FINITE ELEMENTS SOFTWARE AND APPLICATIONS

BY

AHMED SHARAF ELDIN AHMED

A Doctoral Thesis

Submitted in partial fulfilment of the requirements

for the award of Doctor of Philosophy

of Loughborough University of Technology

July, 1987.

Supervisor: Professor D.J. EVANS, Ph.D.,D.Sc.,
Department of Computer Studies

External Supervisor: Professor A.S. NOUH

ABSTRACT

The contents of this thesis are a detailed study of the software for the finite element method. In the text, the finite element method is introduced from both the engineering and mathematical points of view. The computer implementation of the method is explained with samples of mainframe, mini- and micro-computer implementations. A solution is presented for the problem of limited stack size for both mini- and micro-computers which possess stack architecture.

Several finite element programs are presented. Special purpose programs to solve problems in structural analysis and groundwater flow are discussed. However, an efficient easy-to-use finite element program for general two-dimensional problems is presented. Several problems in groundwater flow are considered that include steady, unsteady flows in different types of aquifers. Different cases of sinks and sources in the flow domain are also considered. The performance of finite element methods is studied for the chosen problems by comparing the numerical solutions of test problems with analytical solutions (if they exist) or with solutions obtained by other numerical methods. The polynomial refinement of the finite elements is studied for the presented problems in order to offer some evidence as to which finite element simulation is best to use under a variety of circumstances.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my supervisor, Professor D.J. Evans, for his considerable guidance, advice and willingness to assist and advice at any time throughout the programme of this work. I am also grateful to Professor A.S. Nouh who acted as an external supervisor during this research. Thanks are also due to Dr. E.I. Elniema for his suggestions.

Finally, my sincere gratitude and thanks to my mother, my wife, Olfat and my children Omniya and Ahmed for their patience and moral encouragement. To them this thesis is dedicated.

CONTENTS

CHAPTER 1

INTRODUCTION

TABLE OF CONTENTS

1.1 INTRODUCTION

The Finite Element Method (FEM) originated as a generalization of the matrix structural analysis method to problems of elastic continua. Despite the fact that the term "Finite Elements" itself was introduced by Clough [1960] in a paper on plane elasticity, the ideas of Finite Element (FE) analysis date back much further. Therefore, it is not possible to mention a specific date when the FEM was invented. However, it seems to be a fact that the challenge met when designing aircraft during the last few decades was the motivation for new methods of structural analysis. The complexity of these structural systems together with the various loading conditions were behind the first ideas of the method where a continuum is discretized into smaller parts for which a solution can be approximated. At a later date, the method was realised to be equivalent to finding approximate solutions to variational problems using piecewise polynomials. This allowed the method to be used for many field problems like those in fluid dynamics, electrostatics and solid mechanics. As the FEM applications increased rapidly in the 1960's, more mathematicians became interested in giving the method a firm mathematical foundation. Meanwhile, since the FEM is a computational technique which requires the use of a digital computer for its implementation, many computer specialists started developing software for the method. Due to the rapid advances in computer hardware and software technology, several FE research and software projects are being developed to cope with these advances.

Unquestionably, today the FEM is a well-established technique and is considered as one of the most powerful engineering analysis tools.

1.2 THE BASIC CONCEPTS

It seems to be a general approach of the thinking methodology in life: to study a complicated problem, try to partition it to simpler subproblems for which solutions can be found much easier and then construct the final solution to the original problem by the assemblage of the subproblems solution.

In the FEM a discretization technique is employed through which a complex region is divided into simpler subregions called finite elements. This discretization process will convert a continuum, for example, of infinite degrees of freedom to a system of finite elements of finite degrees of freedom. These elements are connected at specified points called nodes. It is possible to have fewer nodes within elements as well. The field variable is specified in terms of approximate functions within each element. These functions are called interpolation or basis functions and are usually polynomials. The interpolation functions are defined in terms of the values of the field variables at nodes. Thus, the problem now becomes how to determine the field values at nodes. It is obvious that: (1) these interpolation functions must satisfy some continuity conditions across element boundaries, (2) certain boundary conditions must also be satisfied. The major factors that affect the obtained solution are the types, number and size of elements, the elements distribution and grading and the used interpolation functions. Four approaches can be used to obtain the element characteristics: the direct, the variational, the weighted residual and the energy balance approaches. The direct approach was the first to be used for structural analysis problems and can be used in relatively simple cases

only. In this approach, a direct formulation of the element characteristics can be done based on the principles of structural mechanics in applications of solid mechanics, say. This will be explained in greater detail in Chapter 2 of this thesis.

In the variational approach, a functional is extremised. The functional in structural mechanics is usually the potential energy. This approach can be used for more complicated problems provided that a variational principle exists and is known. If there is no variational principle known for a problem, then the weighted residual approach can be used to obtain the element characteristics. In this case, the starting point is the partial differential equation itself. Finally, the energy balance approach, requires no variational principle and thus can be used for a wider range of problems where the balance of thermal and/or mechanical energy of the system is utilized to formulate the element characteristics.

Despite the approach used to formulate the element characteristics, a general strategy of finite element solution can be stated as follows:

(1) Discretization of the domain into suitable finite elements and specification of the nodes. Generally, for one dimensional problems, line segments are used, triangles or rectangles are used in two dimensional problems, while tetrahedrons or hexahedrons are used in three-dimensional problems. Several types of these elements are used within this thesis for one and two dimensional problems.

(2) Selection of interpolation functions to express the field variable, which may be scalar, vector or higher-order tensor over the element.

(3) Computation of the element characteristics using one of the above mentioned approaches.

(4) Assemblage of the element characteristics to obtain the overall system characteristics, then the incorporation of boundary conditions and then the resulting set of equations is solved to determine the value of the field variable at the nodes.

(5) Performing some post processing functions like the computations of other important values, the display or plotting of the results, etc.

1.3 DOMAIN OF APPLICATIONS OF THE FINITE ELEMENT METHOD

Although the FEM was originally developed by structural engineers as a solution technique for structural analysis, it has spread rapidly to cover many fields in Engineering, Physics and Applied Mathematics. This may be due to the general nature of its theory as a method for the solution of boundary value problems. However, it should be emphasised that although most branches of engineering analysis can be considered as potential users of the FEM, it is not always the magic and the best solution method to ALL these problems. In fact, every solution technique has its merits and disadvantages. Despite that, it can be safely said that the FEM is usually superior to other competitive techniques in most cases.

It is impossible to list all the applications of the FEM and, therefore, a partial list of the titles of the most well known areas in engineering analysis is given as follows:

(1) Structural Engineering:

* Static and dynamic analysis of various types of structures composed of different materials.
* Stability analysis of structures
* Response of structures to periodic loads
* Elasticity problems

(2) Strength of Materials:

* Creep and fatigue analysis of materials
* Pond stresses in composite sections

(3) Heat transfer:

* Steady state temperature distribution in solids and fluids
* The analysis of transient state in heat transfer problems like heat flow in rocket nozzles and turbine blades.

(4) Hydraulics and Hydrodynamics:

* Analysis of laminar and turbulent flows
* Subsonic and ultrasonic flows
* Analysis of hydraulic structures like dams
* Lake and dam interaction.

(5) Water Resources:

* Analysis of potential flows
* Free surface flows
* Seepage analysis
* Flow in aquifers and porous media.

(6) Geomechanics:

* Analysis of different types of foundations
* Analysis of soil structures interaction
* Rock and soil mechanics

(7) Electrical Engineering:

* Electric and magnetic potential
* Analysis of power transmission systems

(8) Mechanical Engineering:

* Fracture mechanics
* Analysis of mechanical systems, including frequency and modal analysis.
* Stress concentration problems
* Lubrication problems

(9) Nuclear Engineering:

* Analysis of nuclear pressure vessels and containment studies
* Structural analysis of reactors
* Neutron flux distribution

(10) Biomedical Engineering:

* Stress analysis of: bones, eye balls and teeth
* Mechanics of heart valves
* Structural behaviour of the skull

(11) Industrial Engineering:

* Welding analysis
* Manufacturing process of machine tools

(12) Chemical Engineering:

* Melting of solids in chemical plants
* Heat, mass and momentum transfer problems in chemical engineering processes.

Within the body of this thesis some problems in structural and water resources engineering are considered.

1.4 SOFTWARE FOR THE FINITE ELEMENT METHOD

Since the FEM is a numerical technique for which the use of a computer is essential, a potential user of the method is faced by the problem of finding the software suitable for his application. Generally, in most of the structural analysis and heat conduction problems there exist many available software packages that can be used to solve many classical practical problems in these two fields. However, this statement does not exclude the modification of existing software to suit particular needs or even the development of new special purpose programs. The situation in other relatively new fields of applications is quite different. In these fields there is no general purpose software but rather software for particular problems. To illustrate, there is no general purpose software known in the open literature that can be used for general problems in fluid mechanics. This, in no case, means that the current general purpose finite element packages cannot be used to solving some particular problems in fluid mechanics as will be demonstrated within this thesis. The aspects of computer implementation of the FEM will be detailed in Chapter 4. However, it may be useful to give here a brief list of some of the sources of information about FE software [Noor, 1981] and [Huebner and Thornton, 1982]:

- ASIAC : Aerospace Structures Information and Analysis Center, AFFDL/FBR Wright-Patterson Air Force Base, Dayton, OH 45433, U.S.A.
- CEPA : Society for Computer Application in Engineering, Planning and Architecture, Inc., 358 Hungerford Drive, Rockville, MA 20850, U.S.A.

- COSMIC : Computer Software Management and Information Center, 112 Barrow Hall, University of Georgia, Athens, GA 30602, U.S.A.
- Finite Element News : Robinson and Associates, Horton Road, Woodlands, Wimborne, Dorset, BH21 6NB, U.K.
- ICES : ICES Users Group Inc., P.O. Box 8243, Cranston, RI 02920, U.S.A.
- ICP : International Computer Programs Inc., 9000 Keystone Crossing, Indianapolis, IN 46240,U.S.A.
- NISEE : National Information Service for Earthquake Engineering, 519 Davis Hall, University of California, Berkeley, CA 94720, U.S.A.
- NTIS : National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161, U.S.A.

However, a more practical way to search for this information is to use any of the available on-line search facilities that conduct several international databases. The search strategy of these databases can be summarized as follows:

(1) Specify keywords for the topic to be searched for.

(2) Each relevant database is searched for the qualifying entries of each of these keywords separately. The actual entries are not retrieved but rather data sets containing pointers to these entries are created. These data sets are called the hit sets.

(3) The required entries are the intersection of the hit sets.

This is better clarified by an example. Suppose that a search is

required for "Flow towards wells in a quifers using finite elements". The formulation of the searching strategy for this request is:

Set A represents the set of citations retrieved using the keyword "Flow".

Set B is that using the keyword "Well"

Set C " " " " " "A quifer"

Set D " " " " " "Finite Element"

The required set of citation is thus $A \cap B \cap C \cap D$.

It should be noticed that synonyms must be considered to retrieve all the possible citations. For example, the Set B in fact should be replaced by the union of the sets B1, B2 and B3 defined as:

B1 is the set representing the citations retrieved using the keyword "Well"

B2 " " " " " " " " " " "

"Sinks"

B3 " " " " " " " " " " "

"Sources

$$B = B_1 \cup B_2 \cup B_3$$

and the required set of citations is thus:

$$A \cap (B_1 \cup B_2 \cup B_3) \cap C \cap D$$

1.5 THE STRUCTURE OF THE THESIS

This thesis is concerned with various aspects of the computer implementation of the finite element method. The vast amount of developments of computational algorithms used in finite elements makes it impossible to cover all the methods and techniques used in the computer implementation of finite elements and therefore, an arbitrary selection has to be done. The selection is based on the importance, applicability and the range of applications. In addition to that, problems in ground water hydraulics are solved to demonstrate the versatility and utility of the presented software.

In Chapter 2 an engineering approach for the FEM is presented which gives a logical step by step approach to the basic ideas and concepts of FE analysis. This is followed by a mathematical approach in the next chapter.

In Chapter 4, the computer implementation of FE is considered in greater detail. This includes the types of FE software and the data structures required in FE programming. Afterwards, proposed extensions to the standard Fortran are given which will make FE programming more efficient and easier. The different algorithms for the solution of FE equations are presented. Four different FE programs are then presented as examples of computer implementation of FE on mainframe, mini- and micro-computers. The features of these implementations are extracted and demonstrated. The problems of computer implementation on mini- and micro-computers are highlighted and proposed solutions are given. The CPU time analysis of some test problems is reported in order to know the distribution of processing time among the different modules in a class of FE programs. Pre- and post-processors are discussed in depth

with an attempt to define the functions and types of each. The rest of this chapter is devoted to the selection of FE software based on three different quantitative approaches.

In Chapter 5, the problem of limited stack size in many of the mini- and micro-computers is demonstrated. A proposed solution is given; namely the virtual stack facility. This software is implemented and tested. A new replacement algorithm for a virtual stack is implemented and proved to be more efficient than some of the known replacement algorithms implemented for virtual storage systems.

In Chapter 6, a general programming system for the solution of a wide class of second order partial differential equations based on finite elements is presented. The computational aspects of this software are explained. This software is then used to solve some problems in ground water flow in the next chapter.

In the last chapter conclusions are drawn of this work and the scope for further work proposed.

CHAPTER 2

THE FINITE ELEMENT METHOD:

AN ENGINEERING APPROACH

TABLE OF CONTENTS

2.1 HISTORICAL BACKGROUND

The main concept in the finite element method is to replace a complex continuous system by discretized simpler elements whose behaviour is already known. This concept is very old in the history of the subject. The roots of this concept can be back-dated to Archimedes who used it to find an approximate value of π. He replaced a circle by a polygon in order to compute the perimeter of a circle. This concept is actually a form of one dimensional line elements modelling. To calculate the area of the circle, Archimedes used triangles originating from the centre of the circle to the vertices of the polygon. This could be considered as a two-dimensional triangular element model. Beckmann [1971] gave full details of the calculation of π. A brief discussion of Archimedes's method to compute the areas of plane figures and volumes of solids and the work of other ancient scientists can be found in Hogben [1967].

Despite this very old origin, the finite element method (FEM) in its modern shape was discovered as a generalisation of the matrix methods of Structural Analysis. These methods were used for solving skeletal structural systems like trusses and frames which yield exact solutions. However, in order to solve elastic continua, a similar approach was used as an approximation. One of the methods for such approximation is to use a lattice of framework to model the actual continuum [e.g. Hrenikoff (1941), Yettram and Husain (1966)]. Thus, in this approach a plate loaded in plane like that shown in Figure 2.1 which possesses infinite degrees of freedom is modelled as a frame of a finite number of degrees of freedom which can be solved using the standard structural analysis methods as shown in Figure 2.2.

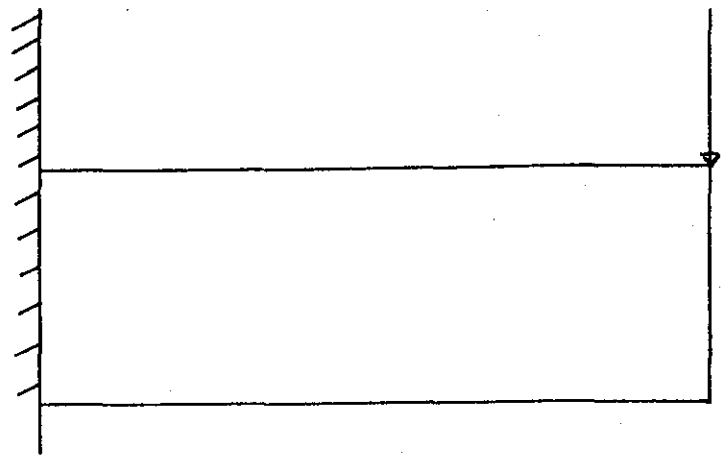


FIGURE 2.1: A plate with in-plane point load

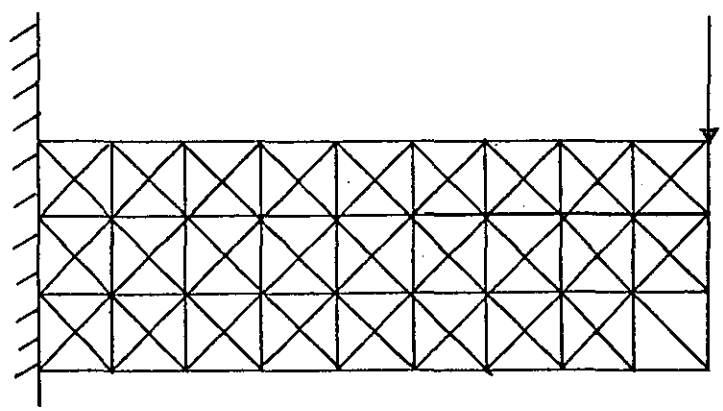


FIGURE 2.2: A lattice of framework model

Another approach which is different in concept was adopted by other scientists and engineers; [Clough, 1960] who introduced the term finite element for the first time. In this approach, instead of approximating the continuum into a framework whose stiffness matrices are known exactly; an approximate value of the stiffness matrix of a triangular element is derived and by an assembly of these triangular elements the whole structure is modelled.

In the body of this chapter a description of the FEM as an extension to the matrix structural analysis is given. This will give one an insight description of the method and the steps used to solve problems using it. In section 2.2, the stiffness method of structural analysis is briefly reviewed. In section 2.3, the assembly process by which the global structure is formed from its individual elements is explained whilst dealing with the boundary conditions is discussed in section 2.4. The solution of the resulting algebraic equations is reviewed in 2.5. The determination of other element data like stresses is given in 2.6. Throughout a simple truss problem is used as a vehicle to exemplify the method. In section 2.7 extensions to other types of problems are presented, whereas section 2.8 presents the conclusions.

2.2 THE STIFFNESS METHOD FOR STRUCTURAL ANALYSIS

The matrix methods of structural analysis give a unified approach to solve structural systems. The most widely used approach is the stiffness method. Naturally it is the advent of digital computers that makes such methods practically applicable. The details of such methods can be found in the standard text books on the subject like [Livesly (1964) and Przemieniecki (1968)]. However, for the sake of completeness, a rather short description is given.

In the matrix methods of Structural Analysis, the stiffness matrix of each member of the structure is calculated. The stiffness matrix relates the applied forces to displacements. The elements of this matrix are functions of the geometry and material properties of that element. Then, a global stiffness matrix for the whole structure is assembled. Equating the external forces vector by the product of the stiffness matrix and the displacement vectors will yield the determination of the unknown displacements. This is expressed in the following matrix equation:

$$f = Kd \ , \tag{2.1}$$

where f is the load vector, K is the global stiffness matrix and d is the unknown displacement vector. Hereafter, a plane truss will be considered to exemplify the various aspects of the finite element process. Consider the truss element shown in Figure 2.3. There are several ways to derive the stiffness matrix of this element. In the early days of the FEM, the stiffness matrix was derived element by element based on its definition as a force-displacement characteristic. For example, since k_{ij} is defined as the force associated with node i that produces a unit displacement at j, then to determine k_{ij} we may

simply impose this unit displacement and calculate the corresponding force. It is usually much easier to derive the stiffness matrix in a local coordinate system rather than in the global one, then using the necessary transformation to relate it to the global coordinate system.

For the truss element the local coordinate system is chosen along the axis of the member itself as shown in Figure 2.3. To derive the stiffness matrix K we note that a truss element is subjected to axial forces only thus we have two displacements along the member and none perpendicular to it. To derive the elements of this matrix we need to derive k_{11} and k_{12} only, while $k_{21}=k_{12}$ according to the reciprocal theorem and k_{22} could be concluded by induction. To calculate k_{11} we know that k_{11} is equal to the force that must be applied at node i in order to produce a unit displacement at the same node along the member axis. From elementary theory of structures, $k_{11} = \frac{EA}{L}$, where E is the Young's modulus of elasticity, A is the cross sectional area of the member and L is its length. At the same time and due to equilibrium k_{21} will be $= -\frac{EA}{L}$. Similarly, $k_{22} = \frac{EA}{L}$. Thus the stiffness matrix in the local coordinate system is,

$$K = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.2}$$

In order to express the stiffness matrix in the global x-y coordinate system, we assume that $\bar{K}$, $\bar{d}$ and $\bar{f}$ are the stiffness matrix, displacement vector and force vector in the global coordinate system, respectively. The displacement vectors d and $\bar{d}$ are related by:

$$d = \lambda \bar{d} , \tag{2.3}$$

where λ is a transformation matrix.

It must be noted that d is a two element vector:

$$d = \begin{Bmatrix} d_i \\ d_j \end{Bmatrix} \tag{2.4}$$

while $\bar{d}$ is a four element one since the displacements are in two dimensions with reference to the global x-y system.

$$\bar{d} = \begin{Bmatrix} \bar{d}_{2i-1} \\ \bar{d}_{2i} \\ \bar{d}_{2j-1} \\ \bar{d}_{2j} \end{Bmatrix} \tag{2.5}$$

Denoting the direction cosines between the x-axis and the $\bar{x},\bar{y}$ axis by ℓ and m respectively, then,

$$d_i = \bar{d}_{2i-1} \cdot \ell + \bar{d}_{2i} m$$

and

$$d_j = \bar{d}_{2j-1}\ \ell + \bar{d}_{2j}\ m \tag{2.6}$$

or in matrix form,

$$d = \begin{bmatrix} d_i \\ d_j \end{bmatrix} = \begin{bmatrix} \ell & m & 0 & 0 \\ 0 & 0 & \ell & m \end{bmatrix} \begin{bmatrix} \bar{d}_{2i-1} \\ \bar{d}_{2i} \\ \bar{d}_{2j-1} \\ \bar{d}_{2j} \end{bmatrix} \tag{2.7}$$

Therefore, the transformation matrix λ is,

$$\lambda = \begin{bmatrix} \ell & m & 0 & 0 \\ 0 & 0 & \ell & m \end{bmatrix} \tag{2.8}$$

If the angle between the x-axis and $\bar{x}$-axis is α then,

$$\ell = \cos\alpha$$

and

$$m = \sin\alpha \tag{2.9}$$

Thus,

$$\lambda = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ 0 & 0 & \cos\alpha & \sin\alpha \end{bmatrix} \tag{2.10}$$

If a virtual displacement vector $\delta\bar{d}$ is introduced on the element then from (2.3) it follows that:

$$\delta d = \lambda \delta \bar{d} \ , \tag{2.11}$$

The resulting virtual work - being a scalar quantity - must be obviously independent on the coordinate system and it follows that:

$$\delta \bar{d}^T \bar{f} = \delta d^T f \ . \tag{2.12}$$

Substituting in (2.12) from (2.11) we get:

$$\delta \bar{d}^T \bar{f} = (\lambda \delta \bar{d})^T f \ , \tag{2.13}$$

or

$$\delta \bar{d}^T \bar{f} = \delta \bar{d}^T \lambda^T f \ , \tag{2.14}$$

i.e.,

$$\delta \bar{d}^T (\bar{f} - \lambda^T f) = 0 \ . \tag{2.15}$$

Since $\delta\bar{d}$ is arbitrary, it follows that,

$$\bar{f} - \lambda^T f = 0 \ . \tag{2.16}$$

Substituting from equation (2.1) for f in equation (2.16) gives,

$$\bar{f} - \lambda^T K d = 0 \ . \tag{2.17}$$

Substituting for d from equation (2.3) into (2.17) gives,

$$\bar{f} - \lambda^T K \lambda \bar{d} = 0 \ ,$$

or

$$\bar{f} = (\lambda^T K \lambda) \bar{d} \ . \tag{2.18}$$

Comparing (2.18) with (2.1) gives,

$$\bar{K} = \lambda^T K \lambda \ , \tag{2.19}$$

substituting for λ, λ^T and K from (2.2) and (2.10) gives,

$$\bar{K} = \frac{EA}{L} \begin{bmatrix} \cos\alpha & 0 \\ \sin\alpha & 0 \\ 0 & \cos\alpha \\ 0 & \sin\alpha \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ 0 & 0 & \cos\alpha & \sin\alpha \end{bmatrix}$$

i.e.,

$$\overline{K} = \frac{EA}{L}\begin{bmatrix} \cos^2\alpha & \sin\alpha\cos\alpha & -\cos^2\alpha & -\sin\alpha\cos\alpha \\ \cos\alpha\sin\alpha & \sin^2\alpha & -\sin\alpha\cos\alpha & -\sin^2\alpha \\ -\cos^2\alpha & -\cos\alpha\sin\alpha & \cos^2\alpha & \sin\alpha\cos\alpha \\ -\sin\alpha\cos\alpha & -\sin^2\alpha & \sin\alpha\cos\alpha & \sin^2\alpha \end{bmatrix} \qquad (2.20)$$

The element axis and the global x-axis are shown in Figure 2.3.

Consider the sample problem in Figure 2.4 which represents a simple truss of constant cross sectional area A and modulus of elasticity E. The first step to solve this problem is to number each node and element in the structure. Figure 2.5 shows the element and node numbers. A global x-y coordinate system is chosen for the whole structure passing through node number 1. The element stiffness matrices for each member is then computed according to equation (2.2b) as follows,

$$K^{(1)} = \frac{EA}{10\sqrt{2}}\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} = K^{(3)}$$

$$K^{(2)} = \frac{EA}{10\sqrt{2}}\begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

$$K^{(4)} = \frac{EA}{10\sqrt{2}}\begin{bmatrix} \frac{1}{2} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Now, each element matrix is formed. We should assemble these matrices to form the overall structure matrix. This procedure is of a general nature in FEM.

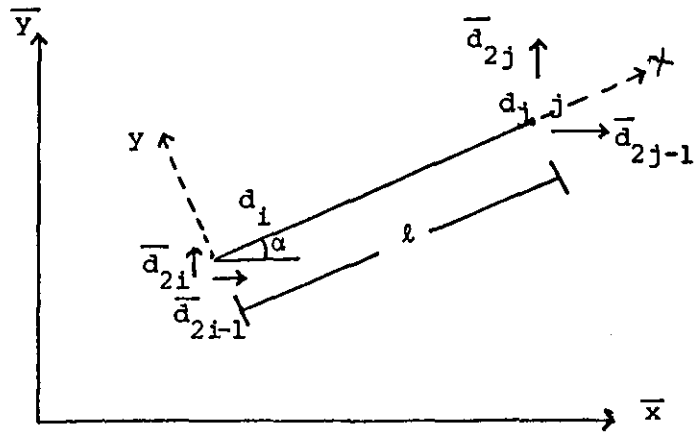


FIGURE 2.3: A truss element

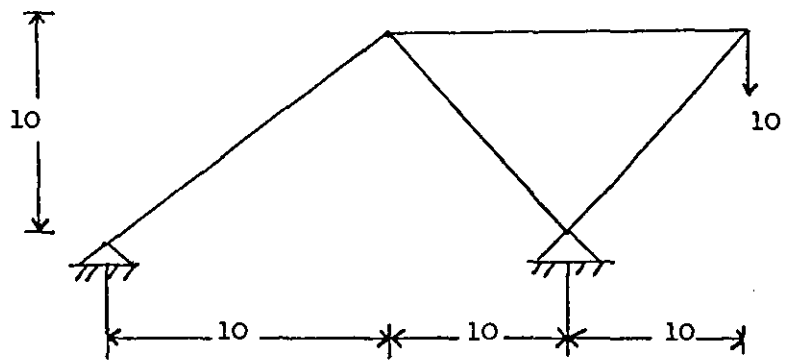


FIGURE 2.4: A truss problem

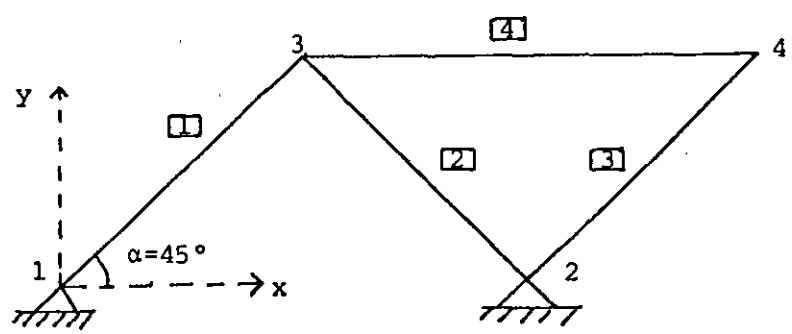


FIGURE 2.5: Nodes and elements numbering

2.3 ASSEMBLY OF ELEMENTS

The main idea of the assemblage of elements is to map the contribution of each element into the global stiffness matrix in a manner that preserves the compatibility at element nodes. In other words, for a node which is common to more than one element; the nodal stiffnesses and nodal loads for all the elements sharing this node are added to obtain the net stiffness and net load at that node. To illustrate this, we assemble the overall stiffness matrix of the considered truss problem.

Since we have four nodes with two degrees of freedom (dof) at each, it results that the global stiffness matrix is of size (8×8). We start by zeroing a (8×8) matrix that will hold the global stiffness matrix. To map elements of the first element matrix $K^{(1)}$ we note that element [1] is connected to nodes 1 and 3. Thus $K^{(1)}$ will be mapped to the corresponding cells in the global stiffness matrix; i.e. to cells corresponding to nodes 1 and 3 only as shown in Figure 2.6. Note that each submatrix K_{11}, K_{13}, K_{31} and K_{33} is a (2×2) matrix.

In the same manner, other element matrices could be assembled in the overall global matrix. This is shown in Figure 2.7. The numerical value of K will therefore be:

$$K^{(1)} = \begin{bmatrix} K_{11}^{(1)} & & K_{13}^{(1)} \\ & & \\ K_{31}^{(1)} & & K_{33}^{(1)} \end{bmatrix}$$

$$K = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} K_{11}^{(1)} & & K_{13}^{(1)} & \\ & & & \\ K_{31}^{(1)} & & K_{33}^{(1)} & \\ & & & \end{bmatrix} \end{array}$$

FIGURE 2.6: Map of $K^{(1)}$ into the overall stiffness matrix

$$K = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} K_{11}^{(1)} & & K_{13}^{(1)} & \\ & K_{22}^{(2)}+K_{22}^{(3)} & K_{23}^{(2)} & K_{24}^{(3)} \\ K_{31}^{(1)} & K_{32}^{(2)} & K_{33}^{(1)}+K_{33}^{(2)}+K_{33}^{(4)} & K_{34}^{(4)} \\ & K_{42}^{(3)} & K_{43}^{(4)} & K_{44}^{(3)}+K_{44}^{(4)} \end{bmatrix} \end{array}$$

FIGURE 2.7: Formation of the global stiffness matrix

$$K = \frac{EA}{10\sqrt{2}} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2}\frac{1}{2} & -\frac{1}{2}\frac{1}{2} & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & -\frac{1}{2}\frac{1}{2} & \frac{1}{2}\frac{1}{2} & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & \frac{1}{2}\frac{1}{\sqrt{2}} & -\frac{1}{2}-\frac{1}{2} & -\frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & \frac{1}{2}-\frac{1}{2} & \frac{1}{2}\frac{1}{2} & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{\sqrt{2}} & 0 & \frac{1}{2}+\frac{1}{\sqrt{2}} & \frac{1}{2}+0 \\ 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & \frac{1}{2}+0 & 0+\frac{1}{2} \end{bmatrix}$$

i.e.,

$$K = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & 1+\frac{1}{\sqrt{2}} & -1 & -\frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{\sqrt{2}} & 0 & \frac{1}{2}+\frac{1}{\sqrt{2}} & \frac{1}{2} \\ 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

At this point it is useful to note some properties of the assembled matrix. The first property is that K is symmetrical. Second, it is a sparse banded matrix. This is due to the structural connectivity of the elements. It is worthwhile to mention that the global stiffness matrix - as the elemental ones - is a singular matrix. This reflects the fact that so far, we did not impose any boundary conditions on the problem, so the structure will have a rigid body motion under any applied loads.

2.4 IMPOSING OF BOUNDARY CONDITIONS

Since our main unknown in the problem is the displacement vector d and since at supports we know the displacements in advance as being zeros, it follows that when solving equation 2.1 we must cater for those known displacements. First let us form the f vector for the nodal loads. In our example, this is quite simple:

$$f = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -10 \end{Bmatrix}$$

We know that the displacements in the x and y directions at nodes 1 and 2 are zeros, thus d should look like:

$$d = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ d_5 \\ d_6 \\ d_7 \\ d_8 \end{Bmatrix}$$

In other words, we have four unknowns only rather than eight. Usually there are two approaches to impose this in the considered FE solution. In the first approach, the equations corresponding to the known displacements are omitted and the remaining equations could then be solved for the actual unknowns. In our example, this means that the first four rows and columns of K are removed and we will be left with:

$$f = Kd \ ,$$

where,

$$f = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ -10 \end{Bmatrix} \ , \qquad d = \begin{Bmatrix} d_5 \\ d_6 \\ d_7 \\ d_8 \end{Bmatrix} \ ,$$

and

$$K = \frac{EA}{10\sqrt{2}} \begin{bmatrix} 1+\frac{1}{\sqrt{2}} & -1 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{2}+\frac{1}{\sqrt{2}} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

which gives the solution vector d:

$$d = \frac{100}{EA} \begin{Bmatrix} \sqrt{2} \\ 0 \\ 1+\sqrt{2} \\ -3-\sqrt{2} \end{Bmatrix} \ .$$

Another method of imposing the boundary conditions on the considered problem is to force the known displacements to be equal to their known values. This is done by multiplying the diagonal terms of K that correspond to a known displacement by a very large number and multiplying the corresponding element in the load vector by the same large number and the corresponding diagonal element in K and the prescribed displacement. Thus, we force the solution to give us as a solution, the prescribed displacement given. This approach is referred to as the penalty modifications for nodal constraints.

In our example we get,

$$K = \frac{EA}{10\sqrt{2}} \begin{bmatrix} \frac{1}{2}\times 10^{20} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2}\times 10^{20} & 0 & 0 \\ 0 & 0 & 1\times 10^{20} & 0 \\ 0 & 0 & 0 & 1\times 10^{20} \end{bmatrix}$$

Other elements of K are unchanged. Since the prescribed displacements in this case are zeros, the corresponding nodal vector f will be:

$$f = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -10 \end{Bmatrix}$$

Solving the equation f=Kd for d will result in practically zero values for the first four elements in d.

2.5 SOLUTION OF EQUATIONS

As it is shown, the FEM will lead to the solution of a set of algebraic equations. These equations could be linear if the original problem is linear by nature otherwise it will be non-linear. There are many methods used to solve the FE algebraic equations. Generally, we can classify two distinct approaches. In the first approach, exact or direct methods of solution are tried while in the second approach, approximate or iterative methods of solution are being used. In the exact methods a solution is guaranteed on the completion of a fixed amount of arithmetical operations, whilst the iterative methods generally involve a repetitive sequence of simple matrix-vector operations in which a guess vector is successively improved until the solution is obtained to a specified accuracy [Evans, 1973].

2.5.1 Direct Methods

Direct methods of solving linear algebraic equations are primarily based on the Gauss elimination method. In a standard Gauss elimination method, the system of equations Ax=b is solved by reducing the matrix A to an upper triangular form with unity values on the main diagonal. Then by backward substitution, the last unknown is first determined and consequently other unknowns are determined by backward substitution [see Fox (1966), for example].

Consider the system of equations,

$$Ax = b \ , \tag{2.21}$$

where A (n×n) is the matrix of coefficients, x (n×1) is the unknown solution vector and b (n×1) is the known vector of constants. In FE structural analysis systems; A is usually the stiffness matrix; x is

the unknown nodal displacement and b is the nodal force.

The elimination of the unknown x_i (i=1,2,...,n) by the Gauss elimination method is done by modifying the elements of the matrix A and the vector b as follows:

$$a_{jk} = a_{jk} - a_{ji} * a_{ik} / a_{ii} \tag{2.22}$$

$$b_j = b_j - a_{ji} * b_i / a_{ii} \ , \tag{2.23}$$

for j=i+1 to n

and k=i+1 to n.

The last equation will be in the form of

$$x_n = b_n / a_{nn}$$

and thus x_n is determined directly while other x's are determined by backward substitution using,

$$x_i = (b_i - \sum_{j=i+1}^{n} a_{ij} x_j) / a_{ii}, \text{ for } i=n-1 \text{ to } 1. \tag{2.24}$$

It is clear that this method will fail if any of the elements a_{ii} become zero during the elimination process. In the meanwhile if the elements a_{ii} are too small, big round-off errors are expected. To avoid this situation pivoting is used. The main idea of pivoting is to re-arrange the equations to be solved so that the elements a_{ii} are chosen to be the largest in absolute value sense among other elements at each reduction step. Pivoting can be done by searching the complete matrix a_{ij} elements for the maximum value and take that to be the pivot. In this case both row and column interchanges are needed and the pivoting process is called complete pivoting. If the search is limited to the largest element per column, pivoting is called partial pivoting and can be done by row interchanges only. In many cases partial pivoting is found to be sufficient to obtain a successful elimination.

In many of the practical FE problems the matrix A is positive definite and symmetrical and pivoting is not required.

Another method of solution of linear equations is that by the Gauss-Jordan method where at each elimination step the variable x_i is eliminated not only from the equations i+1,i+2,...,n as before, but also it is eliminated from the equations 1,2,...,i-1. Thus the coefficient matrix A is reduced to a diagonal form and the solution of the unknowns x_i is determined directly by dividing, i.e. b_i/a_{ii}.

A variant of the reduction of A to a triangular form by elimination is the factorization of A to LU matrices, where L is a lower triangular matrix and U is an upper triangular matrix with unity values among the main diagonal,

$$A = LU \ . \tag{2.25}$$

Thus equation (2.21) will be,

$$LUx = b \ , \tag{2.26}$$

In the Crout algorithm [Stabrowski, (1981)] an auxiliary vector y is calculated during the decomposition of A from the equation,

$$Ly = b \ , \tag{2.27}$$

and by backward substitution, x is determined from:

$$Ux = y \ , \tag{2.28}$$

The elements of L, U, y and x could be determined from:

$$\ell_{ij} = a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} u_{kj} \ , \quad i \geq j, \ i=1,2,\ldots,n \tag{2.29}$$

For j=1, $\ell_{il}=a_{il}$

$$u_{ij} = \frac{1}{\ell_{ii}}\left(a_{ij} - \sum_{k=1}^{i-1} \ell_{ik} u_{kj}\right) \ , \ i<j, \ j=2,3,\ldots,n \tag{2.30}$$

and $u_{ij}=1$ for i=j.

For i=1 $$u_{1j} = \frac{a_{1j}}{\ell_{11}} = \frac{a_{1j}}{a_{11}}$$

$$y_i = \frac{1}{\ell_{ii}} \left(b_i - \sum_{k=1}^{i-1} \ell_{ik} y_k\right) \tag{2.31}$$

and

$$x_i = y_i - \sum_{k=i+1}^{n} u_{ik} x_k \ . \tag{2.32}$$

This method could be used to solve a family of systems of equations having the same A matrix simultaneously. In this case the elements of L and U will be the same and elements of y and x could be calculated for all the r.h.s. at the same time. Equations (2.31) and (2.32) will be:

$$y_{im} = \frac{1}{\ell_{ii}} \left(b_{im} - \sum_{k=1}^{i-1} \ell_{ik} y_{km}\right) \ , \tag{2.33}$$

and

$$x_{im} = y_{im} - \sum_{k=i+1}^{n} u_{ik} x_{km} \ , \tag{2.34}$$

where m runs from 1 to r; the total number of r.h.s.

It is worth mentioning that although the coefficient matrix A is factorized into two matrices of the same order, the storage required is minimal. This is accomplished by storing the non-zero elements only. The ones on the diagonal of the U matrix are not stored as well. In fact the elements of U, except the diagonal, are stored in place of the zeros of the matrix L as shown below,

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \rightarrow \begin{bmatrix} \ell_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ \ell_{21} & \ell_{22} & u_{23} & \cdots & u_{2n} \\ \vdots & \vdots & & & \vdots \\ \ell_{n1} & \ell_{n2} & \cdots & \cdots & \ell_{nn} \end{bmatrix}$$

A similar method of the Crout's algorithm is that known as Doolittle's method. Here, the matrix A is decomposed into an LU pair

where L has the ones on its diagonal instead of U as before. The Choleski's method requires that the diagonal elements of both L and U are the same i.e. $\ell_{ii}=u_{ii}$ $\forall$ i. If the coefficient matrix A is positive definite i.e. $z^TAz>0$ for all non-zero vectors z and $A^T=A$ i.e. A is symmetrical, the resulting factorization yields $U=L^T$ and $A=LL^T$.

