C
C     OUTPUT GENERALIZED EIGENVALUE PROBLEM.
```

```
C
      WRITE(3) NEQNS, IBW, LENA, IDSET, LAST
      WRITE(3) (IBETA(I), I = 1, NEQNS)
      WRITE(3) (MAP(I), AL(I), BL(I), I = 1, LENA)
C
      RETURN
C
C
C     SOLVE GENERATED LINEAR SYSTEM USING PROFILE METHODS.
C
    9 CALL SOLVPI(NEQNS, IBW, MU, IBETA, MAP, AL, UL,
     *            RHS, R, U, H)
C
      CALL OUTPUT(IBASE, IDOPE, IP, ISB1, ISB2, VALS, U, RHS, Z)
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE MODIFY(IDIM, ISUB1, IDOPE, ISB2,
     *                  A, B, RHS1, VALS)
C-----------------------------------------------------------------
C IF A DERIVATIVE BOUNDARY CONDITION IS SPECIFIED IN A DIRECTION
C NOT ALONG A COORDINATE AXIS,WE MODIFY A DERIVATIVE PARAMETER SO
C SO THAT IT IS IN THE REQUIRED DIRECTION. THEN THE CONDITION CAN
C BE IMPOSED AND THE PARAMETER ELIMINATED FROM THE PROBLEM.
C THIS ROUTINE CARRIES OUT THE NECESSARY TRANSFORMATIONS
C ON THE MATRIX A, B AND RHS1.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, NBNDS, IBUG, N,
     *   IPRINT, NCEN, NPS, M, NEQNS, NVALS, NDOPE, IBW, NCR,
     *   LAST, LENL, LENA, KPTS, IEIG, IRHS, ISOL(3), IBND(20,3)
C-----------------------------------------------------------------
      INTEGER*4 IDIM
      INTEGER*2 ISUB1(1), IDOPE(1), ISB2(1), I1/1/
      REAL*8 C1, S1, C2, S2, B11, B12, B21, B22, T1,
     *       RHS1(1), A(IDIM,1), B(IDIM,1), VALS(3,1)
C-----------------------------------------------------------------
      DO 1 I = 1, N
        K = ISUB1(I)
        IDP = IDOPE(K)
        IADP = KABS(IDP)
        IF (IADP .NE. 2) GO TO 1
        IF (IDOPE(K+1) .LT. 0 .AND. IDP .LT. 0) GO TO 1
        K1 = ISB2(K)
        K2 = ISB2(K+1)
        C1 = VALS(1,K1)
        S1 = VALS(2,K1)
        C2 = VALS(1,K2)
        S2 = VALS(2,K2)
        B22 = C1*S2 - C2*S1
        B11 = S2/B22
        B12 = -S1/B22
        B21 = -C2/B22
        B22 = C1/B22
        CALL TRANS2(IDIM, I, I+I1, N, B11, B12, B21, B22, A)
C
        IF (IEIG .GT. 0)
     *  CALL TRANS2(IDIM, I, I+I1, N, B11, B12, B21, B22, B)
C
        IF (IRHS .EQ. 0) GO TO 1
```

```
        T1 = RHS1(I)
        RHS1(I) = T1*B11 + RHS1(I+1)*B21
        RHS1(I+1) = T1*B12 + RHS1(I+1)*B22
    1 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE OUTPUT(IBASE, IDOPE, IP, ISB1, ISB2,
     *                  VALS, U, RHS, Z)
C-----------------------------------------------------------------
C     THIS ROUTINE PRINTS OUT INFORMATION OF THE SOLUTION.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, NBNDS, IBUG, N,
     *   IPRINT, NCEN, NPS, M, NEQNS, NVALS, NDOPE, IBW, NCR,
     *   LAST, LENL, LENA, KPTS, IEIG, IRHS, ISOL(3), IBND(20,3)
C-----------------------------------------------------------------
      COMPLEX*16 Z(1), TZ
      REAL*8 TX(2), ERROR, EMAX(3), RHS(1), VALS(3,1), U(1)
      INTEGER*2 IBASE(1), IDOPE(1), IP(1), ISB1(1), ISB2(1)
      INTEGER*2 ISOL1(2,3)/0,0,0,1,1,0/
      EQUIVALENCE (TZ, TX(1))
C-----------------------------------------------------------------
C
      CALL PAGE(' NUMERICAL SOLUTION.')
C
      DO 4 I = 1, 3
        EMAX(I) = 0.0D0
    4 CONTINUE
C
      DO 1 I = 1, KPTS
        K1 = IP(I)
        TZ = Z(K1)
        IF(IPRINT .EQ. 1) WRITE(6, 100) TX(1), TX(2)
  100   FORMAT(' COORDINATES:', 2F12.7)
        KB = IBASE(K1) + 1
        KB1 = NDOPE
        IF (IP(I+1) .NE. 0) KB1 = IBASE(IP(I+1))
        DO 2 J = KB, KB1
          IDP = KABS(IDOPE(J))
          IF (J .GT. KB .AND. IPRINT .EQ. 1) WRITE(6, 104)
  104     FORMAT(' ')
          KSB2 = ISB2(J)
          KSUB = ISB1(J)
          IF (KSUB .EQ. 0) UVAL = VALS(3, KSB2)
          IF (KSUB .GT. 0) UVAL = U(KSUB)
          IF (IPRINT .EQ. 1) WRITE(6, 101) UVAL, IDOPE(J)
  101     FORMAT('+', T40, ' VALUE:', F12.8, ' TYPE:', I3)
          K2 = ISOL(IDP)
          IF (K2 .EQ. 0 .OR. KSUB .EQ. 0) GO TO 2
          TSIN = ISOL1(1,IDP)
          TCOS = ISOL1(2,IDP)
          IF ( KSB2 .EQ. 0 ) GO TO 6
          TSIN = VALS(2, KSB2)
          TCOS = VALS(1, KSB2)
    6     TEMP = FRND(K2, TX(1), TX(2), TSIN, TCOS)
          ERROR = DABS (TEMP - UVAL)
          EMAX(IDP) = DMAX1(ERROR, EMAX(IDP))
          IF (IPRINT .EQ. 1) WRITE(6, 102) TEMP, ERROR
```

```
 102      FORMAT('+',T70,' TRUE VALUE:',F12.8,' ERROR:',F12.8)
   2    CONTINUE
C
   1  CONTINUE
C
   3  DO 5 I = 1, 3
        IF (ISOL(I) .EQ. 0) GO TO 5
        WRITE(6, 103) I, EMAX(I)
 103    FORMAT(/, ' MAXIMUM ERROR IN PARAMETER OF TYPE ',I1,
     *                 ' IS:', 1PD15.6)
   5  CONTINUE
C
      CALL TIMER('OUTPUT DONE.          ')
C
      RETURN
      END
C*******************************************************************
      SUBROUTINE ANGLE(DZ, SINN, COSN, SINS, COSS)
C-------------------------------------------------------------------
C     DZ IS A COMPLEX NUMBER TREATED AS A VECTOR.
C-------------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMPLEX*16  DZ, TZ, PI/(-1D0,0D0)/, PIB2/(0D0, 1D0)/
      REAL*8 X(2)
      EQUIVALENCE (TZ, X(1))
C-------------------------------------------------------------------
C
C     TANGENTIAL DIRECTION
C
      TZ = DZ
      IF (X(1) .LT. 0 .OR. (X(1) .EQ. 0D0 .AND. X(2) .LT. 0D0))
     *      TZ = TZ * PI
C
      SINS = X(2)
      COSS = X(1)
C
C     NORMAL DIRECTION
C
      TZ = TZ * PIB2
      IF (X(1) .LT. 0.0D0)  TZ = TZ * PI
      SINN = X(2)
      COSN = X(1)
      RETURN
      END
C*******************************************************************
      SUBROUTINE BOUNDY(IDIM, ISUB1, ISUB2, NPND, ISB2, IDOPE,
     *       INDEX, IBASE, ITBL2, ISIDE, ICRNR, VALS, Z)
C-------------------------------------------------------------------
C     THIS ROUTINE USES THE INFORMATION ABOUT THE BOUNDARY
C     OBTAINED FROM PHASE ONE ALONG WITH INPUT INFORMATION IN
C     THE ARRAY IBND TO GENERATE INFORMATION  TO ENABLE THE
C     ROUTINE MODIFY TO MODIFY THE STIFFNESS MATRIX, AND TO
C     ENABLE THE ROUTINE UPDATE TO INCORPORATE BOUNDARY
C     CONDITIONS AS THEY ARE ENCOUNTERED.
C-------------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, NBNDS, IBUG, N,
     *   IPRINT, NCEN, NPS, M, NEQNS, NVALS, NDOPE, IBW, NCP,
     *   LAST, LENL, LENA, KPTS, IEIG, IRHS, ISOL(3), IBND(20,3)
C-------------------------------------------------------------------
      COMPLEX*16 Z(1), DZ, Z1, Z2, TZ
      REAL*8 VALS(3,1), CDABS, SINN, COSN, SINS, COSS, X(2)
      EQUIVALENCE (TZ, X(1))
      INTEGER*4 IDIM
      INTEGER*2 Q(3)/0,1, 1/, IO/0/, ISIDE(4,1), ICRNR(2,1),
     *          ISB2(1), ITBL2(IDIM,1), ISUB2(IDIM,1), ISUB1(1),
     *          INDEX(1), NPND(1), IBASE(1), IDOPE(1)
C-------------------------------------------------------------------
C
      DO 1 I = 1, NSIDE
        ITR = ISIDE(1, I)
        ITR1 = ITR * M
C
        CALL DPTRS(IDIM, ITR, IBASE, INDEX, ISUB1, ISUB2, NPND)
C
        K1 = ISIDE(4, I)
        Z1 = Z(INDEX(K1+ITR1))
        K2 = K1 + 1
        IF (K2.GT.3) K2 = 1
        Z2 = Z(INDEX(K2+ITR1))
        DZ = Z2 - Z1
        DZ = DZ / CDABS(DZ)
C
C       GET NORMAL AND TANGENTIAL DIRECTIONS.
C
        CALL ANGLE(DZ, SINN, COSN,SINS, COSS)
C
C       GET THE BOUNDARY REFERENCE NUMBER .
C
        IBND1 = ISIDE(2, I)
C
        DO 2 K = 1, M
          IF (ITBL2(K, K1) .NE. 1) GO TO 2
C
C         WE HAVE A BOUNDARY NODE - INFO IS IN IBND(IBNDL, *)
C
          TZ = Z(INDEX(K+ITR1))
          NP = NPND(K)
C
          DO 3 L = 1, NP
            KL = ISUB2(K,L)
            IOP = IDOPE(KL)
            IADP = KABS(IOP)
            IFN = IBND(IBND1, IADP)
C
            IF (IFN.EQ.0) GO TO 3
C
C           BOUNDARY VALUE IS SPECIFIED.
C
            GO TO (10, 10, 20), IADP
C
C           NORMAL DERIVATIVE CONDITION
C
 10         CALL BNDRY1(IFN, KL, Q(IADP), IDOPE, ISB2,
     *                  VALS, SINN, COSN, X(1), X(2))
            CALL BNDRY1(IO, KL+Q(IADP), IO, IDOPE, ISB2,
     *                  VALS, SINS, COSS, 0D0, 0D0)
            GO TO 3
C
C           TANGENTIAL DERIVATIVE CONDITION
```

```
C
  20      CALL ANDRYI(IFN, KL, Q(IADP), IDOPE, ISB2,
     *                VALS, SINS, COSS, X(1), X(2))
          CALL BNDRYI(IG, KL+Q(IADP), IO, IDOPE, ISB2,
     *                VALS, SINN, COSN, ODO, OCO)
    3     CONTINUE
C
    2   CONTINUE
C
    1 CONTINUE
C
C     PROCESS ISOLATED-NODE BOUNDARY CONDITIONS ...
C
      IF (NCR .EQ. 0) RETURN
C
      DO 21 I = 1, NCR
C
C       GET THE REFERENCE NUMBER.
C
        IBND1 = ICRNR(1,I)
        IZ = Z(ICRNR(2,I))
        K1 = IBASE(ICRNR(2,I))
        J1 = NPND(I)
        DO 22 J = 1, J1
          KL = K1 + J
          IDP = IDOPE(KL)
          IADP = KABS(IDP)
          IFN = IBND(IBND1, IADP)
          IF (IFN .EQ. 0) GO TO 22
C
C         BOUNDARY VALUE IS SPECIFIED.
C
          GO TO (30, 30, 40), IADP
   30     CALL ANDRYI(IFN, KL, Q(IADP), IDOPE, ISB2,
     *                VALS, COO, IOO, X(1), X(2))
          GO TO 22
   40     CALL ANDRYI(IFN, KL, Q(IADP), IDOPE, ISB2,
     *                VALS, IOC, ODO, X(1), X(2))
   22   CONTINUE
   21 CONTINUE
C
      RETURN
      END
C*********************************************************************
      SUBROUTINE ANDRYI(IFN, KLT, II, IDOPE, ISB2,
     *                  VALS, SN, CS, X, Y)
C--------------------------------------------------------------------
C     AUXILIARY ROUTINE USED BY ANDRY.
C--------------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, NBNDS, IBUG, N,
     *   IPRINT, NCEN, NPS, M, NEQNS, NVALS, NDOPE, IBW, NCR,
     *   LAST, LEML, LENA, KPTS, IEIG, IRHS, ISOL(3), IBND(20,3)
C--------------------------------------------------------------------
      INTEGER*2 IDOPE(1), ISB2(1)
      REAL*8 VALS(3,1)
C--------------------------------------------------------------------
C
      KL = KLT
C
C     CHECK TO SEE IF PARAMETER IS ALREADY SPECIFIED.
C
      IF (IDOPE(KL) .LT. 0) GO TO 10
C
C     IT IS, SO CHECK TO SEE IF IT IS A DERIVATIVE PARAMETER.
C     IF IT IS, MAYBE THERE IS ANOTHER TO ASSIGN VALUE TO.
C
      IF (II .EQ. 0) RETURN
C
      K = ISB2(KL)
      KL = KL + II
      IF (IDOPE(KL) .GT. 0  .OR.
     *    DABS(VALS(1,K) - CS) .LT. 0.1D0) RETURN
C
C     CHECK TO SEE IF PARAMETER IS ALSO SPECIFIED.
C
   10 IF (ISB2(KL) .GT. 0) GO TO 20
      NVALS = NVALS + 1
      ISB2(KL) = NVALS
C
C     STORE DIRECTION ASSOCIATED WITH THE KL-TH PARAMETER.
C
   20 K = ISB2(KL)
      VALS(1,K) = CS
      VALS(2,K) = SN
C
      IF (IFN .EQ. 0) RETURN
C
C     MARK PARAMETER AS KNOWN, AND STORE ITS VALUE IN VALS(3,K).
C
      IDOPE(KL) = - IDOPE(KL)
      VALS(3,K) = FBND(IFN, X, Y, SN, CS)
      RETURN
      END
C*********************************************************************
      SUBROUTINE TABLES(IDIM, IBASE, IP, IDOPE, ISB1, ISB2,
     * NPND, ISUB1, ISUB2, INDEX, MU, IBETA, MAP, VALS)
C--------------------------------------------------------------------
C     THIS ROUTINE GENERATES THE ARRAYS ISB1, MU, IBETA
C     AND MAP.  FOR DETAILS SEE THE COMMENTS IN THE MAINLINE
C     AND CHAPTER 4, SECTION 2.
C--------------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, NBNDS, IBUG, N,
     *   IPRINT, NCEN, NPS, M, NEQNS, NVALS, NDOPE, IBW, NCR,
     *   LAST, LEML, LENA, KPTS, IEIG, IRHS, ISOL(3), IBND(20,3)
C--------------------------------------------------------------------
      INTEGER*4 IDIM
      INTEGER*2 IBASE(1), IP(1), IDOPE(1), ISB1(1), ISB2(1),
     *          NPND(1), ISUB1(1), ISUB2(IDIM,1), INDEX(1),
     *          MU(1), IBETA(1), MAP(1)
      REAL*8 VALS(3,1)
C--------------------------------------------------------------------
C
C     GENERATE SUBSCRIPTS...            (ISB1)
C
      KSUB = 0
      DO 4 I = 1, KPTS
        L = IBASE(IP(I)) + 1
        L1 = NDOPE
```

```
      IF (IP(I+1) .NE. 0) L1 = IBASE(IP(I+1))
      DO 5 J = L, L1
        IF (IOPER(J) .GT. 0) GO TO 5
        KSUB = KSUB + 1
        ISB1(J) = KSUB
    5   CONTINUE
    4 CONTINUE
C
C     FIND MAXIMUM BANDWIDTH ...         (IBW)
C
      NEQNS = KSUB
      IBW = 0
      DO 6 I = 1, NTRSM1
      CALL OPTRS(IDIM, I, IBASE, INDEX, ISUB1, ISUB2, NPND)
        DO 7 J = 1, N
          K1 = ISB1(ISUB1(J))
          DO 8 K = 1, N
            K2 = ISB1(ISUB1(K))
            IF (K1.EQ.0.OR.K2.EQ.0.OR.K2.GT.K1) GO TO 8
            K3 = K1 - K2
            IF (K3 .GT. IBW) IBW = K3
            IF (MU(K1) .LT. K3) MU(K1) = K3
    8     CONTINUE
    7   CONTINUE
    6 CONTINUE
C
      WRITE(6, 101) IBW
  101 FORMAT(' BANDWIDTH:', I5/)
C
C     GENERATE MU.
C
      L = 0
      DO 12 I = 1, NEQNS
        L = L + MU(I) + 1
        MU(I) = L
   12 CONTINUE
C
      WRITE(6, 104) L
  104 FORMAT(' STORAGE REQUIRED FOR L :', I7, ' WORDS'/)
C
C     GENERATE THE MAPPING FUNCTION (MAP) TO ENABLE US TO
C     STORE ONLY THE NON-ZERO ELEMENTS OF THE COEFFICIENT
C     MATRIX.
C
      DO 14 I = 1, NTRSM1
        CALL OPTRS(IDIM, I, IBASE, INDEX, ISUB1, ISUB2, NPND)
        DO 15 J = 1, N
          DO 16 K = 1, N
            K1 = ISB1(ISUB1(J))
            K2 = ISB1(ISUB1(K))
            IF (K1.EQ.0.OR.K2.EQ.0.OR.X2.GT.K1) GO TO 16
            MAP(MU(K1) - K1 + K2) = 1
   16     CONTINUE
   15   CONTINUE
   14 CONTINUE
C
      MAP(1)=0
      IBETA(1) = 1
      K3 = 1
      DO 18 I = 2, NEQNS
        K1 = MU(I-1) + 1
        K2 = MU(I)
        DO 19 J = K1, K2
          IF (MAP(J) .EQ. 0) GO TO 19
          K3 = K3 + 1
          MAP(K3) = K2 - J
   19   CONTINUE
        IBETA(I) = K3
   18 CONTINUE
C
      LENA = IBETA(NEQNS)
      LENL = MU(NEQNS)
C
      RETURN
      END
C************************************************************************
      SUBROUTINE OPTRS(IDIM, ITR, IBASE, INDEX,
     *                 ISUB1, ISUB2, NPND)
C-----------------------------------------------------------------------
C     THIS ROUTINE FILLS ISUB1 AND ISUB2 WITH POINTERS INTO
C     THE ARRAY IDOPE(CORRESPONDING TO TRIANGLE # ITR) .
C-----------------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, NBNDS, IBUG, N
     *   IPRINT, NCEN, NPS, M, NEQNS, NVALS, NDOPE, IBW, NCR,
     *   LAST, LENL, LENA, KPTS, IEIG, IRHS, ISOL(3), IBND(20,
C-----------------------------------------------------------------------
      INTEGER*4 IDIM
      INTEGER*2 IBASE(1), INDEX(1), ISUB1(1),
     *          ISUB2(IDIM,1), NPND(1)
C-----------------------------------------------------------------------
C
      ITR1 = M * ITR
      L = 0
      DO 1 I = 1, M
        K = IBASE(INDEX(I+ITR1))
        J1 = NPND(I)
        DO 2 J = 1, J1
          L = L + 1
          ISUB1(L) = K + J
          ISUB2(I, J) = ISUB1(L)
    2   CONTINUE
    1 CONTINUE
C
      RETURN
      END
C************************************************************************
      SUBROUTINE GENDOP(IDIM, INDEX, IBASE, NPND, ITBL1,
     *                  ITBL2, IDOPE, ISIDE, ICRNR, IP, Z)
C-----------------------------------------------------------------------
C     THIS ROUTINE READS THE OUTPUT FROM PHASE1 AND THE
C     PARAMETERS OF PHASE2. IT ALSO GENERATES THE VECTORS
C     IDOPE, IBASE, AND THE ARRAY ITBL2.
C-----------------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, NBNDS, IBUG, N,
     *   IPRINT, NCEN, NPS, M, NEQNS, NVALS, NDOPE, IBW, NCR,
     *   LAST, LENL, LENA, KPTS, IEIG, IRHS, ISOL(3), IBND(20,
C-----------------------------------------------------------------------
      NAMELIST/PARMS1/ NPTS, NTRS. NSIDE, NPS, NCEN, M, N, IRHS
```

```
C-----------------------------------------------------------------
      INTEGER*4 IDIM
      INTEGER*2 IBASE(1), INDEX(1), NPND(1), ITBL1(IDIM,1),
     *   ITBL2(IDIM,1), IDOPE(1), ISIDE(4,1), ICRNR(2,1), IP(1)
      COMPLEX*16 Z(1)
C-----------------------------------------------------------------
C
      READ (1) (Z(I), I=1, NPTS)
      READ (1) I, (INDEX(K), K = 1, I)
      READ (1) ((ISIDE(K, I), K = 1, 4), I = 1, NSIDE)
      READ (1) NCR, ((ICRNR(J,I), J = 1, 2), I = 1, NCR)
      READ (1) (IP(I), I = 1, NPTS)
      READ (2) (NPND(K), K = 1, M),
     *         ((ITBL1(I,J), J=1,3), I = 1, M)
      NTRSM1 = NTRS - 1
      NVALS = 0
      WRITE(6,PARMS1)
C
      IF (IBUG .EQ. 0) GO TO 17
C
      CALL PAGE('DEBUG INFORMATION...')
      WRITE(6, 99)
   99 FORMAT(T35, ' POINT NUMBERS AND THEIR COORDINATES:'/)
      WRITE(6, 101) ((I, Z(I)), I = 1, NPTS)
  101 FORMAT(1X, I4, 2F12.8, I4, 2F12.8, I4, 2F12.8, I4, 2F12.8)
      I = M * (NTRS + 1)
      WRITE(6, 102) I, (INDEX(K), K = 1, I)
  102 FORMAT(1X, 30I4)
      WRITE(6, 102) ((ISIDE(K, I), K = 1, 4), I = 1, NSIDE)
      WRITE(6, 102) (IP(I), I = 1, NPTS)
C
      DO 4 I = 1, M
        K = NPND(I)
        WRITE(6, 104) I, K, (ITBL1(I,J), J = 1, K)
  104   FORMAT(' =>', 7I4)
    4 CONTINUE
C
C     GENERATE IBASE.
C
   17 NDOPE = 0
      DO 2 I = 1, NTRSM1
        L = I * M
        DO 3 J = 1, M
          IBASE(INDEX(L+J)) = J
    3   CONTINUE
    2 CONTINUE
C
C     GENERATE IDOPE.
C
      DO 13 I = 1, KPTS
        K = IP(I)
        K2 = IBASE(K)
        K1 = NPND(K2)
        IBASE(K) = NDOPE
        DO 15 J = 1, K1
          NDOPE = NDOPE + 1
          IDOPE(NDOPE) = -ITBL1(K2,J)
   15   CONTINUE
   13 CONTINUE
C
   14 CONTINUE
      NEQNS = NDOPE
C
C     GENERATE ITBL2.  SEE COMMENTS IN MAINLINE FOR DETAILS.
C
      ITBL2(1,1) = 1
      ITBL2(1,3) = 1
      ITBL2(2,1) = 1
      ITBL2(2,2) = 1
      ITBL2(3,2) = 1
      ITBL2(3,3) = 1
      IF (NPS .EQ. 0) RETURN
      DO 9 I = 1, 3
        DO 10 J = 1, NPS
          ITBL2(3 + J + (I-1)*NPS, I) = 1
   10   CONTINUE
    9 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE TIMER (HD)
C-----------------------------------------------------------------
C     PRINT THE TOTAL ACCUMULATED TIME AND THE TIME SINCE THIS
C     ROUTINE WAS LAST CALLED.
C-----------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /TIME/ TSTRT, TOLD, TNEW
C-----------------------------------------------------------------
      REAL*4 HD(5)
      INTEGER*4 TIMER1
C-----------------------------------------------------------------
C
      TOLD = TNEW
      TNEW = TIMER1(1)
      SEC1 = (TNEW - TOLD) * 0.001
      SEC2 = (TNEW - TSTRT) * 0.001
C
      WRITE(6, 100) HD, SEC1, SEC2
  100 FORMAT(/, 1X, 5A4, '  TIME USED:', F8.3, ' SECONDS.',
     *  '           ACCUMULATED TIME:', F8.3, ' SECONDS'/)
      RETURN
      END
C*****************************************************************
      SUBROUTINE TRANS2(IDIM, K1, K2, N, B11, B12, B21, B22, A)
C-----------------------------------------------------------------
C     THIS ROUTINE APPLIES THE TRANSFORMATION B (2 X 2) TO
C     THE COLUMNS AND ROWS K1 AND K2 .
C-----------------------------------------------------------------
      INTEGER*4 IDIM
      INTEGER*2 K1, K2, N
      REAL*8 B11, B12, B21, B22, A(IDIM,IDIM), T1, T2
C-----------------------------------------------------------------
C
      DO 1 K = 1, N
         T1 = A(K,K1)
         T2 = A(K,K2)
         A(K,K1) = B11 * T1 + B21 * T2
         A(K,K2) = B12 * T1 + B22 * T2
    1 CONTINUE
```