It should be noted, however, that the Gauss elimination is not implemented as such in FE systems. This is due to the fact that: first the stiffness matrix is very sparse and it is normally symmetric and positive definite. Another factor that should be considered is that when solving the FE algebraic equations the stiffness matrix size is usually large enough so that it will not fit in the computer's fast memory (core) and the use of auxiliary storage will be necessary. The methods used for computer implementation of these methods will be discussed in Chapter 4 of this thesis.

2.5.2 Indirect Methods

Indirect methods for solving systems of linear equations are primarily based on the Gauss-Seidel iteration. If the system of equations to be solved is Ax=b, and an initial approximate solution vector is $x^{(1)}$ then the iterative procedure is defined by:

$$x_i^{(n+1)} = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(n+1)} - \sum_{j=i+1}^{N} \frac{a_{ij}}{a_{ii}} x_j^{(n)} \quad , n=1,2,... \qquad (2.35)$$

where the superscript (n+1) denotes the iteration cycle number n+1.

A sufficient condition for convergence is that [Gerald, 1978]

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{N} |a_{ij}| \quad , i=1,2,...,N \quad . \qquad (2.36)$$

Re-writing equation (2.35) as

$$x_i^{(n+1)} = \frac{1}{a_{ii}} \left\{ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(n+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(n)} \right\} \qquad (2.37)$$

$$= x_i^{(n)} + \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(n+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(n)} \right). \qquad (2.38)$$

If the second term in (2.38) is multiplied by some factor ω we get the successive over-relaxation method (SOR) which will converge much faster than the standard Gauss-Seidel method and then the iterative equations become,

$$x_i^{(n+1)} = x_i^{(n)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(n+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(n)} \right) \qquad (2.39)$$

The over-relaxation factor ω must be between 1 and 2, the optimum value is problem dependent.

It is a fact that iterative methods of solution are not popular in the well-known finite element computer packages [Rao, (1982)]. In a survey of 36 of the most well-known FE computer packages by [Noor, (1981)], none uses iterative methods to solve systems of linear equations resulting during the solution process. Although the iterative methods of solution requires less memory space than the elimination methods and usually better for small size equation systems (500-1000) fitting into the RAM of the computer [Stabrowski, (1981)] but for large systems when using backing storage is unavoidable the direct methods are better. This is almost certainly due to the lack of knowledge on how many iterations are necessary to achieve an acceptable solution. Another disadvantage of iterative methods is the choice of a good over-relaxation factor (ω value) which is very sensitive in relation to the rate of convergence. There is no guarantee of convergence for

unsymmetric problems. Another very important factor is that in iterative methods of solving linear systems of equations the re-solution is almost as expensive as the solution itself although you do possess a near starting solution. In engineering design, re-solution is frequent for different right hand sides [e.g. different cases of loading in a structural analysis problem] or in non-linear analysis.

2.5.3 Solution of Non-Linear Equations

If the problem to be solved is not linear, the resulting system of equations will be non-linear. Solution of a system of non-linear equations is not, in general, possible by direct methods. Iterative procedures are used to solve such equations. Many methods have been devised [e.g. Ortega and Rheinboldt, (1970)] samples of which are given only which represent those used in practical FE programs.

Consider the set of equations,

$$\begin{aligned} f_1(X) &= 0 \\ f_2(X) &= 0 \\ f_3(X) &= 0 \\ &\vdots \\ f_n(X) &= 0 \end{aligned} \tag{2.40}$$

where X is the vector of unknowns $=\{x_1, x_2, \ldots, x_n\}$ and $f_1, f_2, \ldots, f_n$ are non-linear functions. These set of equations can be written as,

$$F(X) = 0 \; . \tag{2.41}$$

The problem is to find the solution vector X with sufficient accuracy. The main essence of the iterative procedures to be described is to start by a guessing solution vector $X^{(0)}$ which is close enough to the exact solution X* and try to generate a sequence of vectors $X^{(1)}, X^{(2)}, \ldots X^{(m)}$ that converges to X*.

The simplest and oldest method to solve the system of equations (2.41) is the fixed point iterative method. In this method the equations are re-written in the form,

$$X = G(X) \tag{2.42}$$

The initial guess vector $X^{(0)}$ is chosen and subsequently updated using the iterative process defined by,

$$X^{(i+1)} = G(X^{(i)}) \ , \tag{2.43}$$

where the superscript i denotes the iteration cycle. The iteration is proceeded until an assigned criterion is satisfied. Practically, two limits are set to accept the solution vector in iterative procedures: (i) a preset error limit is satisfied and (ii) a maximum number of iterations is not exceeded. It is clear that this method is of linear convergence. However, it is possible to accelerate the convergence by using the most updated values of the components of the vector X. In other words, when computing $x_m^{(i+1)}$ the values $x_1^{(i+1)}, x_2^{(i+1)}, \ldots, x_{m-1}^{(i+1)}$ are used instead of the values $x_1^{(i)}, x_2^{(i)}, \ldots, x_{m-1}^{(i)}$.

Another method of solving the system of equations defined by (2.41) is the Newton's method. In this case the iteration process is defined by:

$$X^{(i+1)} = X^{(i)} - J^{(i)-1} F(X^{(i)}) \ , \tag{2.44}$$

where $X^{(i+1)}$ is the updated solution vector at iteration number i and $J^{(i)-1}$ is the inverse of the Jacobian matrix computed at $X^{(i)}$. The Jacobian $J^{(i)}$ is defined by:

$$J^{(i)} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1^{(i)}} & \frac{\partial f_1}{\partial x_2^{(i)}} & \cdots & \frac{\partial f_1}{\partial x_n^{(i)}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1^{(i)}} & \frac{\partial f_n}{\partial x_2^{(i)}} & \cdots & \frac{\partial f_n}{\partial x_n^{(i)}} \end{bmatrix} \tag{2.45}$$

It is known that Newton's method is of quadratic convergence. However, the convergence of this method depends on the initial guessing vector $X^{(0)}$. This is particularly noticed for highly nonlinear equations. Practically the Jacobian is not inverted at each iteration cycle since matrix inversion is expensive in terms of computer cost. Rather, the Jacobian is evaluated and the system of equations:

$$J^{(i)}C^{(i)} = F(X^{(i)}) \ , \tag{2.46}$$

are solved for $C^{(i)}$ which represents the correction vector to update the solution vector $X^{(i)}$. Finally, the updated vector $X^{(i+1)}$ is computed as:

$$X^{(i+1)} = X^{(i)} - C^{(i)} \ . \tag{2.47}$$

There are many methods which are all based on Newton's method with some modifications. Among these is the damped Newton's method. In this method we introduce a damping factor $\alpha^{(i)}$ to be multiplied by the correction vector $C^{(i)}$ such that the residual error after iteration cycle (i+1) is always less than that in iteration cycle (i), i.e.,

$$||F(X^{i+1})|| < ||F(X^{(i)})|| \tag{2.48}$$

Another method which is based on Newton's method is that by Broyden [1965] where the Jacobian matrix is replaced by an approximate one which is updated at each iteration cycle. The iterative procedure is defined by,

$$X^{(i+1)} = X^{(i)} - A^{(i)-1}F(X^{(i)}) \ . \tag{2.49}$$

The approximate Jacobian $A^{(i)}$ is computed from:

$$A^{(i)-1} = A^{(i-1)-1} + \frac{(S^{(i)} - A^{(i-1)-1}y^{(i)})S^{(i)T}A^{(i-1)-1}}{S^{(i)T}A^{(i-1)-1}y^{(i)}} \tag{2.50}$$

and,

$$S^{(i)} = X^{(i)} - X^{(i-1)} \tag{2.51}$$

$$y^{(i)} = F(X^{(i)}) - F(X^{(i-1)}) \ . \tag{2.52}$$

It is clear that in this method the solution of equations defined by (2.46) is replaced by simple matrix operations (multiplication, addition and subtraction) which are executed faster as in equations (2.50), (2.51) and (2.52). On the other hand, the quadratic convergence of Newton's method is degraded.

2.6 DETERMINATION OF OTHER ELEMENT DATA

The previous procedures will result in the determination of the unknown displacements at nodes. Usually it is more important to know the stress values at different points in the structure. The procedure could be summarized as follows:

(i) Stress-strain relationship:

Generalization of Hook's law results in,

$$\sigma = D\varepsilon \tag{2.53}$$

where $\sigma = [\sigma_x \sigma_y \sigma_z \sigma_{xy} \sigma_{yz} \sigma_{zx}]^T$ is the stress vector

ε is the strain vector $= [\varepsilon_x \varepsilon_y \varepsilon_z \varepsilon_{xy} \varepsilon_{yz} \varepsilon_{zx}]^T$

D is the elasticity matrix and ε is the strain vector.

For a linearly elastic material, D is given by [Pr zemieniecki, (1968)],

$$D = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{2.54}$$

(ii) Strain-displacement relationship,

$$\varepsilon = Bd \tag{2.55}$$

where ε is the strain vector as before

d is the general displacement vector $= \begin{Bmatrix} u \\ v \\ w \end{Bmatrix}$

B is a matrix of differential operators given by,

$$B = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{bmatrix} \qquad (2.56)$$

But since what is available is the node displacements only, then, we must relate the displacements within the element to the nodal ones. This is done using a function of the position called the shape function or the interpolation function. Thus denoting the calculated nodal displacements with d^e then the displacement field d is given by:

$$d = N^e d^e \qquad (2.57)$$

The choice of the shape functions $N^e(x,y,z)$ is in the hands of the user and depends on the type of element under consideration [Davis, (1980)]. The shape functions are said to be conforming functions if they satisfy the following conditions [Majid, (1980)]:

(i) The displacement and the resulting derived strains must be continuous functions (C_o continuity).

(ii) The shape function must give rise to uniform strains within the element (constant first derivative).

(iii) No strains should be produced due to rigid body movement.

(iv) The function must satisfy the conditions of compatibility inside the element, at the nodes where elements meet and along the sides.

The role of the shape functions in the FE modelling is important

and usually only polynomials are used due to their ease of manipulation symbolically and computationally. It is clear that combining equations (2.53) through (2.57) the stresses can be computed.

2.7 EXTENSIONS TO NON-STRUCTURAL APPLICATIONS

After the FEM was used successfully for linear problems in 2-D structural analysis, the natural extension to 3-D analysis was developed [Argyris, (1964)]. Non-linear problems, both in geometry and/or material, were considered afterwards. After discovering that FEM could be interpreted in terms of variational techniques, the method was used to solve problems outside the structural domain. General field problems were solved [Zienkiewicz and Cheung, (1965)]. Another dimension added to the range of applications that could be handled by FE was after it was discovered that FEM can be formulated as one of the methods of weighted residuals (MWR) such as Galerkin's method. This idea paved the way to solve problems for which variational principles do not exist or were difficult to find. The text of [Oden, (1972)] gives a comprehensive account of the application of FE to non-linear problems. A classical reference of the FEM that covers a wide range of applications is that of [Zienkiewicz, (1977)]. Some of the recent applications are: Biomechanics [Gallagher, et al, (1982)], Coupled Problems [e.g. Borsetto et al, (1981) and Hinton et al (1981)]. In an excellent paper by [Zienkiewicz and Kelly, (1982)] the role of finite elements as a unified problem solving and information transfer method has been stressed.

2.8 CONCLUSIONS

The FEM was originated by structural engineers to solve complex structural problems. The main motivation for the development of the method was the challenge problems posed by relatively high-speed, jet-powered aircraft. High-speed digital computers coupled with space exploration money for basic research helped to process the present advance in FEM in the last two decades [Kaldjian et al, (1982)]. Generally, three main types of problems were found in Engineering and Applied Mathematics that could be handled by FEM. These are [Huebner and Thornton, (1982)]:

(i) Equilibrium problems:

These are systems that do not vary with time. Examples are: linear structural analysis and steady state fluid flow in porous media.

(ii) Eigenvalue problems:

These are equilibrium problems whose solution often requires the determination of natural frequencies and modes of vibration of solids and fluids. Examples are: Stability of structures and modes of vibration of dynamic systems.

(iii) Propagation problems:

These are time dependent problems. It could arise from the above mentioned categories when a time variation is considered. Examples are: creep analysis of structures and non-steady flow of fluids in porous media.

The main steps in FE analysis could be summarized as:

(i) The discretization of the domain into appropriate finite elements

(ii) Evaluation of each element stiffness and load matrices

(iii) Assemblage of element stiffness and load matrices into the global stiffness and load matrices.

(iv) Application of suitable constraints or boundary conditions.

(v) Solution of the resulting simultaneous equations for the unknown nodal variables.

(vi) Evaluation of other element quantities of interest in the problem.

The element stiffness matrix of an element could be derived by direct methods in simple cases like truss elements, constant strain triangles (CST) etc. In these cases, it is relatively easy to derive the terms of the stiffness matrix in algebraic form. For more complicated cases it is usually better, and sometimes the only possible alternative to evaluate the element stiffness matrix using numerical integration. Of the many available numerical integration schemes, the Gauss-Legendre method is the most popular technique used in FEM. This is primarily due to its high accuracy and ease of computer implementation.

As a general view of the current research in the FEM, three major fields of specialization could be identified. In the engineering aspects, the development of new elements specially for shell structures is one of the many active areas of research. Extending the FEM for new applications could also be observed. Solution of interfacing and coupling problems is a trend in the literature. Frequently new methods for solving free surface and moving boundary problems occur. Methods for non-linear analysis are being refined and developed that give acceptable results with reasonable costs. In the mathematical aspects, error analysis specially the discretization error is an area of active research. The theoretical study of the convergence of the method for non-linear analysis is frequently discussed. In the computational aspects new methods and algorithms are necessary for: data structures, equation

solving, programming systems, the use of microcomputers and systems of microprocessors and the use of advanced computer architectures such as pipeline processors, single instruction multiple data streams (SIMD) and multiple instructions multiple data streams (MIMD). Preprocessors and postprocessors are also urgently required in FEM. Although several systems are becoming available, it seems that many more are required.

CHAPTER 3

THE FINITE ELEMENT METHOD: A MATHEMATICAL APPROACH

TABLE OF CONTENTS

3.1 PRELIMINARIES

The aim of this section is to introduce some of the basic concepts in linear algebraic theory, partial differential equations and variational calculus which are required in other parts of the thesis. The illustration is by no means neither complete nor comprehensive.

3.1.1 Basics of Linear Algebraic Theory

(1) Vector Spaces

Given a non-empty set X, the totality of vectors that can be constructed by scalar multiplication and vector addition from the vectors in X is called a vector space. The scalar multiplication must satisfy the following conditions:

$$\text{(i)} \quad \alpha(x+y) = \alpha x+\alpha y \ , \tag{3.1}$$

$$\text{(ii)} \quad (\alpha+\beta)x = \alpha x+\beta x \ , \tag{3.2}$$

$$\text{(iii)} \quad (\alpha\beta)x = \alpha(\beta x) \ , \tag{3.3}$$

$$\text{(iv)} \quad 1.x = x \ , \tag{3.4}$$

where, x,y are two vectors in X and α,β are arbitrary scalars.

The vector addition must satisfy the following conditions:

$$\text{(i)} \quad x+y = y+x \ , \tag{3.5}$$

$$\text{(ii)} \quad x+(y+z) = (x+y)+z \ , \tag{3.6}$$

(iii) There exists the zero element $O \in X$ such that:

$$O+x = x+O \ , \quad \forall\, x \in X \tag{3.7}$$

(iv) $\forall x \in X$; there exist a negative $-x$ such that:

$$x+(-x) = O \ . \tag{3.8}$$

A set of vectors is said to span the space if they can generate the vector space by the use of these operations. If the set consists of the least number of vectors that span the space, it is called a basis

for the space. The number of vectors in the basis is called the dimensionality of the space.

Assuming n basis vectors exist in X, an n-dimensional space can be generated. Any subset of m basis vectors; m<n; forms the basis of an m-dimensional subspace. A necessary and sufficient condition that a set of n vectors be confined to a subspace is that the set be linearly dependent, i.e., there exists coefficients c_i, not all zeros, such that,

$$\sum_{i=1}^{n} c_i x_i = 0 \ . \tag{3.9}$$

Otherwise the set is linearly independent.

(2) Matrices and Sets of Linear Equations

Some notations and properties of a square real matrix A (n×n) which are relevant to the solution of the set of linear equations defined in matrix form as:

$$Ax = b \ , \tag{3.10}$$

where x is the unknown vector (n×1) and b is the known vector of constants (n×1) are as follows:

- The matrix A is said to be non-singular if $|A| \neq 0$ where $|A|$ is the determinant of the matrix A.
- A is symmetric if $A=A^T$ where A^T is the transpose of A.
- A is orthogonal if $A^{-1}=A^T$ where A^{-1} is the inverse of A.
- A is null if $a_{ij}=0$ $\forall i$ $\forall j$.
- A is diagonally dominant if $|a_{ii}| \geq \sum_{i \neq j} |a_{ij}|$ $\forall i$
- A is irreducible if there exists no permutation transformation PAP^{-1} which reduces A to the form:

$$\begin{bmatrix} P & O \\ R & Q \end{bmatrix} \ ,$$

where P and Q are square submatrices of order p and q, respectively; p+q=n and O is a (p×q) null matrix.

- Two square matrices A and B are similar if there exists a permutation matrix P such that: $B=P^{-1}AP$.
- The vector space generated by the rows of A is called the row space and that by the columns, the column space. The dimensionality of these two spaces is the same and is called the rank of A.

Considering the set of equations Ax=b, the sufficient condition that this system of n nonhomogeneous linear equations in n unknowns has a unique solution is that A^{-1} exists, i.e. rank[A]=n. When the rank of A is r<n, the system has a solution if it satisfies the consistency condition:

$$\text{rank}[A|b] = \text{rank}[A]$$

i.e. if b is subject to the same linear dependencies as the rows of A. In this case the equations are permuted such that the first r are linearly independent and in the partitioned form the system will be:

$$\begin{array}{c} \\ r \\ n-r \end{array} \begin{array}{c} \begin{array}{cc} r & n-r \end{array} \\ \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \end{array} \begin{bmatrix} x_1 \\ \hline x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ \hline b_2 \end{bmatrix} \qquad (3.11)$$

and the solution of the r independent unknowns will be,

$$x_1 = A_{11}^{-1}b_1 - A_{11}^{-1}A_{12}x_2 \ . \qquad (3.12)$$

In fact this re-arrangement of the system of equations is one of the ways to apply the boundary condition in a finite element analysis as demonstrated in Chapter 2 of this thesis. The known displacements, which are the boundary conditions in structural analysis applications,

are arranged as x_2 in the partitioned form of the system of equations. However, as previously stated this method of imposing the boundary conditions in FE programs is not commonly used.

(3) Determinants

The determinant of a matrix A (n×n) is denoted by $|A|$ and is defined by the Laplace expansion:

$$|A| = \sum(-1)^k a_{1\alpha}a_{2\beta}\cdots a_{n\nu} \tag{3.13}$$

where $\alpha,\beta,\ldots,\nu$ represent one of the permutations of the natural numbers $1,2,\ldots,n$. The total number of terms is n! The exponent k is used only to determine the sign of the permutation, i.e., the term is negative if the permutation is odd otherwise the term is positive. The permutation is said to be odd if the number of pairs of integers which are out of natural order is odd. Thus, the permutation 13245 is an odd permutation since there is only one sequence out of natural order; 32. Since each of the numbers $1,2,\ldots,n$ appears only once as a row subscript and once as a column subscript, any term of the expansion contains only one element from each row and column of A. This is best illustrated by an example of a 3×3 determinant,

$$A = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

The different permutations with their signs are:

123, -132, -213, 231, 312 and 321.

Thus, the expansion is:

$$|A| = a_{11}a_{22}a_{33} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} .$$

Some of the important properties of determinants are as follows:

(1) Taking the transpose does not change the determinant.

(2) If one of the rows of a determinant is all zeros; the determinant value is zero.

(3) Interchanging two rows changes the sign of the determinant.

(4) If two rows of a determinant are the same, the determinant is equal to zero.

(5) A determinant with two proportional rows is equal to zero.

(6) A determinant remains unchanged if to the elements of one of its rows we add corresponding elements of another row multiplied by the same number.

(4) Quadratic Forms

A function of n variables $x_1,\ldots,x_n$ in quadratic form is defined as:

$$F(x_1,\ldots,x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j \qquad (3.14)$$

$$= a_{11}x_1^2 + a_{12}x_1x_2 + \ldots + a_{1n}x_1x_n + \ldots$$
$$+ a_{21}x_2x_1 + \ldots + a_{2n}x_2x_n + a_{n1}x_nx_1 + \ldots$$
$$+ a_{nn}x_n^2 .$$

This form is usually encountered whenever the energy of a continuous system is expressed in a set of discretized coordinates of the system. It is more convenient to write the quadratic form F in matrix notation as:

$$F(x_1,\ldots,x_n) = x^T A x , \qquad (3.15)$$

where,

$$x = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} .$$

A quadratic form $F(x_1,\ldots,x_n)$ is positive definite if F is non-

negative for all possible combinations of real x_i (i=1,...,n) and if F is zero only when every x_i is zero. A property of a positive definite quadratic form is that the determinants of the coefficients a_{ij} and all of its principal minors are positive, i.e.,

$$a_{11} > 0 , \quad \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} > 0, \ \ldots, \quad \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & & & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{vmatrix} > 0 .$$

If the variables $x_1,\ldots,x_n$ are subjected to a linear transformation defined by:

$$x_i = \sum_{k=1}^{n} q_{ik}y_k , \quad i=1,2,\ldots,n \tag{3.16}$$

or in matrix notation:

$$x = Qy . \tag{3.17}$$

Thus, the quadratic form F will be:

$$F = x^T A(Qy) ,$$

since $x^T = y^T Q^T$, then,

$$F = y^T(Q^T AQ)y$$

or
$$F = y^T By \tag{3.18}$$

where,
$$B = Q^T AQ . \tag{3.19}$$

If the linear transformation defined by Q is non-singular, the rank of $Q^T AQ$ will be equal to the rank of A and thus the rank of a quadratic form does not change under a non-singular linear transformation.

If the quadratic form F is changed by a system of non-singular linear transformations to a sum of squares of the variables, it is called its canonical form, i.e.:

$$\begin{aligned} F &= b_1y_1^2 + b_2y_2^2 + \ldots + b_ny_n^2 \\ &= y^T By . \end{aligned} \tag{3.20}$$

In this case, the matrix B will be a diagonal matrix.

It is always possible to reduce any quadratic form to canonical form by means of non-singular linear transformations.

(5) Eigenvalues and Eigenvectors

Given a matrix A (n×n), the eigenproblem is to find the eigenvalues (also called characteristic roots or latent roots), λ_i and the eigenvectors x such that:

$$Ax = \lambda x \ , \tag{3.21}$$

where the eigenvalues λ are the roots of the characteristic equation given by:

$$|A-\lambda I| = 0 \ . \tag{3.22}$$

For each λ_i, if $x \neq 0$ and satisfy,

$$(A-\lambda I)x = 0 \ , \tag{3.23}$$

then x is an eigenvector of A corresponding to the eigenvalue λ_i.

The spectral radius $\rho(A)$ is defined as:

$$\rho(A) = \max_{1 \leq i \leq n} |\lambda_i| \ . \tag{3.24}$$

In other words the spectral radius is the largest eigenvalue of the matrix A. Since the eigenvalues may be complex numbers, in general, then $|\lambda_i|$ is $\sqrt{a_i^2+b_i^2}$, where $\lambda_i = a_i+\sqrt{-1}\, b_i$.

(6) The Calculus of Matrices

If the elements of a matrix A are functions of n independent variables $x_1, x_2, \ldots, x_n$ then the matrix A is a matrix function of $x_1, \ldots, x_n$. The derivatives of A with respect to any of these variables is done by differentiating every element with respect to the same variables, e.g.,

$$\frac{\partial A}{\partial x_2} = \begin{bmatrix} \frac{\partial a_{11}}{\partial x_2} & \frac{\partial a_{12}}{\partial x_2} & \cdots & \frac{\partial a_{1n}}{\partial x_2} \\ \vdots & \vdots & & \vdots \\ \frac{\partial a_{n1}}{\partial x_2} & \cdots & & \frac{\partial a_{nn}}{\partial x_2} \end{bmatrix} \tag{3.25}$$

In the same sense, the integral of a function matrix A exists only when the integral of each element of the matrix A exists.

A quadratic functional $I(x_1,\ldots,x_n)$ is defined in matrix form as:

$$I(x_1,x_2,\ldots,x_n) = \tfrac{1}{2}x^TAx - x^Tb \ , \tag{3.26}$$

where $x_1,x_2,\ldots,x_n$ are n independent variables, A is a symmetric square matrix and b is a column vector.

To make the functional stationary, which is used frequently in developing the element characteristics in the FEM, the n derivatives of I with respect to $x_1,\ldots,x_n$ must be equated to zeros.

$$[\frac{\partial I}{\partial x}] = \frac{\partial I}{\partial x_i} = 0 \ , \ i=1,2,\ldots,n \tag{3.27}$$

since,

$$\frac{\partial}{\partial x_i} x^TAx = 2Ax \ , \tag{3.28}$$

and

$$\frac{\partial}{\partial x_i} x^Tb = b \tag{3.29}$$

substituting we get:

$$\frac{\partial I}{\partial x_i} = Ax - b = 0, \quad i=1,2,\ldots,n \tag{3.30}$$

This is in fact a system of linear equations.

(7) Norms

The norm of a vector $x = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix}$ is denoted by $||x||$ and is a real nonnegative number such that:

$$||x|| = 0 \quad \text{iff } x=0$$

$$||cx|| = |c|\ ||x|| \quad \text{for all scalars } c$$

$$||x_1+x_2|| \leq ||x_1|| + ||x_2|| \text{ for all vectors } x_1 \text{ and } x_2.$$

There are many norms for vectors the most commonly used ones are:

$$||x||_1 = \sum_{i=1}^{n} |x_i| \tag{3.31}$$

$$||x||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2} \tag{3.32}$$

$$||x||_\infty = \max_i |x_i| \tag{3.33}$$

The matrix norms can be defined in a similar manner as:

$$||A||_1 = \max_j \sum_i |a_{ij}| \tag{3.34}$$

$$||A||_2 = (\text{maximum eigenvalue of } A^T A)^{\frac{1}{2}} \tag{3.35}$$

$$||A||_\infty = \max_i \sum_j |a_{ij}| \tag{3.36}$$

The 2-norm is often called the spectral norm, and for any real symmetric matrix A (n×n) this norm is $\rho(A)$. The matrix norm satisfies similar properties to those of vector norms, i.e.,

$$||cA|| = |c|\ ||A||$$

$$||Ax|| \leq ||A||\ ||x||$$

$$||AB|| \leq ||A||\ ||B||$$

$$||A+B|| \leq ||A|| + ||B||$$

(8) Computational Errors in the Solution of Linear Algebraic Equations

Due to the finite word length of computers all the numerical computations are done using finite arithmetic precision. Consequently, errors occur during the numerical computations. If the system of linear equations are ill-conditioned, i.e. the matrix of coefficients is nearly

singular the effect of these computational errors became more serious. In order to conclude simple bounds on these errors consider the system of equations defined by:

$$Ax = b \ . \tag{3.37}$$

Assume perturbation Δx in x due to a variation Δb in b, then,

$$A(x+\Delta x) = b+\Delta b \ ,$$

or $$A(\Delta x) = \Delta b$$

or $$\Delta x = A^{-1}\Delta b \ , \tag{3.38}$$

whence,

$$||\Delta x|| \leq ||A^{-1}||\ ||\Delta b|| \tag{3.39}$$

but since $$||A||\ ||x|| \geq ||b||$$

by division the bound on the relative perturbation can be found as:

$$\frac{||\Delta x||}{||x||} \leq ||A||\ ||A^{-1}||\ \frac{||\Delta b||}{||b||} \tag{3.40}$$

The value $||A||\ ||A^{-1}||$ is called the condition number of A and is denoted k(A). Since the norm $||AB|| \leq ||A||\ ||B||$ and putting $B=A^{-1}$ then: $||AA^{-1}|| \leq ||A||\ ||A^{-1}||$.

The norm of the identity matrix is 1 thus $||A||\ ||A^{-1}|| \geq 1$., i.e. k(A) is ≥ 1 always. Thus,

$$\frac{||\Delta x||}{||x||} \leq \beta \frac{||\Delta b||}{||b||} \tag{3.41}$$

where $\beta \geq 1$.

This shows that if β is a large number then small variations in b (which can be due to computational errors) will result in relatively large variations in the solution.

In a similar manner it is possible to study the effect of perturbation in the matrix A itself on the solution. Assuming the perturbation ΔA then:

$$(A+\Delta A)\hat{x} = b \ , \tag{3.42}$$

where $\hat{x}$ is the perturbed solution. A measure of the error in the solution may be $e=x-\hat{x}$ and the residual r is defined by: $r=b-A\hat{x}$,

$$\therefore \quad r = \Delta A\hat{x} \tag{3.43}$$

$$\therefore \quad ||r|| \leq ||\Delta A||\ ||\hat{x}|| \tag{3.44}$$

$$\therefore \quad Ae = A(x-\hat{x}) = b-A\hat{x} = r$$

$$\therefore \quad e = A^{-1}r$$

$$\therefore \quad ||e|| \leq ||A^{-1}||\ ||r|| \ . \tag{3.45}$$

Substituting for $||r||$ we have,

$$\frac{||e||}{||\hat{x}||} \leq ||A^{-1}||\ ||\Delta A||\ \frac{||A||}{||A||}$$

$$\text{i.e.} \quad \frac{||e||}{||\hat{x}||} \leq k(A)\ \frac{||\Delta A||}{||A||} \ , \tag{3.46}$$

which indicates that depending on the value of the condition number k(A) the relative error in the solution can be very large due to variations in the matrix of coefficients A.

3.1.2 Preliminaries of Partial Differential Equations

In many of the problems in engineering and science the physical phenomena to be studied can be formulated mathematically as a partial differential equation or as a set of these equations. In these problems two or more independent variables exist and the rates of changes of the dependent variables are related to these independent variables through differential operators.

The order of a partial differential equation (PDE) is the order of the highest derivative in the equation. Thus, the Laplace equation

$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0$ is a second order partial differential equation. A partial differential equation is said to be linear if the highest degree of the variables and their derivatives is one. Thus the Laplace equation is linear while an equation like: $\frac{\partial^2 u^2}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y,u)$ is a non-linear partial differential equation of the second order.

The problems which will be solved using the FEM within this thesis can be written in the general form:

$$L(\phi) - f = 0 \ , \tag{3.47}$$

where f is a known function and L is a differential operator. The solution is required for some domain D bounded by the surface Σ and ϕ is the field variable (dependent). ϕ can be a scalar function, e.g. hydraulic head in a fluid problem or a vector e.g. displacement in a structural mechanics problem. The differential operator L may be linear or non-linear. Many of the physical problems can be modelled using a second-order differential equation. This may be because many of the physical problems deal with one form of the conservation principle e.g. energy, mass or momentum conservation. Considering n dimension space, L can be written as:

$$L(\ ) = \sum_{i=1}^{n} A_i \frac{\partial^2(\ )}{\partial x_i^2} + \sum_{i=1}^{n} B_i \frac{\partial(\ )}{\partial x_i} + C_i(\ ) + D, \tag{3.48}$$

where the coefficients A_i, B_i, C_i and D may be functions. This operator is linear if A_i, B_i, C_i and D are functions of the independent variables $(x_1, x_2, \ldots, x_n)$ only. L is quasilinear if A_i, B_i, C_i and D are functions of x_i and the dependent variable as well as the first derivatives of the dependent variable. Considering the case of two independent variables x and y, a general second-order partial differential equation may be written as,

$$A\frac{\partial^2\phi}{\partial x^2} + 2B\frac{\partial^2\phi}{\partial x\partial y} + C\frac{\partial^2\phi}{\partial y^2} = D\left(x,y,\phi,\frac{\partial\phi}{\partial x},\frac{\partial\phi}{\partial y}\right) , \qquad (3.49)$$

where A,B and C are functions of x and y only.

This equation can be linear or non-linear depending on the terms in D. However, it is classified as:

Elliptic equation if $B^2-AC < 0$

Parabolic equation if $B^2-AC = 0$

Hyperbolic equation if $B^2-AC > 0$.

Since A,B and C may be functions of x and y, this classification may change from point to point in the solution region. A well known example of these three classes are:

Laplace $\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0$, which is elliptic,

Diffusion equation $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$ which is parabolic,

and Wave equation $\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$ which is hyperbolic.

For parabolic and hyperbolic equations, the solution domains are usually open, while for the elliptic equations it is closed. In general, elliptic equations are associated with steady-state phenomena and require a knowledge of values of the unknown function or its derivatives on the boundary of the region of interest. On the other hand, hyperbolic equations are generally associated with propagation problems. The parabolic equations are generally associated with problems in which the quantity of interest varies slowly.

In order to be able to get a solution for a PDE, some boundary conditions must be specified. The important types of boundary conditions are:

(1) Type I usually known as a Dirichlet boundary condition is where the function u(x,y) is prescribed along the boundary i.e. u is given on the boundary ∂R. If the function assumes zeros along the boundary, this condition is termed homogeneous Dirichlet.

(2) Type II which is known as a Neumann condition is where the normal derivative $\frac{\partial u}{\partial n}$ is specified along the boundary.

(3) Type III which is known as a mixed condition is where the function u(x,y) and its normal derivative $\frac{\partial u}{\partial n}$ are specified along the boundary ∂R.

For an elliptic operator L, the problem is said to be properly posed when only one of these conditions holds at each point of the boundary.

Consider the PDE in a 2-D region R expressed in the form,

$$L(\phi) = f \ , \tag{3.50}$$

where L is a differential operator defined as:

$$L(\phi) \equiv A \frac{\partial^2 \phi}{\partial x^2} + B \frac{\partial^2 \phi}{\partial x \partial y} + C \frac{\partial^2 \phi}{\partial y^2} + D(x,y,\phi, \frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}) \tag{3.51}$$

and f(x,y) is a given function of position.

Then the operator L is said to be self-adjoint iff the expression,

$$\iint_R \psi \, L(\phi) dxdy - \iint_R \phi \, L(\psi) \, dxdy \tag{3.52}$$

is a function of ϕ, ψ and their derivatives evaluated on the boundary. For homogeneous boundary conditions, L is self-adjoint iff

$$\iint_R \psi L(\phi) dxdy = \iint_R \phi \, L(\psi) dxdy \ . \tag{3.53}$$

The operator L is said to be positive definite, iff for all ϕ:

$$\int \int_R \phi \, L(\phi) \, dxdy \geq 0 \ . \tag{3.54}$$

The equality to zero occurs only iff $\phi \equiv 0$.

3.1.3 Preliminaries of Variational Calculus

The aim of this part is to introduce some principles and notation and to show the correspondence between differential equations and the variational problem formulation. The variational calculus (or calculus of variations) is concerned with the determination of minima or maxima of functionals. Thus, the basic problem in variational calculus is to find the function $\phi(x)$ which makes the functional I defined by:

$$I[\phi] = \int_{x_1}^{x_2} F(x,\phi,\phi_x,\phi_{xx})dx \ , \tag{3.55}$$

stationary.

In order to solve (3.55) for the value of $\phi(x)$ which makes the functional $I[\phi]$ stationary we notice that a necessary condition is to have:

$$\delta I[\phi] = 0 \ , \tag{3.56}$$

where the variational operator δ is similar to the differential operator d. The operation of variation is commutative with respect to both integration and differentiation, i.e.,

$$\delta\left(\int F dx\right) = \int (\delta F).dx \tag{3.57}$$

and

$$\delta \frac{d\phi}{dx} = \frac{d}{dx}\delta\phi \tag{3.58}$$

The variation in ϕ, i.e. $\delta\phi$, is defined as an infinitesimal, arbitrary change in ϕ for a fixed value of the variable x, i.e. for $\delta x=0$. The variation of a functional or a function of several variables is defined in a similar manner to the calculus definition of a total differential, i.e.,

$$\delta F = \frac{\partial F}{\partial x}.\delta x + \frac{\partial F}{\partial \phi}.\delta\phi + \frac{\partial F}{\partial \phi_x}.\delta\phi_x + \frac{\partial F}{\partial \phi_{xx}}.\delta\phi_{xx} \tag{3.59}$$

since $\delta x=0$, then,

$$\delta F = \frac{\partial F}{\partial \phi}.\delta\phi + \frac{\partial F}{\partial \phi_x}.\delta\phi_x + \frac{\partial F}{\partial \phi_{xx}}.\delta\phi_{xx} \ , \tag{3.60}$$

and substituting in (3.56) we get,

$$\delta I[\phi] = 0 = \int_{x_1}^{x_2} \left(\frac{\partial F}{\partial \phi}\cdot \delta\phi + \frac{\partial F}{\partial \phi_x}\cdot \delta\phi_x + \frac{\partial F}{\partial \phi_{xx}}\cdot \delta\phi_{xx}\right) dx. \tag{3.61}$$

Integrating the second and third terms by parts we get:

$$\int_{x_1}^{x_2} \frac{\partial F}{\partial \phi_x}\cdot \delta\phi_x \, dx = \int_{x_1}^{x_2} \frac{\partial F}{\partial \phi_x}\cdot \delta\left(\frac{\partial \phi}{\partial x}\right) dx = \int_{x_1}^{x_2} \frac{\partial F}{\partial \phi_x}\cdot \frac{\partial}{\partial x}(\delta\phi)\, dx$$

$$= \frac{\partial F}{\partial \phi_x}\cdot \delta\phi \Big|_{x_1}^{x_2} - \int_{x_1}^{x_2} \frac{d}{dx}\left(\frac{\partial F}{\partial \phi_x}\right)\cdot \delta\phi \, dx \tag{3.62}$$

and:

$$\int_{x_1}^{x_2} \frac{\partial F}{\partial \phi_{xx}}\cdot \delta\phi_{xx}\, dx = \int_{x_1}^{x_2} \frac{\partial F}{\partial \phi_{xx}}\cdot \frac{\partial}{\partial x}(\delta\phi_x)\, dx$$

$$= \frac{\partial F}{\partial \phi_{xx}}\cdot \delta\phi_x \Big|_{x_1}^{x_2} - \int_{x_1}^{x_2} \frac{d}{dx}\left(\frac{\partial F}{\partial \phi_{xx}}\right)\cdot \delta\phi_x \, dx$$

$$= \frac{\partial F}{\partial \phi_{xx}}\cdot \delta\phi_x \Big|_{x_1}^{x_2} - \frac{d}{dx}\left(\frac{\partial F}{\partial \phi_{xx}}\right)\cdot \delta\phi \Big|_{x_1}^{x_2} + \int_{x_1}^{x_2} \frac{d^2}{dx^2}\left(\frac{\partial F}{\partial \phi_{xx}}\right)\delta\phi \, dx \tag{3.63}$$

$$\therefore \quad \delta I[\phi] = \int_{x_1}^{x_2} \left[\frac{\partial F}{\partial \phi} - \frac{d}{dx}\left(\frac{\partial F}{\partial \phi_x}\right) + \frac{d^2}{dx^2}\left(\frac{\partial F}{\partial \phi_{xx}}\right)\right] \delta\phi\, dx$$

$$+ \left[\frac{\partial F}{\partial \phi_x} - \frac{d}{dx}\left(\frac{\partial F}{\partial \phi_{xx}}\right)\right]\cdot \delta\phi \Big|_{x_1}^{x_2}$$

$$+ \left[\left(\frac{\partial F}{\partial \phi_{xx}}\cdot \delta\phi_x\right]\right|_{x_1}^{x_2} = 0 . \tag{3.64}$$

Since $\delta\phi$ is arbitrary, each term must vanish individually,

$$\frac{\partial F}{\partial \phi} - \frac{d}{dx}\left(\frac{\partial F}{\partial \phi_x}\right) + \frac{d^2}{dx^2}\left(\frac{\partial F}{\partial \phi_{xx}}\right) = 0 , \tag{3.65}$$

and $$\left[\frac{\partial F}{\partial \phi_x} - \frac{d}{dx}\left(\frac{\partial F}{\partial \phi_{xx}}\right)\right] \Big|_{x_1}^{x_2} = 0 , \tag{3.66}$$

and $$\frac{\partial F}{\partial \phi_{xx}} \Big|_{x_1}^{x_2} = 0 . \tag{3.67}$$

Equation (3.65) is the governing differential equation for the given

problem and is called the Euler or Euler-Lagrange equation while the other two equations (3.66) and (3.67) give the associated boundary conditions and are called NATURAL boundary conditions. If they are satisfied they are called free boundary conditions. If the natural boundary conditions are not satisfied, we must have:

$$\delta\phi(x_1) = 0 \ , \ \delta\phi(x_2) = 0$$

and
$$\delta\phi_x(x_1) = 0, \ \delta\phi_x(x_2) = 0 \tag{3.68}$$

These are called the geometric or essential or forced boundary conditions.

To illustrate, consider the functional I[y] defined by:

$$I[y] = \int_a^b \sqrt{1+(y')^2}\,dx \ , \tag{3.69}$$

where $y(a)=y_0$ and $y(b)=y_1$.

What we need is to find the function g that minimizes I[y]. In this case, the functional defines the distance between the two points a and b along the curve y=f(x). The obvious solution of this problem is a straight line connecting the two points a and b where $y(a)=y_0$ and $y(b)=y_1$. In this case

$$F \equiv F(x,y,y') = (1+(y')^2)^{\frac{1}{2}} \tag{3.70}$$

$$\frac{\partial F}{\partial y} = 0 \quad \text{and} \quad \frac{\partial F}{\partial y'} = \tfrac{1}{2}[1+(y')^2]^{-\frac{1}{2}}.2y'. \tag{3.71}$$

The Euler equation is, therefore,

$$-\frac{d}{dx}\ \frac{y'}{(1+y'^2)^{\frac{1}{2}}} = 0 \ , \tag{3.72}$$

and by integrating once, we have,

$$\frac{y'}{(1+y'^2)^{\frac{1}{2}}} = A$$

or

$$(y')^2 = A^2(1+y'^2)$$

i.e.

$$(y')^2 = \frac{A^2}{1-A^2} = B^2$$

i.e., $y' = B$.

Integrating once more, we obtain,

$$y = Bx + c \ , \tag{3.73}$$

which is a straight line as expected.

In the case of two variables x,y, the functional $I[\phi]$ will be of the form:

$$I[\phi] = \iint_R F(x,y,\phi,\phi_x,\phi_y,\phi_{xx},\phi_{xy},\phi_{yy})\,dxdy \tag{3.74}$$

and the corresponding Euler equation is:

$$\frac{\partial^2}{\partial x^2}\left(\frac{\partial F}{\partial \phi_{xx}}\right) + \frac{\partial^2}{\partial x \partial y}\left(\frac{\partial F}{\partial \phi_{xy}}\right) + \frac{\partial^2}{\partial y^2}\left(\frac{\partial F}{\partial \phi_{yy}}\right)$$

$$- \frac{\partial}{\partial x}\left(\frac{\partial F}{\partial \phi_x}\right) - \frac{\partial}{\partial y}\left(\frac{\partial F}{\partial \phi_y}\right) + \frac{\partial F}{\partial \phi} = 0 \ . \tag{3.75}$$

3.2 APPROXIMATE SOLUTIONS FOR PARTIAL DIFFERENTIAL EQUATIONS

3.2.1 Introduction

Obtaining exact analytical solutions to most of the problems expressed in partial differential equations is usually not possible. Only a few problems with regular geometry can be solved by direct integration methods using, for example, separation of variables or Fourier or Laplace transformation techniques. It is therefore a necessity to try to find approximate solutions for the PDE. One of the approaches to approximate solutions is the use of perturbation methods. However, since these methods are basically useful when the problem contains relatively small nonlinear terms, their applicability is limited.

The advent of computers make the numerical approximate solutions more attractive and easier. It can be said that the three currently outstanding methods for obtaining approximate numerical solutions of high accuracy are the method of weighted residuals, the finite difference method, and the finite element method [Huebner and Thornton, 1982]. It is worthwhile to mention that these methods can be related to each other as we shall be demonstrating later in this chapter, where the finite element method can be derived as a special method of weighted residuals. The choice of a particular technique is a function of many parameters concerning the problem to be solved, the required accuracy, the availability of software and hardware suitable for the technique and the cost.

In what follows a brief presentation is given for the methods of weighted residuals and finite differences.

3.2.2 The Method of Weighted Residuals

The method of weighted residuals is an approximate technique used for solving PDE's. Consider the boundary-value problem defined by,

$$L(\phi) = f \text{ in } R \ , \tag{3.76}$$

subject to the boundary conditions,

$$\phi = g(s) \ , \tag{3.77}$$

on some part C_1 of the boundary, and

$$\frac{\partial \phi}{\partial n} + q(s)\phi = h(s) \tag{3.78}$$

on the remainder C_2.

Assume an approximate solution $\hat{\phi}$ is to be found, then the difference from the exact solution ϕ is called the residual r, thus,

$$r = L(\hat{\phi}) - f \tag{3.79}$$

The approximate solution $\hat{\phi}$ is expressed in terms of a suitable complete set of linearly independent trial functions ψ_i which are chosen to satisfy the boundary conditions as ϕ,

$$\phi = \sum_{i=1}^{n} C_i \psi_i \ . \tag{3.80}$$

The method of weighted residuals is based on the determination of the C_i parameters in such a way that the weighting average of the residual over the whole domain vanishes. This is accomplished by choosing n linearly independent weighting functions w_i and the weighted residual is therefore

$$\int_R r \, w_i \, dR = 0 \quad , i=1,2,\ldots \tag{3.81}$$

Once these weighting functions are specified, and substituting in equations (3.80) we get a system of equations for the parameters C_i. These equations are linear algebraic if the operator L is linear otherwise they are non-linear. Moreover, if the dependent variable

in the problem to be solved is a function of both spatial and temporal variables, the c_i will be functions of time and the resulting equations will be ordinary differential equations rather than algebraic ones.

There are many ways to choose the weighting functions w_i and, consequently many methods of weighted residuals. The most popular methods are:

(i) The collocation method, where the weighting function is the delta function. Thus,

$$w_i(x,y) = \delta(x-x_i, y-y_i) \tag{3.82}$$

This means that we are forcing the residual to vanish at specified points $(x_1,y_1),\ldots,(x_n,y_n)$. These points are called the collocation points. In this way the c_i parameters can be determined by solving the resulting n equations. In practice, it is often that m collocation points are chosen where m>>n. Thus, the resulting system of equations will be overdetermined and the solution can be obtained by a least squares approach.

(ii) The least squares method where the parameters c_i are chosen to minimize the residual r in a least square sense. In this approach the integral,

$$I = \iint_R r^2 dxdy \ , \tag{3.83}$$

is to be minimized with respect to the unknown parameters c_i, i.e.,

$$\frac{\partial I}{\partial c_i} = 0 \ , \ i=1,2,\ldots \tag{3.84}$$

These equations are then used to solve for the c_i.

(iii) The Galerkin method where the weighting functions w_i are chosen to be the same as the trial functions. This approach is the most

popular and, indeed, it will be used in the derivation of the FEM equations within this thesis as explained in Chapter 7.

3.2.3 The Finite Difference Method

The finite difference method is an approximating technique used to solve partial differential equations numerically. It is amongst the first methods used in this context and is a well established method. The main idea of the method is to approximate the derivatives by difference quotients over a small interval i.e. $\frac{\partial \phi}{\partial x}$ is replaced by $\frac{\delta \phi}{\delta x}$ where δx is small enough. Practically, to use this technique to solve a PDE in a region R in two dimensions, a system of rectangular meshes formed by two sets of equally spaced lines, one set is parallel to the x axis and the other to the y axis, are overlaid on the region R as shown in Figure (3.1). The points of intersection of the parallel lines are called mesh points (also named: grid, lattice or nodal points). The solution of the PDE is determined at these points.

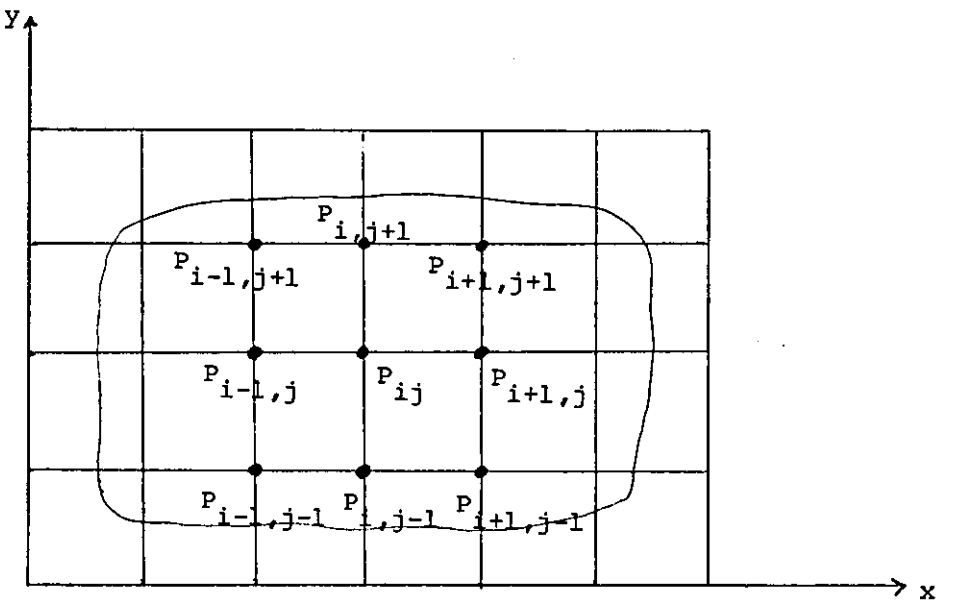


FIGURE 3.1: Finite difference grid

The main idea is to approximate the derivatives at each point P_{ij} by the difference quotients expressed in terms of the function values at the neighbour points to P_{ij} as explained later. Ultimately, this process will result in n algebraic equations for the n unknowns which are $\phi_1, \phi_2, \ldots, \phi_n$ at the nodal points. The accuracy of the solution can be improved as usual by refining the mesh or by expressing the derivatives more accurately in terms of finite differences.

In the case of parabolic or hyperbolic equations, we notice that the solution proceeds from each time value to the next time step and the finite difference approximation is, therefore, applied in both spatial and temporal planes.

Derivative Approximations

There are many approximations for the derivatives. They stem from approximating the Taylor expansion in the neighbourhood of a point (x_o, y_o) as follows:

$$\phi(x_o+\xi,\ y_o+\eta) = \phi(x_o,y_o) + \sum_{j=1}^{k} \frac{1}{j!}\left(\xi\frac{\partial}{\partial x} + \eta\frac{\partial}{\partial y}\right)^j \phi\Big|_{\substack{x=x_o \\ y=y_o}} + \frac{1}{(k+1)!}\left(\xi\frac{\partial}{\partial x} + \eta\frac{\partial}{\partial y}\right)^{k+1} \phi\Big|_{\substack{x=x_o+r\xi \\ y=y_o+r\eta}} \qquad (3.85)$$

for some $0 \leq r \leq 1$.

Considering the first derivatives only, we can write,

$$\phi(x_o+h,y_o) \simeq \phi(x_o,y_o) + h\,\frac{\partial\phi}{\partial x} \qquad (3.86)$$

or

$$\left.\frac{\partial\phi}{\partial x}\right|_{x_o,y_o} \simeq \frac{1}{h}\{\phi(x_o+h,y_o) - \phi(x_o,y_o)\} \qquad (3.87)$$

This is the forward difference approximation for the derivative $\frac{\partial\phi}{\partial x}$. Similar expressions can be obtained for $\frac{\partial\phi}{\partial y}$:

$$\left.\frac{\partial\phi}{\partial y}\right|_{x_o,y_o} \simeq \frac{1}{h}\{\phi(x_o,y_o+h) - \phi(x_o,y_o)\} \qquad (3.88)$$

The backward difference approximation for the derivatives are:

$$\left.\frac{\partial\phi}{\partial x}\right|_{x_o,y_o} \approx \frac{1}{h}\{\phi(x_o,y_o)-\phi(x_o-h,y_o)\} \tag{3.89}$$

and $$\left.\frac{\partial\phi}{\partial y}\right|_{x_o,y_o} \approx \frac{1}{h}\{\phi(x_o,y_o)-\phi(x_o,y_o-h)\} \tag{3.90}$$

Expanding $\phi(x_o+h,y_o)$ and $\phi(x_o-h,y_o)$ about (x_o,y_o) and subtracting gives the central difference approximation for the derivative $\frac{\partial\phi}{\partial x}$:

$$\left.\frac{\partial\phi}{\partial x}\right|_{x_o,y_o} \approx \frac{1}{2h}\{\phi(x_o+h,y_o)-\phi(x_o-h,y_o)\} \tag{3.91}$$

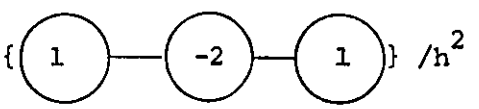
Similarly,

$$\left.\frac{\partial\phi}{\partial y}\right|_{x_o,y_o} \approx \frac{1}{2h}\{\phi(x_o,y_o+h)-\phi(x_o,y_o-h)\}. \tag{3.92}$$

Adding the expansions of $\phi(x_o+h,y_o)$ and $\phi(x_o-h,y_o)$ about (x_o,y_o) gives the finite difference approximation of $\frac{\partial^2\phi}{\partial x^2}$:

$$\left.\frac{\partial^2\phi}{\partial x^2}\right|_{x_o,y_o} \approx \frac{1}{h^2}\{\phi(x_o+h,y_o)-2\phi(x_o,y_o)+\phi(x_o-h,y_o)\}. \tag{3.93}$$

It is convenient to represent this equation in a molecular form as:



It is obvious that the local truncation error in the forward and backward difference approximation is of order h while in the central difference it is of order h^2.

To illustrate the use of these derivative approximations consider the elliptic PDE for the torsion of a long solid elastic cylinder:

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + 2 = 0 , \tag{3.94}$$

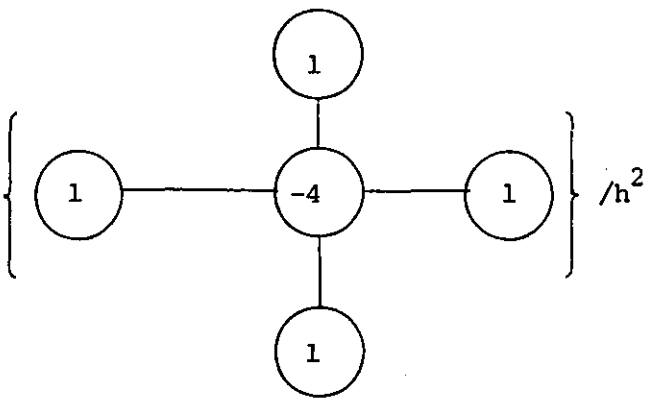
The derivatives $\frac{\partial^2\phi}{\partial x^2}$ and $\frac{\partial^2\phi}{\partial y^2}$ at a general point P_{ij} (Figure 3.1) are approximated as before and the resulting finite difference equations will be of the form:

$$\frac{1}{h^2}[\phi_{i+1,j}-2\phi_{i,j}+\phi_{i-1,j}] + \frac{1}{h^2}[\phi_{i,j+1}-2\phi_{i,j}+\phi_{i,j-1}] + 2 = 0$$

or

$$\frac{1}{h^2}[\phi_{i+1,j}+\phi_{i-1,j}+\phi_{i,j+1}+\phi_{i,j-1}-4\phi_{i,j}] = -2 \qquad (3.95)$$

In a molecular form this equation is represented by:



In the case of parabolic and hyperbolic PDE's the discretization process is done spatially and temporally as shown in the following sections.

Explicit Method for Parabolic and Hyperbolic PDE's

An explicit method in finite differences means that in each finite difference equation, at step i, for example, one unknown nodal value is expressed directly in terms of the known nodal values in previous steps. To illustrate this method consider the parabolic PDE:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}\ ,\ 0<x<1 \text{ and } t>0\ . \qquad (3.96)$$

One of the finite difference approximations for this equation may be:

$$\frac{1}{k}\{u_{i,j+1}-u_{i,j}\} = \frac{1}{h^2}\{u_{i+1,j}-2u_{i,j}+u_{i-1,j}\} + O(k+h^2) \qquad (3.97)$$

Where the time steps are k and the x steps are h i.e. x=0,h,2h,... and t=0,k,2k,... . Let $r=\frac{k}{h^2}$ then the finite difference equation can be written as:

$$u_{i,j+1} \simeq u_{i,j} + r(u_{i+1,j}-2u_{i,j}+u_{i-1,j}) \ . \tag{3.98}$$

It is obvious that the solution at the (j+1) interval is determined by using the values at the j^{th} interval. This is why this method is called explicit. Given the boundary conditions at t=0, the values of u at t=k can be determined and so on. A condition for the convergence of this procedure is that $r\leq\frac{1}{2}$ i.e. the obtained solution $\hat{u}$ will converge to the exact solution u as h tends to 0. This finite-difference scheme is also numerically unstable for $r>\frac{1}{2}$ [Smith, 1969].

In the case of hyperbolic PDE's like the wave equation:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} \ , \ t>0 \ , \tag{3.99}$$

Using the same technique as above the explicit formulation for the finite differences will be:

$$u_{i,j+1} = q^2 u_{i-1,j}+2(1-q^2)u_{i,j}+q^2 u_{i+1,j}-u_{i,j-1} \tag{3.100}$$

where $q=\frac{k}{h}$. This method is convergent and stable when $q\leq 1$ [Smith, 1969].

Recall again that for a parabolic PDE the ratio of time step k to the spatial step h must be $r=\frac{k}{h^2}\leq\frac{1}{2}$. This means that $k\leq\frac{1}{2}h^2$ to get a valid approximation. This is not convenient since it necessitates that the time step must be very small. If, for example when h is .1 then k must be $\leq .005$. This motivates the introduction of a method which is convergent and stable for all values of r which is the Crank-Nicolson method.

Crank-Nicolson Implicit Method for Parabolic PDE's

This method is an implicit one, i.e. the finite difference equation contains two or more unknown values at step i in terms of the known values at step (i-1). Applying this implicit difference equation at each nodal point a system of simultaneous algebraic equations will result and the solution of which gives the values of the unknowns at all the points in step i.

Crank and Nicolson used an average value of the finite-differences for the term $\frac{\partial^2 u}{\partial x_2^2}$ in the time steps j+1 and j. Thus the parabolic equation $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$ is now approximately,

$$\frac{1}{k}(u_{i,j+1}-u_{i,j}) = \frac{1}{2h^2}\{u_{i+1,j+1}-2u_{i,j+1}+u_{i-1,j+1}+u_{i+1,j}-2u_{i,j} + u_{i-1,j}\} + O(k^2+h^2) ,$$

or,

$$-ru_{i-1,j+1}+2(r+1)u_{i,j+1}-ru_{i+1,j+1} \approx ru_{i-1,j}+2(1-r)u_{i,j}+ru_{i+1,j} , \qquad (3.101)$$

where $r = \frac{k}{h^2}$ as before.

Note that the lefthand side of the difference equation now contains three unknowns instead of one as in the explicit method. This means that at each time step a system of N equations must be solved. However, for many practical problems the matrix of coefficients is independent of the time step.

A more general finite difference approximation that combines both the implicit and the explicit forms for the considered parabolic equation is the weighted average approximation, where:

$$\frac{1}{k}\{u_{i,j+1}-u_{i,j}\} \approx \frac{1}{h^2}\{\theta(u_{i+1,j+1}-2u_{i,j+1}+u_{i-1,j+1})+(1-\theta)(u_{i+1,j}-2u_{i,j}+u_{i-1,j})\} . \qquad (3.102)$$

where $0 \leq \theta \leq 1$.

When $\theta=0$ we get the explicit approximation, for $\theta=\frac{1}{2}$ the Crank-Nicolson and for $\theta=1$, a fully implicit backward time-difference method. The approximation is unconditionally valid for $\frac{1}{2}\leq\theta\leq 1$ but for $0\leq\theta<\frac{1}{2}$ the ratio $r = \frac{k}{h^2}$ must be $\leq \frac{1}{2(1-2\theta)}$.

Dealing with Derivative Boundary Conditions

If some of the boundary conditions are expressed in terms of the derivatives of the dependent variable, then fictitious nodal points are considered to approximate the derivative boundary condition on the boundary itself. This condition can be expressed as central-difference quotients (say) using the fictitious nodes. To illustrate this consider the grid shown in Figure (3.2). Assuming that derivative boundary conditions are specified along the two horizontal sides the fictitious nodes are introduced as shown circled.

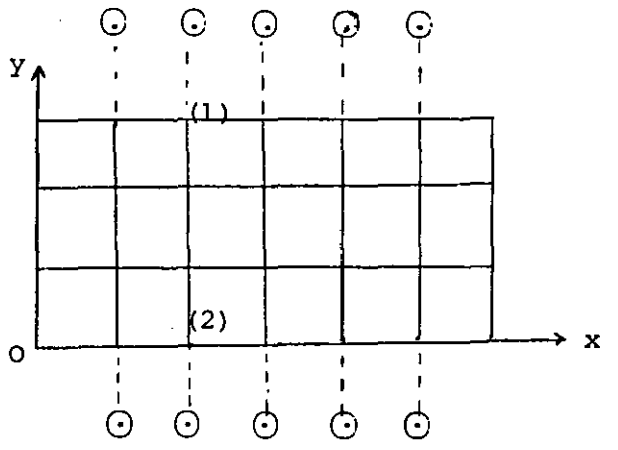


FIGURE 3.2: Finite difference grid with derivative boundary conditions at sides (1) and (2). Fictitious nodes are circled.

Handling of Curved Boundaries

If the boundary of the region is curved or, in general, cannot be overlaid exactly by the rectangular mesh the previous equations cannot

be used at these boundary points. Let us consider the general case where the region near the point u_o is irregular as shown in Figure (3.3). The curved boundary AB does not coincide with the normal grid points. To approximate the derivatives at the point O which is the closest one to the boundary BA where the function value u is

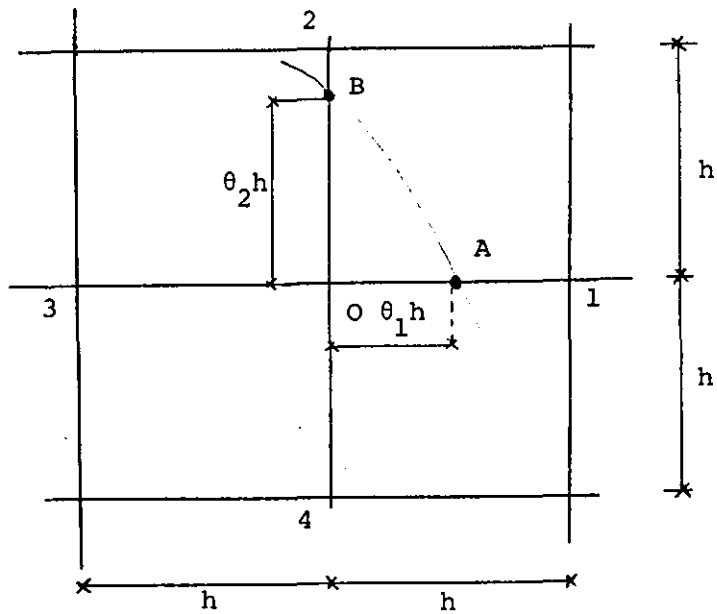


FIGURE 3.3: Curved boundary finite difference

prescribed we use the Taylor expansion for u_A and u_3

$$u_A = u_o + (\theta_1 h)\frac{\partial u_o}{\partial x} + \tfrac{1}{2}(\theta_1 h)^2 \frac{\partial^2 u_o}{\partial x^2} + O(h^3)$$

$$u_3 = u_o - h\frac{\partial u_o}{\partial x} + \tfrac{1}{2} h^2 \frac{\partial^2 u_o}{\partial x^2} + O(h^3)$$

Thus:

$$\frac{\partial u_o}{\partial x} \simeq \frac{1}{h}\left\{\frac{u_A}{\theta_1(1+\theta_1)} - \frac{(1-\theta_1)u_o}{\theta_1} - \frac{\theta_1 u_3}{1+\theta_1}\right\} \qquad (3.103)$$

and

$$\frac{\partial^2 u_o}{\partial x^2} \simeq \frac{2}{h^2}\left\{\frac{u_A}{\theta_1(1+\theta_1)} + \frac{u_3}{1+\theta_1} - \frac{u_o}{\theta_1}\right\} \qquad (3.104)$$

Similar expressions can be obtained for $\frac{\partial u_o}{\partial y}$ and $\frac{\partial^2 u_o}{\partial y^2}$.

Improving the Accuracy of Solutions

There are several approaches to improve the accuracy of solutions obtained by finite difference methods. The main approaches can be divided into:

(i) Mesh refinement:

It is expected, as usual, that the finer the finite difference mesh, the more accurate the solution. This h-version of finite differences will, however, result in ever increasing number of equations since the number of grid points is proportional to $\frac{1}{h^2}$.

(ii) Richardson Extrapolation:

This approach can be used if the discretization error can be estimated as being proportional to the mesh length and two estimated solutions are obtained, in order to get a more accurate one.

Let u be the exact solution of the P.D.E. and u_1 and u_2 are approximate solutions at the same nodal points but with mesh lengths h_1 and h_2 respectively. If the discretization error is proportional to h^p then:

$$u-u_1 = C\, h_1^p \tag{3.105}$$

and

$$u-u_2 = C\, h_2^p \tag{3.106}$$

where C is some constant. Elimination of C gives,

$$\frac{u-u_1}{u-u_2} = \frac{h_1^p}{h_2^p}$$

or

$$u = \frac{h_2^p u_1 - h_1^p u_2}{h_2^p - h_1^p} \tag{3.107}$$

In the case of the five-point star finite difference approximation for the Laplace equation, the discretization error for a rectangular region with smooth known boundary values is known to be proportional to h^2, i.e.

p is 2 in this case. Assume that the problem is solved twice with the mesh length halved, i.e. $h_2 = \frac{1}{2}h_1$, then:

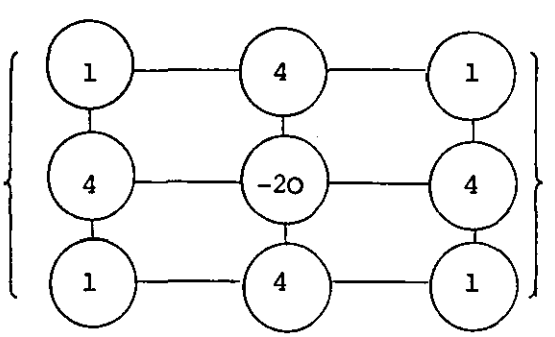
$$u = u_2 + \frac{1}{3}(u_2 - u_1) \quad . \tag{3.108}$$

If the value of p is not known, an estimate can be done with the expense of a third solution set u_3.

This approach will be employed in the general finite element programming system presented in Chapter 6 of this thesis where the time discretization is treated using finite differences rather than finite elements as done spatially.

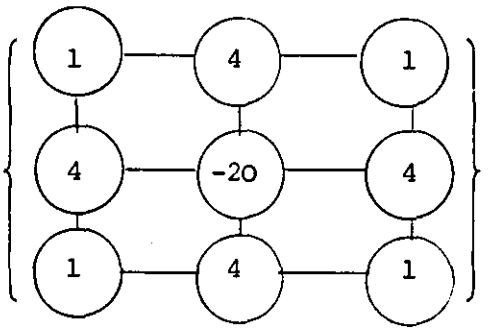
(iii) Using higher accuracy finite difference equations:

The accuracy of the approximate solutions obtained by finite difference methods can be improved by representing the P.D.E. by a higher-order finite difference approximation designed to minimize the truncation errors. This of course, will increase the number of nodal values at each step. Many of these formulae have been devised. For example, the following molecule, known as the nine-point molecule, has a truncation error of the order of h^4 rather than the five-point which has a truncation error of the order of h^2. For the Poissons equation $\nabla^2 u = f$.



$$u = 6h^2 f + \tfrac{1}{2}h^4 \nabla^2 f$$

If f is a constant then the nine-point formula will be,



$$u + 6h^2 f = 0 \qquad (3.109)$$

with truncation error in the order of h^6.

3.3 VARIATIONAL APPROACH OF THE FEM

The engineering approach presented in Chapter 2 for the FEM gave an insight to the understanding of the method as was originally developed and helped in giving an orderly step-by-step formulation of the method. However, this approach which is a direct one, cannot be applied as such to other engineering and scientific applications. The variational approach for formulating the FE equations gives a broader range of applications to be solved. The variational basis of the FEM dictates the criteria to be satisfied by the element interpolation functions and enables us to make definitive statements about the convergence of the results as we use an ever increasing number of smaller and smaller elements.

Variational principles occur frequently in many engineering and physical problems, and, historically, these methods are among the oldest means of obtaining approximate solutions to these problems. Since the FEM formulated from a variational principle can be considered as a special case of the Rayleigh-Ritz method when the interpolation functions satisfy some continuity requirements over elements, it may be convenient to explain the Rayleigh-Ritz method first.

3.3.1 The Rayleigh-Ritz Method

The Rayleigh-Ritz (R.R.) method is one of the methods used to minimize a functional. The method is based on the choice of a suitable complete set of linearly independent basis functions $\psi_i(x,y)$ for i=1, 2,... . The exact solution ϕ_0 is approximated by a sequence of trial functions:

$$\phi_n = \sum_{i=1}^{n} c_i \psi_i \qquad (3.110)$$

where C's are chosen to minimize the functional $I(\phi_n)$. This procedure is said to be convergent to the solution if $\phi_n \to \phi_0$ as $n \to \infty$. This method is best illustrated by an example. Consider the Poisson's equation,

$$\nabla^2\phi = -f \ , \tag{3.111}$$

subject to the boundary conditions:

$$\phi = g(s) \text{ on some part of the boundary } \Gamma_1 \tag{3.112}$$

$$\text{and } \frac{\partial\phi}{\partial n} + \sigma(s)\phi = h(s) \text{ on } \Gamma_2 \ , \tag{3.113}$$

where n is the direction of the outward normal on the boundary Γ_2.

The functional $I[\phi]$ corresponding to this equation [Davies, 1980]:

$$I[\phi] = \iint_R \{(\frac{\partial\phi}{\partial x})^2 + (\frac{\partial\phi}{\partial y})^2 - 2\phi f\}\, dxdy + \int_{\Gamma_2} (\sigma\phi^2 - 2\phi h)ds \tag{3.114}$$

The problem now becomes how to find the function $\phi(x,y)$ that minimizes the functional $I[\phi]$. We choose a linearly independent set of basis functions ψ_i which satisfy the homogeneous Dirichlet condition, i.e., $\psi_i = 0$ on Γ_1, then a sequence of trial functions which satisfy the non-homogeneous Dirichlet condition on Γ_1 is:

$$\phi_n = g + \sum_{i=1}^{n} C_i\psi_i \ . \tag{3.115}$$

This can be rewritten as:

$$\phi_n = \sum_{i=1}^{n+1} C_i\psi_i \ , \tag{3.116}$$

where $\psi_{n+1} = g$ and $C_{n+1} = 1$.

The functional $I[\phi]$ can now be written in terms of the C's and ψ's as:

$$I(c_1, c_2, \ldots, c_{n+1}) = \iint_R \{(\Sigma c_i \frac{\partial\psi_i}{\partial x})^2 + (\Sigma c_i \frac{\partial\psi_i}{\partial y})^2 - 2\Sigma c_i\psi_i f\}dxdy$$

$$+ \int_{\Gamma_2} \{\sigma(\Sigma c_i\psi_i)^2 - 2(\Sigma c_i\psi_i)h\}ds \ . \tag{3.117}$$

In order to minimize I we must have:

$$\frac{\partial I}{\partial c_i} = 0, \text{ for } i=1,2,\ldots,n. \tag{3.118}$$

We notice that c_{n+1} is known to be 1 as stated before. Performing the differentiation:

$$\frac{\partial I}{\partial c_i} = 2A_{ii}c_i + 2\sum_{j\neq i} A_{ij}c_j - 2h_i + 2S_{ii}c_i + 2\sum_{j\neq i} S_{ij}c_j - 2k_i = 0 ,$$

$$i=1,2,\ldots,n , \tag{3.119}$$

where,

$$A_{ij} = \iint_R \left(\frac{\partial \psi_i}{\partial x}\frac{\partial \psi_j}{\partial x} + \frac{\partial \psi_i}{\partial y}\frac{\partial \psi_j}{\partial y}\right) dxdy , \tag{3.120}$$

$$h_i = \iint_R \psi_i \, f dxdy , \tag{3.121}$$

$$S_{ij} = \int_{\Gamma_2} \sigma \psi_i \psi_j \, ds , \tag{3.122}$$

and

$$k_i = \int_{\Gamma_2} \psi_i \, h \, ds . \tag{3.123}$$

The set of equations resulting from equating $\frac{\partial I}{\partial c_i}$ to zero can be written in a matrix form as:

$$Bc = g , \tag{3.124}$$

where,

$$B_{ij} = A_{ij} + S_{ij} , \tag{3.125}$$

and

$$g_i = h_i + k_i . \tag{3.126}$$

Solving this set of equations, the values of the c_i, i=1,...,n, can be determined and hence the solution of the original PDE is obtained. The success of this method depends strongly on the choice of the trial functions. Generally, the larger the size of the family of trial functions (i.e. the number of adjustable parameters) is, the more accurate is the solution. Although the trial functions are usually polynomials, it is possible to choose a different class of trial functions e.g. trigonometric for some problems as has been explained by Hildebrand [1965].

It is known [Finlayson and Scriven, 1966], that for linear self-adjoint operators L the application of a variational principle (R.R. method say) will give an identical solution to that obtained by the Galerkin method of weighted residuals. This can be illustrated by the following example.

Consider the Poisson's equation,

$$\nabla^2\phi = -f \ . \tag{3.127}$$

For simplicity assume that we have the homogeneous condition on the boundary $\phi=0$. The variational formulation of this problem has been done before and it was shown to be:

$$I[\phi] = \iint [(\frac{\partial\phi}{\partial x})^2 + (\frac{\partial\phi}{\partial y})^2 - 2\phi f]dxdy \ . \tag{3.128}$$

Assume the solution ϕ is expressed in terms of the trial functions N_i, i.e.,

$$\phi = \sum_{i=1}^{n} N_i\phi_i \tag{3.129}$$

Substituting in $I[\phi]$ we get:

$$I[\phi] = \iint \{(\sum_{i=1}^{n} \phi_i \frac{\partial N_i}{\partial x})^2 + (\sum_{i=1}^{n} \phi_i \frac{\partial N_i}{\partial y})^2 - 2f \sum_{i=1}^{n} \phi_i N_i\}dxdy \ . \tag{3.130}$$

To minimize the functional $I[\phi]$ then $\frac{\partial I}{\partial \phi_j} = 0$ for $j=1,2,\ldots,n$. This results in:

$$\frac{\partial I}{\partial \phi_j} = 0 = \iint [2\frac{\partial N_j}{\partial x}(\sum_{i=1}^{n} \phi_i \frac{\partial N_i}{\partial x}) + 2\frac{\partial N_j}{\partial y}(\sum_{i=1}^{n} \phi_i \frac{\partial N_i}{\partial y}) - 2fN_j]dxdy. \tag{3.131}$$

If we start to solve the same problem using the Galerkin method then the Poisson's equation $\nabla^2\phi=-f$ will produce the following weighted residual statement:

$$\iint N_i [\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + f]dxdy = 0 \tag{3.132}$$

Applying Green's theorem to the first two terms we get:

$$\int N_i \left[\frac{\partial \phi}{\partial x} \ell_x + \frac{\partial \phi}{\partial y} \ell_y\right] ds - \iint \left\{ \frac{\partial N_i}{\partial x} \frac{\partial \phi}{\partial x} + \frac{\partial N_i}{\partial y} \frac{\partial \phi}{\partial y} - f N_i \right\} dxdy = 0 \quad (3.133)$$

Since the functions N_i are chosen to satisfy the homogeneous boundary conditions then the first term will be a zero and we get:

$$\iint \left\{ \frac{\partial N_i}{\partial x} \frac{\partial \phi}{\partial x} + \frac{\partial N_i}{\partial y} \frac{\partial \phi}{\partial y} - f N_i \right\} dxdy = 0 , \quad (3.134)$$

Since the weighting functions are the same as the interpolation functions in the Galerkin's procedure, then ϕ is expressed in terms of N_i, i.e. $\phi = \sum_{i=1}^{n} N_i \phi_i$. Thus equation (3.134) can be written as:

$$\iint \left\{ \frac{\partial N_i}{\partial x} \sum \phi_i \frac{\partial N_i}{\partial x} + \frac{\partial N_i}{\partial y} \sum \phi_i \frac{\partial N_i}{\partial y} - f N_i \right\} dxdy = 0 \quad (3.135)$$

which is identical to the same equation obtained using the variational approach.

The consequence of this result is that it is possible to formulate the FEM equations directly from the PDE rather than considering the corresponding variational principle which may not be easily found or unknown as will be explained later.

3.3.2 Merits and Limitations of Variational Formulations

Despite the fact that variational formulation of continuum problems were amongst the first methods used to solve these problems it has some limitations that can be summarized as follows:

(1) It is not possible, generally, to find one function (or a sequence of functions) that satisfy certain essential boundary conditions for an irregular-shaped boundary. Thus the variational approach

is suited for fairly simple geometries.

(2) The need to have high order trial functions, even in the case of simple geometrical domains, since, in general, very high order polynomials would be required to approach the exact behaviour of the unknown over the whole domain.

(3) Difficulties in the handling of singularities, since in the variational methods all parts of the domain are covered using the same trial functions, and no special treatment is performed or allowed to areas that require more attention.

(4) Weak coupling of points, which are distant from one another. This will yield dense matrices in the final analysis.

The merits of the variational formulation can be summarized as [Rao, 1982]:

(1) The variational principle usually possesses a clear physical interpretation in most of the practical problems.

(2) The functional can contain lower order derivatives of the field variable compared to the governing differential equations and hence an approximate solution can be obtained using a larger class of functions.

(3) It is possible to prove the existance of solution in some cases using the variational formulation.

(4) The variational formulation permits the treatment of complicated boundary conditions implicitly as natural boundary conditions and thus we need to explicitly impose the geometric or forced boundary conditions.

(5) Sometimes, the problem may possess a dual variational formulation in which case the solution can be sought either by minimizing

(or maximizing) the functional I or by maximizing (or minimizing) its dual functional. In such cases it is possible to find upper and lower bounds to the solution.

3.3.3 The Variational Formulation of the FEM

The finite element method and the Rayleigh-Ritz method are essentially similar. The main difference is that in the R.R. method the assumed trial functions are defined over the whole domain and have to satisfy the boundary conditions. However, in the FEM the assumed trial functions are defined over each element and they have to satisfy some continuity conditions over elements. This shows the greater flexibility of the FEM over the R.R. technique which, in fact, can be used for fairly simple geometries only. So, if the functional for a given problem can be expressed as the sum of functionals evaluated for all elements, it is sufficient to consider an isolated element to derive the equations describing its behaviour. To do so, interpolation functions are assumed to define the field variable ϕ_e in terms of its values at the nodes of the element. Then, the functional over the element I_e is evaluated by substituting the assumed form for ϕ_e and its derivatives and doing the integration over the element domain. At last, the differentiation of the functional I_e, now expressed in terms of nodal values of ϕ, is done with respect to these nodal values. This can be summarized in the following steps:

Assume the problem to be solved is expressed in a variational principle as:

$$I[\phi] = \int_R F_1(\phi,\phi_x,\ldots,)dR + \int_{\partial R} F_2(\phi,\phi_x,\ldots,)ds \qquad (3.136)$$

(1) The first step is to divide the domain R into n smaller non-

overlapped parts that cover the whole domain R. These smaller parts are the finite elements. If $I[\phi]$ can be expressed as a summation of elemental contributions, then

$$I[\phi] = \Sigma\, I^e[\phi] \ , \quad \forall \text{ elements} \in R \tag{3.137}$$

(2) The unknown field variable ϕ (which is a vector in general) is assumed to vary in each element and can be expressed in terms of its nodal values. Assume the element has r nodes, then:

$$\phi = \sum_{i=1}^{r} N_i \phi_i \tag{3.138}$$

where N_i are the shape functions.