```
C
      DO 2 K = 1, N
         T1 = A(K1,K)
         T2 = A(K2,K)
         A(K1,K) = B11 * T1 + B21 * T2
         A(K2,K) = B12 * T1 + B22 * T2
    2 CONTINUE
C
      RETURN
      END
C*******************************************************************
      SUBROUTINE PAGE(HD)
C-------------------------------------------------------------------
C     SKIP TO A NEW PAGE AND PRINT A HEADING.
C-------------------------------------------------------------------
      REAL*4 HD(5)
C-------------------------------------------------------------------
      WRITE(6, 100)
  100 FORMAT('1FINITE ELEMENT SOLUTION OF PDE-S : PHASE THREE ',
     *       ' (ASSEMBLY OF EQUATIONS AND SOLUTION) *******'/)
C
      WRITE(6, 101) HD
  101 FORMAT(T20, 5A4/)
C
      RETURN
      END
C*******************************************************************
      INTEGER FUNCTION KABS*2 (IARG)
C-------------------------------------------------------------------
C     SHORT INTEGER VERSION OF IABS FUNCTION.
C-------------------------------------------------------------------
      INTEGER*2 IARG
C-------------------------------------------------------------------
      KABS = IARG
      IF (IARG .LT. 0) KABS = -IARG
      RETURN
      END
C*******************************************************************
      SUBROUTINE UPDATE (IDIM, ISUB1, ISB1, ISB2, IDOPE,
     *                   IBETA, MAP, AL, BL, A, B, RHS, RHS1, VALS)
C-------------------------------------------------------------------
C     UPDATE THE OVERALL STIFFNESS (MASS) MATRIX FROM A(B).
C     DITTO FOR THE LOAD VECTOR.
C-------------------------------------------------------------------
      IMPLICIT INTEGER*2 (I-N), REAL*8 (A-H,O-Y), COMPLEX*16 (Z)
      COMMON /FIXED/ NPTS, NTRS, NTRSM1, NSIDE, NPNDS, IBUG, N,
     *   IPRINT, NCFN, NPS, M, NEQNS, NVALS, NDOPE, IBW, NCR,
     *   LAST, LENL, LENA, KPTS, IEIG, IRHS, ISOL(3), IBND(20,3)
C-------------------------------------------------------------------
      INTEGER*4 IDIM
      INTEGER*2 ISUB1(1), ISB1(1), ISB2(1),
     *          IDOPE(1), IBETA(1), MAP(1)
      REAL*8 AL(1), BL(1), A(IDIM,1), B(IDIM,1),
     *       RHS(1), RHS1(1), VALS(3,1)
C-------------------------------------------------------------------
C
      DO 1 I = 1, N
        I1 = ISUB1(I)
C
C       CHECK IF PARAMETER IS SPECIFIED BY BOUNDARY CONDITIONS
C
        IF (IDOPE(I1) .GT. 0) GO TO 1
        IS = ISB1(I1)
        IF (IRHS .GT. 0) RHS(IS) = RHS(IS) - RHS1(I)
C
        DO 2 J = 1, N
          J1 = ISUB1(J)
          IF (IDOPE(J1) .GT. 0) GO TO 3
          JS = ISB1(J1)
          IF (IS .LT. JS) GO TO 2
          KT = IS - JS
          K1 = 1
          IF (IS .NE. 1) K1 = IBETA(IS-1) + 1
          K2 = IBETA(IS)
          DO 5 K = K1, K2
            IF (MAP(K) .NE. KT) GO TO 5
            AL(K) = AL(K) + A(I,J)
            IF (IEIG .GT. 0) BL(K) = BL(K) + B(I,J)
            GO TO 2
    5     CONTINUE
C
    3     IF(IEIG .EQ. 0)
     *       RHS(IS) = RHS(IS) - A(I, J) * VALS(3, ISB2(J1))
    2   CONTINUE
    1 CONTINUE
C
      RETURN
      END
C*******************************************************************
      SUBROUTINE UNPACK(N, MU, IBETA, MAP, AL, ABIG)
C-------------------------------------------------------------------
C     UNPACK THE MATRIX AL, WHICH IS STORED IN COMPACT FORM,
C     AND PLACE IT IN ABIG IN THE FORMAT EXPECTED BY
C     THE ROUTINES LUSPRS AND SPRSLV.
C-------------------------------------------------------------------
      REAL*8 AL(1), TEMP, ABIG(1)
      INTEGER*2 MU(1), IBETA(1), MAP(1), N
C-------------------------------------------------------------------
C
      ABIG(1) = AL(1)
      DO 2 I = 2, N
        K1 = IBETA(I-1) + 1
        K2 = IBETA(I)
        K = MU(I)
        DO 3 J = K1, K2
          ABIG(K - MAP(J)) = AL(J)
    3   CONTINUE
    2 CONTINUE
C
      RETURN
      END
C*******************************************************************
      SUBROUTINE SOLVR1(N, IBW, MU, IBETA, MAP, AL,
     *                  UL, RHS, R, X, H)
C-------------------------------------------------------------------
C     DRIVER PROGRAM FOR SOLUTION OF THE LINEAR EQUATIONS.
C     ITERATIVE REFINEMENT IS PERFORMED AS WELL.
C-------------------------------------------------------------------
      INTEGER*2 N, MU(1), IBETA(1), MAP(1), IBW
      REAL*8 UL(1), AL(1), RHS(1), R(1), X(1), H(1), XNRM, HNRM
```

```
C-----------------------------------------------------------------
      ITER=1
      CALL UNPACK(N, MU, IBETA, MAP, AL, UL)
      CALL TIMER('UNPACK DONE.           ')
C
      CALL LUSPRS(N, IBW, MU, UL, UL)
      CALL TIMER('DECOMP DONE.           ')
C
      CALL SPRSLV(N, MU, UL, X, RHS)
      CALL TIMER('SOLVE  DONE.           ')
C
      XNRM = 0.0D0
      DO 1 I = 1, N
        XNRM = DMAX1(XNRM, DABS(X(I)))
    1 CONTINUE
C
   10 CALL RESID(N, IBETA, MAP, AL, RHS, X, R)
      CALL SPRSLV(N, MU, UL, H, R)
C
      HNRM = 0.0D0
      DO 2 I = 1, N
        HNRM = DMAX1(HNRM, DABS(H(I)))
        X(I) = X(I) + H(I)
    2 CONTINUE
C
      ITER = ITER + 1
      WRITE(6, 100) ITER, HNRM
  100 FORMAT(' ITERATION',I3,' MAX NORM OF CHANGE:', E14.6/)
C
      IF (HNRM / XNRM .GE. 1D-10 .AND. ITER .LE. 4) GO TO 10
      CALL TIMER('REFINE DONE.           ')
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE RESID(N, IBETA, MAP, AL, RHS, X, R)
C-----------------------------------------------------------------
C     COMPUTE THE RESIDUAL VECTOR OF THE COMPACTLY STORED
C     LINEAR SYSTEM (IN DOUBLE-DOUBLE PRECISION).
C-----------------------------------------------------------------
      REAL*8 AL(1), RHS(1), X(1), R(1), T1
      INTEGER*2 N, IBETA(1), MAP(1), I1
      INTEGER IEXP
C-----------------------------------------------------------------
C
      IEXP = 0
      R(1) = 0D0
      R(2) = RHS(1)
      CALL VPR2(-X(1), AL(1), R(2), IEXP)
      DO 2 I = 2, N
        K1 = IBETA(I-1) + 1
        K2 = IBETA(I) - 1
        R(I*2) = RHS(I)
        R(I*2+1) = 0D0
        CALL VPR2(-X(I), AL(K2+1), R(I*2), IEXP)
        IF (K1 .GT. K2) GO TO 2
        DO 3 J = K1, K2
          I1 = I - MAP(J)
          CALL VPR2(-X(I1), AL(J), R(2*I), IEXP)
          CALL VPR2(-X(I), AL(J), R(2*I1), IEXP)
    3   CONTINUE
    2 CONTINUE
C
      DO 4 I = 1, N
        R(I) = R(I*2)
    4 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE LUSPRS(N, IBW, MU, A, UL)
C-----------------------------------------------------------------
C THIS SUBROUTINE DECOMPOSES A POSITIVE DEFINITE SYMMETRIC MATRI
C INTO IT'S CHOLESKY DECOMPOSITION. THE LOWER HALF OF THE MATRIX
C (INCLUDING THE DIAGONAL) IS STORED ROW BY ROW IN THE
C VECTOR A. THE POSITION OF THE DIAGONAL ELEMENT OF THE I-TH
C ROW IS FOUND IN MU(I).  THE  LOWER TRIANGULAR FACTOR OF THE
C DECOMPOSITION IS STORED IN THE SAME MANNER IN THE ONE
C DIMENSIONAL VECTOR UL.
C IBW SHOULD BE SET TO MAX ( MU(I) - MU(I-1))
C                         1 < I < N+1
C   IT SHOULD AT LEAST BE AS LARGE AS THAT VALUE, AND AT THE COS
C OF SOME LOSS IN EFFICIENCY, CAN BE SET TO N.
C-----------------------------------------------------------------
      REAL*8 UL(1), A(1), T1, T2, DSQRT
      INTEGER*2 MU(1), N, IBW
C-----------------------------------------------------------------
C
      OPC = 0
      DO 1 J = 1, N
        JD = MU(J)
        T1 = A(JD)
C
C       COMPUTE THE J-TH DIAGONAL ELEMENT OF UL .....
C
        J1 = 1
        IF (J .NE. 1 ) J1 = MU(J-1) + 1
        IF (J1 .EQ. JD) GO TO 10
        JDM1 = JD - 1
        OPC = OPC + JD - J1
        DO 2 I = J1, JDM1
          T2 = UL(I)
          T1 = T1 - T2*T2
    2   CONTINUE
C
   10   T1 = DSQRT(T1)
        UL(JD) = T1
        IF (J .EQ. N) GO TO 1
C
C       COMPUTE THE J-TH COLUMN OF UL .....
C
        JP1 = J + 1
        JDEL = JD - J1
        JPB = JP1 + IBW
        IF (JPB .GT. N) JPB = N
C
        DO 3 I = JP1, JPB
          IDEL = MU(I) - MU(I-1)
          IF (IDEL .LE. (I-J)) GO TO 3
          IDEL = IDEL - I + J - 1
```

```
          IF (JDEL .LT. IDEL) IDEL = JDEL
          KT = MU(I) - I + J
          T2 = A(KT)
          IF (IDEL .EQ. 0) GO TO 20
C
          DO 4 K = 1, IDEL
            T2 = T2 - UL(KT-K) * UL(JD-K)
    4     CONTINUE
C
   20     UL(KT) = T2 / T1
          OPC = OPC + IDEL + 1
C
    3   CONTINUE
    1 CONTINUE
C
      WRITE(6, 100) N, OPC
  100 FORMAT(/,' LUSPRS:', /, '      NUMBER OF EQUATIONS:', I4/,
     *  '      NUMBER OF OPERATIONS REQUIRED(*,/):', F12.1/)
C
      RETURN
      END
C*********************************************************************
      SUBROUTINE SPRSLV(N, MU, UL, X, B)
C---------------------------------------------------------------------
C     THIS ROUTINE SOLVES UL * UL(T) X = B, WHERE UL IS A LOWER
C     TRIANGULAR MATRIX STORED AS DESCRIBED IN LUSPRS.
C---------------------------------------------------------------------
      REAL*8 X(1), B(1), T1, UL(1)
      INTEGER*2  MU(1), N
C---------------------------------------------------------------------
C
C     SOLVE UL * X = B ....
C
      OPC = 1
      X(1) = B(1) / UL(1)
      DO 1 I = 2, N
        ID = MU(I)
        K1 = ID - MU(I-1) - 1
        T1 = B(I)
        IF (K1 .EQ. 0) GO TO 3
        OPC = OPC + K1+ 1
C
        DO 2 J = 1, K1
          T1 = T1 - X(I-J) * UL(ID-J)
    2   CONTINUE
C
    3   X(I) = T1 / UL(ID)
    1 CONTINUE
C
C     SOLVE UL(T) * X = X ....
C
      DO 4 K = 2, N
        I = N - K + 2
        ID = MU(I)
        K1 = ID - MU(I-1) - 1
        T1 = X(I) / UL(ID)
        X(I) = T1
        IF ( K1 .LE. 0 ) GO TO 4
C
        DO 5 J = 1, K1
          X(I-J) = X(I-J) - T1 * UL(ID-J)
    5   CONTINUE
C
    4 CONTINUE
      X(1) = X(1) / UL(1)
C
      OPC = 2 * OPC
      WRITE(6, 100) OPC
  100 FORMAT(' OPERATION COUNT (*,/) IN SPRSLV:', F12.1/)
C
      RETURN
      END
```

Inverse Iteration Using a

Band Linear Equation Solver

```
C***********************************************************************
C      DRIVER PROGRAM FOR INVERSE ITERATION.  THIS ROUTINE
C      READS THE PARAMETERS OF THE GENERALIZED EIGENVALUE
C      PROBLEM, ALLOCATES THE STORAGE AND THE CALLS INVERS.
C-----------------------------------------------------------------------
       REAL*8 A(32000)
       INTEGER*2 N, M, LENGTH, IDSET, LAST
       INTEGER TIMER1, LB, LBETA, LMAP, LU, LL, LV1, LV2, IDIM
C-----------------------------------------------------------------------
       TSTRT = TIMER1(0)
       REWIND 2
       LAST = 0
C
   30  IF (LAST .EQ. 1) STOP
C
C      READ PROBLEM PARAMETERS FROM UNIT TWO.
C
       READ(2, END=2) N, M, LENGTH, LAST
       WRITE(6, 100) N, M, LENGTH
  100  FORMAT('1 N=', I3, ' M=', I3, ' LENGTH=', I6/)
C
C      COMPUTE SOME CONSTANTS FOR STORAGE ALLOCATION ....
C
       IDIM = N
       LB = LENGTH + 1
       LBETA = LB + LENGTH
       LU = LBETA + N / 4 + 1
       LL = LU + N * (2 * M + 1)
       LV1 = LL + N * M
       LV2 = LV1 + N
       LPS = LV2 + N
       LMAP = LPS + N / 4 + 1
       KK = LMAP + LENGTH / 4 + 1
       WRITE(6, 105) KK
  105  FORMAT(' NUMBER OF LONG WORDS OF STORAGE REQUIRED:', I7/)
       IF (KK .GT. 32000) STOP
C
C      ZERO OUT THE MEMORY WE ARE GOING TO USE ...
C
       DO 1 I = LU, KK
         A(I) = 0.0D0
    1  CONTINUE
C
       CALL INVRS(IDIM, N, M, LENGTH, A, A(LB), A(LU),
     *  A(LL), A(LV1), A(LV2), A(LMAP), A(LBETA), A(LPS))
C
       GO TO 30
C
    2  STOP
C
       END
C***********************************************************************
       SUBROUTINE INVRS(IDIM, N, M, LENGTH, A, B, UU, UL,
     *                  V1, V2, MAP, IBETA, IPS)
C-----------------------------------------------------------------------
C      INVERSE ITERATION ROUTINE.
C-----------------------------------------------------------------------
       INTEGER IDIM, TIMER1
       INTEGER*2 N, M, LENGTH, IBETA(1), IPS(1), MAP(1)
       REAL*8 A(1), B(1), UU(1), UL(1), V1(1), V2(1), T,
     *        XAX, XBX, IP2, SHIFT, OLDT, DIFF
C-----------------------------------------------------------------------
       NAMELIST /PARMS/ SHIFT
C-----------------------------------------------------------------------
C
C      READ THE PROBLEM TO BE SOLVED.
C
       READ(2) (IBETA(K), K = 1, N)
       READ(2) (MAP(K), A(K), B(K), K = 1, LENGTH)
C
   10  READ(5, PARMS)
       WRITE(6, 104) N, M
  104  FORMAT('1 INVERSE ITERATION ..N=', I3, ' M=', I3/)
       WRITE(6, PARMS)
       T = SHIFT
       IF (SHIFT .LE. 0.0) RETURN
C
       V1(N) = 1.0D0
       V2(N) = 1.0D0
       T1 = TIMER1(1)
C
       CALL UNPACK(IDIM, N, M, IBETA, MAP, A,
     *             B, SHIFT, UU)
C
       CALL BNDCMP(IDIM, N, M, UU, UL, IPS)
C
       ITER = 0
   20  ITER = ITER + 1
       WRITE(6, 102) ITER
  102  FORMAT(/, ' ITERATION #', I3)
C
       CALL BNDSOL(IDIM, N, M, UU, UL, V1, IPS)
C
       CALL SCALE(N, V1)
C
       CALL IPSPRS(N, IBETA, MAP, A, V1, V2)
C
       XAX = IP2(N, V1, V2)
C
       CALL IPSPRS(N, IBETA, MAP, B, V1, V2)
C
       XBX = IP2(N, V1, V2)
C
       OLDT = T
       T = XAX / XBX
       DIFF = DABS (OLDT - T)
       WRITE(6, 101) T, DIFF
  101  FORMAT(' EIGENVALUE=', F20.14, ' CHANGE=', F14.10)
C
       IF (DIFF .LT. 1D-10 .OR. ITER .GT. 10) GO TO 30
C
       DO 2 I = 1, N
         V1(I) = V2(I)
    2  CONTINUE
       GO TO 20
C
   30  T1 = (TIMER1(1) - T1) / 1000.0
       WRITE(6, 123) T1
  123  FORMAT(' ITERATION DONE, TIME USED=', F8.4, ' SEC.')
       GO TO 10
```

```
      END
C***********************************************************************
      SUBROUTINE BNDCMP(IDIM, N, M, A, UL, IPS)
C-----------------------------------------------------------------------
C     L U DECOMPOSITION OF AN UNSYMMETRIC BAND MATRIX A
C     HAVING BANDWIDTH M = MAX |I-J| FOR A(I,J) .NE. 0.
C     THE DIAGONALS OF THE MATRIX ARE ASSUMED TO BE
C     STORED IN THE COLUMNS OF A AS DESCRIBED IN SECTION
C     2 OF CHAPTER 4 (METHOD 4). A IS REPLACED BY THE
C     UPPER TRIANGLE U, AND L IS STORED IN UL, WITH THE
C     I-TH COLUMN PLACED IN UL(I,*).
C-----------------------------------------------------------------------
      INTEGER*2 N, M, IPS(1)
      INTEGER*4 IDIM, TIMER1
      REAL*8 A(IDIM,1), UL(IDIM,1), NORM, X
C-----------------------------------------------------------------------
      T1 = TIMER1(1)
      M1 = M + 1
      M2 = M + 2
      MM1 = M + M + 1
      NORM = 0.0
      DO 1 I = 1, N
        X = 0.0
        DO 2 J = 1, MM1
          X = X + DABS(A(I,J))
    2   CONTINUE
        IF (X .GT. NORM) NORM = X
    1 CONTINUE
C
      L = M
      DO 3 I = 1, M
        K = M2 - I
        DO 4 J = K, MM1
          A(I, J-L) = A(I,J)
    4   CONTINUE
        L = L - 1
        K = MM1 - L
        DO 5 J = K, MM1
          A(I,J) = 0.0D0
    5   CONTINUE
    3 CONTINUE
C
C     END INITIALIZATION
C
      L = M
      DO 6 K = 1, N
        X = DABS(A(K,1))
        I = K
        IF (L .LT. N) L = L + 1
        KP1 = K + 1
        IF (KP1 .GT. L) GO TO 12
C       SEARCH FOR INTERCHANGE ...
        DO 7 J = KP1, L
          IF (DABS(A(J,1)) .LE. X) GO TO 7
          X = DABS(A(J,1))
          I = J
    7   CONTINUE
   12   IPS(K) = I
        IF (X .EQ. 0.0) A(K,1) = NORM * 1E-6
        IF (I .EQ. K) GO TO 8
C       INTERCHANGE ROWS I AND K ...
        DO 9 J = 1, MM1
          X = A(K,J)
          A(K,J) = A(I,J)
          A(I,J) = X
    9   CONTINUE
C
    8   IF (KP1 .GT. L) GO TO 6
        DO 10 I = KP1, L
          X = A(I,1) / A(K,1)
          UL(K,I-K) = X
          DO 11 J = 2, MM1
            A(I,J-1) = A(I,J) - X * A(K,J)
   11     CONTINUE
          A(I,MM1) = 0.0
   10   CONTINUE
    6 CONTINUE
C
      T1 = (TIMER1(1) - T1) / 1000.0
      WRITE(6, 123) T1
  123 FORMAT(' DECOMP DONE, TIME USED=', F8.4, ' SECONDS.')
      RETURN
      END
C***********************************************************************
      SUBROUTINE BNDSOL(IDIM, N, M, A, UL, B, IPS)
C-----------------------------------------------------------------------
C     SOLVE UL.A.X = B AND PLACE THE RESULT IN B.
C     WE USE THE DECOMPOSITION PRODUCED BY BNDCMP. THE
C     LOWER TRIANGULAR FACTOR IS FOUND IN UL AND THE
C     UPPER TRIANGULAR FACTOR IS IN A.
C-----------------------------------------------------------------------
      INTEGER IDIM, TIMER1
      INTEGER*2 N, M, IPS(1)
      REAL*8 A(IDIM,1), UL(IDIM,1), B(1), X
C-----------------------------------------------------------------------
      T1 = TIMER1(1)
C
C     SOLVE UL . X = B AND PLACE THE RESULT IN B.
C
      MM1 = M + M + 1
      L = M
      DO 1 K = 1, N
        I = IPS(K)
        IF (I .EQ. K) GO TO 2
        X = B(K)
        B(K) = B(I)
        B(I) = X
    2   IF (L .LT. N) L = L + 1
        KP1 = K + 1
        IF (KP1 .GT. L) GO TO 1
        DO 3 I = KP1, L
          B(I) = B(I) - UL(K,I-K) * B(K)
    3   CONTINUE
    1 CONTINUE
C
C     SOLVE A . X = B AND PLACE THE RESULT IN B.
C
      L = 1
      DO 4 J = 1, N
        I = N + 1 - J
```

```
      IM1 = I - 1
      X = B(I)
      IF (I .EQ. N) GO TO 5
      IF (L .LT. MM1) L = L + 1
      DO 6 K = 2, L
        X = X - A(I,K) * B(IM1+K)
    6 CONTINUE
    5 B(I) = X / A(I,1)
    4 CONTINUE
C
      T1 = (TIMER1(1) - T1) / 1000.0
      WRITE(6, 123) T1
  123 FORMAT(' SOLVE  DONE, TIME USED=', F8.4, ' SECONDS.')
      RETURN
      END
C*********************************************************************
      SUBROUTINE SCALE ( N, A)
C---------------------------------------------------------------------
C     DIVIDE THE VECTOR A BY ITS ELEMENT OF MAXIMUM MODULUS.
C---------------------------------------------------------------------
      INTEGER*2  N
      REAL*8 A(1), DABS, FMAX
C---------------------------------------------------------------------
C
      FMAX = 0.0D0
      DO 1 I = 1, N
        IF ( FMAX .LT. DABS(A(I))) FMAX = DABS(A(I))
    1 CONTINUE
C
      DO 2 I = 1, N
        A(I) = A(I) / FMAX
    2 CONTINUE
C
      RETURN
      END
C*********************************************************************
      SUBROUTINE UNPACK(IDIM, N, M, IBETA, MAP, A, B, LAMDA, A1)
C---------------------------------------------------------------------
C     UNPACK A AND B (OF THE A X = LAMDA B X PROBLEM) AND
C     PLACE A - LAMDA * B IN THE ARRAY A1 IN THE FORMAT
C     ACCEPTABLE TO BNDCMP AND BNDSOL.
C---------------------------------------------------------------------
      INTEGER*2 N, M, IBETA(1), MAP(1)
      REAL*8 A1(IDIM,1), A(1), B(1), LAMDA, T
C---------------------------------------------------------------------
      MP1 = M + 1
      DO 1 I = 1, N
        K1 = 1
        IF (I .GT. 1) K1 = IBETA(I-1) + 1
        K2 = IBETA(I)
        DO 2 J = K1, K2
          K3 = MAP(J)
          T = A(J) - LAMDA * B(J)
          A1(I,MP1-K3) = T
          A1(I-K3,MP1+K3) = T
    2   CONTINUE
    1 CONTINUE
C
      RETURN
      END
C*********************************************************************
      SUBROUTINE IPSPR(N, IBETA, MAP, A, X, R)
C---------------------------------------------------------------------
C     THIS ROUTINE MULTIPLIES A VECTOR X BY THE MATRIX A
C     STORED IN COMPACT FORM AS DESCRIBED IN SEC. 4.2 (METHOD 1)
C     AND PLACES THE RESULT IN THE ARRAY R.
C---------------------------------------------------------------------
      REAL*8 A(1), X(1), R(1), T1
      INTEGER*2 N, IBETA(1), MAP(1)
C---------------------------------------------------------------------
C
      R(1) = X(1) * A(1)
      DO 2 I = 2, N
        K1 = IBETA(I-1) + 1
        K2 = IBETA(I)  - 1
        R(I) =  X(I) * A(K2+1)
        IF ( K1 .GT. K2 ) GO TO 2
        DO 3 J = K1, K2
          I1 = I - MAP(J)
          R(I) = R(I) + X(I1) * A(J)
          R(I1) = R(I1) + X(I) * A(J)
    3   CONTINUE
    2 CONTINUE
C
      RETURN
      END
C*********************************************************************
      DOUBLE PRECISION FUNCTION IP2( N, V1, V2)
C---------------------------------------------------------------------
C     COMPUTE THE DOUBLE-DOUBLE PRECISION INNER PRODUCT OF
C     THE VECTORS V1 AND V2.
C---------------------------------------------------------------------
      INTEGER*2 N
      REAL*8 V1(1), V2(1), SUM(2)
C---------------------------------------------------------------------
C
      SUM(1) = 0.0D0
      SUM(2) = 0.0D0
      DO 1 I = 1, N
        CALL VPR2( V1(I), V2(I), SUM, IEXP)
    1 CONTINUE
C
      IP2 = SUM(1)
      RETURN
      END
```

Inverse Iteration Using a

Symmetric Indefinite Linear Equation

Solver Based on the Work of J. R. Bunch