(3) To minimize the functional $I[\phi]$ then $\frac{\partial I}{\partial \phi}$ is equated to 0. Since I is the sum ΣI^e, then $\frac{\partial I}{\partial \phi}$ will be:

$$\frac{\partial I}{\partial \phi_i} = \sum_{e=1}^{E} \frac{\partial I^e}{\partial \phi_i} = 0 \ , \ i=1,2,\ldots,N \tag{3.139}$$

where E is the total number of elements.

In the special case where I is a quadratic function of ϕ and its derivatives, the element equations can be written as:

$$\frac{\partial I^e}{\partial \phi^e} = K^e \phi^e - f^e \ , \tag{3.140}$$

where K^e is the element characteristic matrix and f^e is the element characteristic vector. These are the corresponding terms to the stiffness matrix and equivalent nodal loads vector in the direct formulation of the FEM.

(4) The overall equations of the system can be written by the summation of the elements contributions. This is known as the assembly process. Thus, we reach the set of equations defined by,

$$\frac{\partial I}{\partial \phi} = K\phi - f = 0 \ , \tag{3.141}$$

where,

$$K = \sum_{e=1}^{E} K^e \tag{3.142}$$

and

$$f = \sum_{e=1}^{E} f^e \tag{3.143}$$

The assembly process has been explained in Chapter 2 and it will not be repeated here again.

(5) After imposing the boundary conditions, the system of equations $K\phi=f$ can be solved for ϕ. If I was not quadratic in ϕ, the resulting set of equations will be non-linear.

3.4 A WEIGHTED RESIDUAL APPROACH TO THE FEM

The main difficulty with the variational formulation of the FEM is that it relies on having a variational principle for the problem to be solved. This is not always possible in general. A more flexible and general approach is by using the method of weighted residuals (MWR) for the FE equation formulation starting from the governing PDE directly. As previously explained, the FEM can be considered as a special case of the MWR and the variational methods for solving PDE's in the sense that the assumed trial functions need not be defined over the whole domain, but rather, on finite elements only. In addition to that, they have to satisfy some continuity conditions but nothing for the boundary conditions. The Galerkin's method is usually used among the other MWR techniques in the FEM. In the Galerkin method, the weighting functions are chosen to be the same as the trial functions themselves. Thus the equations governing the behaviour of a finite element according to the Galerkin method is:

$$\int_{D^e} \{L(\hat{\phi}^e)-f^e\}\, N_i^e \, dD^e = 0, \quad i=1,2,\ldots,r \tag{3.144}$$

where r is the number of unknown parameters assigned to the element. The interpolation functions N_i^e are defined over the element and usually polynomials are chosen. These sets of equations can be written for every element. The N_i functions must satisfy interelement continuity. This requires that the ϕ values, as well as the derivatives up to the highest order minus one of the derivatives in the expression to be integrated, are continuous over element boundaries. Since the higher the order of continuity to be satisfied by the interpolation functions, the narrower the class of functions that can be chosen, it is desirable to lower the

highest-order derivative appearing in the element equations. This is done by integrating equation (3.144) by parts. This will have two advantages: first, the resulting expressions will contain lower-order derivatives which implies that lower-order interelement continuity needs to be satisfied by the interpolation functions, and second, it offers a convenient way to introduce the natural boundary conditions that must be satisfied on some portion of the boundary. The fixed boundary conditions can be introduced after assembly of elements in a similar manner to that presented in Chapter 2 of this thesis.

These concepts are better clarified by a simple example. A more complicated one will be given in Chapter 7 of this thesis for the FE formulation for some ground water problems.

Consider the one dimensional Poisson equation defined by:

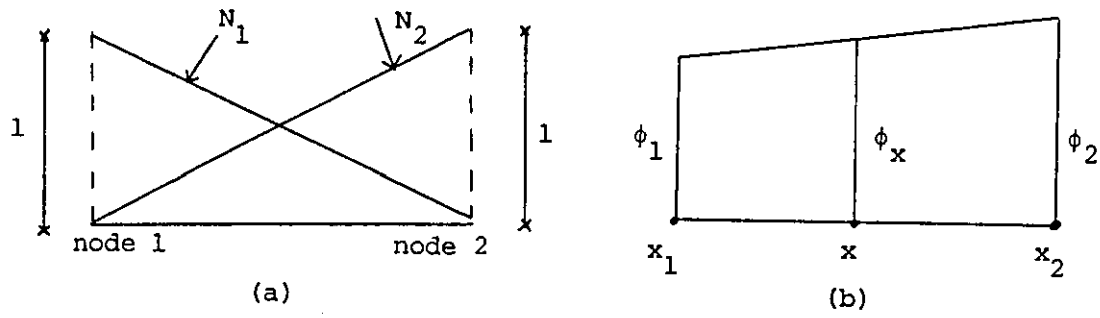
$$\frac{d^2\phi}{dx^2} + f(x) = 0 \ , \tag{3.145}$$

with boundary conditions $\phi(a)=A$ and $\phi(b)=B$.

This problem will now be solved by the FEM based on the two approaches which have been explained i.e. starting from a variational formulation and by the Galerkin's MWR. The solution by the variational formulation is simply to find the function $\phi(x)$ that minimizes the function $I[\phi]$ defined by:

$$I[\phi] = \int_a^b [\tfrac{1}{2}(\frac{d\phi}{dx})^2 - f(x)\phi(x)]dx \tag{3.146}$$

Since the problem is 1-D, then line elements can be used. The simplest of these line elements are those with linear interpolation functions. Thus two nodes at the ends of each line element are required. A typical element is shown in Figure (3.4).



(a) The interpolation functions N_1 and N_2

(b) Linear variation of ϕ over the element

FIGURE 3.4: Two nodes line element

The value of ϕ at a typical point x within the element can be expressed in terms of the nodal values ϕ_1 and ϕ_2 using:

$$\phi_{(x)} = \left(\frac{x_2-x}{x_2-x_1}\right)\phi_1 + \left(\frac{x-x_1}{x_2-x_1}\right)\phi_2$$

or
$$\phi^{(e)} = [N_1 \quad N_2]\begin{bmatrix}\phi_1\\ \phi_2\end{bmatrix} = [N][\phi]^{(e)} \tag{3.147}$$

where,
$$N_1(x) = \frac{x_2-x}{x_2-x_1} \tag{3.148}$$

and,
$$N_2(x) = \frac{x-x_1}{x_2-x_1} \quad . \tag{3.149}$$

The overall functional $I[\phi]$ for the whole domain can be derived by summing the contributions of each element, i.e.,

$$I[\phi] = \Sigma I[\phi^e] \qquad \forall \text{ elements.} \tag{3.150}$$

In order to determine the expression $I[\phi^e]$ we substitute the expression of ϕ^e into equation (3.146) for the $I[\phi]^e$, i.e.,

$$I[\phi^e] = \int_{x_1}^{x_2} \{\tfrac{1}{2}([N_1'N_2']\begin{bmatrix}\phi_1\\ \phi_2\end{bmatrix})^2 - f(x)\ [N_1 \quad N_2]\ \begin{bmatrix}\phi_1\\ \phi_2\end{bmatrix}\}dx \tag{3.151}$$

N_1' is $\frac{dN_1}{dx}$ and N_2' is $\frac{dN_2}{dx}$.

Minimizing $I[\phi^e]$ with respect to the nodal values ϕ_1 and ϕ_2 requires that $\frac{\partial I}{\partial \phi_1} = \frac{\partial I}{\partial \phi_2} = 0$.

$$\frac{\partial I^e}{\partial \phi_1} = \int_{x_1}^{x_2} \{N_1'[N_1' \quad N_2'] \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} - fN_1\} dx = 0 \tag{3.152}$$

and,

$$\frac{\partial I^e}{\partial \phi_2} = \int_{x_1}^{x_2} \{N_2'[N_1' \quad N_2'] \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} - fN_2\} dx = 0 \ . \tag{3.153}$$

Rewriting these equations as:

$$\int_{x_1}^{x_2} N_1'(\phi_1 N_1' + \phi_2 N_2') dx = \int_{x_1}^{x_2} fN_1 \, dx \tag{3.154}$$

and

$$\int_{x_1}^{x_2} N_2'(\phi_1 N_1' + \phi_2 N_2') dx = \int_{x_1}^{x_2} fN_2 \, dx \tag{3.155}$$

These two equations can be combined in a matrix form as:

$$[K]^e [\phi]^e = [F]^e \tag{3.156}$$

where,

$$K^e = \int_{x_1}^{x_2} \begin{bmatrix} N_1'N_1' & N_1'N_2' \\ N_2'N_1' & N_2'N_2' \end{bmatrix} dx \tag{3.157}$$

$$\phi^e = \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} \tag{3.158}$$

and

$$[F]^e = \int_{x_1}^{x_2} \begin{bmatrix} fN_1 \\ fN_2 \end{bmatrix} dx \tag{3.159}$$

Thus we have obtained the element characteristics using the variational principle. In order to derive these equations based on the Galerkin's MWR we start with the differential equation itself and assume that the approximate solution $\hat{\phi}$ is described in terms of the nodal values ϕ_i

and the interpolation functions N_i then,

$$\hat{\phi} = \sum_{i=1}^{n} N_i(x)\phi_i \ , \tag{3.160}$$

where n is the number of nodes in the element. Since we use two nodes linear element, n is = 2. In the Galerkin's method the weighting functions are chosen to be the same as the interpolation functions. The integration of the weighted residual is equated to zero which is expressed as,

$$\int_{x_1}^{x_2} \left(\frac{d^2\hat{\phi}}{dx^2} + f(x)\right) N_i(x)dx = 0, \quad i=1 \text{ and } 2 \ , \tag{3.161}$$

i.e.,

$$\int_{x_1}^{x_2} \left(\frac{d^2\hat{\phi}}{dx^2}\right) N_i(x)dx + \int_{x_1}^{x_2} f(x)N_i(x)dx = 0 \ . \tag{3.162}$$

The first term can be integrated by parts to give,

$$\int_{x_1}^{x_2} \left(\frac{d^2\hat{\phi}}{dx^2}\right) N_i(x)dx = \left| N_i \frac{d\hat{\phi}}{dx} \right|_{x_1}^{x_2} - \int_{x_1}^{x_2} \frac{d\hat{\phi}}{dx}\frac{dN_i}{dx}\,dx. \tag{3.163}$$

Thus,

$$\left| N_i \frac{d\hat{\phi}}{dx} \right|_{x_1}^{x_2} - \int_{x_1}^{x_2} \frac{d\hat{\phi}}{dx}\frac{dN_i}{dx}\,dx + \int_{x_1}^{x_2} f(x)N_i(x)dx = 0 \ , \quad i=1 \text{ and } 2, \tag{3.164}$$

but,

$$\frac{d\hat{\phi}}{dx} = \sum_{i=1}^{n} \frac{dN_i}{dx}\phi_i = [N_1' \quad N_2'] \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} \tag{3.165}$$

$\therefore$

$$\int_{x_1}^{x_2} (N_1' \quad N_2') \frac{dN_i}{dx}\,dx \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} = \left| N_i \frac{d\hat{\phi}}{dx} \right|_{x_1}^{x_2} + \int_{x_1}^{x_2} f(x)N_i(x)dx \ , \quad i=1 \text{ and } 2, \tag{3.166}$$

but,

$$\left| N_i \frac{d\hat{\phi}}{dx} \right|_{x_1}^{x_2} = N_1(x_2) \left.\frac{d\hat{\phi}}{dx}\right|_{x_2} - N_1(x_1) \left.\frac{d\hat{\phi}}{dx}\right|_{x_1} \quad \text{for } i=1.$$

and since N_1 at x_2 is zero and N_1 at x_1 is 1 by definition of this

interpolation function, then,

$$\left| N_1 \frac{d\hat{\phi}}{dx} \right|_{x_1}^{x_2} = - \left. \frac{d\hat{\phi}}{dx} \right|_{x_1} = - \frac{d\hat{\phi}}{dx}(x_1) \quad . \tag{3.167}$$

Similarly:

$$\left| N_2 \frac{d\hat{\phi}}{dx} \right|_{x_1}^{x_2} = \frac{d\hat{\phi}}{dx}(x_2) \tag{3.168}$$

$$\therefore \quad \left| N_i \frac{d\hat{\phi}}{dx} \right|_{x_1}^{x_2} = \begin{bmatrix} - \frac{d\hat{\phi}}{dx}(x_1) \\ \frac{d\hat{\phi}}{dx}(x_2) \end{bmatrix} \tag{3.169}$$

$$\therefore \quad \int_{x_1}^{x_2} [N_1' \quad N_2'] \frac{dN_i}{dx} dx \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} = \begin{bmatrix} - \frac{d\hat{\phi}}{dx}(x_1) \\ \frac{d\hat{\phi}}{dx}(x_2) \end{bmatrix} + \int_{x_1}^{x_2} f(x) N_i(x) dx \tag{3.170}$$

$i = 1$ and 2,

i.e.,

$$K^e \phi^e = \begin{bmatrix} - \frac{d\hat{\phi}}{dx}(x_1) \\ \frac{d\hat{\phi}}{dx}(x_2) \end{bmatrix}^e + \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}^e \tag{3.171}$$

where,

$$K^e = \int_{x_1}^{x_2} \begin{bmatrix} N_1'N_1' & N_1'N_2' \\ N_2'N_1' & N_2'N_2' \end{bmatrix} dx \tag{3.172}$$

and,

$$\phi^e = \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix}^e \quad \text{and} \quad \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}^e = \int_{x_1}^{x_2} \begin{bmatrix} fN_1 \\ fN_2 \end{bmatrix} dx \quad . \tag{3.173}$$

This set of equations expresses the characteristics of the two nodes line element. The expression for K^e and F^e are identical to those derived using the variational principle. We notice that the natural boundary conditions are taken into account when we assemble the element matrices. During assembly of the matrices, the natural boundary condition terms $\frac{d\hat{\phi}}{dx}$ will cancel at all interior nodes of the solution domain, leaving only the natural boundary conditions at the exterior nodes i.e. at the

points a and b only. In the numerical solution for these classes of problems the boundary conditions at exterior nodes are incorporated as was described in Chapter 2.

3.5 THE DOMAIN DISCRETIZATION PROCESS IN THE FEM

The domain in the FEM must be divided into finite elements as the first step in the solution process. The discretization process is sometimes fairly simple and clear such as in the case of structural analysis of skeletal structures like beams, trusses and frames. However, for other problems like continuum problems, the discretization requires more attention. Choice of finite elements is one of the most important factors that affect the solution obtained in the FEM. Another very important factor is the choice of the interpolation functions. Choosing particular elements and interpolation functions depends heavily on the nature of the problem to be solved, the required accuracy, the available computing resources and the cost. There are no formal fixed sets of rules that can be set to achieve the best discretization of the domain. Sometimes, it is referred to the technique of choosing finite elements as "elementology". Here, we list some major guide lines principles for "elementology" that are based on experience and engineering judgement in the first place.

(i) Nodes should be placed at points of application of external forces in structural analysis problems and in similar situations like heat sources in heat transfer problems. The same is true at corners in the domain.

(ii) Whenever possible, one type of element should be used in the discretization. However, sometimes it becomes impossible to model the continum by one type of element only, as in the case, for example, of a plate supported by springs.

(iii) Usually line elements are used in 1-D problems, triangles in 2-D and tetrahedrons in 3-D. This is due to their simplicity and

ability to represent irregular boundaries. These elements can have any number of exterior and interior nodes as required to satisfy the interpolation functions defined over them. Moreover, they can have curved sides at curved boundaries.

(iv) In regions of the solution domain where the gradient of the field variable is expected to be varying quickly, or at irregular boundaries, more elements should be created i.e. a finer mesh should be constructed.

(v) Ill-proportioned elements should be avoided since they tend to give directional bias solutions that may not be correct. For example, in triangular elements it is recommended that internal angles be around 60°. Generally, an aspect ratio around 1 is the best.

(vi) It is usually true that increasing the number of elements will give more accurate results provided that the elements obey the requirements for a convergent solution. However, this will lead to more expensive solution and a compromise must be done. Sometimes, particularly for new problems, it is a good practice to start solving the problem using a sequence of $n_1, n_2, \ldots$ elements where $n_i > n_{i-1}$ until a stable solution is reached.

(vii) For regions which extend to infinity, a similar technique to that mentioned in (vi) is utilized, i.e. the problem is solved with the boundary located at distances $d_1, d_2, \ldots$ such that $d_i > d_{i-1}$ until an acceptable solution is found.

3.5.1 Element Shapes

A finite element is fully specified if all of the following data

are specified:

(i) its shape i.e. the geometry of the element;

(ii) the number of its nodes and their locations within the element;

(iii) the type of each of its nodes, i.e. exterior or interior;

(iv) the nodal variables to be computed and their nature; and

(v) the type of the interpolation functions.

In this section, the element shapes are described while in later sections other relevant criteria are discussed.

In the case of one-dimensional problems, line elements are used. It is possible to specify m nodes in these line segments depending upon the type of the interpolation functions, the degree of continuity required and the type of nodal variables. A family of these line elements is shown in Figure (3.5).

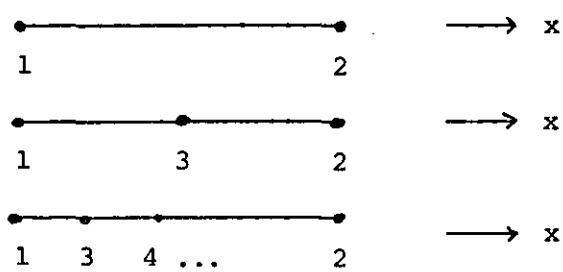
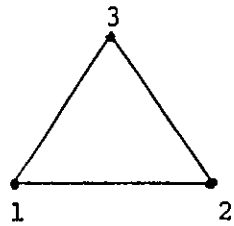


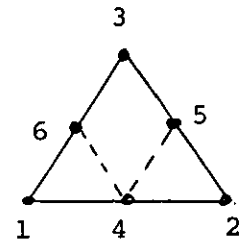
FIGURE 3.5: Family of 1-D elements

In the case of two-dimensional problems the basic element is the linear triangle i.e. the 3-nodes triangle. This, actually, was the first 2-D element used in the FEM. It was first developed for the solution of elasticity problems and known as the constant stress triangle (CST). More advanced triangles are those with mid-side nodes which are the quadratic triangles. It is also possible to use higher

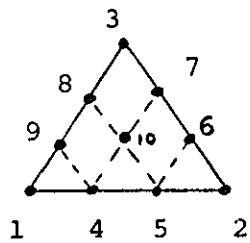
order triangles with interior nodes like cubic and quartic triangles. These are shown in Figure (3.6).



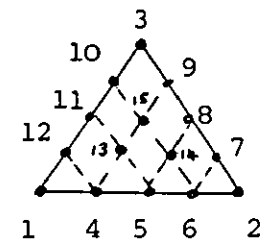
(a) linear triangle- 3 nodes



(b) quadratic triangle - 6 nodes



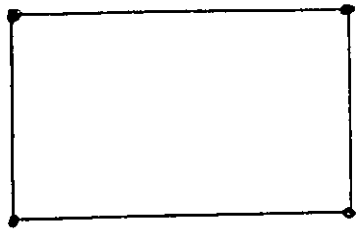
(c) cubic triangle - 10 nodes



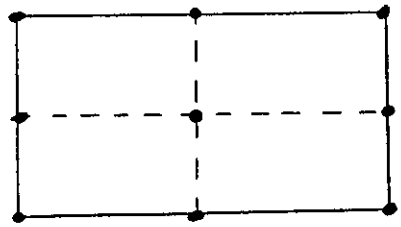
(d) quartic triangle - 15 nodes

FIGURE 3.6: Family of triangular elements

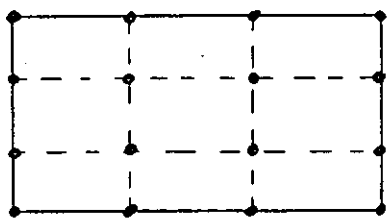
A less frequently used 2-D element is the rectangular element. A family of rectangular elements that correspond to the above-mentioned triangular elements is shown in Figure (3.7). The main advantage of rectangular elements is that they can be created automatically by a fairly simple preprocessor. However, since many sophisticated pre-processors are now available that can subdivide the domain to triangular elements automatically with the minimal user input, it seems that triangular elements will still be dominant in 2-D FE modelling.



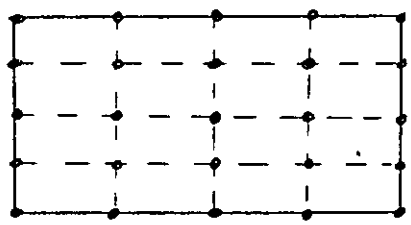
(a) linear rectangle - 4 nodes



(b) quadratic rectangle - 9 nodes



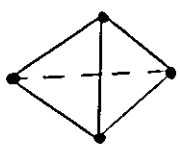
(c) cubic rectangle - 16 nodes



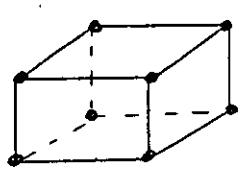
(d) quartic rectangle - 25 nodes

FIGURE 3.7: Family of rectangular elements

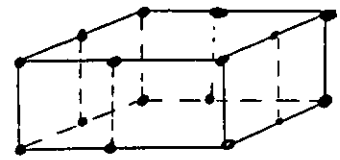
For the three-dimensional case tetrahedron elements are generally used. Less frequently, right prism elements can be used. Modelling of thick shells is usually done by the 3-D element of 16 nodes. These elements are shown in Figure (3.8).



(a) tetrahedron - 4 nodes



(b) right prism - 8 nodes



(c) brick element - 16 nodes

FIGURE 3.8: Some three-dimensional elements

If the boundary is not linear, it is possible to have the element sides curved as well. These elements will be explained later. It is worthwhile to mention that in some instants it is possible to decrease the dimensionality of the problem by one. This is in the case of axisymmetric problems, where axial symmetry exists in cylindrical coordinates. Many practical engineering problems are axisymmetric like storage tanks, pistons and shafts. A problem is considered as axisymmetric where all of its parameters are invariant with respect to any plane passing through the symmetry axis of the solution domain.

3.5.2 Nodes

Nodes can be classified as exterior or interior. Exterior nodes are those positioned at the corners of elements or along the edges (or on the surfaces in the 3-D case). Elements can be connected at exterior nodes only. In contrast, the interior nodes are those inside the element itself and are not connected to any other elements. It is obvious that in the case of 1-D linear elements, the two nodes of this element are both exterior. In the case of 3-nodes linear elements, we have 2 exterior nodes and one interior node. In the case of cubic

triangular elements we have 1 interior node and 9 exterior ones. If the field variable is continuous along element interfaces we say that we have C^0 continuity. If, in addition to that, the first derivatives are also continuous, we have C^1 continuity and so on. The number of nodal variables is called the degree of freedom associated with the node. The roots of this term is due to structural analysis problems where the degrees of freedom of a node are the number of available displacements at this node.

3.5.3 Interpolation Functions

One of the most crucial factors in finite element analysis is the choice of interpolation functions. These functions describe the behaviour of the field variable within the finite element itself in terms of their values at nodes. Although polynomials are mostly used, it is possible to use other functions like trigonometric. The reasons which give preference to polynomials over other types of functions are their ease of computation both symbolically and numerically, ease of differentiation and integration, ease of controlling the required accuracy by increasing its degree and ease of programming. Polynomials are also attractive since any continuous function can be approximated, arbitrarily closely, by a polynomial. This is known as the Weierstrass approximation theory.

The choice of the polynomial to be used to describe the field variable behaviour within the element is dependent on many factors. Here some guidelines are given for choosing "good" polynomials.

(i) The number of terms in the polynomial must be equal to the total number of degrees of freedom associated with the element,

otherwise the polynomial may not be unique. To illustrate, consider a line element with three nodes and assume that the field variable associated with each node is a scalar quantity, then a polynomial like:

$$\phi(x) = a_o + a_1 x + a_2 x^2 \tag{3.174}$$

should be used.

(ii) The polynomial representation within an element should be geometrically invariant or geometrically isotropic. This means that the polynomial should not possess any preference for either the x or the y directions over the other. Consequently, the polynomial should contain terms which do not violate the symmetry of the complete polynomials as shown in Figure (3.9).

Polynomial	Behaviour	No. of terms
1	Constant	1
$x \quad y$	Linear	3
$x^2 \quad xy \quad y^2$	Quadratic	6
$x^3 \quad x^2y \quad xy^2 \quad y^3$	Cubic	10
$x^4 \quad x^3y \quad x^2y^2 \quad xy^3 \quad y^4$	Quartic	15
$x^5 \quad x^4y \quad x^3y^2 \quad x^2y^3 \quad xy^4 \quad y^5$	Quantic	21
$x^6 \quad x^5y \quad x^4y^2 \quad x^3y^3 \quad x^2y^4 \quad xy^5 \quad y^6$	Hexadic	28

FIGURE 3.9: Complete polynomials in two-dimensions

(iii) The interpolation polynomial should satisfy the convergence requirements, i.e. the unknown field variable must be continuous within the element itself.

In one-dimension a general complete nth order polynomial can be expressed as:

$$p_n(x) = \sum_{i=o}^{n} a_i x^i \quad . \tag{3.175}$$

Samples of interpolation functions in one dimension are, therefore:

$$P_0(x) = a_0 \qquad \text{constant}$$

$$P_1(x) = a_0 + a_1 x \qquad \text{linear}$$

$$P_2(x) = a_0 + a_1 x + a_2 x^2 \qquad \text{quadratic}$$

In the two-dimensional case the complete nth order polynomial can be expressed as:

$$P_n(x,y) = \sum_{k=1}^{m} a_k x^i y^j , \quad i+j \leq n, \quad m=(n+1)/(n+2)/2 \tag{3.176}$$

Samples of interpolation functions in two-dimension are therefore,

$$P_0(x,y) = a_1 \qquad \text{constant}$$

$$P_1(x,y) = a_1+a_2x+a_3y \qquad \text{linear}$$

$$P_2(x,y) = a_1+a_2x+a_3x^2+a_4y+a_5y^2+a_6xy \qquad \text{quadratic}$$

In the three-dimensional case the complete nth order polynomial can be expressed as:

$$P_n(x,y,z) = \sum_{\ell=1}^{m} a_\ell x^i y^j z^k , \quad i+j+k \leq n, \quad m = \frac{(n+1)(n+2)(n+3)}{6} \tag{3.177}$$

Samples of interpolation functions in three dimensions are, therefore,

$$P_0(x,y,z) = a_1 \qquad \text{constant}$$

$$P_1(x,y,z) = a_1+a_2x+a_3y+a_4z \qquad \text{linear}$$

$$P_2(x,y,z) = a_1+a_2x+a_3y+a_4z+a_5xy+ a_6xz+a_7yz+a_8x^2+a_9y^2+a_{10}z^2 \qquad \text{quadratic}$$

Some examples of the determination of the unknown coefficients (the a's) and consequently the determination of the interpolation functions will be given later in this chapter and the next chapter.

3.5.4 Natural Coordinate Systems

Natural coordinate systems are a family of coordinate systems which possess the property that they are local within the element geometry and can assume values between 0 and 1 only. In these systems the value of the coordinate is one at a particular node while at other nodes it is zero. The main idea of a natural coordinate system is to express the location of a point inside an element in terms of coordinates associated with the nodes of the element. This is particularly useful in deriving the interpolation functions (N_i) at each node in a curve-sided element which will be discussed later in this chapter. If an element contains n external nodes; then, n natural coordinates are used.

Assume a linear element in one-dimension with two nodes, the natural coordinate system for this element denoted as L_1 and L_2 can be derived as follows.

For a general point x on the element its global coordinate x is related to the natural coordinates by,

$$x = L_1 x_1 + L_2 x_2 \ . \tag{3.178}$$

We notice that, by definition,

$$L_1 = 1 \text{ at } x=x_1$$
$$= 0 \text{ at } x=x_2$$

and

$$L_2 = 1 \text{ at } x=x_2$$
$$= 0 \text{ at } x=x_1$$

Both L_1 and L_2 varies linearly along the element 1-2 (Figure 3.10)

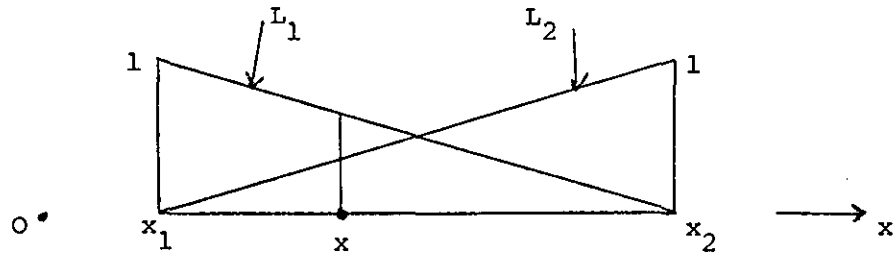


FIGURE 3.10: Natural coordinates for two nodes line elements

Therefore,

$$L_1(x) = \frac{x_2-x}{x_2-x_1} \quad \text{and} \quad L_2(x) = \frac{x-x_1}{x_2-x_1} \tag{3.179}$$

and $$L_1(x) + L_2(x) = 1 \ . \tag{3.180}$$

Since the functions L_1 and L_2 are simply ratios of lengths they are often called length coordinates. It should be noted that since the problem is one dimensional in position (x only); then L_1 and L_2 must be related to each other to keep the dimensionality unchanged. In fact, L_1 and L_2 are weighting functions that relate the coordinates of the end nodes to the coordinates of any interior point. Thus we must have, $L_1+L_2=1$ as explained earlier. It is easier in the derivation of the natural coordinates to start by imposing the relationship between the L_i functions as their sum at any point is always 1.

Similarly, natural coordinates in two-dimensions can be formulated. Assume a 3-node triangle element, the natural coordinate system in this case comprises 3 coordinates L_1,L_2 and L_3 where L_i assumes the value of 1 at node i and zeros at other nodes. The original cartesian coordinates of a point in the element should be linearly related to the new natural coordinates by:

$$x = L_1x_1 + L_2x_2 + L_3x_3 \tag{3.181}$$

and $$y = L_1y_1 + L_2y_2 + L_3y_3 \tag{3.182}$$

where, $$L_1 + L_2 + L_3 = 1. \tag{3.183}$$

These 3 equations can be solved for L_1, L_2 and L_3 which results in:

$$L_1(x,y) = \frac{1}{2\Delta}(a_1+b_1x+c_1y) \tag{3.184}$$

$$L_2(x,y) = \frac{1}{2\Delta}(a_2+b_2x+c_2y) \tag{3.185}$$

$$L_3(x,y) = \frac{1}{2\Delta}(a_3+b_3x+c_3y) \tag{3.186}$$

where,

$$\Delta = \frac{1}{2}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} = \text{area of the triangle whose vertices are } (x_1,y_1),\ (x_2,y_2) \text{ and } (x_3,y_3) \tag{3.187}$$

and the constants $a_1, a_2, a_3, b_1, \ldots$, etc. are given by,

$$a_1 = x_2y_3 - x_3y_2 \ , \ b_1 = y_2 - y_3 \text{ and } c_1 = x_3 - x_2 \tag{3.188}$$

The other coefficients are obtained by cyclically permuting the subscripts.

It should be noticed that L_1, L_2 and L_3 are ratios of area and hence they are often called area coordinates (Figure 3.11).

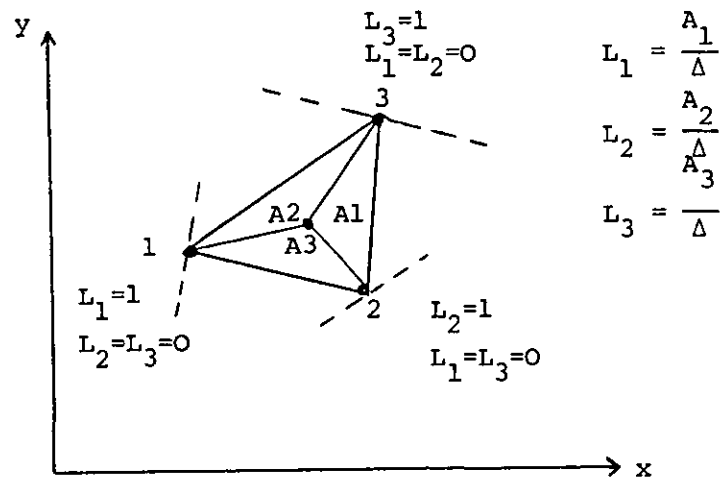


FIGURE 3.11: Area coordinates for a triangle with 3 nodes

3.6 THE TWO-DIMENSIONAL TRIANGULAR ELEMENTS

The family of the two-dimensional triangular elements are very popular and most widely used in 2-D problems. One reason for this situation may be because irregular boundaries can be better approximated by triangles. In addition to this, the complete nth order polynomial,

$$U = a_1+a_2x+a_3y+a_4xy+a_5x^2+a_6y^2+ \ldots +a_my^n \qquad (3.189)$$

where $a_1,a_2,\ldots,m$ are the coefficients of the polynomial, also known as generalized coordinates; n is the degree of the polynomial and $m = \sum_{i=1}^{n+1} i$ can be used to interpolate a function u at $\frac{1}{2}(n+1)(n+2)$ symmetrically placed nodes in a triangle.

Throughout this thesis most of the FE modelling will be done utilizing triangular elements with different degrees, viz: linear, quadratic,... etc.

In the linear case, the value of the interpolation function can be determined if its values at three nodes, typically the vertices, are known. For the higher degree polynomials, the required nodes can be generated by taking (n-1) equally spaced lines parallel to each side and placing the nodes at the intersections of these lines with each other and with the sides of the triangle as shown in Figure 3.6.

To compute the interpolation functions for a triangular element, let us start by the linear triangular elements. We must notice that all the family of the triangular elements possess the advantage that they have the sufficient number of nodes to uniquely specify a complete polynomial of the order necessary to retain C° continuity, i.e. inter-element continuity of the field variable ϕ along element boundaries. Hence, the compatibility, completeness and geometric isotropy requirements are satisfied.

Let the coordinates of the three nodes of the linear triangle element be: $(x_1,y_1),(x_2,y_2)$ and (x_3,y_3) with the values of the field variable ϕ at the three nodes ϕ_1,ϕ_2 and ϕ_3, respectively. The field variable ϕ defined over the element being a linear function can be expressed by,

$$\phi(x,y) = a_1 + a_2x + a_3y \ . \tag{3.190}$$

To evaluate the values of a_1,a_2 and a_3 we substitute in (3.190) by the value of ϕ_1,ϕ_2 and ϕ_3 and thus we obtain the system of linear equations,

$$\phi_1 = a_1 + a_2x_1 + a_3y_1 \tag{3.191}$$

$$\phi_2 = a_1 + a_2x_2 + a_3y_2 \tag{3.192}$$

$$\phi_3 = a_1 + a_2x_3 + a_3y_3 \ . \tag{3.193}$$

Solving these equations for a_1,a_2 and a_3 gives:

$$a_1 = \frac{1}{2A}(a_1\phi_1 + a_2\phi_2 + a_3\phi_3) \tag{3.194}$$

$$a_2 = \frac{1}{2A}(b_1\phi_1 + b_2\phi_2 + b_3\phi_3) \tag{3.195}$$

$$a_3 = \frac{1}{2A}(c_1\phi_1 + c_2\phi_2 + c_3\phi_3) \tag{3.196}$$

where A is the area of the element given by

$$A = \frac{1}{2}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} \tag{3.197}$$

and

$$a_1 = x_2y_3 - x_3y_2 \tag{3.198}$$

$$a_2 = x_3y_1 - x_1y_3 \tag{3.199}$$

$$a_3 = x_1y_2 - x_2y_1 \tag{3.200}$$

$$b_1 = y_2 - y_3 \tag{3.201}$$

$$b_2 = y_3 - y_1 \tag{3.202}$$

$$b_3 = y_1 - y_2 \tag{3.203}$$

$$c_1 = x_3 - x_2 \tag{3.204}$$

$$c_2 = x_1 - x_3 \tag{3.205}$$

$$c_3 = x_2 - x_1 \tag{3.206}$$

Substituting in the original expression for ϕ results in

$$\begin{aligned}\phi(x,y) = &\frac{1}{2A}(a_1+b_1x+c_1y)\phi(x_1,y_1) \\ &+ \frac{1}{2A}(a_2+b_2x+c_2y)\phi(x_2,y_2) \\ &+ \frac{1}{2A}(a_3+b_3x+c_3y)\phi(x_3,y_3)\end{aligned} \tag{3.207}$$

which can be rewritten as

$$\phi(x,y) = \sum_{i=1}^{3} N_i^{\ell}(x,y)\phi_i(x,y) \tag{3.208}$$

where,

$$N_i^{\ell}(x,y) = \frac{1}{2A}(a_i+b_ix+c_iy), \quad i=1,2,3 \tag{3.209}$$

and

$$\phi_i(x,y) = \phi(x_i,y_i), \quad i=1,2,3. \tag{3.210}$$

The functions $N_i^{\ell}(x,y)$, i=1,2,3 are the interpolation functions associated with nodal degrees of freedom for the linear triangle. These nodal interpolation functions have the value of unity at the associated node and zeros at all other nodes of the element. Thus, $N_1(x,y)$, for example, will have the value of 1 at node 1 and zeros at nodes 2 and 3.

In a similar way, the interpolation functions for other higher-order triangular elements can be determined. Huebner and Thornton [1982], presented a systematic method to derive these interpolation functions using a triple-index numbering scheme. The idea of this scheme is to denote the nodes of the triangular element by a three-digit label $\alpha\beta\gamma$ where, α,β and γ are integers satisfying $\alpha+\beta+\gamma=n$ and n

is the order of the interpolation polynomial used. These integers define constant coordinate lines in the area coordinate system and indicate the number of steps or levels by which a particular node is located from a side of the triangle. This is best illustrated as shown in Figure (3.12) for the case of the quadratic triangular element.