```
C*****************************************************************
C     DRIVER PROGRAM FOR INVERSE ITERATION.  THIS ROUTINE
C     READS THE PARAMETERS OF THE GENERALIZED EIGENVALUE
C     PROBLEM, ALLOCATES THE STORAGE AND THE CALLS INVERS.
C-----------------------------------------------------------------
      REAL*8 A(32000)
      INTEGER*2 N, M, LENGTH, LAST
      INTEGER TIMER1, LB, LBETA, LMAP, LU, LV1, LV2, LPRM, LDET
C-----------------------------------------------------------------
C
      TSTRT = TIMER1(0)
      REWIND 2
      LAST = 0
C
   30 IF (LAST .EQ. 1) STOP
C
C     READ PROBLEM PARAMETERS FROM UNIT TWO.
C
      READ(2, END=2) N, M, LENGTH, LAST
      WRITE(6, 100) N, M, LENGTH
  100 FORMAT('1 N=', I3, ' M=', I3, ' LENGTH=', I6/)
C
C     COMPUTE SOME CONSTANTS FOR STORAGE ALLOCATION ....
C
      LB = LENGTH + 1
      LBETA = LB + LENGTH
      LU = LBETA + N / 4 + 1
      LV1 = LU + N * (N+1) / 2
      LV2 = LV1 + N
      LPS = LV2 + N
      LMAP = LPS + N / 4 + 1
      LPRM = LMAP + LENGTH / 4 + 1
      LDET = LPRM + N/4 + 1
      KK = LDET + N
      WRITE(6, 105) KK
  105 FORMAT(' NUMBER OF LONG WORDS OF STORAGE REQUIRED:', I7/)
      IF (KK .GT. 32000) STOP
C
C     ZERO OUT THE MEMORY WE ARE GOING TO USE ...
C
      DO 1 I = LU, KK
        A(I) = 0.0D0
    1 CONTINUE
C
      CALL INVRS(N, LENGTH, A, A(LB), A(LU), A(LV1), A(LV2),
     *     A(LDET), A(LPRM), A(LMAP), A(LBETA), A(LPS))
C
      GO TO 30
C
    2 STOP
C
      END
C*****************************************************************
      SUBROUTINE INVRS(N, LENGTH, A, B, UU, V1, V2, DET,
     *                 PERM, MAP, IBETA, IPS)
C-----------------------------------------------------------------
C     INVERSE ITERATION ROUTINE.
C-----------------------------------------------------------------
      INTEGER TIMER1
      INTEGER*2 N, LENGTH, IBETA(1), IPS(1), MAP(1), PERM(1)
      REAL*8 A(1), B(1), UU(1), DET(1), V1(1), V2(1), T,
     *       XAX, XBX, IP2, SHIFT, OLDT, DIFF
C-----------------------------------------------------------------
      NAMELIST /PARMS/ SHIFT
C-----------------------------------------------------------------
C
C     READ THE PROBLEM TO BE SOLVED.
C
      READ(2) (IBETA(K), K = 1, N)
      READ(2) (MAP(K), A(K), B(K), K = 1, LENGTH)
C
   10 READ(5, PARMS)
      WRITE(6, 104) N
  104 FORMAT('1 INVERSE ITERATION ...    N=', I3/)
      WRITE(6, PARMS)
      T = SHIFT
      IF (SHIFT .LE. 0.0) RETURN
C
      V1(N) = 1.0D0
      V2(N) = 1.0D0
      T1 = TIMER1(1)
C
      CALL UNPACK(N, IBETA, MAP, A, B, SHIFT, UU)
C
      CALL SYMLDL(N, UU, IPS, PERM, DET)
C
      ITER = 0
   20 ITER = ITER + 1
      WRITE(6, 102) ITER
  102 FORMAT(/, ' ITERATION #', I3)
C
      CALL SYMSLV(N, UU, IPS, PERM, DET, V1)
C
      CALL SCALE(N, V1)
C
      CALL IPSPRS(N, IBETA, MAP, A, V1, V2)
C
      XAX = IP2(N, V1, V2)
C
      CALL IPSPRS(N, IBETA, MAP, B, V1, V2)
C
      XBX = IP2(N, V1, V2)
C
      OLDT = T
      T = XAX / XBX
      DIFF = DABS (OLDT - T)
      WRITE(6, 101) T, DIFF
  101 FORMAT(' EIGENVALUE=', F20.14, ' CHANGE=', 1F14.10)
C
      IF (DIFF .LT. 1D-10 .OR. ITER .GT. 10) GO TO 30
C
      DO 2 I = 1, N
        V1(I) = V2(I)
    2 CONTINUE
      GO TO 20
C
   30 T1 = (TIMER1(1) - T1) / 1000.0
      WRITE(6, 123) T1
  123 FORMAT(' ITERATION DONE, TIME USED=', F8.4, ' SECONDS.')
      GO TO 10
```

```
      END
C*****************************************************************
      SUBROUTINE MAXD (N, A, K, J, M1)
C-----------------------------------------------------------------
C     FIND M1 = MAX |A(I,I)| FOR K-1 < I < N+1. J IS THE LEAST
C     INTEGER SUCH THAT M1 = |A(J,J)|.
C-----------------------------------------------------------------
      INTEGER*2 N, K, J
      REAL*8 A(1), M1
C-----------------------------------------------------------------
C
      L = K * (K+1) / 2
      M1 = DABS(A(L))
      J = K
      IF (K .GE. N) RETURN
      KP1 = K + 1
      DO 1 I = KP1, N
        L = L + I
        IF (DABS(A(L)) .LE. M1) GO TO 1
        M1 = DABS(A(L))
        J = I
    1 CONTINUE
      RETURN
      END
C*****************************************************************
      SUBROUTINE MAXA(N, A, R, S, MO)
C-----------------------------------------------------------------
C      FIND MO = MAX |A(I,J)| FOR 0 < I,J < N+1. THE INTEGERS
C      R AND S ARE THE LEAST INTEGERS FOR WHICH MO = |A(R,S)|.
C      THIS ROUTINE IS ONLY USED ONCE. AFTER THE FIRST STEP
C      OF THE REDUCTION, MO IS DETERMINED AS THE REDUCTION
C      PROCEEDS IN SYMLDL.
C-----------------------------------------------------------------
      REAL*8 TEMP, MO, A(1)
      INTEGER*2 N, R, S
C-----------------------------------------------------------------
C
      MO = A(1)
      R = 1
      S = 1
      IF (N .EQ. 1) RETURN
      IT = 1
      DO 20 I = 2, N
         DO 10 J = 1, I
            IF (DABS(A(IT+J)) .LE. MO) GO TO 10
            MO = DABS(A(IT+J))
            R = I
            S = J
   10    CONTINUE
         IT = IT + I
   20 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE SWITCH(N, A, K, I)
C-----------------------------------------------------------------
C     INTERCHANGE ROWS AND COLUMNS I AND K, WHERE WE ASSUME
C     THAT I < K.
C-----------------------------------------------------------------
      REAL*8 TEMP, A(1)
      INTEGER*2 N, K, I
C-----------------------------------------------------------------
C
      IM1 = I - 1
      KT = K * (K-1) / 2
      IT = I * (I-1) / 2
C
      K1 = K + 1
      IF (K1 .GT. N) GO TO 4
      JT = K * K1 / 2
      DO 101 J = K1, N
         TEMP = A(JT+K)
         A(JT+K) = A(JT+I)
         A(JT+I) = TEMP
         JT = JT + J
  101 CONTINUE
C
    4 IF (I .EQ. 1) GO TO 1
      DO 3 J = 1, IM1
         TEMP = A(IT+J)
         A(IT+J) = A(KT+J)
         A(KT+J) = TEMP
    3 CONTINUE
C
    1 I1 = I + 1
      K1 = K - 1
      IF (I1 .GT. K1) GO TO 2
C
      JT = I * I1 / 2
      DO 102 J = I1, K1
         TEMP = A(JT+I)
         A(JT+I) = A(KT+J)
         A(KT+J) = TEMP
         JT = JT + J
  102 CONTINUE
C
    2 TEMP = A(IT+I)
      A(IT+I) = A(KT+K)
      A(KT+K) = TEMP
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE SYMLDL(N, A, PIVOT, PERM, DET)
C-----------------------------------------------------------------
C     THIS ROUTINE USES THE DIAGONAL PIVOTING ALGORITHM OF
C     J. R. BUNCH TO COMPUTE THE L-D-L(TRANSPOSE) DECOMPOSITION
C     OF A, WHERE L IS LOWER TRIANGULAR AND D IS A DIAGONAL
C     MATRIX OF 1X1 AND 2X2 BLOCKS.  CONSECUTIVE ROWS OF THE
C     LOWER TRIANGLE OF THE MATRIX TO BE DECOMPOSED ARE
C     ASSUMED TO BE IN A, WITH THE I-TH ROW IN POSITIONS
C     A(K), K = L1, ... L2, WHERE L1 = I*(I-1)/2 AND
C     L2 = I*(I+1)/2.  A IS ASSUMED INDEFINITE.
C       A IS REPLACED BY L AND D, WHERE L(I,I-1) = 0 IF
C     D(I,I-1) .NE. 0.
C       FOR DETAILS ON THE ALGORITHM, CONSULT BUNCH'S
C     PH.D. DISSERTATION, U.C. BERKELEY, 1965.
C-----------------------------------------------------------------
      REAL*8 A(1), DET(1), MO, M1, ALPHA, TEMP, SAVE, TSTRT
```

```
      INTEGER*2 N, R, S, I, J, K, I1, PERM(1), PIVCT(1)
      INTEGER TIMER1
      LOGICAL TWC
C-----------------------------------------------------------------
C
      ALPHA = (1DO + DSQRT(17DO)) / 8DC
      TSTRT = TIMER1(1)
C
      I = 1
      IT1 = 1
      IT = 0
      CALL MAXA(N, A, R, S, MO)
C
C     BEGIN MAIN LOOP  ..
C
    1 CALL MAXD(N, A, I, K, M1)
      TWO = MO * ALPHA .GT. M1
      MO = 0D0
      I1 = I + 1
      IF (TWO) GO TO 200
C
C     USE A 1 X 1 PIVOT....
C
      IF (K .NE. I) CALL SWITCH(N, A, K, I)
      PERM(I) = K
      IF (A(IT+I) .EQ. 0D0) GO TO 999
      IF (I1 .GT. N) GC TC 110
C
      TEMP = A(IT+I)
      JT = IT1
      DC 101 J = I1, N
         A(JT+I) = A(JT+I) / TEMP
         JT = JT + J
  101 CONTINUE
C
      JT = IT1
      DC 103 J = I1, N
         KT = IT1
         SAVE = A(JT+I) * TEMP
         DO 102 K = I1, J
            A(JT+K) = A(JT+K) - A(KT+I) * SAVE
            KT = KT + K
            IF (MO .LE. DABS(A(JT+K))) GC TC 102
            R = J
            S = K
            MO = DABS(A(JT+K))
  102    CONTINUE
         JT = JT + J
  103 CONTINUE
C
  110 PIVOT(I) = 1
      IT = IT + I
      IT1 = IT1 + I1
      I = I1
      IF (I .LE. N) GO TO  1
C
      GC TO 500
C
C     2 X 2 PIVOT USED ....
C
  200 PERM(I) = S
      IF (S .NE. I) CALL SWITCH(N, A, S, I)
      PERM(I1) = R
      IF (R .NE. I1) CALL SWITCH(N, A, R, I1)
C
      DET(I) = A(IT+I) * A(IT1+I1) - A(IT1+I) * A(IT1+I)
      IF (DET(I).EQ. 0D0) GC TC 999
C
      I2 = I + 2
      IF (I2 .GT. N) GC TC 210
C
      JT = I1 * I2 / 2
      DO 202 J = I2, N
         J1 = J - 1
         SAVE = A(JT+I)
         TEMP = A(JT+I1)
         IF (I2 .GT. J1) GC TO 203
         KT = I2 * I1 / 2
         DO 201 K = I2, J1
            A(JT+K) = A(JT+K) - A(KT+I) * SAVE - A(KT+I1) * TEMP
            KT = KT + K
            IF (MO .LE. DABS(A(JT+K))) GO TO 201
            R = J
            S = K
            MO = DABS(A(JT+K))
  201    CONTINUE
C
  203    A(JT+I) = (A(IT1+I1) * SAVE - A(IT1+I) * TEMP)/ DET(I)
         A(JT+I1) = (A(IT+I) * TEMP - A(IT1+I) * SAVE) / DET(I)
         A(JT+J) = A(JT+J) - A(JT+I) * SAVE - A(JT+I1) * TEMP
         JT = JT + J
  202 CONTINUE
C
  210 PIVCT(I) = 2
      PIVOT(I1) = 0
      IT = IT + I + I1
      IT1 = IT1 + I1 + I2
      I = I2
      IF (I .LE. N) GC TC 1
C
  500 TSTRT = (TIMER1(1) - TSTRT) / 1000.0D0
      WRITE(6, 600) TSTRT
  600 FORMAT(' DECOMP DONE. TIME USED (SECONDS):', F10.6/)
C
      RETURN
C
  999 WRITE(6, 123)
  123 FORMAT(' SINGULAR MATRIX')
      RETURN
      END
C*****************************************************************
      SUBROUTINE SYMSLV(N, A, PIVCT, PERM, DET, B)
C-----------------------------------------------------------------
C     THIS ROUTINE SOLVES A.X = B, WHERE A CONTAINS THE L-D-L(T)
C     DECOMPOSITION PRODUCED BY SYMDCL.
C-----------------------------------------------------------------
      REAL*8 A(1), DET(1), MC, M1, B(1), TEMP, SAVE
      INTEGER*2 N, R, S, PERM(1), PIVOT(1)
C-----------------------------------------------------------------
C
```