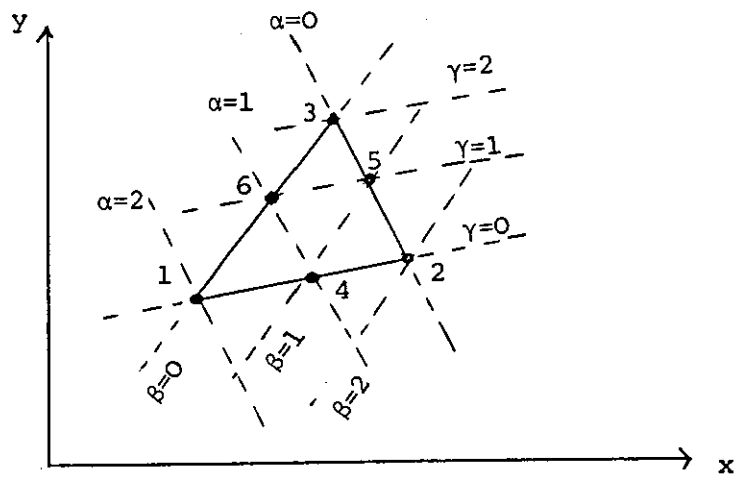


FIGURE 3.12: Node labelling in quadratic triangular element

In this case node number 1 will be labelled as node 200, while a mid-side node like number 4 will be labelled as 110. The interpolation functions can now be written in the area coordinates L_1,L_2 and L_3 as triple subscript function $N_{\alpha\beta\gamma}$ associated with the node $\alpha\beta\gamma$ as follows:

$$N_{\alpha\beta\gamma}(L_1,L_2,L_3) = N_\alpha(L_1)N_\beta(L_2)N_\gamma(L_3) \ , \tag{3.211}$$

where,

$$N_\alpha(L_1) = \prod_{i=1}^{\alpha} \left(\frac{nL_1-i+1}{i}\right) \quad \text{for } \alpha \geq 1 \tag{3.212}$$

$$= 1 \qquad \text{for } \alpha=0.$$

$N_\beta(L_2)$ and $N_\gamma(L_3)$ have similar equations. These equations can be used to derive the interpolation functions for higher-order triangular elements in area coordinates in an easy way. For example, for the six-

node quadratic triangle we have the node labels: 200, 020 and 002 for the vertices and 110,011 and 101 for the mid-side nodes. The associated interpolation functions: $N_{200}, N_{020}, N_{002}, N_{110}, N_{011}$ and N_{101} can be computed using n=2 (the degree of the polynomial) using the above equations. To illustrate consider N_{200}:

$$N_{200} = N_2(L_1)N_0(L_2)N_0(L_3) \tag{3.213}$$

$$N_2(L_1) = \prod_{i=1}^{2} \left(\frac{2L_1 - i + 1}{i}\right) = 2L_1\left(\frac{2L_1 - 1}{2}\right) = L_1(2L_1 - 1) \tag{3.214}$$

$$N_0(L_2) = 1, \; N_0(L_3) = 1. \tag{3.215}$$

Thus, we have,

$$N_{200} = L_1(2L_1 - 1) \; , \tag{3.216}$$

and similarly,

$$N_{020} = L_2(2L_2 - 1) \tag{3.217}$$

$$N_{002} = L_3(2L_3 - 1) \tag{3.218}$$

$$N_{110} = 4L_1L_2 \tag{3.219}$$

$$N_{011} = 4L_2L_3 \tag{3.220}$$

$$N_{101} = 4L_1L_3 \; . \tag{3.221}$$

These equations can be rewritten in a more compact form as for vertices,

$$N_i = L_i(2L_i - 1), \quad i=1,2,3 \tag{3.222}$$

and for mid-side nodes,

$$N_4 = 4L_1L_2, \; N_5 = 4L_2L_3 \quad \text{and} \quad N_6 = 4L_3L_1 \tag{3.223}$$

In a similar way for the cubic element the interpolation functions will be, for vertices,

$$N_i = \frac{L_i}{2}(3L_i - 1)(3L_i - 2) \quad , \; i=1,2,3 \; , \tag{3.224}$$

for side nodes,

$$N_4 = \frac{9}{2} L_1 L_2 (3L_1 - 1) \ , \tag{3.225}$$

$$N_5 = \frac{9}{2} L_1 L_2 (3L_2 - 1) \tag{3.226}$$

$$N_6 = \frac{9}{2} L_2 L_3 (3L_2 - 1) \tag{3.227}$$

$$N_7 = \frac{9}{2} L_2 L_3 (3L_3 - 1) \tag{3.228}$$

$$N_8 = \frac{9}{2} L_3 L_1 (3L_3 - 1) \tag{3.229}$$

$$N_9 = \frac{9}{2} L_3 L_1 (3L_1 - 1) \tag{3.230}$$

$$N_{10} = 27 L_1 L_2 L_3 \ . \tag{3.231}$$

For the quartic triangular element the shape function will be, for vertices,

$$N_i = \frac{1}{6} L_i (4L_i - 1)(4L_i - 2)(4L_i - 3) \ , \ i=1,2,3 \tag{3.232}$$

for side nodes:

$$N_4 = \frac{8}{3} L_1 L_2 (4L_1 - 1)(4L_1 - 2) \tag{3.233}$$

$$N_5 = 4 L_1 L_2 (4L_1 - 1)(4L_2 - 1) \tag{3.234}$$

$$N_6 = \frac{8}{3} L_1 L_2 (4L_2 - 1)(4L_2 - 2) \tag{3.235}$$

... etc.

For internal nodes:

$$N_{13} = 32 \ L_1 L_2 L_3 (4L_2 - 1) \tag{3.236}$$

$$N_{14} = 32 \ L_1 L_2 L_3 (4L_3 - 1) \tag{3.237}$$

and

$$N_{15} = 32 \ L_1 L_2 L_3 (4L_1 - 1) \ . \tag{3.238}$$

It is worthwhile to mention that it is sometimes customary to write the equations of the interpolation functions for the higher-order elements in terms of those of the linear one. This in the case of the triangular elements can be simply done by substituting N_1^{ℓ} which means

the interpolation function N_1 in the case of a linear polynomial for every L_1, N_2^{ℓ} for L_2 and N_3^{ℓ} for L_3. This can be achieved if we consider the case of a linear triangular element with node labelling 100, 010 and 001 and the associated interpolation functions are,

$$N_{100} = L_1 \tag{3.239}$$

$$N_{010} = L_2 \tag{3.240}$$

and

$$N_{001} = L_3 \, . \tag{3.241}$$

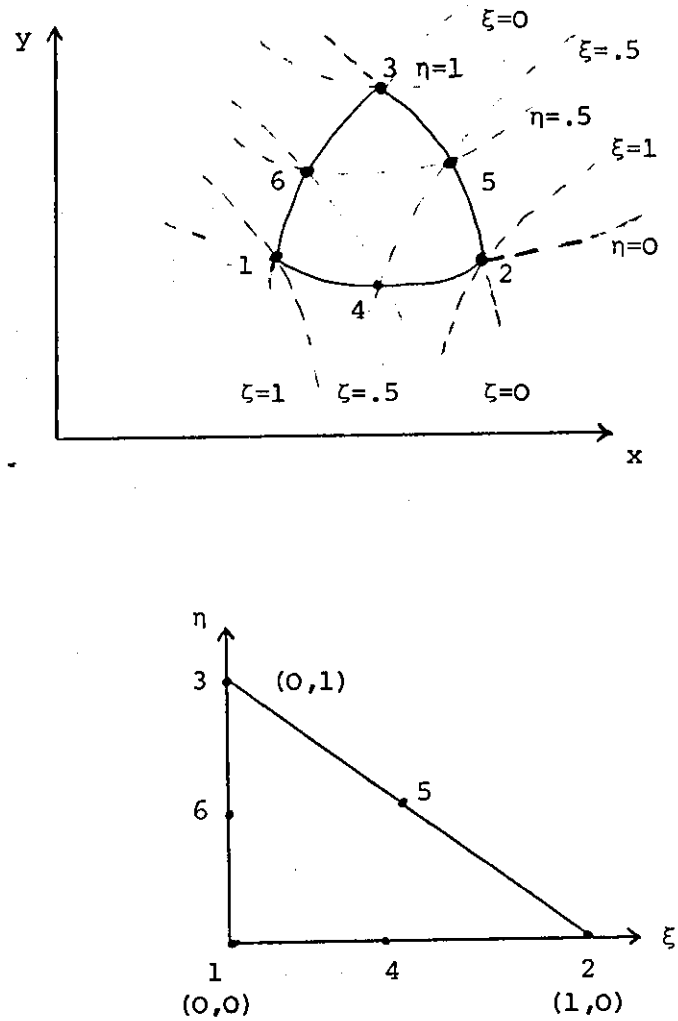
For all these triangular elements the function values, the ϕ's, along a side are uniquely determined by the nodal values along that side and this is the reason for denoting such elements as conforming elements. The evaluation of the matrices for the higher-order elements is usually carried out by numerical integration.

It is important to note that for elements with internal nodes, like the cubic and quartic triangles, it is advantageous to eliminate their degrees of freedom before assembly. This is known as condensation. This can be done because these nodes are not, by definition, connected to any other elements and thus their degrees of freedom do not affect inter-element continuity. This process is desirable because it saves computational effort and the cost that will occur due to the resulting reduced master matrix size after assembly. This is particularly useful if a band solver is to be used in solving the resulting FE equations. However, if a frontal algorithm is to be used its effect is much less. A similar technique called substructuring can be used to divide a hyper complex structure into substructures each of which is still a complex structure but can be solved within the available computing resources. The details of these techniques can be found in Wilson [1974].

3.7 THE ISOPARAMETRIC ELEMENTS

When the boundary of the region is not straight i.e. curved, the mesh must be refined enough to accommodate the curvature of the boundary. Consequently, many straight sided elements can substitute for a curved boundary. A much better approach is to use finite elements with curved sides at curved boundaries. This will not only give better accuracy, but also a lesser number of elements can be used. Among the first pioneers of this idea were Irons [1966] and Ergatoudis et al [1968]. The basic idea behind the isoparametric elements is to map simple geometric shapes in local coordinates into distorted shapes in the global coordinates. The same interpolation functions used to define the field variable within the element are used to define the element shape and hence the naming isoparametric. Linear isoparametric elements can have straight sides only while higher order elements have curved sides. Although it is possible to define subparametric and superparametric elements where in the former the interpolation functions used to define the element shape is of lower order than that used for the field variable and in the latter the opposite, they are rarely used. Since the line and the quadrilateral elements will be covered in Chapter 4 of this thesis, it may be useful to discuss the triangular element here.

Consider the quadratic 6-nodestriangular elements with curved sides as shown in Figure (3.13a). The mapped triangle is shown in Figure (3.13b) in the ξ-η plane. The analysis is essentially following that proposed by Mitchell et al [1971]. We notice that out of the three local coordinates ξ-ζ-η [xi-zeta-eta] only two are independent since at any point $\xi+\zeta+\eta=1$. Assuming a quadratic shape function



(a) The transformed (distorted element)

(b) The parent element

FIGURE 3.13: The isoparametric quadratic triangular element

of the form,

$$S(x,y) = a_1+a_2x+a_3y+a_4xy+a_5x^2+a_6y^2 \quad , \tag{3.242}$$

where the 6 unknown coefficients are uniquely determined knowing the values of S(x,y) at the six nodes. The transformation from the x-y plane to the ξ-η plane is given by,

$$x = \zeta(2\zeta-1)x_1+\xi(2\xi-1)x_2+\eta(\eta-1)x_3+4\xi\eta x_4+4\eta\zeta x_5+4\zeta\xi x_6 \tag{3.243}$$

and $$y = \zeta(2\zeta-1)y_1+\xi(2\xi-1)y_2+\eta(2\eta-1)y_3+4\xi\zeta y_4+4\eta\xi y_5+4\zeta\eta y_6 \quad (3.244)$$

or in a more compact form,

$$S(x,y) = \zeta(2\zeta-1)s_1+\xi(2\xi-1)s_2+\eta(2\eta-1)s_3+4\xi\zeta s_4+4\eta\xi s_5+4\zeta\eta s_6 \quad (3.245)$$

Since $\zeta=1-\xi-\eta$ this equation can be written as

$$S(x,y) = s_1+\xi(4s_4-3s_1-s_2)+\eta(4s_6-3s_1-s_3)+\xi^2(2s_1+2s_2-4s_4)$$
$$+\eta^2(2s_1+2s_3-4s_6)+4\xi\eta(s_1-s_4-s_6+s_5) \quad (3.246)$$

It is clear that ξ and η can be determined from the global coordinates (x,y) by solving the quadratic equations simultaneously though it is not easy to have a closed form expression for these equations.

In the special case where one side only of the triangle is curved, these expressions can be simplified. Assume that side 2-3 is the only curved one, then:

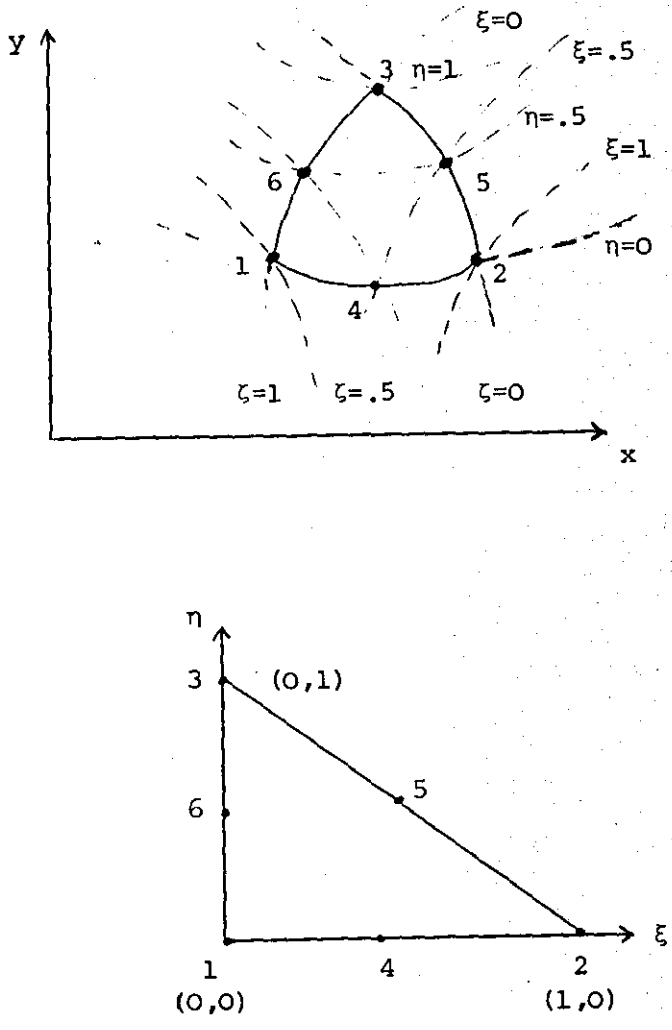
$$x = x_1-a\xi\eta+(x_2-x_1)\xi+(x_3-x_1)\eta \quad (3.247)$$

and $$y = y_1-b\xi\eta+(y_2-y_1)\xi+(y_3-y_1)\eta \ , \quad (3.248)$$

where, $$a = 2x_2 + 2x_3 - 4x_5 \quad (3.249)$$

and $$b = 2y_2 + 2y_3 - 4y_5 \quad (3.250)$$

It should be noticed that by this quadratic approximation to the original curved side it is possible to model any region with curved boundaries using the quadratic triangular elements with one curved side only at the curved boundary. In fact, this element will be used in some of the problems solved in this thesis.



(a) The transformed (distorted element)

(b) The parent element

FIGURE 3.13: The isoparametric quadratic triangular element

of the form,

$$S(x,y) = a_1+a_2x+a_3y+a_4xy+a_5x^2+a_6y^2 \quad , \tag{3.242}$$

where the 6 unknown coefficients are uniquely determined knowing the values of S(x,y) at the six nodes. The transformation from the x-y plane to the ξ-η plane is given by,

$$x = \zeta(2\zeta-1)x_1+\xi(2\xi-1)x_2+\eta(2\eta-1)x_3+4\xi\zeta x_4+4\eta\xi x_5+4\zeta\eta x_6 \tag{3.243}$$

and $$y = \zeta(2\zeta-1)y_1+\xi(2\xi-1)y_2+\eta(2\eta-1)y_3+4\xi\eta y_4+4\eta\zeta y_5+4\zeta\xi y_6 \quad (3.244)$$

or in a more compact form,

$$S(x,y) = \zeta(2\zeta-1)s_1+\xi(2\xi-1)s_2+\eta(2\eta-1)s_3+4\xi\eta s_4+4\eta\zeta s_5+4\zeta\xi s_6 \quad (3.245)$$

Since $\zeta=1-\xi-\eta$ this equation can be written as

$$S(x,y) = s_1+\xi(4s_6-3s_1-s_2)+\eta(4s_5-3s_1-s_3)+\xi^2(2s_1+2s_2-4s_6)$$
$$+\eta^2(2s_1+2s_3+4s_5)+4\xi\eta(s_1+s_4-s_6-s_5) \quad (3.246)$$

It is clear that ξ and η can be determined from the global coordinates (x,y) by solving the quadratic equations simultaneously though it is not easy to have a closed form expression for these equations.

In the special case where one side only of the triangle is curved, these expressions can be simplified. Assume that side 2-3 is the only curved one, then:

$$x = x_1+a\xi\eta+(x-x_2)\xi+(x-x_3)\eta \quad (3.247)$$

and $$y = y_1+b\xi\eta+(y-y_2)\xi+(y-y_3)\eta \ , \quad (3.248)$$

where, $$a = 2x_2 + 2x_3 - 4x_4 \quad (3.249)$$

and $$b = 2y_2 + 2y_3 - 4y_4 \quad (3.250)$$

It should be noticed that by this quadratic approximation to the original curved side it is possible to model any region with curved boundaries using the quadratic triangular elements with one curved side only at the curved boundary. In fact, this element will be used in some of the problems solved in this thesis.

3.8 CONVERGENCE OF THE FEM

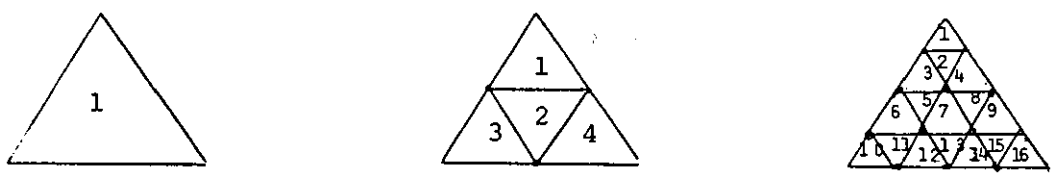
In the FEM, convergence may be thought of by different approaches [Babuska and Szabo, 1982]:

(i) The basis functions of each finite element can be fixed and the diameter of the largest element, denoted by h_{max}, is decreased. This mode is called the h-convergence and its computer implementation is called the h-version of the FEM.

(ii) The finite element mesh can be kept fixed and the minimum order of the polynomial basis functions, denoted by p_{min}, is increased. This mode is called the p-convergence and its computer implementation is called the p-version of the FEM.

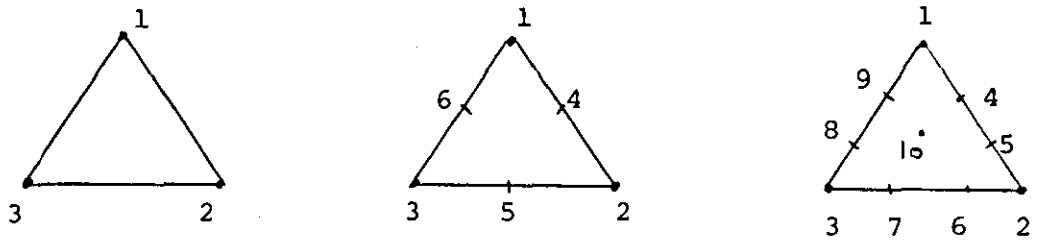
(iii) A mixture of the two approaches, let us call it the h-p-convergence, and its computer implementation the h-p-version of the FEM.

As an example, Figure(3.14) shows the two versions for a simple triangular element. In Figure (3.14a), the initial triangular element is subdivided into smaller triangles all of them are of the same type, using linear basis functions. In Figure (3.14b), the initial triangular element is refined by increasing the order of the basis function.



1-linear element 4-linear elements 16-linear elements

(a) The h-version of FE



Linear element Quadratic element Cubic element

(b) The p-version of FE

FIGURE 3.14: h- and p-versions of the FEM

In practice, however, most of the computer software implement the h-version of the FEM. This may be due to the following factors:

(i) Programming the h-version is substantially easier compared to the p-version.

(ii) The structure of the master matrix in the h-version will be essentially the same, i.e. the sparseness pattern will be the same in the h-version. On the other hand, this pattern is not retained in the p-version.

(iii) The sparseness in the h-version is bigger compared to that in the case of the p-version. However, we must notice that the overall matrix size may not be bigger since in the p-version a smaller number of elements are used.

(iv) In practice, it is easier to decrease the value of h_{max}, i.e., refine the mesh many times. On the other hand, the value of p_{max} cannot be increased to more than 3 or 4 at most.

Nevertheless, the p-version usually gives better convergence than the h-version. In the case of corner singularities, the rate of

convergence of the p-version is exactly twice that of the h-version [Babuska and Szabo, 1982]. More details about the theoretical foundations for the rate of convergence of both the h- and p-versions can be found in Babuska et al [1981], Babuska and Door [1981] and Babuska et al [1979]. Here, a practical example is used to demonstrate the rate of convergence of both approaches.

Consider the Poisson's equation,

$$\nabla^2\phi = 4-2(x+y) \quad , \tag{3.251}$$

in a unit square as shown in Figure(3.15). The boundary conditions are:

$$\phi(0,y) = y^2 \tag{3.252}$$

$$\phi(x,0) = x^2 \tag{3.253}$$

$$\frac{\partial\phi}{\partial x}(1,y) = 2-2y-y^2$$

and

$$\frac{\partial\phi}{\partial y}(x,1) = 2-2x-x^2 \tag{3.254}$$

The exact solution for this test problem is:

$$\phi(x,y) = x^2+y^2-xy(x+y) \tag{3.255}$$

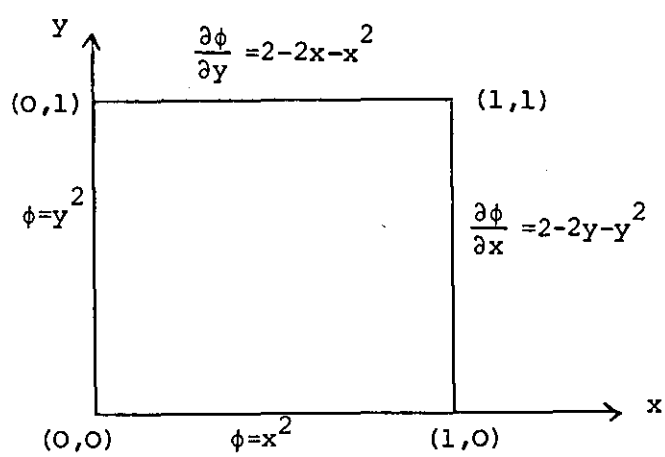


FIGURE 3.15: Sample problem

This problem is solved using both approaches, i.e. the h-version and the p-version. The region is divided up to triangles of increasing number and for each discretization triangles of different orders are used. The values are computed for values of (x,y) of spacing .2 in both directions, i.e. at points (0,0), (0,.2),(0,.4),...,(.2,0),(.2,.2),..., (1,1). The error norm L_2 is defined by:

$$L_2 = \sqrt{\sum_{\forall i} \sum_{\forall j} (\phi_{ij} - \hat{\phi}_{ij})^2} \quad , \tag{3.256}$$

where ϕ_{ij} is the exact value of ϕ at the point (x_i, y_j) and $\hat{\phi}_{ij}$ is the computed value obtained by the FE modelling. The results are shown in Table 3.1. It is evident that with a fewer number of elements of higher order better accuracy can be obtained. We must notice also that since the exact solution of ϕ is a polynomial of order 3, the cubic triangular elements give the best results with a fewer number of elements. More problems will be solved using both approaches in Chapter 7 of this thesis.

Number of Elements	L_2 Error Norm		
	Quadratic	Cubic	Quartic
4	.2143	$.4712\times10^{-3}$	$.4591\times10^{-3}$
8	$.4273\times10^{-2}$	$.1463\times10^{-5}$	$.1892\times10^{-5}$
16	$.576\times10^{-3}$	$.1941\times10^{-6}$	$.2073\times10^{-6}$
32	$.6103\times10^{-4}$	$.9362\times10^{-7}$	$.8250\ 10^{-7}$

TABLE 3.1: Comparison of h- and p-versions for test problem

One of the simple, yet powerful, tests of the convergence of elements is the patch test [Irons and Razzaque, 1973]. This test can be stated in several ways. In its general case it can be formulated as follows [Davis, 1980]:

Given a solution ϕ_o for the PDE defined by $L\phi=f$, assume that round the perimeter of any arbitrary patch of elements values of ϕ are chosen to be equal to ϕ_o; then, if the approximate solution $\hat{\phi}$ to this problem inside the patch, is identical with ϕ_o there, then the test is passed and the element will yield convergence. This is best illustrated by a simple example, [Davies, 1980].

Consider the solution of Laplace's equation in the region shown in Figure 3.16.

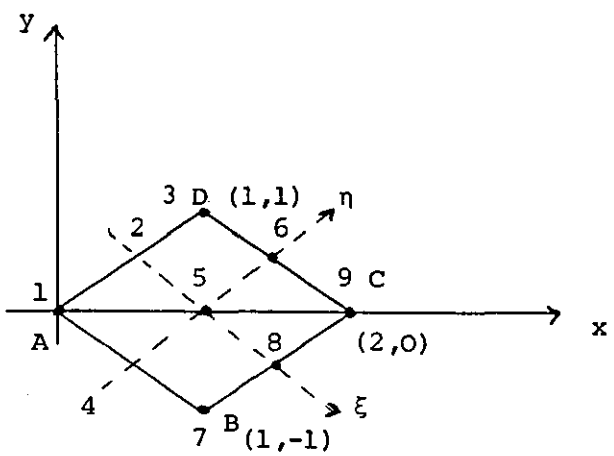


FIGURE 3.16: A square region

What we like to test is the bilinear rectangular element. In the patch shown we assume a test solution x-y+1, this will lead to nodal values as follows:

$$\phi_1 = \phi_2 = \phi_3 = 1; \quad \phi_4 = \phi_6 = 2 \text{ and } \phi_7 = \phi_8 = \phi_9 = 3$$

and $\phi_5=2$.

The element interpolation functions for this element are given in

[Huebner and Thornton, 1982], viz.

$$\phi^e(\xi,\eta) = \sum_{i=1}^{4} N_i(\xi,\eta)\phi_i \ , \tag{3.257}$$

where,

$$N_1(\xi,\eta) = \frac{1}{4}(1-\xi)(1-\eta) \ , \tag{3.258}$$

$$N_2(\xi,\eta) = \frac{1}{4}(1+\xi)(1-\eta) \ , \tag{3.259}$$

$$N_3(\xi,\eta) = \frac{1}{4}(1+\xi)(1+\eta) \ , \tag{3.260}$$

$$N_4(\xi,\eta) = \frac{1}{4}(1-\xi)(1+\eta) \ . \tag{3.261}$$

Consider the element whose nodes are 1,4,5 and 2. In this element ϕ will be:

$$\begin{aligned}\phi^1 &= N_1+2N_2+2N_3+N_4 \\ &= \frac{1}{4}(1-\xi-\eta+\xi\eta+2+2\xi-2\eta-2\xi\eta+2+2\xi+2\eta+2\xi\eta+1-\xi+\eta-\xi\eta) \\ &= \frac{1}{2}(\xi+3) \ .\end{aligned}$$

But inside this element the relationship between the local coordinate ξ and the global coordinate system is:

$$\xi = 2x-2y-1 \ , \tag{3.262}$$

thus,

$$\phi^1 = \frac{1}{2}(2x-2y-1+3) = x-y+1 \ . \tag{3.263}$$

In a similar manner it is possible to find that ϕ^2,ϕ^3 and ϕ^4 all will have the same solution x-y+1 which is the assumed for the patch when this element passes the patch test.

3.9 ERROR ESTIMATES IN FE

3.9.1 Sources of Errors

Starting from a physical problem and ending by a numerical FE solution a number of approximations have to be done. Assume that a physical system is characterised by a field variable $\bar{\phi}(t)$ in which ϕ is, in general, a vector and t is the time. Assume that the FE solution of this system gives the solution $\hat{\phi}(t)$. The difference between the actual physical quantity $\bar{\phi}(t)$ and the corresponding numerical solution $\hat{\phi}(t)$ is the total error E:

$$E = \bar{\phi}(t) - \hat{\phi}(t) \quad . \tag{3.264}$$

This error is due to the following approximations:

(i) Modelling of the physical problem:

What is solved by a mathematical technique is a mathematical model of a physical problem. During the derivation of the system equations many simplifying assumptions are done whether in the underlying theory, in the geometry or in other parameters of the system. Such assumptions are found in almost every physical problem. They are, however, usually very small and can be neglected provided that these assumptions are reasonable and usually supported by experimental evidence. Examples of these reasonable assumptions are: flow in aquifers is assumed to be essentially horizontal, a plane section in the beam theory remains plane after deformation. These assumptions are valid, of course, within prescribed ranges. For example, to apply the beam theory, the height of the section must be sufficiently small compared to the span otherwise the obtained results will be erroneous. In the following discussion the mathematical modelling errors are assumed to be negligible and therefore they will not be considered, i.e. hereafter solution errors do not

include mathematical modelling errors.

(ii) Discretization Errors:

The actual mathematical system possesses infinite degrees of freedom while its FE model has a finite number of degrees of freedom. Moreover, the FE model may have also simplifications of the geometry of the domain and modifications of the boundary conditions. In addition, to that, in most FE analysis, the terms of the element matrix are computed by numerical integration rather than having explicit exact terms.

(iii) Roundoff Errors:

These are due to the finite word length of computers. The current generation of computers used in FE analysis have a word length that ranges from 16 bits to 64 bits. Consequently, most of the real numbers are represented in these machines approximately. Since most of the FE calculations are done in floating point arithmetic it implies that roundoff errors do exist in almost every FE analysis. In the floating point arithmetic, a number is represented as a mantissa and an exponent.

If error estimates can be established prior to the FE solution they are called a-priori estimates. On the other hand, if error estimates are based on the information obtained from the FE solution they are called a-posteriori estimates.

3.9.2 Error Measures

From an analysis point of view, it is usually more convenient to decompose the solution errors to its simpler constituents as follows. At time t, the difference between the exact solution of the mathematical model $\phi(x,t)$ and the obtained numerical solution $\hat{\phi}(x,t)$ is denoted by r which is the residual [Utku and Melosh, 1984].

$$r = \phi(x,t) - \hat{\phi}(x,t) \quad . \tag{3.265}$$

This residual can be decomposed to three components: r_d, r_r and r_i, where: r_d is the error due to discretization, r_r is the error due to round-off and r_i is the inherited errors at time t and represents the effects of all equation errors of the earlier times. These error components are at any time t, but in fact when solving a time-dependent problem there is a possibility of having another source of error during the solution process, this is the manipulation error [Melosh, 1973]. This error is due to the used solution algorithm. During the solution process, depending upon the solution algorithm, intrinsic characteristics of the system may be altered. It is possible to have a large manipulation error in an intrinsically stable system due to a wrong choice of the solution algorithm. The total solution error at a time t=T is defined now as the sum of equation errors at times $t \leq T$ and the manipulation errors.

Errors are usually measured by one of the error norms which have been explained earlier this chapter. In general a p-norm is defined as,

$$||e||_p = (|e_1|^p + |e_2|^p + \ldots |e_n|^p)^{1/p} \quad , \text{ for } p=1,2 \text{ or } \infty \tag{3.266}$$

A brief discussion of measuring the round-off and the discretization errors is given in the next section.

3.9.3 Round-Off Errors

The exact computation of the round-off errors is not possible. So what is possible is to get a reasonable estimate of its value. Let us assume the actual number to be represented in the computer to be x and its machine representation in floating point is x_f; then the round-off

error ε will be:

$$\varepsilon = x - x_f \ , \tag{3.267}$$

The lower bound of ε can be zero in case the number is one of those which can be represented exactly within the computer word length. On the other hand, the upper bound of the error ε can be computed as follows:

Assume that the real numbers are represented in floating point with mantissa of m bits for the normalised fractional part, and e bits for the exponent. Most computers chop the extra bits rather than perform the rounding operations. Thus the maximum value of the chopped bits will be the sequence,

$$2^{-(m+1)} + 2^{-(m+2)} + \ldots \infty \quad .$$

This is a geometric series and its sum is,

$$\frac{2^{-m-1}}{1-2^{-1}} = 2^{-m}.$$

Therefore, the upper bound of ε is 2^{-m}*exponent. The exponent part is $|x|$, thus:

$$0 \leq |\varepsilon_{number}| \leq 2^{-m} * |x| \tag{3.268}$$

Considering two floating point numbers x and y, the bounds for the round-off errors of their sum is:

$$0 \leq |\varepsilon_{sum}| \leq (|x|+|y|) * 2^{-m} \ , \tag{3.269}$$

and for their multiplication is:

$$0 \leq |\varepsilon_{mult}| \leq |x| * |y| * 2^{-m} \ . \tag{3.270}$$

For a scalar product of two vectors a and b each of n elements the bounds of the round-off error will be:

$$0 \leq |\varepsilon| \leq n * ||a|| * ||b|| * 2^{-m} \ . \tag{3.271}$$

The same inequality applies for matrix multiplication since each element

of the resultant matrix is the scalar products of two vectors of the same length.

From the above equations it is clear that in order to minimize the effect of round-off errors we have to increase the number of bits assigned for the mantissa, i.e. m which can be realised through the use of computers of larger word length or by using double (or higher) length precision arithmetic. In addition to that, small magnitude quantities will give smaller round-off errors. In the case of matrix and vector operations, in addition to the previously mentioned solutions, the length of vector n should be small.

The relative error in the computed value $(x-y)_f$ for two floating point numbers x and y will be:

$$r = \frac{(x-y)-(x-y)_f}{(x-y)_f} = \frac{\varepsilon}{(x-y)_f} \tag{3.272}$$

Using the above inequalities for ε this equation can be written as the inequality:

$$0 \leq r \leq \frac{(|x|+|y|)*2^{-m}}{(x-y)_f} \tag{3.273}$$

From this equation it is clear that if x-y is too small, the effect of round-off errors may be too large. A solution for this situation may be the avoidance of a too refined finite element mesh and the use of double precision arithmetic in critical quantities. In the Gaussian elimination process for the solution of a set of n linear algebraic equations, which is usually used in FEM, it has been proved by Wilkinson [1963] that the cumulative effect of rounding errors on the solution obtained can be related to the effect of rounding errors in the representation of the matrix of coefficients and the vector of constants.

Consider the set of equations defined by:

$$Ax = b \ , \tag{3.274}$$

where A is the ($n \times n$) matrix of coefficients, b is the ($n \times 1$) vector of constants and x is ($n \times 1$) vector of unknowns. Assume that both A and b are normalized such that their elements are all of absolute value <1, and their values are correct to m binary places (the mantissa). Assume further that the arithmetic operations of the Gaussian elimination method with pivoting is carried out using $m+\log_2 n$ binary places. Then, the obtained solution to m binary places is the exact solution of a set of n linear algebraic equations whose coefficients and constants differ from those in the original equations by less than the possible rounding error in the data, i.e. by less than 2^{-m-1}.

3.9.4 Discretization Error

In the FEM the field variable $\phi(x,t)$ is approximated by trial solutions of the form $\psi(x,t)$. Let $\Delta\psi(x,t)$ denote the difference between the actual field value and its assumed trial value, i.e.,

$$\Delta\psi(x,t) = \phi(x,t) - \psi(x,t) \ , \tag{3.275}$$

Considering the functional $I[\phi(x,t)]$, it is possible to write:

$$I[\phi(x,t)] = I[\psi(x,t)+\Delta\psi(x,t)] + \Delta I[\psi(x,t)+\Delta\psi(x,t)] \tag{3.276}$$

where ΔI represents the effect of some probable errors like: (a) the sum of the spatial domain represented by FE may be different than the actual spatial domain, (b) the boundary conditions may not be exactly satisfied by the approximation. Due to the existance of ΔI and $\Delta\psi$ the solution obtained from the functional $I[\psi(x,t)]$ will be deviated from

that obtained from $I[\phi(x,t)]$. The difference between the two sets of equations is the equation discretization error vector r_d. It can be shown that [Becker et al, 1982]:

$$||r_d|| \cong O(h^p) \ , \tag{3.277}$$

where h is the mesh size and p is a power related to the highest order of the interpolation functions used in the approximation. This actually gives the basis of the two versions usually adopted in FE refining, viz. the h-version and the p-version which have been discussed before. It should be noted that if the discretization error is monotonically decreasing with decreasing the mesh size, then, using Richardson's extrapolation it is possible to get a better solution as explained earlier in this chapter. It is only recently [Dunavant and Szabo, 1983; Kelly, et al, 1983] where some a-posteriori error estimates for some FE problems have been established. However, two points should be noted: (a) Almost none of the existing FE software available handles the error problem. An exception to this is the FEARS (Finite Element Adaptive Research Solver) which is a research-type FE software developed at the University of Maryland, U.S.A., (b) practically, in order to assess the errors by computer experiments, particularly for new problems solved by FEM; the same problem should be solved several times (at least three), with a gradually refined mesh. Sometimes, it may be useful to use a Richardson's extrapolation technique to conclude a better approximation. Utilizing higher order elements can then be applied for best results.

3.10 SPECIAL PROBLEMS IN FE ANALYSIS

In this section some special problems in the FEM are briefly highlighted.

3.10.1 Time-Dependent Problems

Two approaches can be used to model time-dependent problems using the FEM. These are:-

(i) Considering time as an extra dimension for the problem to be solved. Thus, a 2-D problem will have now 3 dimensions: x,y and t. The shape functions are defined in terms of these dimensions, i.e.,

$$\phi^e(x,y,t) = \sum_{i=1}^{n} N_i(x,y,t)\phi_i \ , \tag{3.278}$$

where ϕ^e stands for the elemental field variable ϕ, N_i the shape functions and ϕ_i the nodal field variables. This approach can be considered as a natural extension to the steady-state FE formulation already discussed. It should be noted that the cost of computation with this extra dimension is usually very high so that this approach is rarely used in practice.

(ii) Considering the problem at any one instant of time and the nodal variables are considered as functions of time while the space variables are used in the FE analysis. This approach leads to a system of ordinary differential equations which can be solved using other techniques, usually finite differences. This approach is usually used in practice and indeed it is the one to be used within this thesis as will be explained in later chapters. In this case the FE model will have the form,

$$\phi^e(x,y,t) = \sum_{i=1}^{n} N_i(x,y)\phi_i(t) \tag{3.279}$$

3.10.2 Mixed and Hybrid Elements

In most of the FE formulations for structural and solid mechanics the field variable is chosen to be the displacement field. Consequently, the displacements are assumed, and forced to satisfy certain continuity compatibility conditions. The stresses computed after the solution for displacements are nearly continuous. Solutions for such situations should be covered by an ideal postprocessor. This will be explained in a later chapter. What we are going to explain here is that it is possible, though not used in practice, to define other variables as the field variable and hence an appropriate FE formulation can be developed. Perhaps the most known formulation other than the displacement one is that based on assumed stress field. The associated functional in this case is the complementary energy and the primary unknowns are the nodal stresses. FE models based on this formulation are termed stress-based FE models. If a mix of stress and displacement quantities are considered as independent unknowns the resulting FE formulation is termed mixed elements FE model. An example for a mixed element is the plate bending element shown in Figure (3.17), where the nodal variables are the lateral

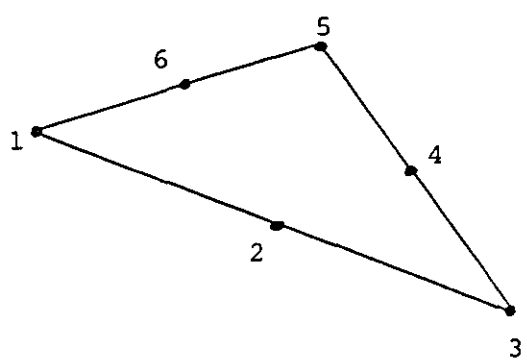


FIGURE 3.17: Mixed element for plate bending

displacements at nodes 1,3,5 and the bending moments normal to the edge at other nodes, i.e. at nodes 2,4 and 6. The associated functional in this case is the Stationary Reissner energy.

Hybrid FE models are obtained if in addition to one field variable the displacement or stress, other variable (i.e. stress or displacement) are introduced and the parameters that correspond to the additional variables are eliminated at the element stage before assembling the element equations. For example, it is possible in a plate element to consider the field variable within the element to be the moments M_x,M_y and M_{xy} [Rao, 1982] while the lateral displacements of any edge is quadratic or cubic in the edge-parallel coordinates and governed by rotations and displacements of nodes on the edge. More information on this approach can be found in Zienkiewicz [1977].

3.10.3 Infinite Finite Elements

When modelling an infinite region using finite elements, only a finite part of the region is considered and the solution at infinity is approximated by that at the boundaries of the finite part or conversely, the boundary conditions at infinity are assumed to happen at the boundaries of the finite part. Consider, for example, a well in an aquifer where the aquifer itself is assumed to extend to ∞ in the x-y plane. Similar situations occur in electromagnetic field problems and in ocean engineering models. In general, the region R is partitioned into two subregions R_c and R_i, where $R = R_c \cup R_i$. The subregion R_c stands for a closed domain that extends up to the range within which the solution varies significantly while R_i stands for the rest of the region which extends to ∞. Of course, we do not know the boundaries of R_i in advance.