```
      DO 301 I = 1, N
         SAVE = B(I)
         IC = PERM(I)
         B(I) = B(IC)
         B(IC) = SAVE
  301 CONTINUE
C
C     SOLVE L.X = B AND PLACE THE RESULT BACK IN B.
C
      I = 0
  400 IM1 = I
      I = I + 1
      IF (I .GT. N) GO TO 499
      IF (PIVOT(I) .EQ. 0) IM1 = IM1 - 1
      IF (IM1 .LT. 1) GO TO 400
C
      IT = I * (I-1) / 2
      DO 401 J = 1, IM1
        B(I) = B(I) - A(IT+J) * B(J)
  401 CONTINUE
      GO TO 400
C
C     SOLVE D.X = B AND PLACE THE RESULT IN B.
C
  499 I = 1
      IT = 0
  500 IF (PIVOT(I) .EQ. 2) GO TO 502
      B(I) = B(I) / A(IT+I)
      I = I + 1
      GO TO 509
C
  502 TEMP = B(I)
      I1 = I + 1
      IT1 = I * I1 / 2
      SAVE = B(I1)
      B(I) = (TEMP * A(IT1+I1) - SAVE * A(IT1+I)) / DET(I)
      B(I1) = (SAVE * A(IT+I) - TEMP * A(IT1+I)) / DET(I)
      I = I + 2
  509 IT = I * (I-1) / 2
      IF (I .LE. N) GO TO 500
C
C     SOLVE L(T).X = B AND PLACE THE RESULT IN B.
C
      I = N
  600 I1 = I
      I = I - 1
      IF (I .LE. 0) GO TO 1000
      IF (PIVOT(I) .EQ. 2) I1 = I1 + 1
      IF (I1 .GT. N) GO TO 600
C
      JT = I * I1 / 2
      DO 601 J = I1, N
        B(I) = B(I) - A(JT+I) * B(J)
        JT = JT + J
  601 CONTINUE
      GO TO 600
C
 1000 I = N
  700 IC = PERM(I)
      SAVE = B(I)
      B(I) = B(IC)
      B(IC) = SAVE
      I = I - 1
      IF (I .GT. 0) GO TO 700
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE SCALE (N, A)
C-----------------------------------------------------------------
C     DIVIDE THE VECTOR A BY ITS ELEMENT OF MAXIMUM MODULUS.
C-----------------------------------------------------------------
      INTEGER*2  N
      REAL*8 A(1), DABS, FMAX
C-----------------------------------------------------------------
C
      FMAX = 0.0D0
      DO 1 I = 1, N
        IF (FMAX .LT. DABS(A(I))) FMAX = DABS(A(I))
    1 CONTINUE
C
      DO 2 I = 1, N
        A(I) = A(I) / FMAX
    2 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE UNPACK(N, IBETA, MAP, A, B, LAMBDA, A1)
C-----------------------------------------------------------------
C     UNPACK A AND B (CF THE A X = LAMBDA B X PROBLEM) AND
C     PLACE A - LAMBDA * B IN THE ARRAY A1 IN THE FORMAT
C     ACCEPTABLE TO SYMLDL AND SYMSLV.
C-----------------------------------------------------------------
      INTEGER*2 N, IBETA(1), MAP(1)
      REAL*8 A1(1), A(1), B(1), LAMBDA
C-----------------------------------------------------------------
C
      IT = 0
      DO 1 I = 1, N
        K1 = 1
        IT = I + IT
        IF (I .GT. 1) K1 = IBETA(I-1) + 1
        K2 = IBETA(I)
        DO 2 J = K1, K2
          A1(IT - MAP(J)) = A(J) - LAMBDA * B(J)
    2   CONTINUE
    1 CONTINUE
C
      RETURN
      END
C*****************************************************************
      SUBROUTINE IPSPRS(N, IBETA, MAP, A, X, R)
C-----------------------------------------------------------------
C     THIS ROUTINE MULTIPLIES A VECTOR X BY THE MATRIX A
C     STORED IN COMPACT FORM AS DESCRIBED IN METHOD 1 SEC. 4.2
C     AND PLACES THE RESULT IN THE ARRAY R.
C-----------------------------------------------------------------
C
      REAL*8 A(1), X(1), R(1), T1
```

```
      INTEGER*2 N, IBETA(1), MAP(1)
C-----------------------------------------------------------------
C
      B(1) = X(1) * A(1)
      DO 2 I = 2, N
        K1 = IBETA(I-1) + 1
        K2 = IBETA(I)   - 1
        B(I) =  X(I) * A(K2+1)
        IF (K1 .GT. K2) GO TO 2
        DO 3 J = K1, K2
          I1 = I - MAP(J)
          B(I) = B(I) + X(I1) * A(J)
          B(I1) = B(I1) + X(I) * A(J)
    3   CONTINUE
    2 CONTINUE
C
      RETURN
      END
C*****************************************************************
      DOUBLE PRECISION FUNCTION IP2(N, V1, V2)
C-----------------------------------------------------------------
C     COMPUTE THE DOUBLE-DOUBLE PRECISION INNER PRODUCT OF
C     THE VECTORS V1 AND V2.
C-----------------------------------------------------------------
C
      INTEGER*2 N
      REAL*8 V1(1), V2(1), SUM(2)
C-----------------------------------------------------------------
C
      SUM(1) = 0.0D0
      SUM(2) = 0.0D0
      DO 1 I = 1, N
        CALL VPR2(V1(I), V2(I), SUM, IEXP)
    1 CONTINUE
C
      IP2 = SUM(1)
      RETURN
      END
```

Appendix C: Sample Deck Set-ups and Runs.

The following pages contain deck set-ups and the output of the resulting runs for a sample problem. The runs were made on an IBM 360/91 at the Stanford Linear Accelerator Center. All the cards with "//" or "/*" in columns 1-2 are OS/360 job control language cards, and do not change from problem to problem. Thus the actual required input is rather small. For information about the input parameters, see the comments at the beginning of each of the program modules. Extensive use is made of the namelist feature of the IBM Fortran language to avoid the rigidity of formatted input.

Object modules for Phase 1, Phase 2, and Phase 3 are contained in the data sets PUB.JAG.P01, PUB.JAG.P02, and PUB.JAG.TMP, respectively.

The sample problem is the following:

$$u_{xx} + u_{yy} = 4 \text{ in } (0,1) \times (0,1)$$

$$u = x^2 + y^2 \quad \text{on} \quad x = 0,1, \quad 0 \leq y \leq 1,$$
$$y = 0,1, \quad 0 \leq x \leq 1.$$

The solution to this problem is $u = x^2 + y^2$.

The first run solves the problem using piecewise quadratics (element 2-6), and the second run uses piecewise cubics (element 3-4). In both cases the error in the parameters is at rounding error level, as is to be expected.

The final two pages of this Appendix contain a deck set-up for an eigenvalue problem. The deck listed is the one used to produce the quintic entry in Table 5.2.2.

Deck Set-up for Sample Problem.

```
//JAGXXTST JOB 'JAG$CG',54,CLASS=E,REGION=300K
//STP1 EXEC LOADGO,PARM.GO='SIZE=288000'
//GO.SYSLIN2 DD DSNAME=PUB.JAG.P01,DISP=OLD,UNIT=2314,
//              VOLUME=SER=PUB001
//GO.FT01F001 DD DSNAME=JAGCG.OUT1,UNIT=SYSDA,DISP=(NEW,PASS),
//              SPACE=(CYL,(1,1),RLSE)
//GO.SYSIN DD *
 &PARMS NDIVS=2,NPS=1,NCEN=0,LAST=1,IBUG=0, &END
 &POINTS PT(1)=(0,0),PT(2)=(1,0),PT(3)=(1,1),PT(4)=(0,1) &END
 &TR NODES=1,2,3, BND(1)=1, BND(2)=2 &END
 &TR NODES=1,3,4, BND(2)=3, BND(3)=4, ENDTR=T &END
/*
//STP2 EXEC LOADGO,PARM.GO='SIZE=288000'
//GO.SYSLIN2  DD DSNAME=PUB.JAG.P02,DISP=OLD,
//              UNIT=2314,VOLUME=SER=PUB003
//GO.FT01F001 DD DSNAME=JAGCG.OUT1,DISP=(OLD,PASS),UNIT=SYSDA
//GO.FT02F001 DD DSNAME=JAGCG.OUT2,DISP=(NEW,PASS),UNIT=SYSDA,
//              SPACE=(CYL,(2,1),RLSE)
//GO.SYSIN DD *
 &PARMS IBUG=0,IDEG=2, NCP=1, ICP(1)=1, NSP(1)=1, ISP(1,1)=1,
        IRHS=2,UX2=1,UY2=1,U2=0,IEIG=0 &END
/*
//STP3 EXEC FORTHLG
//LKED.JAGP03 DD DSNAME=PUB.JAG.TMP,DISP=OLD,UNIT=2314,
//              VOLUME=SER=PUB001
//LKED.SYSIN DD *
 INCLUDE JAGP03
/*
//GO.FT01F001 DD DSNAME=JAGCG.OUT1,UNIT=SYSDA,DISP=(OLD,DELETE)
//GO.FT02F001 DD DSNAME=JAGCG.OUT2,UNIT=SYSDA,DISP=(OLD,DELETE)
//GO.FT03F001 DD DSNAME=JAGCG.OUT3,UNIT=SYSDA,DISP=(NEW,PASS),
//              SPACE=(CYL,(1,1),RLSE)
//GO.SYSIN DD *
 &PARMS NBNDS=4,IPRINT=1,ISOL(1)=10, &END
  1 10
  2 10
  3 10
  4 10
/*
```

```
FINITE ELEMENT SOLUTION OF PDE-S ............. PHASE1:(TRIANGULATION OF THE DOMAIN)
                                  INPUT ...
&PARMS
NDIVS=     2,IBUG=     0,NPS=     1,NCEN=     0,M=     6,LAST=     1
&END

    INDICES                                 NODE COORDINATES                              BND          CORNER
  1   2   3  0.0      0.0      , 1.000000  0.0       , 1.000000  1.000000,   1   2   0,   0   0   0
  1   3   4  0.0      0.0      , 1.000000  1.000000, 0.0       1.000000,   0   3   4,   0   0   0
  1   3   4  0.0      0.0      , 1.000000  0.0       , 0.0       1.000000,   0   0   0,   0   0   0

CORNERS GENERATED     TIME USED:  -0.000 SECONDS.        ACCUMULATED TIME:  -0.000 SECONDS

FILL-IN DONE          TIME USED:   0.016 SECONDS.        ACCUMULATED TIME:   0.016 SECONDS

STORAGE IN MEM2 USED:    433

MAT GENERATED.        TIME USED:   0.0   SECONDS.        ACCUMULATED TIME:   0.016 SECONDS

ORDERING DONE         TIME USED:   0.017 SECONDS.        ACCUMULATED TIME:   0.033 SECONDS

&PARMS1
NPTS=    31,NTRS=     9,NSIDE=     8
&END

BEGIN OUTPUT ....     TIME USED:   0.0   SECONDS.        ACCUMULATED TIME:   0.033 SECONDS

OUTPUT COMPLETE.      TIME USED:   0.016 SECONDS.        ACCUMULATED TIME:   0.049 SECONDS
```

```
FINITE ELEMENT SOLUTION OF PDE-S ............. PHASE2' (GENERATION OF STIFFNESS MATRICES ETC.)

                                        INPUT INFORMATION
BYTES USED IN Z:      3038.CO

&PARMS1
NPTS=    31,NTRS=     9,NSIDE=     8,NPS=     1,NCEN=     0,M=     6,N=     6,IDEG=     2,IPHS=     2,IEIG=     0
&END

INPUT COMPLETE          TIME USED:   0.099 SECONDS.          ACCUMULATED TIME:   0.099 SECONDS

DEGREE OF PIECEWISE POLYNOMIAL:   2
NUMBER OF PARAMETERS CHARACTERIZING IT:   6

END INITIALIZATION      TIME USED:   0.034 SECONDS.          ACCUMULATED TIME:   0.133 SECONDS

GENERATION COMPLETE     TIME USED:   0.083 SECONDS.          ACCUMULATED TIME:   0.216 SECONDS
```

```
FINITE ELEMENT SOLUTION OF PDE-S : PHASE THREE   (ASSEMBLY OF EQUATIONS AND SOLUTION) *******

                    INPUT INFORMATION

&PARMS
NBNDS=     4,IBUG=      0,ISOL=     10,      0,       0,IPRINT=      1
&END
 1 10  0  0
 2 10  0  0
 3 10  0  0
 4 10  0  0
&PARMS1
NPTS=    31,NTRS=      9,NSIDE=      8,NPS=      1,NCEV=      0,M=      6,N=      6,IRHS=      2
&END

GENDUP DONE.           TIME USED:   0.133 SECONDS.          ACCUMULATED TIME:   0.133 SECONDS

BNDRY DONE.            TIME USED:   0.0   SECONDS.          ACCUMULATED TIME:   0.133 SECONDS

BANDWIDTH:    5

STORAGE REQUIRED FOR L :      32 WORDS

&PARMS2
NEQNS=     9,IBW=      5,IRHS=      2,IEIG=      0,LENA=     27
&END
NUMBER OF BYTES USED IN Z:      2832.00

BEGIN ASSEMBLY...      TIME USED:   0.016 SECONDS.          ACCUMULATED TIME:   0.149 SECONDS

ASSEMBLY COMPLETE.     TIME USED:   0.034 SECONDS.          ACCUMULATED TIME:   0.183 SECONDS

UNPACK DONE.           TIME USED:   0.0   SECONDS.          ACCUMULATED TIME:   0.183 SECONDS

LUSPKS:
   NUMBER OF EQUATIONS:    9
   NUMBER OF OPERATIONS REQUIRED(*,/):        71.0

DECOMP DONE.           TIME USED:   0.0   SECONDS.          ACCUMULATED TIME:   0.183 SECONDS

OPERATION COUNT (*,/) IN SPRSLV:        62.0

SOLVE  DONE.           TIME USED:   0.0   SECONDS.          ACCUMULATED TIME:   0.183 SECONDS

OPERATION COUNT (*,/) IN SPRSLV:        62.0

ITERATION  2 MAX NORM OF CHANGE:  0.247291D-15

REFINE DONE.           TIME USED:   0.016 SECONDS.          ACCUMULATED TIME:   0.199 SECONDS
```

```
FINITE ELEMENT SOLUTION OF PDE-S : PHASE THREE   (ASSEMBLY OF EQUATIONS AND SOLUTION) *******

                    NUMERICAL SOLUTION.

COORDINATES:  1.0000000   1.0000000   VALUE:  2.0000000  TYPE:  1
COORDINATES:  0.7500000   0.7500000   VALUE:  1.1250000  TYPE: -1   TRUE VALUE:  1.12500000  ERROR:  0.00000000
COORDINATES:  0.7500000   1.0000000   VALUE:  1.5625000  TYPE:  1
COORDINATES:  1.0000000   0.7500000   VALUE:  1.5625000  TYPE:  1
COORDINATES:  0.0         0.7500000   VALUE:  0.5625000  TYPE:  1
COORDINATES:  0.2500000   1.0000000   VALUE:  1.0625000  TYPE:  1
COORDINATES:  0.0         1.0000000   VALUE:  1.0000000  TYPE:  1
COORDINATES:  1.0000000   0.2500000   VALUE:  1.0625000  TYPE:  1
COORDINATES:  0.7500000   0.0         VALUE:  0.5625000  TYPE:  1
COORDINATES:  1.0000000   0.0         VALUE:  1.0000000  TYPE:  1
COORDINATES:  1.0000000   0.5000000   VALUE:  1.2500000  TYPE:  1
COORDINATES:  0.7500000   0.5000000   VALUE:  0.8125000  TYPE: -1   TRUE VALUE:  0.81250000  ERROR:  0.00000000
COORDINATES:  0.7500000   0.2500000   VALUE:  0.6250000  TYPE: -1   TRUE VALUE:  0.62500000  ERROR:  0.00000000
COORDINATES:  0.5000000   1.0000000   VALUE:  1.2500000  TYPE:  1
COORDINATES:  0.2500000   0.7500000   VALUE:  0.6250000  TYPE: -1   TRUE VALUE:  0.62500000  ERROR:  0.00000000
COORDINATES:  0.5000000   0.7500000   VALUE:  0.8125000  TYPE: -1   TRUE VALUE:  0.81250000  ERROR:  0.00000000
COORDINATES:  0.5000000   0.5000000   VALUE:  0.5000000  TYPE: -1   TRUE VALUE:  0.50000000  ERROR:  0.00000000
COORDINATES:  0.0         0.5000000   VALUE:  0.2500000  TYPE:  1
COORDINATES:  0.5000000   0.0         VALUE:  0.2500000  TYPE:  1
COORDINATES:  0.5000000   0.2500000   VALUE:  0.3125000  TYPE: -1   TRUE VALUE:  0.31250000  ERROR:  0.00000000
COORDINATES:  0.2500000   0.5000000   VALUE:  0.3125000  TYPE: -1   TRUE VALUE:  0.31250000  ERROR:  0.00000000
COORDINATES:  0.2500000   0.2500000   VALUE:  0.1250000  TYPE: -1   TRUE VALUE:  0.12500000  ERROR:  0.00000000
COORDINATES:  0.0         0.2500000   VALUE:  0.0625000  TYPE:  1
COORDINATES:  0.2500000   0.0         VALUE:  0.0625000  TYPE:  1
COORDINATES:  0.0         0.0         VALUE:  0.0        TYPE:  1

MAXIMUM ERROR IN PARAMETER OF TYPE 1 IS:   2.829681D-14

OUTPUT DONE.        TIME USED:   0.050 SECONDS.        ACCUMULATED TIME:   0.249 SECONDS
```

These cards and input solve the sample problem using piece-wise cubics (element 3-4). The output from this run appears on the following pages.

```
//JAGXXTST JOB 'JAG$CG',54,CLASS=E,REGION=300K
//STP1 EXEC LOADGO,PARM.GO='SIZE=288000'
//GO.SYSLIN2 DD DSNAME=PUB.JAG.P01,DISP=OLD,UNIT=2314,
//               VOLUME=SER=PUB001
//GO.FT01F001 DD DSNAME=JAGCG.OUT1,UNIT=SYSDA,DISP=(NEW,PASS),
//               SPACE=(CYL,(1,1),RLSE)
//GO.SYSIN DD *
 &PARMS NDIVS=3,NPS=0,NCEN=1,LAST=1,IBUG=0, &END
 &POINTS PT(1)=(0,0),PT(2)=(1,0),PT(3)=(1,1),PT(4)=(0,1) &END
 &TR NODES=1,2,3, BND(1)=1, BND(2)=2 &END
 &TR NODES=1,3,4, BND(2)=3, BND(3)=4, ENDTR=T &END
/*
//STP2 EXEC LOADGO,PARM.GO='SIZE=288000'
//GO.SYSLIN2  DD DSNAME=PUB.JAG.P02,DISP=OLD,
//               UNIT=2314,VOLUME=SER=PUB003
//GO.FT01F001 DD DSNAME=JAGCG.OUT1,DISP=(OLD,PASS),UNIT=SYSDA
//GO.FT02F001 DD DSNAME=JAGCG.OUT2,DISP=(NEW,PASS),UNIT=SYSDA,
//               SPACE=(CYL,(2,1),RLSE)
//GO.SYSIN DD *
 &PARMS IBUG=0,IDEG=3,NCP=3,ICP=1,2,3,
        IRHS=2,UX2=1,UY2=1,U2=0,IEIG=0 &END
/*
//STP3 EXEC FORTHLG
//LKED.JAGP03 DD DSNAME=PUB.JAG.TMP,DISP=OLD,UNIT=2314,
//               VOLUME=SER=PUB001
//LKED.SYSIN DD *
 INCLUDE JAGP03
/*
//GO.FT01F001 DD DSNAME=JAGCG.OUT1,UNIT=SYSDA,DISP=(OLD,DELETE)
//GO.FT02F001 DD DSNAME=JAGCG.OUT2,UNIT=SYSDA,DISP=(OLD,DELETE)
//GO.FT03F001 DD DSNAME=JAGCG.OUT3,UNIT=SYSDA,DISP=(NEW,PASS),
//               SPACE=(CYL,(1,1),RLSE)
//GO.SYSIN DD *
 &PARMS NBNDS=4,IPRINT=1,ISOL(1)=10,ISOL(2)=11,ISOL(3)=11,&END
  1 10 11 11
  2 10 11 11
  3 10 11 11
  4 10 11 11
/*
```

```
FINITE ELEMENT SOLUTION OF PDE-S ............. PHASE1:(TRIANGULATION OF THE DOMAIN)
                                    INPUT ...
&PARMS
NDIVS=     3, IBUG=     0,NPS=     0,NCEN=     1,M=     4,LAST=     1
&END

  INDICES                         NODE COORDINATES                            BND          CORNER
 1   2   3  0.0        0.0     , 1.000000  0.0       , 1.000000  1.000000,  1   2   3,   0   0   0
 1   3   4  0.0        0.0     , 1.000000  1.000000, 0.0         1.000000,  0   3   4,   0   0   0
 1   3   4  0.0        0.0     , 1.000000  0.0       , 0.0         1.000000,  0   0   0,   0   0   0

CORNERS GENERATED      TIME USED:  0.049 SECONDS.          ACCUMULATED TIME:  0.049 SECONDS

FILL-IN DONE           TIME USED:  0.0   SECONDS.          ACCUMULATED TIME:  0.049 SECONDS

STORAGE IN MEM2 USED:       537

MAT GENERATED.         TIME USED:  0.0   SECONDS.          ACCUMULATED TIME:  0.049 SECONDS

ORDERING DONE          TIME USED:  0.017 SECONDS.          ACCUMULATED TIME:  0.066 SECONDS

&PARMS1
NPTS=    38,NTRS=    19,NSIDE=    12
&END

BEGIN OUTPUT ....      TIME USED:  0.0   SECONDS.          ACCUMULATED TIME:  0.066 SECONDS

OUTPUT COMPLETE.       TIME USED:  0.0   SECONDS.          ACCUMULATED TIME:  0.066 SECONDS
```

```
FINITE ELEMENT SOLUTION OF PDE-S ............. PHASE2: (GENERATION OF STIFFNESS MATRICES ETC.)
                                    INPUT INFORMATION
BYTES USED IN Z:      6912.00

&PARMS1
NPTS=    38,NTRS=    19,NSIDE=    12,NPS=    0,NCEN=    1,M=    4,N=    10,IDEG=    3,IRHS=    2,IEIG=    0
&END

INPUT COMPLETE          TIME USED:   0.116 SECONDS.          ACCUMULATED TIME:   0.116 SECONDS

DEGREE OF PIECEWISE POLYNOMIAL:   3
NUMBER OF PARAMETERS CHARACTERIZING IT:  10


END INITIALIZATION      TIME USED:   0.050 SECONDS.          ACCUMULATED TIME:   0.166 SECONDS


GENERATION COMPLETE     TIME USED:   0.233 SECONDS.          ACCUMULATED TIME:   0.399 SECONDS
```

```
FINITE ELEMENT SOLUTION OF PDE-S : PHASE THREE   (ASSEMBLY OF EQUATIONS AND SOLUTION) *******
                     INPUT INFORMATION

&PARMS
NONDS=     4,IBUG=  ,  0,ISUL=    10,    11,    11,IPRINT=     1
&END
 1 10 11 11
 2 10 11 11
 3 10 11 11
 4 10 11 11
&PARMS1
NPTS=    38,NTRS=    19,NSIDE=    12,NPS=     0,NCEN=     1,M=     4,N=    10,IRHS=     2
&END

GENDOP DONE.            TIME USED:   0.133 SECONDS.            ACCUMULATED TIME:   0.133 SECONDS

BNDRY DONE.             TIME USED:   0.016 SECONDS.            ACCUMULATED TIME:   0.149 SECONDS

BANDWIDTH:   17

STORAGE REQUIRED FOR L :    226 WORDS

&PARMS2
NEQNS=    30,IBW=    17,IRHS=     2,IEIG=     0,LENA=   159
&END
NUMBER OF BYTES USED IN Z:      7776.00

BEGIN ASSEMBLY...       TIME USED:   0.034 SECONDS.            ACCUMULATED TIME:   0.183 SECONDS

ASSEMBLY COMPLETE.      TIME USED:   0.083 SECONDS.            ACCUMULATED TIME:   0.266 SECONDS

UNPACK DONE.            TIME USED:   0.016 SECONDS.            ACCUMULATED TIME:   0.282 SECONDS

LUSPRS:
   NUMBER OF EQUATIONS:  30
   NUMBER OF OPERATIONS REQUIRED(*,/):       1092.0

DECOMP DONE.            TIME USED:   0.0   SECONDS.            ACCUMULATED TIME:   0.282 SECONDS

OPERATION COUNT (*,/) IN SPRSLV:        434.0

SOLVE  DONE.            TIME USED:   0.0   SECONDS.            ACCUMULATED TIME:   0.282 SECONDS

OPERATION COUNT (*,/) IN SPRSLV:        434.0

ITERATION   2 MAX NORM OF CHANGE:  0.322240D-14

REFINE DONE.            TIME USED:   0.034 SECONDS.            ACCUMULATED TIME:   0.316 SECONDS
```

```
FINITE ELEMENT SOLUTION OF PDE-S : PHASE THREE   (ASSEMBLY OF EQUATIONS AND SOLUTION) *******

                    NUMERICAL SOLUTION.