However, engineering judgement and experience are helpful in many problems. In case no previous experience is available, for the problem to be solved, then the problem is solved several times moving the boundaries of R_c until a satisfactory solution is obtained. Another approach is to use "infinite finite elements" where the elements possess some functions of a decaying nature and the integration is performed over the infinite domain. One of these decaying functions is that proposed by Ungless (1973) given by,

$$f(\xi) = \frac{1}{1+\xi/L} \quad , \tag{3.280}$$

where L is the effective length. This function is used to reduce the magnitude of u as ξ increases where:

$$f(\xi) = 1 \text{ at } \xi=0$$

$$f(\xi) = .5 \text{ at } \xi=L \tag{3.281}$$

and $$f(\xi) \to 0 \text{ as } \xi\to\infty \quad .$$

Bettess (1977) uses Lagrangian polynomials as interpolation functions and multiplies them by an exponential decay function which can best fit for the rapidly decaying phenomena. Assuming the Lagrangian interpolation function to be $G(\xi)$ the modified function will be:

$$f(\xi) = G(\xi)e^{-\xi/L} \quad . \tag{3.282}$$

Again, the decaying multiplier is 1 at $\xi=0$ and $\to 0$ as $\xi\to\infty$.

3.11 COMPARISON OF THE FINITE ELEMENT METHOD WITH OTHER COMPUTATIONAL TECHNIQUES

There are many computational techniques that can be used to solve PDE's and it is impossible to compare the FEM with all other methods. So, the comparison will be limited to what are considered as the most competitive methods: the finite differences and the relatively new method, the boundary element method. Before discussing the criteria used in the comparison it is convenient to give the metalgorithms for the three methods. The word metalogorithm is due to Rice [1975]. It consists of a set of blocks or components which represent a class of algorithms each of which has the form and attributes specified by the metalgorithm. A metalgorithm can be described in English statements or in the form of flowcharts. In fact, metalgorithms are used as a framework or theory to study algorithms [Houstis et al, 1975].

(1) The metalgorithm for the FEM:

(i) The domain is divided into a set of finite elements.

(ii) A choice is done for interpolation functions associated with elements.

(iii) A processor is used to generate a set of algebraic equations from the PDE or an associated functional.

(iv) A processor is used to generate a set of algebraic equations from the auxiliary conditions.

(v) An equation solver for the system of equations generated by components (iii) and (iv) is used.

(vi) Measurement of results and termination of the algorithm.

(2) The metalgorithm for the finite difference methods:

(i) A grid of nodal points are placed on the domain.

(ii) A processor that generates a set of algebraic equations from the PDE is utilized.

(iii) A processor is used to generate a set of algebraic equations from the auxiliary conditions.

(iv) An equation solver is used for the system of equations resulting from (ii) and (iii).

(v) Measurement of the results and termination of the algorithm.

(3) The metalgorithm for the boundary element methods:

(i) The boundary of the region is discretized into a set of boundary elements.

(ii) The original PDE is transformed into an equivalent set of boundary integrals.

(iii) A processor is used to generate a set of algebraic equations from the boundary integrals.

(iv) A processor is used to generate a set of algebraic equations from the auxiliary conditions.

(v) An equation solver is utilized to solve the system of equations resulting in (iii) and (iv).

(vi) Measurements of the results and termination of the algorithm.

The criteria used in comparison are then: (i) domain modelling, (ii) handling of auxiliary conditions, (iii) processor properties and (iv) user convenience.

(i) Domain modelling:

The ability of the FEM to model an arbitrary geometry through the use of the higher order isoparametric elements is an advantage of this method over the others.

Although it is possible to model such geometries using finite

differences (FDM) as explained earlier in this chapter, but the approximation is usually poorer compared to the FEM. It is easier to use a mix of elements of various types, sizes, shapes and gradation in the FEM as compared to other methods. An advantage of the boundary element method (BEM) over the other methods is that a smaller number of elements is required since the discretization process is applied to the boundary only rather than the whole region.

(ii) Handling of auxiliary conditions:

Derivative boundary conditions are treated in the FDM utilizing fictituous nodes that reside outside the domain itself. This is somewhat unnatural. Application of such conditions in other methods do not require fictitious elements.

(iii) Processor properties:

The processors used in each method to generate the set of algebraic equations are quite different. In the FDM the derivatives are replaced by function values at the grid points. This is a straightforward process and no assembly is required for the resulting equations. This is not the case in the other two methods where an elaborate assembly process is required to form the final set of equations. Another factor is the characteristics of the master matrix of the resulting set of equations. In the FEM the master matrix is usually symmetrical, positive definite, sparse and banded or can be transformed to a banded one.

These "good" properties save computer memory and execution time for the solution. On the other hand, in the FDM the master matrix is usually sparse but in many cases is unsymmetric. In the BEM the resulting matrix is normally dense and unsymmetric. However, since the boundary is discretized only, the size of the master matrix is much less

as compared to the other methods.

(iv) User convenience:

Three aspects are considered here. First, the input data required in the analysis is much less in the case of the BEM compared to the other method. This is apparently due to the fact that the boundary is discretized in the BEM rather than the whole domain. Consequently an easier user-interface can be achieved in the BEM.

Secondly, the domain of applications of the FDM and FEM seems to be wider than that of the BEM. The solution obtained in the BEM is that on the boundary only and the solution inside the domain has to be computed. In the other methods, the solution is obtained inside and on the boundary of the domain directly. Third, there are many standard, reliable computer software for the FDM and the FEM that can be used to solve a wide range of problems. This is not the case with the BEM where very few specialized codes are produced.

CHAPTER 4

COMPUTER IMPLEMENTATION OF THE FINITE ELEMENT METHOD

TABLE OF CONTENTS

4.1 INTRODUCTION

The FEM is a computational technique which requires the essential use of a digital computer for practical applications. Since the FEM was originally developed by structural engineers, it is not surprising that the first computer implementation of this method was for structural analysis systems.

FE programs in the sixties were typical in that they had their inputs on punched cards and outputs on line printers with in-core solution techniques. It is possible to identify five milestones in the computer implementation of F.E.:

(i) In the late sixties the use of direct access storage devices like magnetic discs and the building of the virtual memory machines had enlarged the capabilities of F.E. programs to handle fairly big problems, e.g. SAP I [Wilson, 1970].

(ii) In the early seventies the out-of-core techniques and sparse matrix methods opened the way to solve very big problems e.g. NASTRAN [MacNeal, 1970] and George [1971].

(iii) In the mid-seventies the evolution of super mini-computers allowed the FE user to get access to large systems on a relatively cheap machine in a time-sharing environment. The use of interactive programs was dominant.

(iv) In the late 70's and early 80's and due to large scale integration (LSI) and very large scale integration (VLSI) technological advances, micro- and super-computers started to be used in scientific computations. This gives the possibility for in-house F.E. computations on a small size machine which cost less than a time-sharing service with a mainframe. The

use of MIMD and SIMD computers for extremely large problems was also exploited. The use of graphics to enhance man-machine interface is another feature of these F.E. programs.

(v) The last milestone is the current state-of-the-art for which software integration is a major objective. The use of database management systems (DBMS) and some techniques from the field of artificial intelligence (AI) are among the features of these F.E. programs. Other features are: the extensive use of colour graphics and digitizers.

In this chapter F.E. software is classified according to different criteria such as use and size. The data structures necessary for implementing F.E. on computers are reviewed and discussed. Since most of F.E. programming is done in Fortran a proposed set of extensions is given to facilitate scientific programming in general and F.E. programming in particular. Four F.E. programs are presented. The first is a model of a mainframe computer implementation; the second and the third are mini-computer implementations; and the fourth is a micro-computer implementation. The first two are adapted by the author while the last two are developed by him. In choosing these programs the following factors were considered:

(i) The program must be available in source code and not a proprietary one.

(ii) The program must be already loaded and tested on the available computer hardware.

(iii) Program documentations must be available.

In view of the above, the MSAP program [Kaldjian et al, 1982] is chosen as a mainframe computer implementation, the ELASTIC [Sharaf Eldin

and Evans, 1987] and STRAP [Turaby and Sharaf Eldin, 1978] are chosen as samples of mini-computer implementations, and the IFEP [Sharaf Eldin, 1983a and 1985a] program is chosen as a sample of a micro-computer implementation.

The importance of pre- and post-processors for F.E. systems is evident. Two sections are devoted to these two important subjects. Special topics in computer implementation of FEM are discussed which include F.E. on parallel systems, use of database technology in F.E. and standards for F.E. codes. Finally three quantitative approaches for the selection of FE software are given.

4.2 CLASSIFICATION OF FINITE ELEMENT SOFTWARE

It is possible to classify finite element software according to several criteria such as: applicability, function, size and approach.

Considering the domain of applications we can distinguish four different types of F.E. software:

(i) General Purpose Practical Packages:

These are the most widely used packages for F.E. Most of these packages are oriented for structural analysis and are used for practical problems. The main features of such packages are: a broad spectrum of capabilities, rich library elements and most of them are in proprietary code. Examples of such packages are ADINA [Bathe, 1978] and MSAP [Kaldjian et al, 1982].

(ii) Special Purpose Packages:

These packages are usually developed to solve very specialized problems for which no general-purpose package exists. As an example the IFEP [Sharaf Eldin, 1983a] which is used to simulate some ground-water problems. This program will be discussed in Section 4.8 in detail.

(iii) Educational Packages:

These are packages used for teaching purposes. They are usually available in source code. Such packages are simple in design and contain a good user interface to make the communication with the package easy. Examples of such packages are: FEMSKI [Irons and Shrive, 1983], FINEL [Hitchings, 1975] and STRAP [Turaby and Sharaf Eldin, 1978].

(iv) Research-oriented Packages:

These are programs which have been developed for research purposes in F.E. Unfortunately, no general-purpose packages are available that allows the implementation and testing of new algorithms in F.E. in an

easy way. Most of the researchers in computer implementation of FE start their work from an existing package and attempt to modify it for their application such as Sada Costa [1980]. One of the attempts to solve this problem is to add special capabilities for one of the available high level programming languages as language extensions and thus allow the FE researchers to test different algorithms. Example of such an approach is that developed by Collins [1980].

FE systems can be classified according to their size into three different categories:

(i) F.E. packages that require a super or a mainframe computer to run. These programs are usually general purpose large scale software used for practical applications in industry. These systems are usually expensive and most of them are available in binary code only. They require a large memory size supported by DASD for back-storage. In addition, graphics devices are invariably required to obtain the plotted results. It is worthwhile to mention that none of the known F.E. packages fully utilize super-computer capabilities and in particular parallel processing facilities. These are expected soon. Examples of mainframe implementations are ADINA [Bathe, 1978] and MSAP [Kaldjian et al, 1982].

(ii) In the mid 70's the powerful super mini-computers like VAX and PRIME made it possible to have F.E. packages that could be implemented on mini-computers. Due to their relative cheapness compared to mainframe computer prices it sometimes becomes an cost-effective alternative. Examples of programs that can be implemented on mini-computers are: Gattass and Abel [1983] and STRAP [Turaby and Sharaf Eldin, 1978]. Among

the problems encountered when using mini- and microcomputers for F.E. analysis is the limited stack size. This problem is explained and a solution is given in Chapter 5 of this thesis.

(iii) During the last few years and due to the advances in the LSI and VLSI technologies, many micro-computers with greater capabilities have been introduced. However, very few general FE packages are available using such micro-systems. This may be due to the fact that developing a new F.E. code from scratch is time consuming and expensive process and the developments with micro-computer are changing very rapidly. Examples of micro-computer implementations are Yamada and Okumura [1980] and Sharaf Eldin [1985].

It is also possible to classify F.E. systems according to their functions as:

(i) Pre-processor Packages:

These packages are used as a man-machine interface to facilitate the data input to the main F.E. processor. The definition and functions of pre-processors and post-processors will be discussed in detail in later sections of this chapter. Examples of pre-processors are the GIFTS III [Kamel and McCabe, 1976] and PREMSAP [Kaldjian, 1976].

(ii) Processors that perform the actual F.E. analysis

(iii) Post-processor packages:

These are concerned with graphical representation of results and correction and checking of the solution. Examples of post-processors are: MSAPPOST [Kaldjian, 1977] and MENTAT [Marc, 1980].

Finally, it is possible to classify F.E. systems according to their approach. Two types can be figured out: Mathematical and

Engineering. Most of the available F.E. systems are engineering type software where the input to such systems is a collection of elements, nodes, material properties and boundary conditions. The characteristics of elements and the solution steps are all stored in the system. On the other hand a mathematical F.E. software accepts its input as the governing differential equations with the necessary boundary conditions. An example of a mathematical F.E. system is the TWODEPEP package which will be discussed in Chapter 6 of this thesis.

4.3 DATA STRUCTURES FOR FINITE ELEMENTS PROGRAMMING

The problem of data structures for F.E. programming has been given less attention in the literature than it deserves. Many of the existing large finite element software packages have their roots in the late sixties and early seventies. The fast progress that occurred in computer scientific computation is not accompanied by a parallel one in software development. Although progress in software theory is relatively fast, the actual implementation is a great deal slower. As an evidence, most of the existing large F.E. programs do not have built-in interactive input/output facilities. Almost none of them is built around a truely database management system.

One of the main reasons for this situation is that when a F.E. developer designs his system he is faced with problems of data structuring, storage management and data management. The existing programming languages used in F.E. programming, mostly Fortran, do not possess the adequate capabilities to enable the easy command of data structuring, storage and data management. As a consequence, the F.E. developer finds himself involved more and more in the organizing, storing and management of data. The use of a data structure may, therefore, act as an interface between the F.E. developer and the computer.

The first attempt to build a data structure for F.E. systems as reported in the literature was that of Bettess [1977]. In his work a direct access disc file is mapped onto a fixed area of core. The access is granted via a Fortran function NO(I) where the contents of the Ith word of the disc file can be transferred to a core location or vice versa. The data structure has two main parts; a fixed permanent part that holds the title block and the pointer block. The title block holds the problem

title while the pointer block holds pointers to element data, node data and so on. The other parts of the data structure are the entries: node entries, element/node entries and element entries. The node entry holds the node number, the node co-ordinates, and a pointer for the first element/node entry for that node. The element entry holds the element number, type, and a pointer to the first element/node entry for this element. The element/node entry holds the element number, the node number and a pointer to the next element/node entry for this node. A suite of 8 subroutines are available to store and retrieve the nodes, elements and connectivity data. This data structure could be considered as a simple solution for the considered problem. However, it must be emphasized that the use of database management systems (DBMS) to handle data management in F.E. systems is a more efficient way. This will be explained later in more detail.

It is safe to predict that more than 90% of the developed F.E. programs are programmed in Fortran. This may be due to the fact that Fortran is the oldest high level programming language for scientific and engineering applications. A large amount of investment has been spent on the existing software and in training engineers and scientists in Fortran. For these reasons it is believed that Fortran will be dominant for some time in F.E. programming. The question which arises is: does Fortran possess the adequate data structures and management of F.E. programming? Before giving an answer, let us first state what a programmer expects from a programming language for F.E. programming and then examine Fortran in the light of these requirements.

In general a programming language should provide the following capabilities [Browne, 1976]:

- A set of primitive and system defined data structures.
- A set of operations on them.
- A set of composition rules that enable the composing of primitive and system defined data structures into structures appropriate for the application.
- The same but for operations.
- A set of capabilities to manage the computer resources allocated to a program specially the memory size and filing system.
- An interface to capabilities defined in the operating system or in the system libraries.

Before examining Fortran we must know that there are two major versions of Fortran available in the computer community known as Fortran 66 and Fortran 77. Most of the well known large scale F.E. systems are based on Fortran 66 which lacks most of the structuring programming constructs. However, our examination for Fortran will be based on both versions: Fortran 66 and Fortran 77.

The primitive numerical data objects in Fortran are the data types defined explicitly by a type statement or implicitly by its first letter (the I,J,K,L,M and N rule). This includes: integer, real, complex and double precision. In Fortran 77 it is possible to define these data types through an implicit statement as well. A character data type is also available in Fortran 77. The system defined data structure is the array which is a collection of identical objects and can span over a number of dimensions: 3 in Fortran 66 and 7 in Fortran 77. However, most of the existing Fortran compilers permit more than 7 dimensions for the array declaration. The set of operations defined in Fortran are quite powerful for data elements but no operations are allowed on

system data structures. Thus we can write:

$$A = B + C$$

where A,B and C are data objects (scalars) but not arrays. One exception is the I/O of arrays which could be done directly. However, Fortran gives the facility to compose user-defined operations using the system defined operations through functions and subroutines.

The other three capabilities are very limited in Fortran. These are: the composition of system defined data structures into more user-oriented data structures; the capability to manage the allocated computer memory and the adequate interface to operating systemsand system libraries. To exemplify, let us consider the problem of the storage of the master matrix K (stiffness matrix) in a F.E. program. First of all it is well-known that K is sparse and in many cases banded. However no data structure in Fortran is available except the array and thus K is stored as a rectangular array with the necessary vectors that store pointers to different elements of the array. Thus, it is the programmer's responsibility to establish the whole mechanism to manipulate and store this sparse matrix. Another problem is that since Fortran allows the allocation of a fixed amount of memory it implies that K will be dimensioned in the main program so as to fit the largest problem to be handled by this program. This means that when solving problems of smaller size, memory wastage will occur.

This is due to the fact that Fortran does not possess the capability to manage the memory allocated to a program. Fortran does not allow the direct interface with the operating system and system libraries. However, many of the Fortran compilers nowadays define some system calls that give a fairly limited amount of interface to some of the

operating system tables and system libraries. Nevertheless, some 'tricks' have been practiced by engineers to overcome some of these handicaps. One of these 'tricks' is to have the whole memory allocated to the F.E. program in the form of a single vector and hold the suitable pointers to partition this vector.

4.4 PROPOSED FORTRAN EXTENSIONS

The following extensions to Fortran are proposed in order to facilitate F.E. programming, in particular, and scientific computations in general. The underlying philosophy for these extensions are:

- Enrich the system defined data structures
- Increase the capabilities of user-defined data structures
- Define more operations on data structures
- Add the capability to manage the memory allocated to a process
- Add an interface to the operating system
- Enhance the readability of Fortran programs

The definition of arrays:

An array could be defined as it is now in a dimension statement with two more extensions:

(i) Allow dynamic storage allocation by allowing constants, integer variables or expressions to be used in the main segment of a Fortran program.

Examples:

```
DIMENSION X(10,20),Y(1:15,-3:4)
DIMENSION R(N,M),K(1:N+1,-M:N-1)
```

where N and M will be supplied at run-time. This will minimize the wasted memory when running the program. It is worth mentioning that some compilers allow a similar facility through a system call to free unused memory.

(ii) Allow the declaration of a triangular matrix, e.g.

```
DIMENSION X(1:N,1:*)
DIMENSION Y(1:N,*:N)
```

The X array is defined as a lower triangular array while array Y is an

upper triangular one. In the same sense a diagonal matrix can be defined by:

```
DIMENSION Z(1:N,*:*)
```

(iii) Allow the declaration of virtual arrays. These arrays reside in virtual memory. This point will be discussed in greater detail in Chapter 5 of this thesis. Adding this facility is important since paging, done by operating systems, has a basic problem in that it does not reflect the programmer's knowledge of the program structure but rather leaves the whole task to the operating system.

Operations on arrays:

It is interesting to notice that most of the existing BASIC interpreters and compilers allow expressions like:

```
MAT X = A + B
```

where A,B and X are arrays of the same dimensionality. It seems that matrix operations must be allowed in Fortran. The following list of matrix operations is proposed:

(i) Allow the use of arrays in expressions as scalers provided that the correct dimensionality is met, e.g.

```
A = B + C
R = 0
K = IDENT
M = 1
```

where A,B,C,R and K are arrays.

In the first statement array A is set to the sum of arrays B and C. In the second statement array R is zeroed and in the third statement array K is set to the unit matrix. Note that IDENT is a reserved token. In the last statement all the elements of array M are set to 1.

(ii) Allow the use of sub-arrays in a manner analogous to substring manipulation, e.g.

```
A = B(1:N, 1:M)
```

This will copy the submatrix of B defined by rows 1:N and columns 1:M into A,

```
A = B(1:N,1:*)
```

This will copy the lower triangular part of B into A.

```
B(1:N, 1:M) = A(1:N, 1:M)
```

This statement will copy in the subarray B the same N×M part from A.

This facility will be of great help when mapping local stiffness matrices to the global one in F.E. programming. The use of subarrays in the lefthand side is also useful in matrix partitioning.

(iii) Allow the use of arrays as arguments in some of the intrinsic functions like the ABS, SQRT,... functions, e.g.

```
A = ABS(A)

R = AMAX(A)
```

In the first statement all the elements of A will be set to their absolute values while in the second statement the maximum element of array A is stored in the variable R.

(iv) Add more intrinsic functions for matrices like:

```
TRANSPOSE, INVERSE.
```

(v) Allow the use of integer vectors as subscripts for matrices. This is helpful in programming the frontal solution algorithm, e.g.

```
REAL K(1:N, 1:N)
INTEGER ACTIVE (3)
```

If ACTIVE has the value:

```
ACTIVE={1,4,9}
```

then K(N,ACTIVE) will give the elements: K(N,1), K(N,4) and K(N,9).

Interfacing with the operating system:

(i) Allow the user control of real and virtual memory allocated to his program data segment through two new statements:

```
KEEP REAL     list
RELEASE REAL  list
```

These statements are explained in greater detail in Chapter 5 of this thesis.

Enhancement of Fortran readability:

(i) Allow longer names for variables. The current Fortran specifications allow variable names of up to 7 characters. This seems to be very restrictive. Some of the available compilers allow more than 7 characters. It is proposed to increase the variable names up to 30 characters.

(ii) Allow the use of character labels in addition to numerical labels.

The above-mentioned proposal could be used as a basis for a more powerful Fortran which is more adequate to FE programming in particular, and to scientific computations in general.

4.5 COMPUTER SOLUTION OF FINITE ELEMENT EQUATIONS

As explained in Chapter 2, direct methods of solution for the resulting linear equations are usually used in F.E. programs. These methods, although all are variants of Gauss elimination, they are implemented on computers following the three main approaches, viz:

(i) The banded algorithms;

(ii) The general sparse matrix algorithms; and

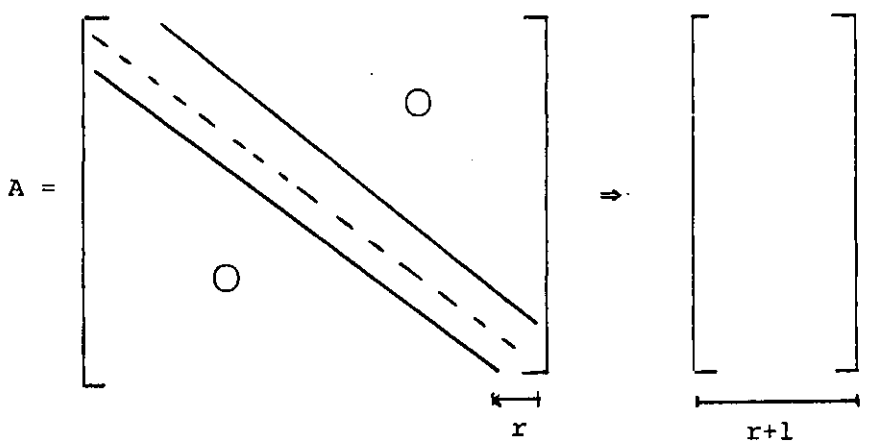
(iii) The frontal algorithm.

4.5.1 Banded Algorithms

In the FEM the stiffness matrix is sparse in general. In many cases it is banded and symmetrical too. Band matrices are stored in a compact form. Assuming that the F.E. system of linear equations to be solved is:

$$Ax = b \ , \qquad (4.1)$$

where A is $n^{\times}n$ banded matrix of semi-bandwidth r. A is stored as a rectangular array of $n^{\times}r$ elements instead of n^2 as in the case of full storage mode. Figure (4.1) shows the storage of a banded symmetrical matrix,



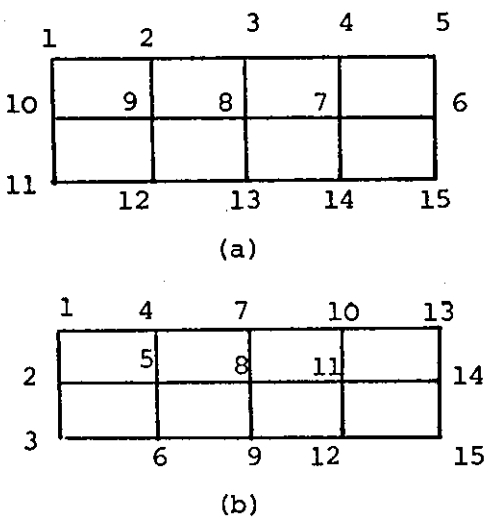
i.e.,

$$A = \begin{bmatrix} a_{11} & & & & \\ a_{21} & a_{22} & & & \\ a_{31} & a_{32} & a_{33} & & \\ 0 & a_{42} & a_{43} & a_{44} & \\ 0 & 0 & a_{53} & a_{54} & a_{55} \end{bmatrix} \rightarrow \begin{bmatrix} - & - & a_{11} \\ - & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} \\ a_{42} & a_{43} & a_{44} \\ a_{53} & a_{54} & a_{55} \end{bmatrix}$$

(3 columns)

FIGURE 4.1: Storage of a banded symmetrical matrix

This will save not only the computer storage but also the execution time required to solve the set of equations. The semi-bandwidth r is a function of the node numbering scheme. As a simple rule, the half-bandwidth is the maximum difference between any two node numbers in the same element as illustrated in Figure 4.2 [Sharaf Eldin, 1983b]. Many techniques are, therefore, used to minimize the bandwidth by node renumbering. These methods will be explained in a later section of this chapter as one of the F.E. preprocessors' functions.



(a) Bad node numbering, semibandwidth = 9

(b) Good node numbering, semibandwidth = 4

FIGURE 4.2: Effect of node numbering on bandwidth

It should be noted that the zeros laying outside the band region remain zeros at all times in the solution process and for that reason they are ignored. Also, we notice that during the Gauss elimination process of the ith row, a limited number of elements of A are affected. Recall equations (2.22) and (2.23) and since A is banded we notice that a_{ik} is zero outside the active triangle shown in Figure (4.3).

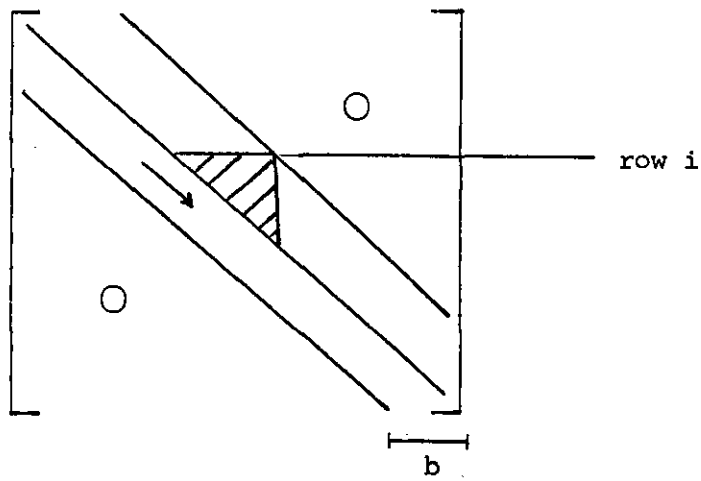


FIGURE 4.3: Active triangle in eliminating the ith row of a banded matrix.

This suggests to keep b rows only of the matrix in core and swapping other rows to backing storage media such as DASD. Examining the active triangle shown in Figure (4.3) we notice that the active triangle is moving downwards. When eliminating the $(i+1)^{th}$ row we thus need to retrieve other elements from the backing store. This will involve heavy I/O operations. In order to decrease the number of these I/O it is preferable to have larger portions of A in-core. A proposed scheme could be to have multiple active triangles in-core. The number of which is dependent on the available core and the size of each active triangle. The storage of a banded matrix as a rectangular array is very simple and quite efficient provided that the number of zeros

inside the band is small. However, this is not always the case. Some problems will give a stiffness matrix which has large variations in the bandwidth from row to row. In this case a more efficient method to store the matrix A is the envelope or profile method. Before discussing this method let us give a definition of the envelope of a matrix following George and Liu [1981] as follows:

For each row i (i=1,2,...,n) in the symmetrical matrix A we define:

$$f_i(A) = \min\{j \mid a_{ij} \neq 0\} \tag{4.2}$$

and

$$\beta_i(A) = i - f_i(A) \tag{4.3}$$

Equation (4.2) gives the column subscript of the first non-zero element in the ith row of A. While equation (4.3) gives the semi-bandwidth of A at the ith row. The semi-bandwidth of A is thus:

$$\beta(A) = \max \beta_i(A) \ , \ i=1,2,\ldots,n \tag{4.4}$$

Considering the variations in $\beta_i(A)$ with i, the envelope (or profile) of A is denoted by Env(A) and defined by,

$$\text{Env}(A) = \{(i,j) \mid 0<i-j\leq\beta_i(A)\} \tag{4.5}$$

In terms of $f_i(A)$ this could be rewritten as,

$$\text{Evn}(A) = \{(i,j) \mid f_i(A)\leq j<i\} \tag{4.6}$$

The total size of envelope of A denoted by $|\text{Env}(A)|$ is

$$|\text{Env}(A)| = \sum_{i=1}^{n} \beta_i(A) \ . \tag{4.7}$$

In the envelope method the non-zero elements within Env(A) are stored. Since by definition $\text{Env}(A) \subset \text{Band}(A)$ it implies that there are savings in the storage of an envelope rather than the band. This is particularly clear if the variations in bandwidth are very large. The most commonly used storage scheme for the profile of a matrix is that given by Jennings

[1966]. Here, in each row of the matrix all the entries from the first non-zero to the diagonal are stored in contiguous locations in a vector ENV. An auxiliary index vector DIAG is used to point to the location of diagonals in the vector ENV. This is best illustrated by the example shown in Figure (4.4).

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & & & \\ & a_{22} & & & & \\ & & a_{33} & a_{34} & & \\ & & & a_{44} & a_{45} & a_{46} \\ & \text{Symmetrical} & & & a_{55} & a_{56} \\ & & & & & a_{66} \end{bmatrix}$$

A

Index:		
	1	a_{11}
	2	a_{12}
	3	a_{13}
	4	a_{22}
	5	a_{33}
	6	a_{34}
	7	a_{44}
	8	a_{45}
	9	a_{46}
	10	a_{55}
	11	a_{56}
	12	a_{66}

Storage of elements of A in vector ENV

Index:		
	1	1
	2	4
	3	5
	4	7
	5	10
	6	12

Pointer vector DIAG giving location of diagonals within ENV

FIGURE 4.4: The storage scheme for matrix envelope

Note that the vector DIAG is of integer type and of length n. The size of the real vector ENV is $|ENV(A)|$. The map which defines the location of an element a_{ij} within the envelope region of A into the

vector ENV is defined by,

$$\{I,J\} \rightarrow DIAG(I)+J-I \ . \qquad (4.8)$$

Note that since A is symmetrical we consider the upper part only, i.e. $J \geq I$. To illustrate the use of (4.8) consider the element a_{45} in Figure (4.4). This element will be stored in ENV in the position:

DIAG(4) + 5 - 4 i.e. 8

If A was not symmetrical, a variant of the above method proposed by George and Liu [1981] is to store the diagonal elements themselves in a separate vector DIAG and store other elements of A within the profile region into another vector ENV. A pointer vector ROW of integer type is used to keep track of the start of each row portion within ENV. Note that the vector DIAG in this case is not of type integer as before. The length of DIAG is n, while the length of ENV is $|ENV(A)|$ and that of ROW is n. However, to have easier indexing when referencing the elements of ENV; the vector ROW is increased by one element to be n+1 and the last element in it i.e. ROW(n+1) is set to $|ENV(A)|+1$. This storage scheme is illustrated in Figure (4.5).

$$\begin{bmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & & & \\ 0 & a_{32} & a_{33} & & \\ & a_{42} & a_{43} & a_{44} & \\ & & & a_{54} & a_{55} \end{bmatrix}$$

A

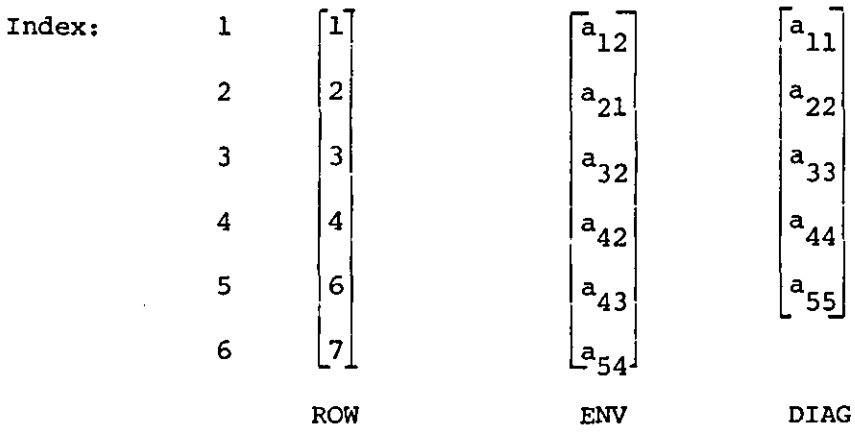


FIGURE 4.5: Another storage scheme for envelopes

Note that ROW(I) is the element number in ENV where the 1st element of the Ith row elements is stored. The mapping of any a_{ij} element within the profile of A onto the vector ENV is defined by:

$$\{I,J\} \rightarrow ROW(I+1)-|I-J| \quad I \neq J \tag{4.9}$$

For example, element a_{54} will be stored in

$$ROW(6) - (5-4) = 6$$

i.e. in ENV(6).

4.5.2 The General Sparse Matrix Algorithms

In the general sparse matrix algorithms use is made of the sparsity of the matrix of coefficients to store nonzero elements only. Here, a well-known problem arises; the fillin's i.e. the zero elements of A may become nonzeros during the solution process. The problem of fill-in's does not exist in banded algorithms since zeros within the band are stored. However, in the general sparse algorithms, the nonzeros of A are stored but due to the fill-in's some additional storage must be added. To illustrate the problem of fill-in's consider the following example [George and Liu, 1981]:

Consider the system of equations Ax=b where A is the symmetric sparse matrix,

$$\begin{bmatrix} 4 & 1 & 2 & .5 & 2 \\ 1 & .5 & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 \\ .5 & 0 & 0 & .625 & 0 \\ 2 & 0 & 0 & 0 & 16 \end{bmatrix}$$

and x is the vector of unknowns and b is the r.h.s. vector of constants,

$$\begin{Bmatrix} 7 \\ 3 \\ 7 \\ -4 \\ -4 \end{Bmatrix}$$

A could be factored to LL^T where,

$$L = \begin{bmatrix} 2 & & & & \\ .5 & .5 & & & \\ 1 & -1 & 1 & & \\ .25 & -.25 & -.5 & .5 & \\ 1 & -1 & -2 & -3 & 1 \end{bmatrix}$$

and the solution is $x = \begin{bmatrix} 2 \\ 2 \\ 1 \\ -8 \\ -.5 \end{bmatrix}$

We notice that the zeros structure of A is not retained in L. Many fill-in's occur. Also note that using banded algorithms for such a problem with this ordering is not useful. This is, in fact, the motivation of using general sparse algorithms.

To minimize the number of fill-in's, re-ordering of the equations is necessary. This is done by multiplying the matrix A by a permutation matrix P. Pre-multiplication PA means re-arranging the rows of A while post-multiplication means permutation of the columns of A. Thus the sparse Gaussian elimination for the solution of sparse symmetric

positive definite equations is usually done in three different steps:

(i) Permutation of A to reduce the fill-in's

(ii) Finding the non-zero structure of the factors of A.
This could be done by symbolic factorization [Schreiber,1982]. It is worth mentioning that the prediction of fill-in's is not possible for a general sparse matrix.

(iii) Using the data structure obtained in (ii), the actual numerical computation is done.

The separation of these three steps is useful since it allows the solution of different problems which possess the same structure but with different values by doing the symbolic factorization only once.

Recall the given example again and use the permutation matrix:

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} = P^T$$

$$PAP^T = \begin{bmatrix} 16 & 0 & 0 & 0 & 2 \\ 0 & .625 & 0 & 0 & .5 \\ 0 & 0 & 3 & 0 & 2 \\ 0 & 0 & 0 & .5 & 1 \\ 2 & .5 & 2 & 1 & 4 \end{bmatrix}$$

The system of equations Ax=b will be:

$$(PAP^T)(Px) = Pb, \text{ i.e. } Cz=d.$$

The factorization of PAP^T to LL^T will give:

$$L = \begin{bmatrix} 4 & 0 & & & \\ 0 & .791 & & & \\ 0 & 0 & 1.73 & & \\ 0 & 0 & 0 & .707 & \\ .5 & .632 & 1.15 & 1.41 & 1.129 \end{bmatrix}$$

which has the same sparsity structure as (PAP^T) itself. We proceed as before to solve the system of equations: $Ly=d$, $L^Tz=y$ and finally $x=P^Tz$.

It should be noticed, however, that:

(i) Finding the best permutation matrix P is not always possible. Some heuristics are used to find a good permutation matrix P.

(ii) The storage requirements for a sparse matrix consists of two main parts: primary and overhead. The primary storage is used to store the non-zero elements of A while the overhead is used to keep the necessary data to access elements of the primary storage like pointers, subscripts, etc. The total storage required is, therefore, the sum of these two portions.

(iii) It may happen that during the reduction of A that some zeros arise which are not due to the sparsity structure of A but due to numerical calculations. These zeros are usually not exploited in sparse matrix methods.

There are many methods to store a sparse matrix which differ in their complexity and execution time. However, an inadequate sparse scheme can lead to very inefficient programs due to the large amounts of data handling involved [Evans, 1973]. One of the well-known data structures used is to store the elements of the lower triangle of A, including the zeros that later will be filled in, in a vector v. A separate vector r is used to record the row number of the corresponding element of array v. Thus if a_{ij} is stored in v(k) then r(k)=i. A is stored columnwise i.e. elements of the same column are stored in contiguous elements in v. Thus we need to keep in another vector c, the diagonal elements of A as stored in v. e.g. if v(6) holds a_{55} then c(5) will be 6.

To illustrate, consider the following example:

$$A = \begin{bmatrix} a_{11} & & & \\ 0 & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & 0 & a_{43} & a_{44} \end{bmatrix}$$

then the v, r and c vectors will be

$$v = \begin{matrix} a_{11} & a_{31} & a_{41} & a_{22} & a_{32} & a_{33} & a_{43} & a_{44} \end{matrix}$$

$$r = \begin{matrix} 1 & 3 & 4 & 2 & 3 & 3 & 4 & 4 \end{matrix}$$

$$c = \begin{matrix} 1 & 4 & 6 & 8 \end{matrix}$$

A survey of 36 of the most well known FE systems in the marketplace [Noor, 1981] indicates that none of them use general sparse methods to solve FE equations. This may be due to: (i) The sparsity structure of the resulting FE equations from general purpose FE software in practical use differs greatly from one problem to another and it is very difficult, if not practically impossible, to tell in advance which solution strategy should be followed: banded, frontal or general sparse. (ii) unless the matrix of coefficients is greatly sparsed, banded algorithms are normally faster and require less computer memory. (iii) If A, the matrix of coefficients is not symmetric positive definite, there is no algorithm known to find a good permutation matrix. (iv) General software for sparse matrices are relatively new compared to banded algorithms.

However, high quality software packages for sparse matrices exist since the mid-seventies. Among these are; the Yale Sparse Matrix Package: YSMP [Eisenstat, et al., 1976] and the SPARSPAK [George and Liu, 1981]. These two packages have similar general structure and solution philosophy. However, they have different data structures.

Consider the system of equations to be solved is Ax=b where A is a sparse symmetric positive definite matrix. In these packages the basic solution steps are:

(i) Input structure of A.

(ii) Permuting A according to one of the available methods in the system library to reduce the fill-in's. At the end of this step, the data structure needed for L is allocated.

(iii) Input of the numerical values of non-zero elements of A.

(iv) Factorise A into LL^T.

(v) Input of righthand side.