COORDINATES:   1.0000000   1.0000000   VALUE:  2.00000000 TYPE:  1
                                       VALUE:  2.00000000 TYPE:  2
                                       VALUE: -2.00000000 TYPE:  3
COORDINATES:   0.7777778   0.8888889   VALUE:  1.39506173 TYPE: -1   TRUE VALUE:  1.39506173 ERROR:  0.00000000
COORDINATES:   0.8888889   0.7777778   VALUE:  1.39506173 TYPE: -1   TRUE VALUE:  1.39506173 ERROR:  0.00000000
COORDINATES:   1.0000000   0.6666667   VALUE:  1.44444444 TYPE:  1
                                       VALUE:  2.00000000 TYPE:  2
                                       VALUE:  1.33333333 TYPE:  3
COORDINATES:   0.8888889   0.4444444   VALUE:  0.98765432 TYPE: -1   TRUE VALUE:  0.98765432 ERROR:  0.00000000
COORDINATES:   0.7777778   0.5555556   VALUE:  0.91358025 TYPE: -1   TRUE VALUE:  0.91358025 ERROR:  0.00000000
COORDINATES:   1.0000000   0.3333333   VALUE:  1.11111111 TYPE:  1
                                       VALUE:  2.00000000 TYPE:  2
                                       VALUE:  0.66666667 TYPE:  3
COORDINATES:   0.8888889   0.1111111   VALUE:  0.80246914 TYPE: -1   TRUE VALUE:  0.80246914 ERROR:  0.00000000
COORDINATES:   1.0000000   0.0         VALUE:  1.00000000 TYPE:  1
                                       VALUE:  0.0        TYPE:  2
                                       VALUE:  2.00000000 TYPE:  3
COORDINATES:   0.7777778   0.2222222   VALUE:  0.65432099 TYPE: -1   TRUE VALUE:  0.65432099 ERROR:  0.00000000
COORDINATES:   0.6666667   1.0000000   VALUE:  1.44444444 TYPE:  1
                                       VALUE:  2.00000000 TYPE:  2
                                       VALUE:  1.33333333 TYPE:  3
COORDINATES:   0.4444444   0.8888889   VALUE:  0.98765432 TYPE: -1   TRUE VALUE:  0.98765432 ERROR:  0.00000000
COORDINATES:   0.5555556   0.7777778   VALUE:  0.91358025 TYPE: -1   TRUE VALUE:  0.91358025 ERROR:  0.00000000
COORDINATES:   0.3333333   1.0000000   VALUE:  1.11111111 TYPE:  1
                                       VALUE:  2.00000000 TYPE:  2
                                       VALUE:  0.66666667 TYPE:  3
COORDINATES:   0.1111111   0.8888889   VALUE:  0.80246914 TYPE: -1   TRUE VALUE:  0.80246914 ERROR:  0.00000000
COORDINATES:   0.0         1.0000000   VALUE:  1.00000000 TYPE:  1
                                       VALUE: -2.00000000 TYPE:  2
                                       VALUE:  0.00000000 TYPE:  3
COORDINATES:   0.2222222   0.7777778   VALUE:  0.65432099 TYPE: -1   TRUE VALUE:  0.65432099 ERROR:  0.00000000
COORDINATES:   0.6666667   0.6666667   VALUE:  0.88888889 TYPE: -1   TRUE VALUE:  0.88888889 ERROR:  0.00000000
                                       VALUE:  1.33333333 TYPE: -2   TRUE VALUE:  1.33333333 ERROR:  0.00000000
                                       VALUE:  1.33333333 TYPE: -3   TRUE VALUE:  1.33333333 ERROR:  0.00000000
COORDINATES:   0.4444444   0.5555556   VALUE:  0.50617284 TYPE: -1   TRUE VALUE:  0.50617284 ERROR:  0.00000000
COORDINATES:   0.5555556   0.4444444   VALUE:  0.50617284 TYPE: -1   TRUE VALUE:  0.50617284 ERROR:  0.00000000
COORDINATES:   0.6666667   0.3333333   VALUE:  0.55555556 TYPE: -1   TRUE VALUE:  0.55555556 ERROR:  0.00000000
                                       VALUE:  1.33333333 TYPE: -2   TRUE VALUE:  1.33333333 ERROR:  0.00000000
                                       VALUE:  0.66666667 TYPE: -3   TRUE VALUE:  0.66666667 ERROR:  0.00000000
COORDINATES:   0.6666667   0.0         VALUE:  0.44444444 TYPE:  1
                                       VALUE:  0.0        TYPE:  2
                                       VALUE:  1.33333333 TYPE:  3
COORDINATES:   0.5555556   0.1111111   VALUE:  0.32098765 TYPE: -1   TRUE VALUE:  0.32098765 ERROR:  0.00000000
COORDINATES:   0.4444444   0.2222222   VALUE:  0.24691358 TYPE: -1   TRUE VALUE:  0.24691358 ERROR:  0.00000000
COORDINATES:   0.3333333   0.6666667   VALUE:  0.55555556 TYPE: -1   TRUE VALUE:  0.55555556 ERROR:  0.00000000
                                       VALUE:  0.66666667 TYPE: -2   TRUE VALUE:  0.66666667 ERROR:  0.00000000
                                       VALUE:  1.33333333 TYPE: -3   TRUE VALUE:  1.33333333 ERROR:  0.00000000
COORDINATES:   0.0         0.6666667   VALUE:  0.44444444 TYPE:  1
                                       VALUE:  0.0        TYPE:  2
                                       VALUE:  1.33333333 TYPE:  3
COORDINATES:   0.1111111   0.5555556   VALUE:  0.32098765 TYPE: -1   TRUE VALUE:  0.32098765 ERROR:  0.00000000
COORDINATES:   0.2222222   0.4444444   VALUE:  0.24691358 TYPE: -1   TRUE VALUE:  0.24691358 ERROR:  0.00000000
COORDINATES:   0.3333333   0.3333333   VALUE:  0.22222222 TYPE: -1   TRUE VALUE:  0.22222222 ERROR:  0.00000000
                                       VALUE:  0.66666667 TYPE: -2   TRUE VALUE:  0.66666667 ERROR:  0.00000000
                                       VALUE:  0.66666667 TYPE: -3   TRUE VALUE:  0.66666667 ERROR:  0.00000000
COORDINATES:   0.3333333   0.0         VALUE:  0.11111111 TYPE:  1
```

```
                                                 VALUE:  0.0          TYPE:  2
                                                 VALUE:  0.66666667   TYPE:  3
COORDINATES:    0.0          0.3333333           VALUE:  0.11111111   TYPE:  1
                                                 VALUE:  0.0          TYPE:  2
                                                 VALUE:  0.66666667   TYPE:  3
COORDINATES:    0.1111111    0.2222222           VALUE:  0.06172840   TYPE: -1   TRUE VALUE:  0.06172840 ERROR:  0.00000000
COORDINATES:    0.2222222    0.1111111           VALUE:  0.06172840   TYPE: -1   TRUE VALUE:  0.06172840 ERROR:  0.00000000
COORDINATES:    0.0          0.0                 VALUE:  0.0          TYPE:  1
                                                 VALUE:  0.0          TYPE:  2
                                                 VALUE:  0.0          TYPE:  3

MAXIMUM ERROR IN PARAMETER OF TYPE 1 IS:   2.058076D-14

MAXIMUM ERROR IN PARAMETER OF TYPE 2 IS:   4.227174D-14

MAXIMUM ERROR IN PARAMETER OF TYPE 3 IS:   3.891332D-14

OUTPUT DONE.         TIME USED:   0.116 SECONDS.          ACCUMULATED TIME:   0.432 SECONDS
```

These cards and input produced the quintic entry in table 5.2.2. Note that the object decks for the inverse iteration code using Bunch's symmetric solver are stored in the data set PUB.JAG.INV.

```
//JAGXXHL5 JOB 'JAG$CG',54,CLASS=E,REGION=300K
//STP1 EXEC LOADGO,PARM.GO='SIZE=288000'
//GO.SYSLIN2 DD DSNAME=PUB.JAG.P01,DISP=OLD,UNIT=2314,
//              VOLUME=SER=PUB001
//GO.FT01F001 DD DSNAME=JAGCG.OUT1,UNIT=SYSDA,DISP=(NEW,PASS),
//              SPACE=(CYL,(1,1),RLSE)
//GO.SYSIN DD *
 &PARMS NDIVS=1, NPS=4, NCEN=6, LAST=1, IBUG=0, &END
 &POINTS PT(1)=(0,0), PT(2)=(1,0), PT(3)=(2,0), PT(4)=(2,1),
  PT(5)=(1.3,.7), PT(6)=(1,.7), PT(7)=(.6,.6), PT(8)=(.9,.9),
  PT(9)=(1.1,.9), PT(10)=(1.2,1), PT(11)=(1,1) &END
 &TR NODES=1,2,7, BND(1)=1 &END
 &TR NODES=7,2,6 &END
 &TR NODES=6,2,5, &END
 &TR NODES=2,3,5, BND(1)=1 &END
 &TR NODES=3,4,5, BND(1)=1 &END
 &TR NODES=7,6,8 &END
 &TR NODES=6,9,8 &END
 &TR NODES=6,5,9 &END
 &TR NODES=8,9,11 &END
 &TR NODES=9,10,11, BND(2)=1 &END
 &TR NODES=9,5,10 &END
 &TR NODES=5,4,10, BND(2)=1, ENDTR=T &END
/*
//STP2 EXEC LOADGO,PARM.GO='SIZE=288000'
//GO.SYSLIN2  DD DSNAME=PUB.JAG.P02,DISP=OLD,
//              UNIT=2314,VOLUME=SER=PUB003
//GO.FT01F001 DD DSNAME=JAGCG.OUT1,DISP=(OLD,PASS),UNIT=SYSDA
//GO.FT02F001 DD DSNAME=JAGCG.OUT2,DISP=(NEW,PASS),UNIT=SYSDA,
//              SPACE=(CYL,(1,1),RLSE)
//GO.SYSIN DD *
 &PARMS IBUG=0, IDEG=5, NCP=1, ICP(1)=1, NSP(1)=1, NSP(2)=1,
  nsp(3)=1, NSP(4)=1, ISP(1,1)=1, ISP(2,1)=1, ISP(3,1)=1,
  isp(4,1)=1, IRHS=0, UX2=1, UY2=1, U2=0, IEIG=1 &END
/*
```

```
//STP3 EXEC FORTHLG
//LKED.JAGP03 DD DSNAME=PUB.JAG.TMP,DISP=OLD,UNIT=2314,
//               VOLUME=SER=PUB001
//LKED.SYSIN DD *
 INCLUDE JAGP03
/*
//GO.FT01F001 DD DSNAME=JAGCG.OUT1,UNIT=SYSDA,DISP=(OLD,DELETE)
//GO.FT02F001 DD DSNAME=JAGCG.OUT2,UNIT=SYSDA,DISP=(OLD,DELETE)
//GO.FT03F001 DD DSNAME=JAGCG.OUT3,UNIT=SYSDA,DISP=(NEW,PASS),
//               SPACE=(CYL,(1,1),RLSE)
//GO.SYSIN DD *
 &PARMS NBNDS=1, IBUG=0 &END
  1  5
/*
//STP5 EXEC FORTHLG
//LKED.JAGP4 DD DSNAME=PUB.JAG.INV,DISP=OLD,UNIT=2314,
//               VOLUME=SER=PUB001
//LKED.SYSIN DD *
    INCLUDE JAGP4
/*
//GO.FT02F001 DD DSNAME=JAGCG.OUT3,UNIT=SYSDA,DISP=(OLD,DELETE)
//GO.SYSIN DD *
 &PARMS SHIFT=9.6 &END
 &PARMS SHIFT=-1 &END
/*
```