(vi) Final solution for x and output of results.

The factorization is done in two steps in the YSMP: symbolic factorization by the subroutine SYMFAC and numerical factorization by the subroutine NUMFAC. The SYMFAC routine determines the fill-in's in the factor L of A while the routine NUMFAC uses this structural information to do the actual numerical computations. In the SPARSPAK package the user allocates a vector of the total memory allocated to the problem and interface subroutines use this vector to allocate memory for different modules during the computation.

4.5.3 The Frontal Algorithm

The other widely used method for the solution of linear equations resulting in FE analysis is the frontal technique. It is a fact that the majority of FE systems available use either banded algorithms (simple or profile) or the frontal technique. The frontal solution technique was originated independently from Irons [1970], Melosh and Bamford [1969] and Hellen [1969]. In this method, the assembly

procedure for the stiffness matrix and nodal forces and the solution of the unknown displacement by means of the Gaussian elimination method are all done in one application. The main and important idea is to assemble the equations and eliminate the variables at the same time. As soon as the coefficients of an equation are completely assembled from the contributions of all relevant elements, the corresponding variable can be eliminated. Therefore, the complete master matrix is never formed as such, since after elimination the reduced equation is immediately transferred to backing storage. So, the frontal method is basically an out-of-core technique. To explain the method we consider the system of linear equations defined by Ax=b. In the Gauss elimination method the elimination of the unknown x_i (i=1,2,...,n) is done by modifying the elements of the coefficient matrix A and the vector b as follows:

$$a_{jk} = a_{jk} - a_{ji} * a_{ik}/a_{ii} \tag{4.10}$$

$$b_j = b_j - a_{ji} * b_i/a_{ii} \tag{4.11}$$

for j=i+1 to n

and k=i+1 to n.

The last equation will be in the form $x_n=b_n/a_{nn}$ which gives x_n directly, while the remaining unknowns $x_1,x_2,\ldots,x_{n-1}$ are determined by backward substitution using:

$$x_i = (b_i - \sum_{j=i+1}^{n} a_{ij} * x_j)/a_{ii} , \tag{4.12}$$

for i=n-1 to 1.

If the matrix A is symmetric, then equation (4.10) can be written in the form:

$$a_{jk} = a_{jk} - a_{ij} * a_{ik}/a_{ii} \tag{4.13}$$

for j=i+1 to n

and k=i+1 to n.

Now it should be noted that:

(i) At the time of elimination a_{jk} and b_j need not be fully assembled provided all the other terms are.

(ii) If either a_{ij} or a_{ik} is zero, then a_{jk} will not be changed by the elimination process. Similarly, b_j is unchanged if a_{ij} is 0.

The frontal algorithm takes advantage of these points by alternating assembly and elimination. A variable x_i is eliminated as soon as all the elements that contribute to any of the terms a_{ij} or b_i have been assembled. After the elimination of x_i its row equation $\sum_{j=1}^{n} a_{ij}x_j=b_i$ will not be needed in the elimination process and thus this row can be swapped to out-of-core backing storage. Only the terms of A and b that correspond to "active" variables need to be kept in main memory. By active variables we mean those variables x_j for which some a_{jk} or b_j has been affected by assembly but are not yet ready for elimination. In the backward substitution phase, the reduced rows of the matrix with the corresponding terms of b are read from the backing storage in the reverse order. This could be done by backspacing through the backing storage device. It is worthwhile to mention, however, that usually direct access storage devices (DASD) like disc devices are used as working files during the frontal operation. For these devices, backspacing is not done physically by repositioning the file one record in the reverse direction as the case in magnetic tapes, but the hardware record address is decremented by one. This makes the execution time for a frontal solver competent and superior, in many cases, to other comparable techniques.

In implementing a frontal solver the symmetric coefficient matrix

A of dimensions (n×n) is considered as a vector v of length n(n+1)/2. Thus, any term a_{ij} is to be mapped to location in v defined by the function:

$f(i,j) = i(i-1)/2 + j$ for $i \geq j$ to store lower triangle

and $f(i,j) = j(j-1)/2 + i$ for $i \leq j$ to store upper triangle (4.14)

In the frontal technique, however, we do not store the whole matrix A in core, rather only the active nodes. So the size of the vector v can be decreased to w(w+1)/2 only, where w is the maximum frontwidth. For each variable x_i in the set of equations to be solved, which corresponds to a degree of freedom, we should calculate a destination d_i which determines where the terms associated with the variable will be placed in the vector v. Thus, the terms can be accessed while they are active by applying the function f to their destinations. In other words, if both x_i and x_j are active, the current value corresponding to a_{ij} will be held in $v(f(d_i,d_j))$. The destinations are calculated by pre-processing the elements' nodal connection data i.e. the connectivity matrices. At the end of the assembly/reduction process, a work file has a size of n records and equal to the number of degrees of freedom in the structure is created. One record corresponding to an eliminated variable equation. The back substitution phase is considered as a frontal process in reverse. The details of the frontal algorithm with the necessary housekeeping and Fortran coding can be found in Hinton and Owen [1979], Cheung and Yeo [1979] and Irons and Ahmad [1980]. However, we give here a brief description of the implementation details on a computer as follows.

The first step in the frontal routine is to determine the last appearance of each node during the assembly/elimination process. This

is known as the pre-front process. The reason why we need to determine the last appearance of each node is that during the "life" of each node in the front procedure it moves through three stages: (1) the inactive status where it is not yet summed; (2) the active status where it is within the front; and (3) the deactivated status where it has been eliminated and removed from the front. The main idea is that during the assembly/elimination process, when an element is to be considered its matrix is formulated and mapped to the appropriate places in the existing equations if all the corresponding nodes are active. However, if some of these nodes were not active then a new equation is formed to cater for them. On the other hand, if any node will appear for the last time, their corresponding equations can be eliminated and moved to a backing storage thus freeing space in the front for a new equation. It is now clear from this discussion that we must know the last time a node will appear during the assembly/elimination process. To do that a loop is done over all elements in the same order in which they will be assembled and the last appearance of a node in this list is marked. An easy way to mark this last appearance is to put a negative sign in front of it, i.e. the node number is negated. This is best illustrated by the example shown in Figure (4.6) which is the same as the one presented in Chapter 2 of this thesis in Figure (2.5).

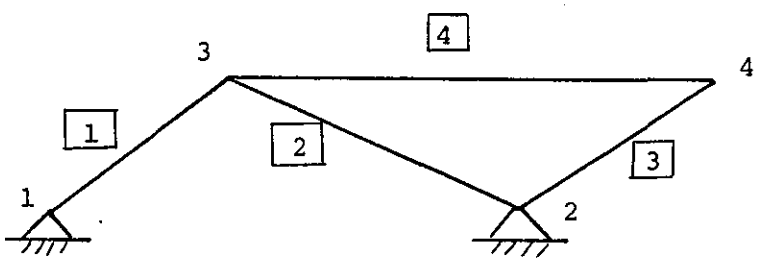


FIGURE 4.6: Sample problem

The connectivity matrix for this problem is:

Element	Nodes	
1	1	3
2	2	3
3	2	4
4	3	4

The last appearance of nodes will be:

-1	3
2	3
-2	4
-3	-4

The second step is to determine the position in the front into which each degree of freedom of a node is to be assembled (mapped) this is known as the destination vector and to determine the list of active variables currently in the front. To determine the destination vector we note that at the beginning, the front is zeros, thus the degrees of freedom corresponding to the nodes of the first element to be assembled are given the first locations in the front. In the considered example, nodal variables 1,2 corresponding to node 1 will be allocated to the first two positions in the front while the nodal variables 5,6 corresponding to node 3 will be allocated to the third and fourth elements in the front. Thus the element destination vector of element 1 is [1,2,3,4]. Elimination is done in conjunction to assembly. If any node of the considered element has a negative sign it means that this is the last appearance of this node and so it is ready for elimination. Elimination is virtually the same as explained earlier except that the equation used for reduction of other equations within the front area is

not on the top but may be anywhere in the front. In our example, we can eliminate equations 1 and 2 which are of node 1. The free space in the front can be utilised when assembling the second element where node 2 appears for the first time. Thus, the second element destination vector will be [1,2,3,4]. After elimination of an equation, its coefficients are stored out-of-core and its position is zeroed to be ready to be occupied by a new equation.

Although the frontal algorithm was devised for FE analysis, the approach is extended to solve sparse symmetric linear equations in general [Duff and Reid, 1983]. The frontal solution requires elaborate housekeeping procedures and higher programming skills as compared to a simple band solver. However, it has many advantages over the band solvers. The solution is not affected by node numbering as in band solvers but rather is dependent on the element ordering and their connectivities. This is very useful for elements with mid-side and internal nodes that will result in an increase in the bandwidth, while in a frontal solution such nodes will be active for a very short time only. A reconstruction of the mesh is not easily implemented for band solvers as the frontal algorithm where the node numbers are unimportant. The frontwidth is normally smaller than the bandwidth but it is very difficult to predict its width. In practice the available memory (core) is utilised to assemble as much elements as possible to reduce the I/O operations which can be done in blocks of records rather than one by one.

4.5.4 Software for the Solution of Equations

Many well known software systems have been developed for the solution of systems of algebraic equations. They differ in their

complexity, capabilities, cost and availability. Two major types of such systems exist: (1) packages which comprise of stand-alone programs that can be used directly by the user. All what he needs to do is to supply the input data. (2) Libraries which contain subroutines that can be called from a user program. The main differences between these two types can be summarized in the following [Sharaf Eldin, 1984]:

(i) Libraries require programming knowledge (Fortran, say), while a package user need not do any programming.

(ii) To use a library a host program must be supplied. However, for a package no host program is required. This host program is normally the main segment of the whole program.

(iii) Libraries are more flexible, they can be modified by the users or used to build more sophisticated software. In other words they can be customised by the user. This is not the case with packages.

Among the well-known systems of equation solvers one can mention YSMP [Eisenstat et al., 1976], IMSL [IMSL, 1984], LINPACK [Dongarra et al., 1979] and MINPACK-1 [More et al., 1980]. Here the IMSL which is a library of Fortran subroutines for mathematical and statistical analysis is presented. The choice of this library is due to its availability in source code, its wide spectrum of applications and being used by many universities and research centres in the world. The general characteristics of the library are:

(i) Several storage modes of matrices are supported. These are band, symmetric, band symmetric and Hermitian. These storage modes will be discussed later.

(ii) All the routines conform to established conventions in coding and documentation.

(iii) Each routine outputs a return code which can be tested to monitor the results given by this routine. This is a good means for error detecting.

(iv) Computer readable documentation is available which permits on-line access to the basic documentation.

(v) Several routines are supplied based on different algorithms to allow a wider choice for the user.

(vi) All routines are available in single and double precision.

(vii) The order of parameters in the argument list of the routines is:

Input parameters,
Input and output parameters,
Output parameters,
Work areas, and
Error parameters (return codes).

(viii) Many routines have more than one version, e.g. in-core and out-of-core versions.

The error detecting facility supplied by the library is based on the "return code" method. In other words, one of the arguments in the routine call (the last one in order) is an integer variable; namely IER. This variable is set automatically by the routine to a value that indicates the conditions met while executing this routine. After the call of the routine it is the responsibility of the user to test the value of the variable IER to check whether the routine was executed normally or any abnormal conditions have occurred. To exemplify, suppose that a system of linear equations are to be solved using one of the solvers available in this library. After calling the solver, the return variable IER must be tested. The solver routine

will place a value in this variable to indicate whether it succeeded in obtaining results with the requested accuracy or not. If the matrix of coefficients was singular another value will be returned by the routine. Three types of errors can be detected by IMSL routines: warning, warning with fix and fatal errors. The warning errors are those which caution the user that, during the course of computations, certain critical conditions were detected. These are not so severe as to suspend subsequent computations. For example, if one requires the solution of a system of linear equations and specify a number of digits for which the solution is correct but during the computation the routine discovers that this condition cannot be satisfied, a warning error will be returned. The value returned is an integer $32<IER\leq64$. The interpretation of these values can be found in the library manuals. Warning errors with fix are those encountered during computation but some attempts have been done by the routine to correct the situation and the computations are continued. The returned value is $64<IER\leq128$. Fatal errors are those of critical nature. Once an error of this type has been detected by the routine it aborts and no further computations are done. An example of this type is a singular matrix for which an inverse is required. The value of the returned value is >128.

4.5.4.1 Matrix Storage Modes

Five storage modes for matrices are supported by the IMSL library routines as follows:

(i) Full storage mode:

This is the normal storage mode of a general matrix. As known in Fortran a matrix is stored in memory in contiguous area column by

column. The adjustable dimensioning feature in Fortran is utilized to pass a submatrix to an IMSL routine. For example, suppose that an array is dimensioned in the main (master segment) program as A(100,100) and a call to a subroutine is done using the submatrix of A with dimensions 20×20 only. In order to have the correct elements to be processed in the subroutine, the row dimension of A must be passed to the routine and used in the adjustable dimensioning in it. Failing to do that will result in processing other submatrices of A in the subroutine as follows:

```
      DIMENSION A(100,100)
        N=20
      CALL SUB1(A,20,SUM)

      SUBROUTINE SUB1(X,N,TOTAL)
      DIMENSION X(N,N)
      TOTAL=0
      DO 10 I=1,N
      DO 10 J=1,N
      TOTAL=TOTAL+X(I,J)
10    CONTINUE
      RETURN
      END
```

Bearing in mind that matrices are stored in column-wise sequence, A will be stored in memory as:

$$a_{11}, a_{21}, a_{31}, \ldots, a_{100,1}, a_{12}, a_{22}, a_{32}, \ldots, a_{100,100}$$

The array X is declared N×N elements and the value of N passed to this routine is 20 i.e. 400 elements. Thus the elements of A that will be used in subroutine SUB1 will be:

$a_{11}, a_{21}, a_{31}, \ldots, a_{100,1}, a_{12}, a_{22}, \ldots, a_{100,4}$ instead of

$a_{11}, a_{21}, \ldots, a_{20,1}, a_{12}, a_{22}, \ldots, a_{20,2}, \ldots, a_{20,20}$.

The solution to this situation is to pass the row dimension of A to the subroutine SUB1 and use it in dimensioning X. Thus the correct code will be:

```
      DIMENSION A(100,100)
      IA=100
C     IA is the row dimension of A
      N=20
      CALL SUB1(A,IA,N,SUM)

      SUBROUTINE SUB1(X,IA,N,TOTAL)
      DIMENSION X(IA,N)
        .
        .
        .
      As before
```

In this case the elements $a_{11}, a_{21}, \ldots, a_{20,1}, a_{12}, a_{22}, \ldots, a_{20,20}$ will be used in SUB1 which are the correct elements.

(ii) Symmetric Storage Mode:

To conserve memory space and reduce the arithmetic operations, a symmetric matrix is stored in vector form. By definition in a symmetric matrix $A_{i,j}=A_{j,i}$, thus all what is stored are the elements on the diagonal and those below it. An N×N symmetric matrix will be stored in symmetric storage mode in only N(N+1)/2 elements as compared to N^2 elements if it were stored in full storage mode. An element i,j in the original matrix A is now the element $\frac{i(i-1)}{2}+j$, for $i \geq j$ in the vector B which is the symmetric storage mode of A. Figure (4.7) represents a symmetric storage mode for a 4×4 symmetric matrix.

$$\begin{bmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad \text{symmetric}$$

A

$$\Rightarrow \quad \begin{bmatrix} a_{11} \\ a_{21} \\ a_{22} \\ a_{31} \\ a_{32} \\ a_{33} \\ a_{41} \\ a_{42} \\ a_{43} \\ a_{44} \end{bmatrix} = B$$

FIGURE 4.7: Symmetric storage mode

It is clear that the saving in memory locations is n(n-1)/2.

(iii) Band Storage Mode:

In this mode an n×n banded matrix with i lower codiagonals and j upper codiagonals is stored in a matrix of dimensions n×(i+j+1). The zero elements outside the band are not stored while the non-zero elements are stored row-wise. Figure (4.8) shows the storage mode of a banded matrix of size 5×5 with 1 upper codiagonal and 2 lower codiagonals. Note that the diagonal elements are stored in the $(i+1)^{th}$ column. The savings in memory locations is n(n-i-j-1).

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} \\ 0 & 0 & a_{53} & a_{54} & a_{55} \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & a_{11} & a_{12} \\ 0 & a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} & a_{45} \\ a_{53} & a_{54} & a_{55} & 0 \end{bmatrix}$$

FIGURE 4.8: Band storage mode

(iv) Band Symmetric Storage Mode:

This mode combines the two modes: symmetric and band. Thus it is suitable for banded symmetric matrices. An n×n symmetric matrix with semi-band width=i is stored in a matrix of size n(i+1). The matrix is stored row-wise so that the diagonal is stored in the last column. Figure 4.9 shows a 5×5 symmetric, banded matrix with semi-bandwidth of 1 and its band symmetric storage mode.

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & a_{11} \\ a_{21} & a_{22} \\ a_{32} & a_{33} \\ a_{43} & a_{44} \\ a_{54} & a_{55} \end{bmatrix}$$

FIGURE 4.9: Band symmetric storage mode

(v) Hermitian Storage Mode:

Hermitian storage mode for complex matrices is analogous to symmetric storage mode for real matrices. Thus an n×n Hermitian matrix is stored in a complex vector of n(n+1)/2 elements.

4.5.4.2 Linear Equation Solvers

There are many subroutines available in the IMSL that can be used to solve systems of linear algebraic equations. However, they can be divided into two main categories: space economizers and high accuracy solutions. The main difference between both versions is that in the space economizer version the solution is obtained without any attempts for iterative improvements. In the high accuracy versions, iterative improvements of the solution can be done if specified by the user. This is costly in terms of computer memory and time. Two subroutines

will be presented here:

(i) LEQT1F which is a space economizer solver and

(ii) LEQT2F which is a high accuracy solution.

These routines are for full storage mode, real matrices of the form,

$$Ax = b \ , \tag{4.15}$$

where A is the (n×n) matrix of coefficients, x is the vector of unknowns (n×1) and b is the righthand side (n×1) vector.

In fact these routines can be used to solve several systems of equations that have the same matrix of coefficients A. In these routines a working area vector is required. Its size is n in the case of the space economizer solvers and n^2+3n in the case of the high accuracy versions. A good feature of these routines is the accuracy test that can be specified. This is achieved by specifying a parameter that determines the number of significant digits to which the elements of A and b are assumed to be correct. This parameter, IDGT, can be set to 0 to bypass the accuracy test.

(i) The subroutine LEQT1F

The purpose of this subroutine is to solve systems of linear equations. It uses a Gaussian elimination (Crout algorithm). Since the routine decomposes the matrix A, several righthand sides can be solved simultaneously. If the IDGT parameter is given a value greater than 0, the elements of A are assumed to be correct to IDGT decimal digits. The solution, x, will be the exact solution without any round-off error to a matrix $\bar{A}$ which agrees with A in the first IDGT decimal digits.

(ii) The subroutine LEQT2F

This is the high accuracy version of the routine LEQT1F. It uses

the Crout algorithm but iterative improvement is performed if the solution obtained is not correct to the IDGT decimal digits. If IDGT is set to 0 the accuracy test is by-passed but iterative improvements are repeated automatically until the answer is correct to the working precision. In most mini-computers this is 7 decimal digits in single precision and 14 in double precision.

4.5.4.3 Test Problems

Some test problems have been designed to test these two subroutines and to determine the CPU time spent in solving systems of linear algebraic equations with different levels of accuracies. The number of equations vary from 10 to 500 in steps of 10. Computations are done specifying 0,1,2,3,4,5,6 and 7 decimal digits accuracy. The results of these runs are plotted in Figures(4.10 to 4.17). It is clear that both routines will take nearly the same time for N (number of equations) $\leq$420. A sharp rise is then noticed in the interval 420$\leq$N$\leq$500.

Note that it is anticipated that this will be true for N>500. A suggested reason for this sharp rise as noticed from the figures is that the required memory size to accommodate the arrays in the program when N is $\geq$420 has exceeded the allocated region whence more page faults occur.

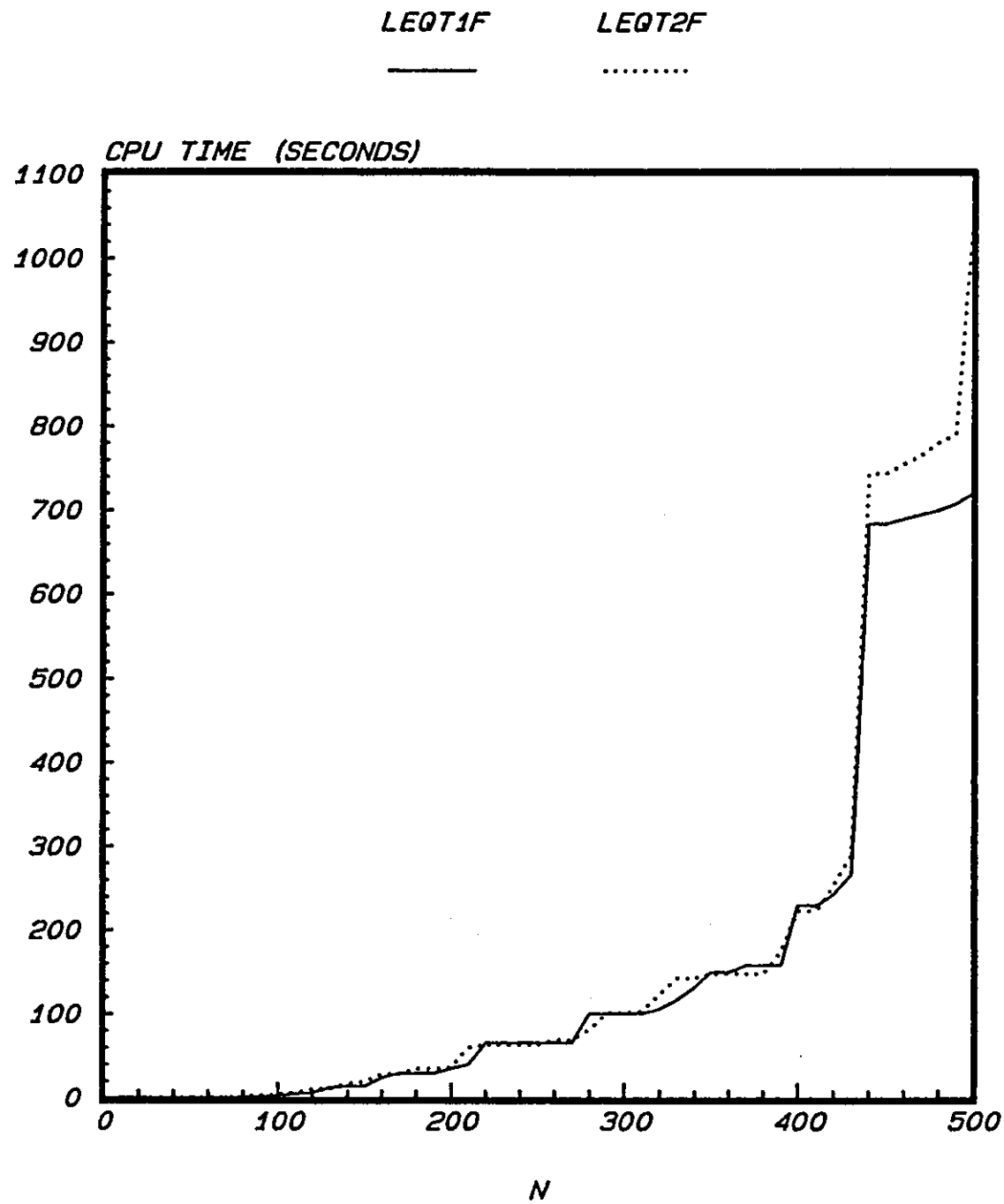


FIG. (4.10) CPU TIME FOR ROUTINES LEQT1F&LEQT2F (0 DP)

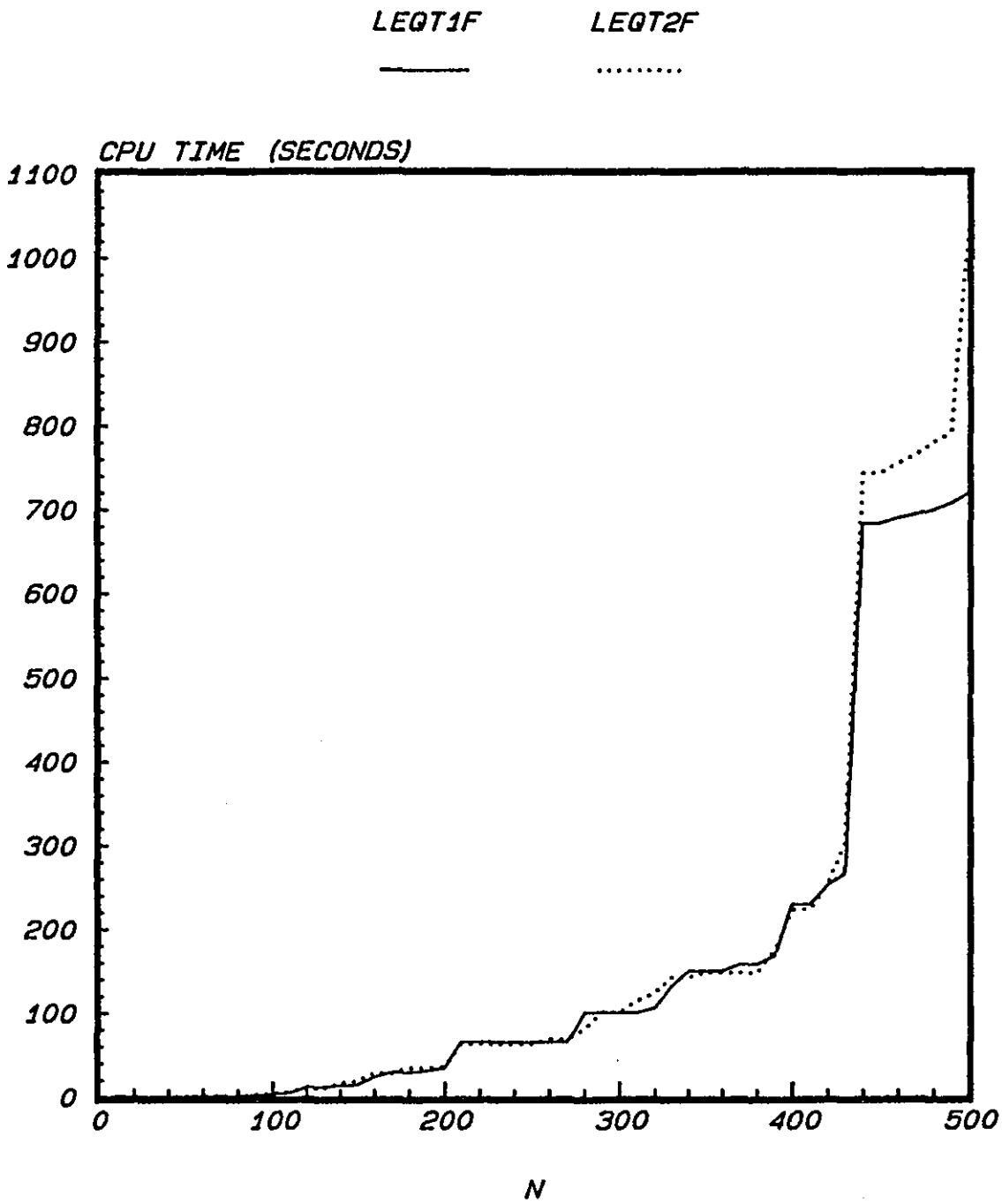


FIG. (4.11) CPU TIME FOR ROUTINES LEQT1F&LEQT2F (1 DP)

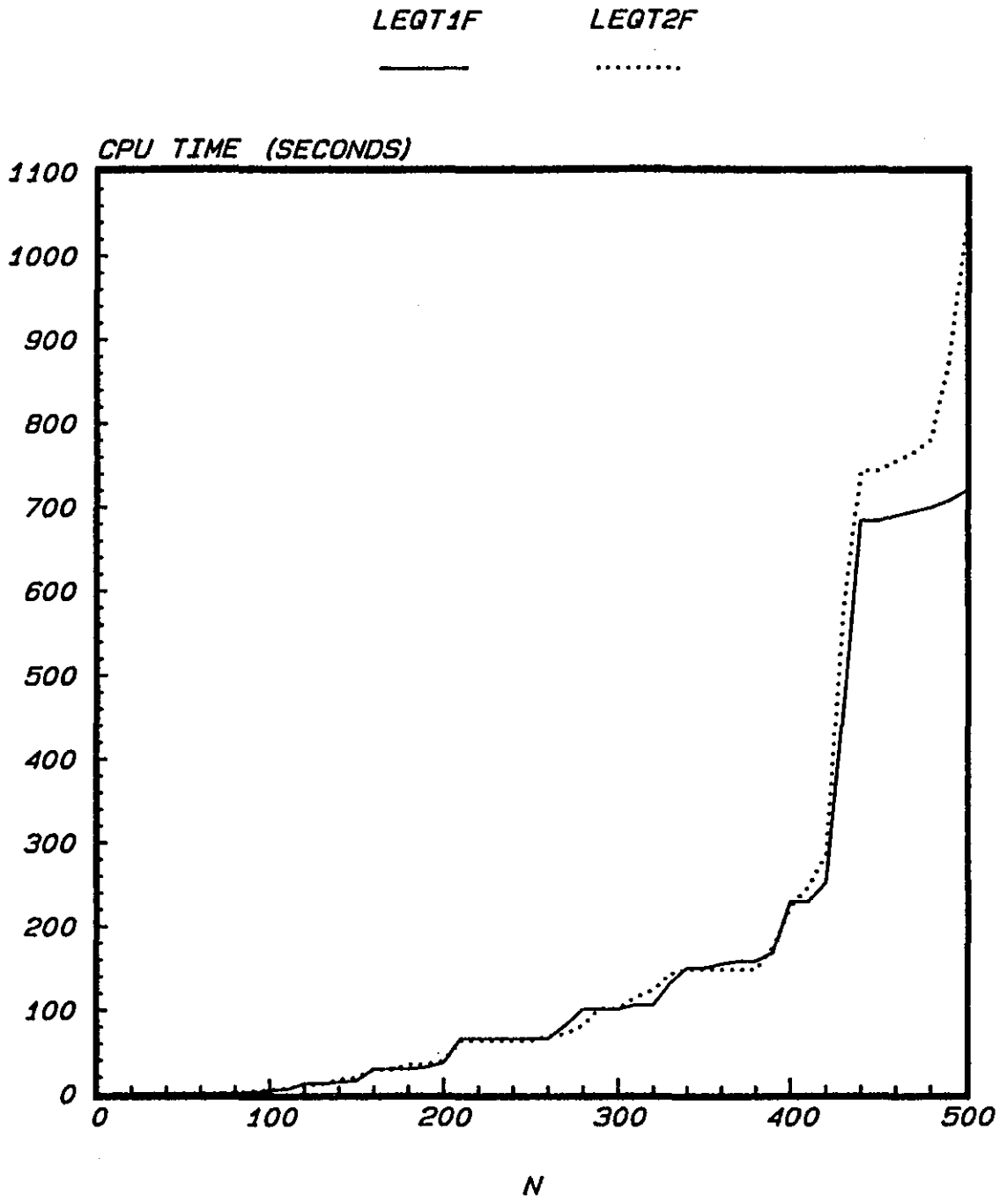


FIG. (4.12) CPU TIME FOR ROUTINES LEQT1F&LEQT2F (2 DP)

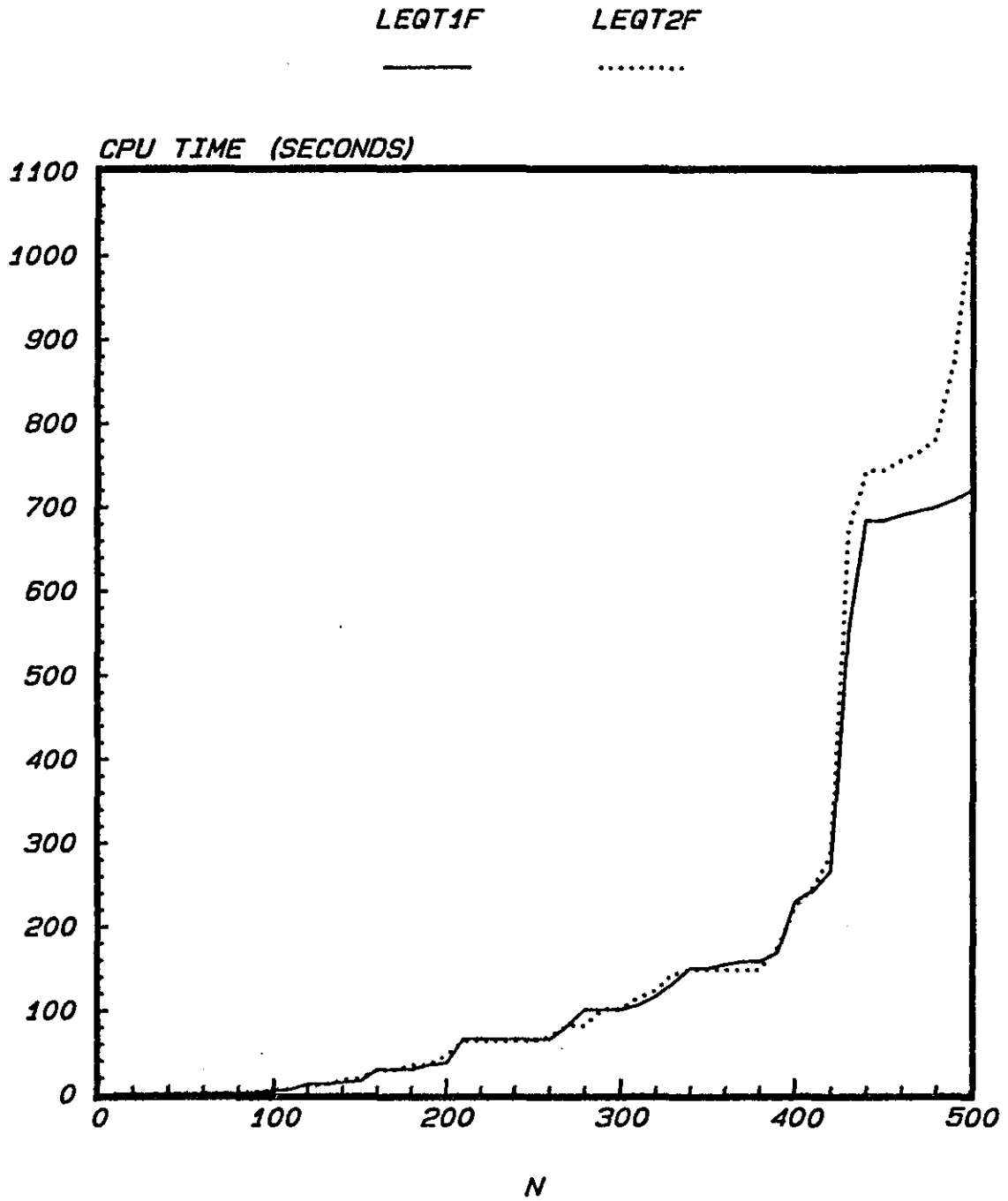


FIG. (4.13) CPU TIME FOR ROUTINES LEQT1F&LEQT2F (3 DP)

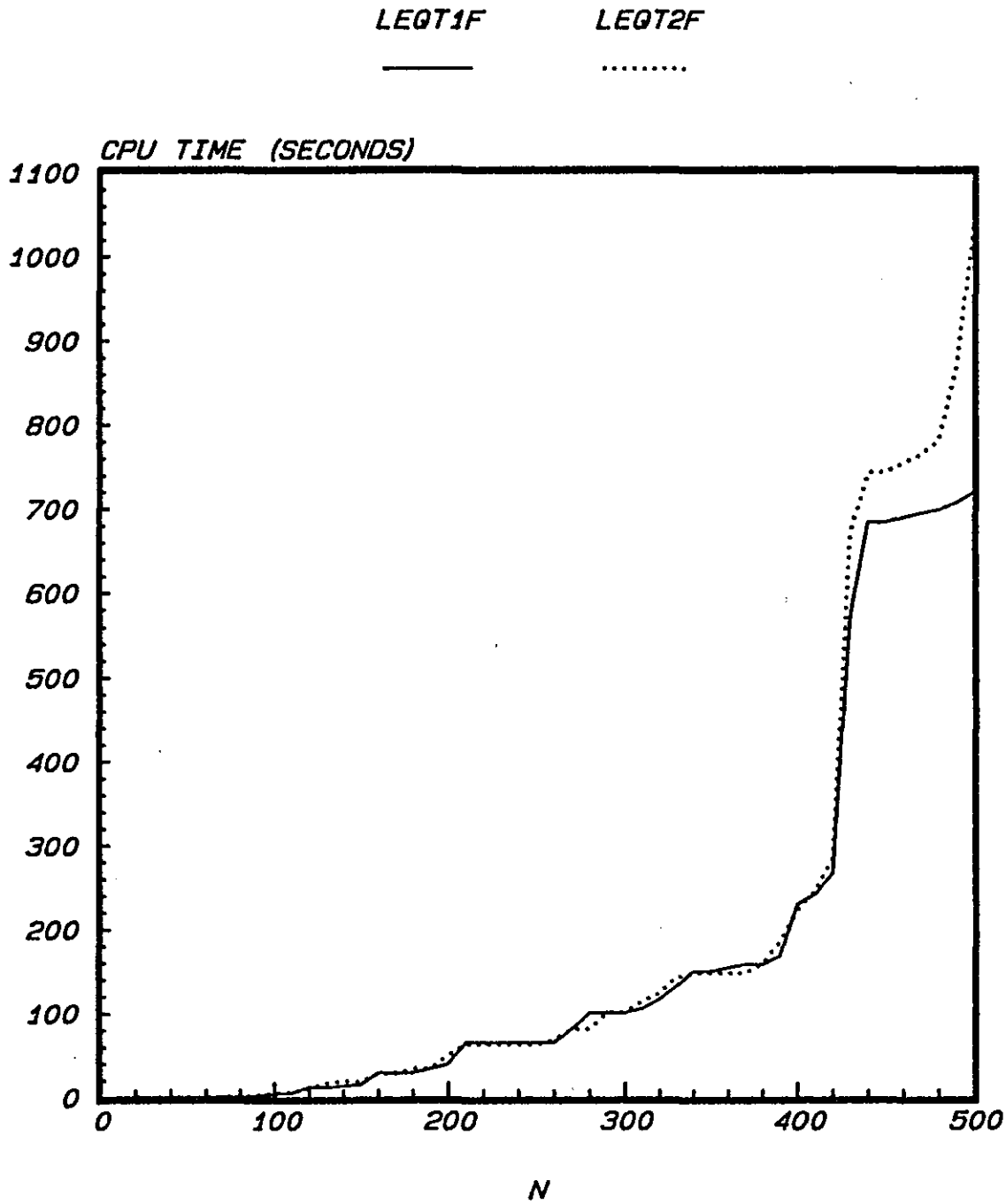


FIG. (4.14) CPU TIME FOR ROUTINES LEQT1F&LEQT2F (4 DP)

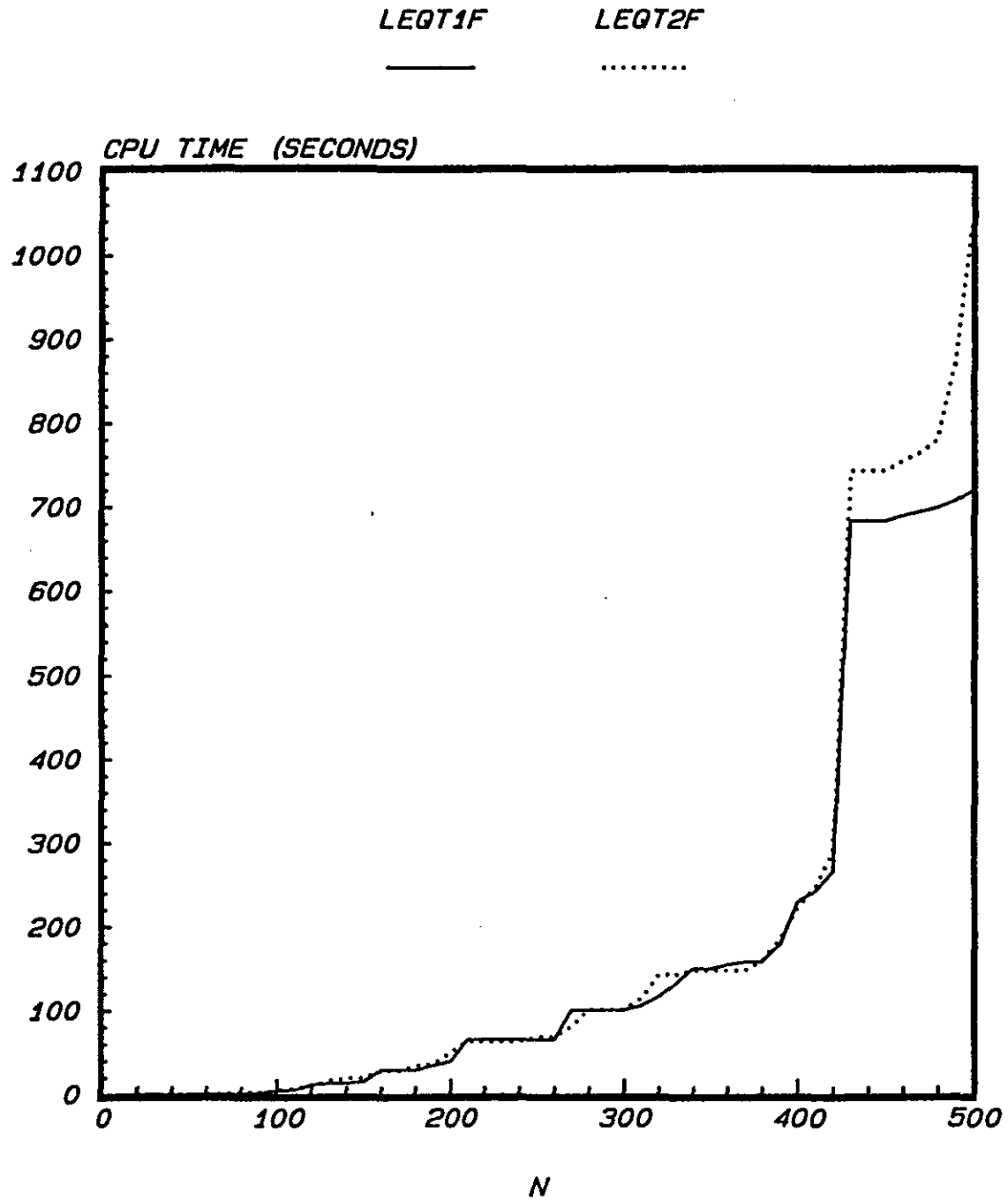


FIG. (4.15) CPU TIME FOR ROUTINES LEQT1F&LEQT2F (5 DP)

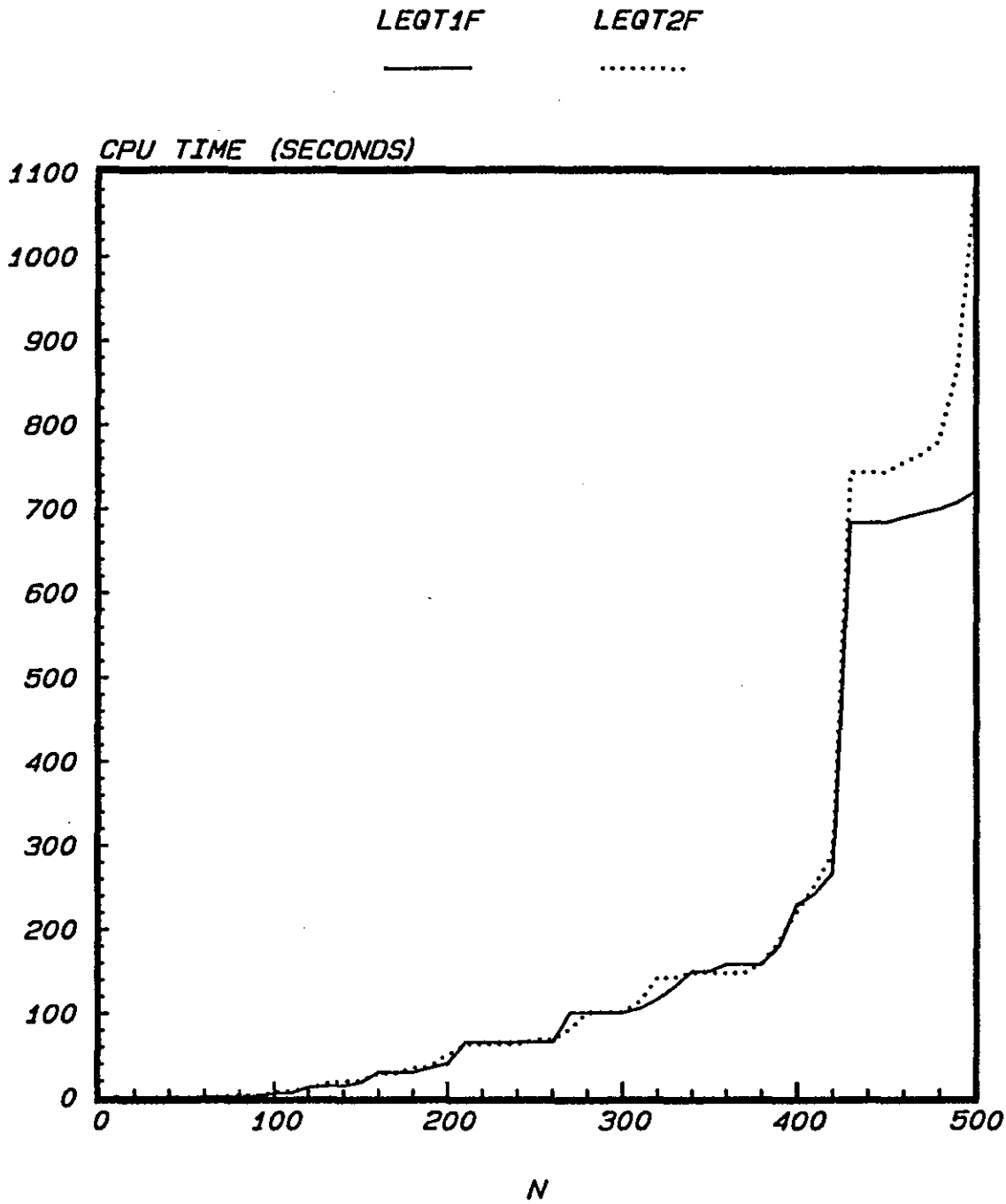


FIG. (4.16) CPU TIME FOR ROUTINES LEQT1F&LEQT2F (6 DP)

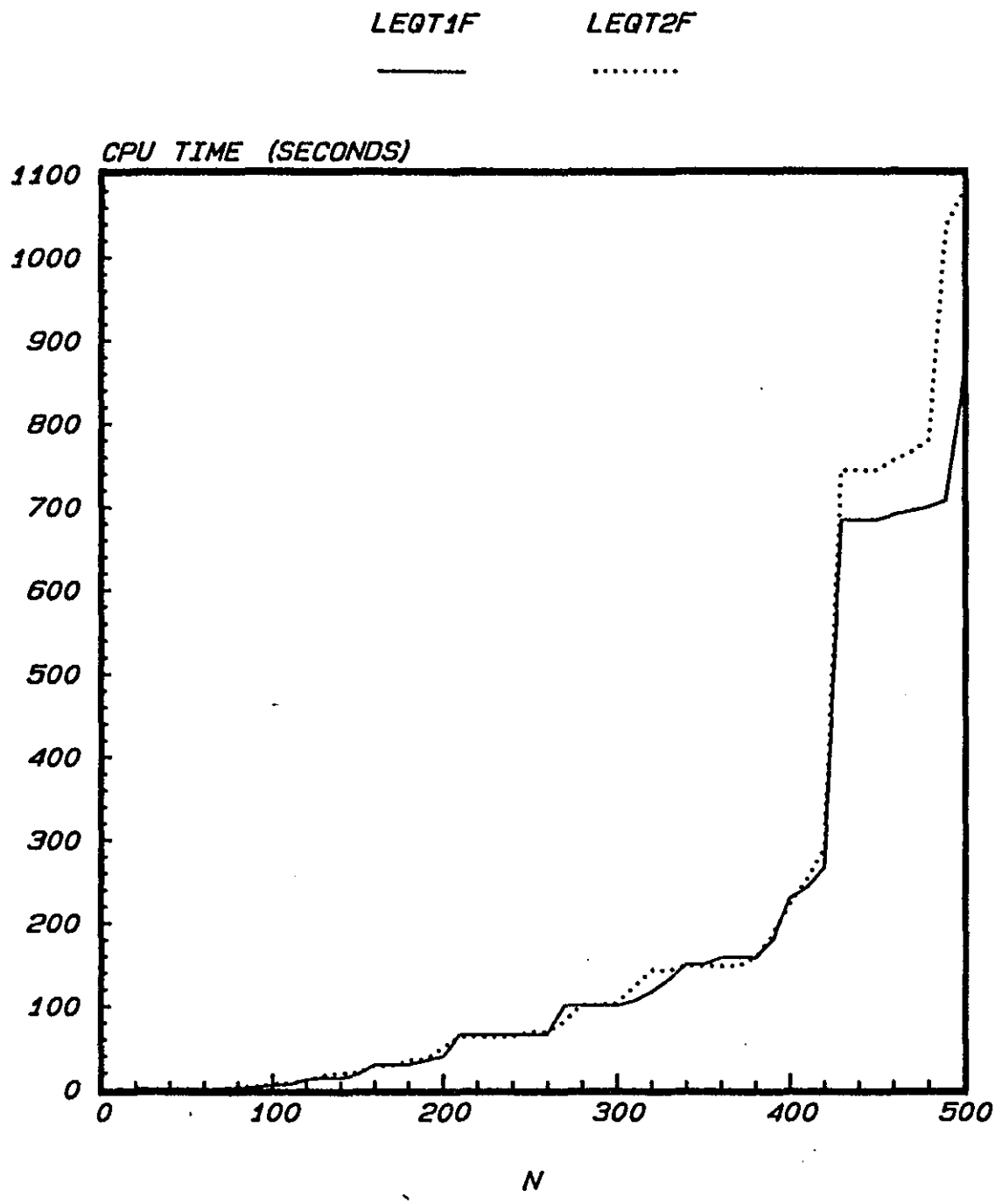


FIG. (4.17) CPU TIME FOR ROUTINES LEQT1F&LEQT2F (7 DP)

4.6 MAIN-FRAME COMPUTER IMPLEMENTATION

4.6.1 Historical Background

The considered model program is the MSAP program. This program is one of the first computer codes for FE analysis. It originated from the SAPIV program developed at Berkley, U.S.A. in 1974 by Bathe et al, which in turn is an advanced version of the SAP program developed by Wilson [1970]. It has high popularity for two main reasons. First it is in the open literature and the source code is available which makes it amenable to modification and adaptation on several different computers. The second is that it is inexpensive. The program costs only about $500. This is due to the fact that it was developed under a grant sponsored by the American Government. The program was developed on the CDC 6400, 6600 and 7600 range of computers. Later on it was modified to run on an IBM 370 computer by Kaldjian, [1982] at Michigan University, Ann Arbor, U.S.A. using the MTS operating system. When Professor Kalidjian visited our university in Riyadh, MSAP was implemented on our local computer system, IBM 3033 running under the MVS operating system where the author participated in that and assisted in courses about Computer Aided Design (CAD) offered by the University of Riyadh. Van Fossen [1978] modified SAP IV also and developed the program FESAP. In this section we present this program as a model for main-frame implementations of the FEM.

The main features of the main-frame computer implementation of the FEM can be summarised in:

(i) Rich element library. Typically 10 or more different elements are expected. Some very big FE systems like MARC [Marcal, 1976] has 50 elements in its element library.

(ii) Normally more than one type of analysis is available, e.g. linear analysis, eigenvalues, non-linear analysis, etc. The procedure library can go up to 15 procedures.

(iii) Sometimes more than one constitutive behaviour is available i.e. different D matrices can be defined. This is called the material library.

(iv) The program size is too big to fit in real storage all the time. Usually automatic overlaying is used and most of the modules are kept in virtual memory instead.

(v) The use of backing storage during the solution process is a must for most practical problems.

(vi) Since most of these packages are developed over a relatively long time span, some of the most recent advances in computer graphics and man-machine interface techniques are not built inside these programs.

(vii) Most of such programs are propriety codes and expensive.

4.6.2 Program Capabilities

MSAP can be used for static and dynamic analysis of linear structural systems. The element library in MSAP contains 10 different elements:

(1) Three-dimensional truss element

(2) Three-dimensional beam element

(3) Plane stress and plane strain elements

(4) Two-dimensional axisymmetric solid

(5) Three-dimensional solid

(6) Variable-number-nodes thick shell and three-dimensional element

(7) Thin plate or thin shell element

(8) Boundary element

(9) Pipe element

(10) Contact element.

Each nodal point in the system can have from zero to six displacement degrees of freedom (DOF). In the static analysis the program solves the equations of equilibrium and computes the element stresses. In the dynamic analysis the frequency calculations are done. It is also possible to obtain the response spectrum analysis. The program itself does not include any pre- or post-processors capabilities except a fairly very simple mesh generation based on equi-dividing element sides. Several types of loads can be handled, i.e. concentrated loads, line, axisymmetric, surface, volume, gravity, thermal and hydrostatic loads. The program has a re-start facility. This enables executing a very long run partially and then resume execution at another time without re-solving the whole problem.

4.6.3 Implementation Details

The general flowchart of the program is shown in Figure 4.18. The first step in the program is the input of the main data. The first input record is for general parameters, i.e. number of nodes, number of elements, etc. The nodal points input data are then read. For each node six boundary conditions codes, 3 coordinates and nodal point temperature are read. The equations associated with each degree of freedom at each node are numbered as they are entered. Nodes which are constrained in any direction are marked and excluded from equation numbering. Knowing the nodal points data, equation numbers for all

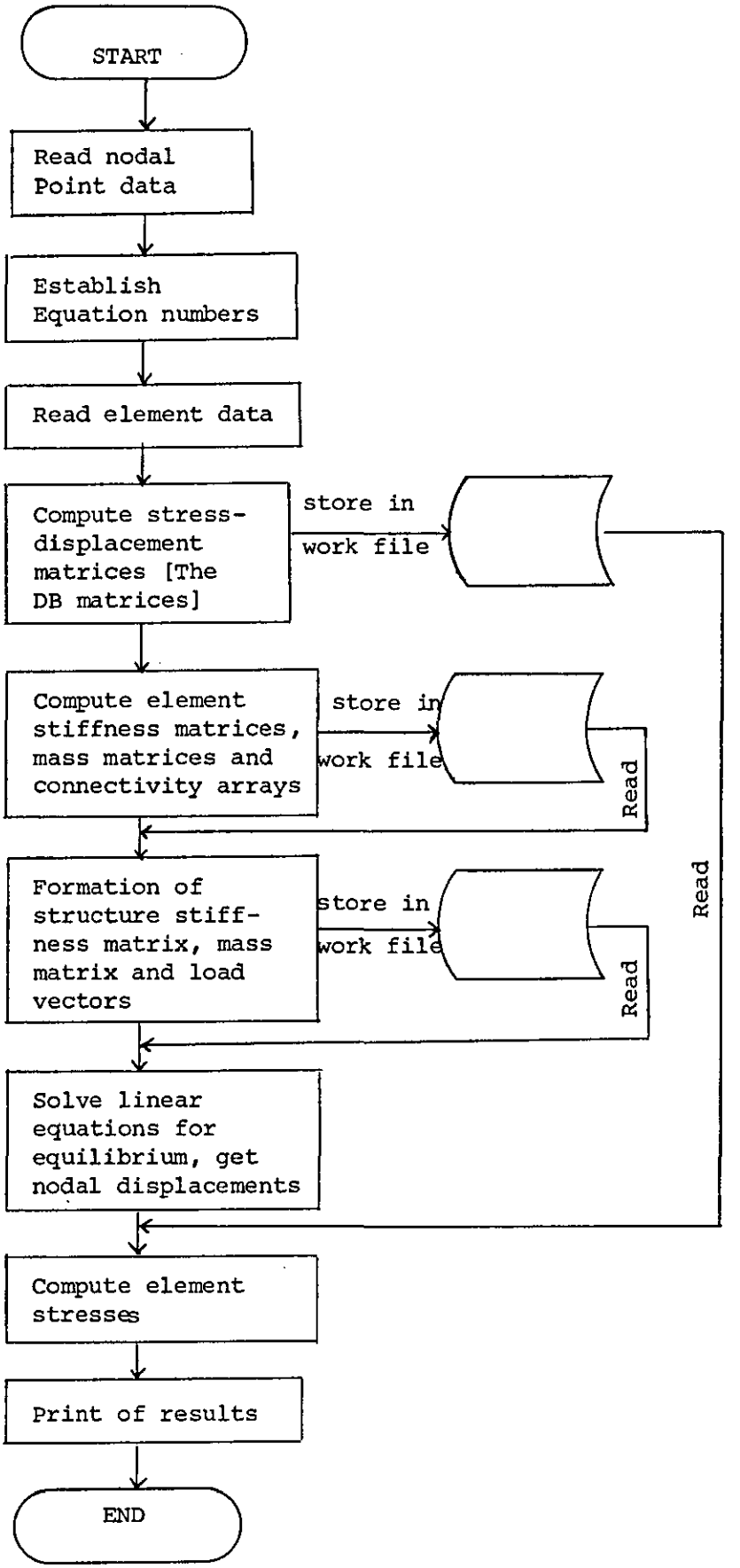


FIGURE 4.18: General flowchart for static analysis in MSAP

degrees of freedom, it is straightforward to compute the element stiffness matrices, the mass and stress-displacement transformation matrices for each structural element. This information and the element connectivity arrays are stored in a work file. In the MSAP program, all the elements of the same type are entered sequentially and grouped together. The static analysis involves the solution of the equilibrium equation f=Kd where f is the load vector, K is the stiffness matrix and d is the unknown displacement vector. The solution is obtained by decomposing K to $L^{T}DL$. It should be noted that such a decomposition is always possible in the linear static structural analysis problems where K is always symmetric and positive definite. After the solution of equations is completed, the stresses are computed using the stress-displacement matrices which have been stored on the work file. Boundary conditions in this program are handled as follows: (i) If a displacement component is zero, e.g. a support condition, then the corresponding equation is not retained in the structure equilibrium equations, and the corresponding element stiffness and mass terms are disregarded. (ii) If a prescribed displacement component is specified at a node with non-zero value, e.g. a settlement of support; then the corresponding component is multiplied by a very big number (1E20) which when solving the equilibrium equation will yield the prescribed displacement as a solution to the considered displacement component. This technique has already been explained in Chapter 2 of this thesis. In the following problem we will show how the equation numbers are established. Figure (4.19) is a truss in two-dimensions (i.e. plane truss) to be solved. The coordinate axes are a right-angled system with x-axis perpendicular to the plane. Recall the flowchart in Figure(4.18) we notice that the

first stage is the input of nodal point data. Nodes are assumed to be labelled with integers ranging from 1 to the total number of nodes in the structure, while elements are numbered serially from 1 but within each element group.

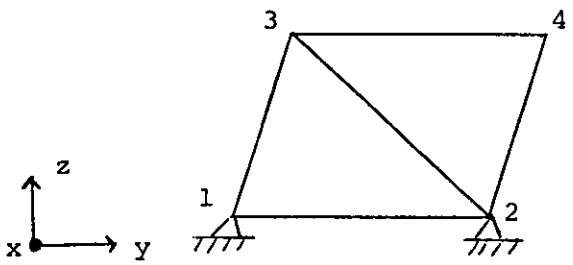


FIGURE 4.19: A truss example

The nodal points data for the truss example shown in Figure (4.19) are defined in the ID array where a "1" indicates a fixed condition i.e. a prescribed displacement of zero, while a "0" indicates a free degree of freedom. The 6 degrees of freedom are for displacements and rotations around the three coordinate axes. Since the considered problem is a two-dimensional one, the displacement in the x direction is always zero. Moreover, since the considered elements are the truss elements, no rotations are allowed in the x,y and z directions. Furthermore, nodes 1 and 2 are fully restrained. Thus ID will be:

$$ID = \begin{array}{c} \\ \text{Node} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccccc} & & & \text{Degrees of Freedom} & & \\ 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Equation numbering is done by scanning the ID array for the zero elements. Since each zero element corresponds to an unknown displacement,

it is evident that the total number of equations will be equal to the number of zero elements. Equations are numbered serially from 1. Since scanning is done row by row, i.e. node by node, it is clear that equation numbers will be assigned in the same manner. Equation numbers for the considered sample problem will be:

$$ID = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 3 & 4 & 0 & 0 & 0 \end{bmatrix}$$

In order to add the contribution of any element to the global stiffness matrix it is necessary to establish the map between element nodal points and equation numbers. This is done by specifying the corresponding equation numbers in the mapping vector Q. In our example, the vector Q for the member 3-4 will be:

$$Q = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 0 \\ 3 \\ 4 \end{bmatrix} \begin{matrix} x_i \\ y_i \\ z_i \\ x_j \\ y_j \\ z_j \end{matrix}$$

Note that x_i and x_j are always 0 for two-dimensional truss elements. Using Q together with the actual element stiffness matrix it will be easy to add the contribution of each element to the corresponding equation.

4.6.4 Installation Procedure

SAP IV was written in Fortran IV (ANSI66) and implemented on CDC computers. The program is about 15000 lines and structured in modules so that it can be easily overlaid. To implement this program on the IBM 3033 computer operating under the MVS operating system, the following points are observed:

(i) To have reasonable accuracy, double precision arithmetic is used.

(ii) To have a good execution speed, a region of 4096K of memory is allocated to the program. This is accomplished in the IBM Job Control Language (JCL) as:

```
// EXEC PGM=MSAP,REGION=4096K
```

(iii) Since MSAP access backing storage devices sequentially only; different files are allocated for different types of data, e.g. mass matrices and element stiffnesses are saved in two different files which are used temporarily during the execution of the program.

(iv) It is possible to have a data-check run where no actual processing took place. In this case all the input records are checked and saved on a file which could be used as an input for other programs or for the processing in a subsequent run.

(v) The JCL statements required to execute the MSAP program and allocating the necessary data sets are stored in a procedure in a library.

4.6.5 Discussion

MSAP is a general analysis tool for the linear static and dynamic analysis of complex structures. The root of this program, SAP I, is one of the first computer implementations of FEM in structural analysis. The main advantages of MSAP are being in the open literature with the source code supplied and its modest price. This makes it ideal for adaptation and modification by several users. However, the program itself suffers from the lack of pre- and post-processors. Generation of nodal coordinates is limited to equal increments along a straight line. No bandwidth reduction is tried. The user interface in SAP is not adequate. All input data must be formatted and arranged in a specific manner. The I/O operation on backing storage is done sequentially. The use of direct I/O will result in more efficient data transfer and will decrease the number of work files used. However, MSAP can be considered as a typical model of the first generation of the main-frame computer implementation of FE.

4.7 MINI-COMPUTER IMPLEMENTATION

4.7.1 Background

In the seventies minicomputers were developed. The term mini-computers as now used covers a wide spectrum of computers that range from the small size 16 bits computers to the bigger ones of 32 bits computers with virtual storage capabilities. What we consider here is a mid-range mini-computers. During the first years of computer implementation of FE on main-frame computers, the main criteria thoughts were speed, core storage size, backing storage size and organisation. However, after the development of mini-computers and the increased interest in interactive computing, other criteria for FE software take on more importance such as: pre- and post-processors, interactive programs and computer ergonomic aspects in general. Two programs are presented in this section. These are the ELASTIC and the STRAP programs. The ELASTIC is a FE package which consists of two programs: a FE analysis program and its interactive pre-processor. The FE analysis program of ELASTIC is an enhanced version of the coding given in the excellent book by Hinton and Owen [1979]. An interactive pre-processor is developed to prepare the input data for this program.

The STRAP program is a simple FE program for beam elements developed for educational purposes [Turaby and Sharaf Eldin, 1978]. The STRAP program was developed on the small mini-computer HP2100S, while the ELASTIC programs were developed on the medium-size mini-computer HP3000. The main features of minicomputers implementation are:

(i) Problems of less size can be handled as compared to the main-frame implementations. This means less element library, material library and procedure library. Typically, 3 to 5 elements are

expected to be supported and single material and procedure.

(ii) Programs are usually interactive with possible pre- and post-processors.

(iii) Computer graphics are supported as part of the FE system.

(iv) Inputs are usually in free format.

(v) Use of backing storage is limited to 4 work files only. Sometimes they are used for pre- and post-processing only while the computations are all in-core.

(vi) Cost is much less than those of main-frame computers.

(vii) Most of the source codes can be obtained. This makes them easy to be modified and adapted.

4.7.2 The ELASTIC Package

This package consists of two programs for the analysis of linear structural systems. The first program is a pre-processor developed for the second program which is the main FE processor. Although the FE processor is built on the subroutines given by Hinton and Owen [1979], however, the following enhancements are done:

(i) Critical variables are defined as double precision.

(ii) A full time-log routine is added to the FE processor that computes the CPU time spent in each step of the solution:

- Data entry and validation
- Element stiffnesses
- Loading data
- Frontal solution of the FE equations
- Stress computation.

(iii) The error handling method is modified to be more versatile

by keeping all the error messages on a separate file which is accessible in read only mode to multi-users. When an error is detected by the program, the appropriate messages are read directly using the record number which is set to the error number and printed. This will shorten the program length and this decreases the memory required to run this program. It also provides a more flexible way to modify the error messages. It is the author's opinion that "hard-coded" error messages should be, in general, avoided as much as possible in large systems where a fairly large number of error messages should be processed.

(iv) A pre-processor program is developed to increase the interaction between the user and the program. This pre-processor is of the ask-and-answer type. It prepares the input data sets for the ELASTIC processor itself.

(v) A procedure is designed to facilitate the operation of the whole package.

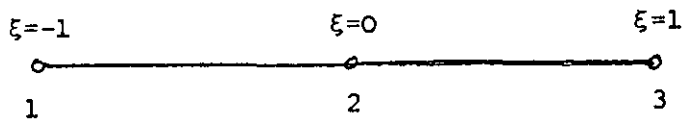
4.7.2.1 Element Library

Three elements are supported in this package: one-dimensional beam, 2-D plane stress, plain strain and plate bending elements. All of them are parabolic isoparametric elements.

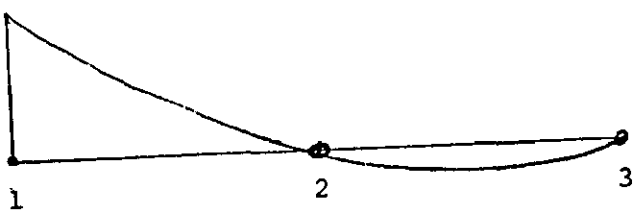
In order to develop the stiffness matrices for these elements, we proceed as explained in Chapter 2, i.e. the interpolation or shape functions are assumed, the strains are defined in terms of nodal displacement and relate the stress to strain. For the 1-D beam element, the parabolic isoparametric thick beam element, we have three nodes:

one at each end and one inbetween. We need three nodes to allow for a parabolic element. As shown in Figure 4.20, the shape functions are defined at each node such that the value of the function is 1 at the node itself and 0 at other nodes. These functions, in terms of the natural coordinates are:

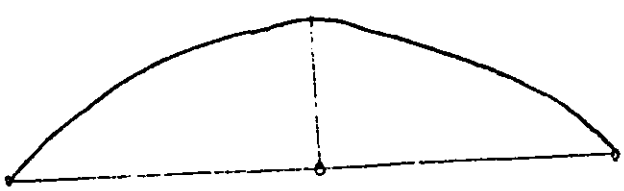
$$N_1(\xi) = -\tfrac{1}{2}\xi(1-\xi)$$
$$N_2(\xi) = (1-\xi)(1+\xi) \qquad (4.16)$$
$$N_3(\xi) = \tfrac{1}{2}\xi(1+\xi)$$



(a) Definition of nodes in the parabolic isoparametric thick beam element



(b) The shape function N_1



(c) The shape function N_2

(d) The shape function N_3

FIGURE 4.20: The parabolic isoparametric thick beam element

These functions can be easily computed as follows:

Since the shape functions are quadratic then we can write it in the form,

$$N_i = a_1\xi^2 + a_2\xi + a_3 \ . \tag{4.17}$$

Consider the case of the N_1 function, we have N_1=1 at ξ=-1 and N_1=0 at ξ=0 and at ξ=1. Thus,

$$\begin{aligned} a_1 - a_2 + a_3 &= 1 \\ a_3 &= 0 \\ a_1 + a_2 + a_3 &= 0 \end{aligned} \tag{4.18}$$

Solving these equations yields:

$$a_1 = \tfrac{1}{2},\ a_2 = -\tfrac{1}{2} \text{ and } a_3 = 0$$

which gives:

$$N_1 = -\tfrac{1}{2}\xi(1-\xi)$$

Similarly, N_2 and N_3 can be computed.

This procedure is of general nature in the determination of the coefficients of the chosen shape functions.

At each node i there are two degrees of freedom: the lateral displacement u_i and the rotation to the normal θ_i; thus for the whole element, i.e.,

$$\delta^e = \begin{Bmatrix} u_1 \\ \theta_1 \\ u_2 \\ \theta_2 \\ u_3 \\ \theta_3 \end{Bmatrix} \ . \tag{4.19}$$

The lateral displacement and rotation at any point can thus be defined in terms of those at the nodes only using the interpolation functions as follows:

$$u(\xi) = \sum_{i=1}^{3} N_i u_i$$

and

$$\theta(\xi) = \sum_{i=1}^{3} N_i \theta_i \quad . \tag{4.20}$$

Since the element is isoparametric, then by definition, the same interpolation functions used to define the displacement field within the element are used also to define its shape. Thus, the x-coordinate is defined by:

$$x(\xi) = \sum_{i=1}^{3} N_i x_i \quad . \tag{4.21}$$

The Jacobian matrix J is computed from:

$$J = \frac{\partial x}{\partial \xi} = \frac{\partial N_1}{\partial \xi} x_1 + \frac{\partial N_2}{\partial \xi} x_2 + \frac{\partial N_3}{\partial \xi} x_3 \tag{4.22}$$

i.e.,

$$J = (\xi - \tfrac{1}{2})x_1 - 2\xi x_2 + (\xi + \tfrac{1}{2})x_3 \tag{4.23}$$

In the special case where the second node (node number 2) is chosen to be at the middle of the element, J will be:

$$J = \frac{x_3 - x_1}{2} = \frac{L}{2} \quad , \tag{4.24}$$

where L is the element length.

In order to define the strains in terms of the nodal displacement i.e. to get the B matrix, we notice that according to the thick beam theory, the lateral displacement u_i and the rotation of the normal θ_i are associated with the relation:

$$\theta_i = \left(\frac{\partial u}{\partial x}\right)_i + \phi_i \tag{4.25}$$

or

$$\phi_i = \theta_i - \left(\frac{\partial u}{\partial x}\right)_i \tag{4.26}$$

where ϕ_i is the effective shear rotation.

Since
$$\frac{\partial N_i}{\partial x} = \frac{\partial N_i}{\partial \xi} \cdot \frac{\partial \xi}{\partial x} \tag{4.27}$$

it is possible to express the strain-nodal displacement relationship in the following equation:

$$\begin{bmatrix} \frac{\partial \theta}{\partial x} \\ \phi \end{bmatrix} = \begin{bmatrix} 0 & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} \\ -\frac{\partial N_1}{\partial x} & N_1 & -\frac{\partial N_2}{\partial x} & N_2 & -\frac{\partial N_3}{\partial x} & N_3 \end{bmatrix} \begin{bmatrix} u_1 \\ \theta_1 \\ u_2 \\ \theta_2 \\ u_3 \\ \theta_3 \end{bmatrix} . \tag{4.28}$$

The last step is to express the stress/strain relationship [the D matrix] for the element which is in the form of [see for example Timoshenko and Goodier, 1951]

$$\begin{bmatrix} M \\ Q \end{bmatrix} = \begin{bmatrix} EI & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} \frac{\partial \theta}{\partial x} \\ \phi \end{bmatrix} , \tag{4.29}$$

where, M is the bending moment

Q is the shearing force

EI is the flexural rigidity

and S is the shear rigidity = $\frac{GA}{\alpha}$

with G is the shear modulus

A is the cross sectional area

α is the warping factor.

The stiffness matrix of the element K^e can be calculated from:

$$K^e = \int [B]^T DB \, dx \tag{4.30}$$

Since all the matrices B and D are computed, it is possible to write the expression for K^e. The integrand $B^T DB$ is:

$$
\begin{bmatrix}
S\left(\frac{\partial N_1}{\partial x}\right)^2 & & & & & \\
-N_1 S \frac{\partial N_1}{\partial x} & EI\left(\frac{\partial N_1}{\partial x}\right)^2 + N_1^2 S & & \text{Symmetric} & & \\
S\left(\frac{\partial N_1}{\partial x}\right)\left(\frac{\partial N_2}{\partial x}\right) & -N_1 S \frac{\partial N_2}{\partial x} & S\left(\frac{\partial N_2}{\partial x}\right)^2 & & & \\
-N_2 S \frac{\partial N_1}{\partial x} & EI \frac{\partial N_1}{\partial x} \frac{\partial N_2}{\partial x} + N_1 N_2 S & -N_2 S \frac{\partial N_2}{\partial x} & EI\left(\frac{\partial N_2}{\partial x}\right)^2 + N_2^2 S & & \\
S\left(\frac{\partial N_1}{\partial x}\right)\left(\frac{\partial N_3}{\partial x}\right) & -N_1 S \frac{\partial N_3}{\partial x} & S \frac{\partial N_2}{\partial x} \frac{\partial N_3}{\partial x} & -S N_2 \frac{\partial N_3}{\partial x} & S\left(\frac{\partial N_3}{\partial x}\right)^2 & \\
-N_3 S \frac{\partial N_1}{\partial x} & EI \frac{\partial N_1}{\partial x} \frac{\partial N_3}{\partial x} + N_1 N_3 S & -N_3 S \frac{\partial N_2}{\partial x} & EI \frac{\partial N_2}{\partial x} \frac{\partial N_3}{\partial x} + N_2 N_3 S & -N_3 S \frac{\partial N_3}{\partial x} & EI\left(\frac{\partial N_3}{\partial x}\right)^2 + N_3^2 S
\end{bmatrix}
$$

(4.31)

Although it is possible to compute these terms and do the integration analytically in this particular case, it is customary to use numerical integration instead. In fact, the use of explicit forms for the evaluation of the elements of the stiffness matrices is done in the very simple cases only like the truss elements as explained in Chapter 2.

One of the methods which are used to do the numerical integration is the Gauss quadrature method which is employed in this package.

In the case of plane stress and plane strain elements we proceed in a similar manner but in this case we have a two-dimensional element. The displacement field and the geometry of the element can be expressed using the same shape functions as follows:-

Figure 4.21 shows the isoparametric element and its parent quadratic element. The general form of the interpolation function is:

$$N_i = a_1+a_2\xi+a_3\eta+a_4\xi^2+a_5\eta^2+a_6\xi\eta+a_7\xi^2\eta+a_8\xi\eta^2 \quad . \tag{4.32}$$

Substituting the value of N at the 8 nodes of the element it is possible to determine the coefficients $a_1, a_2, \ldots, a_8$ in a similar manner to that done for the case of the beam element. The shape functions are found to be:

$$\begin{aligned}
N_1 &= -1/4(1-\xi)(1-\eta)(1+\xi+\eta) \\
N_2 &= 1/2(1-\xi^2)(1-\eta) \\
N_3 &= 1/4(1+\xi)(1-\eta)(\xi-\eta-1) \\
N_4 &= 1/2(1+\xi)(1-\eta^2) \\
N_5 &= 1/4(1+\xi)(1+\eta)(\xi+\eta-1) \\
N_6 &= 1/2(1-\xi^2)(1+\eta) \\
N_7 &= 1/4(1-\xi)(1+\eta)(-\xi+\eta-1) \\
N_8 &= 1/2(1-\xi)(1-\eta^2)
\end{aligned} \tag{4.33}$$

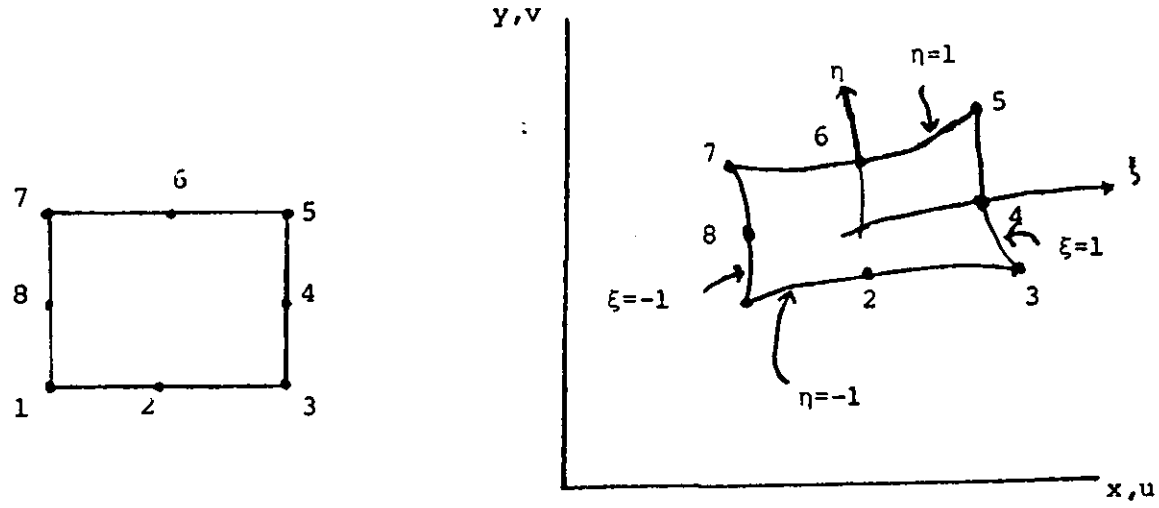


FIGURE 4.21: Quadratic parent element and isoparametric element

The displacements u and v at any point within the element can be expressed in terms of those at nodes using the interpolation functions as follows:

$$u = \sum_{i=1}^{8} N_i u_i$$

$$v = \sum_{i=1}^{8} N_i v_i \qquad (4.34)$$

The coordinates $x(\xi,\eta)$ and $y(\xi,\eta)$ of any point (ξ,η) within the element will also be expressed as:

$$x(\xi,\eta) = \sum_{i=1}^{8} N_i(\xi,\eta).x_i$$

$$y(\xi,\eta) = \sum_{i=1}^{8} N_i(\xi,\eta).y_i \qquad (4.35)$$

The Jacobian matrix $J(\xi,\eta)$ for this case will be,

$$J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix}$$

$$= \sum_{i=1}^{8} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \cdot x_i & \frac{\partial N_i}{\partial \xi} \cdot y_i \\ \frac{\partial N_i}{\partial \eta} \cdot x_i & \frac{\partial N_i}{\partial \eta} \cdot y_i \end{bmatrix} \tag{4.36}$$

The inverse of the Jacobian matrix $[J]^{-1}$ is,

$$[J]^{-1} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} = \frac{1}{\det J} \begin{bmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial y}{\partial \xi} \\ -\frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \xi} \end{bmatrix} \tag{4.37}$$

The strain matrix B which relates the strain to nodal displacement is:

$$B_i = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} \end{bmatrix} \quad , \text{ for } i=1,2,\ldots,8 \tag{4.38}$$

and $B = [B_1,\ldots,B_8]$

The matrix of elastic constants D for the plane stress situation is defined by,

$$D = \frac{E}{(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{4.39}$$

whereas for plane strain situations:

$$D = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{bmatrix} \tag{4.40}$$

where E is the Young's modulus of elasticity and ν is the Poisson's ratio.

Finally, the element stiffness matrix K^e is calculated from:

$$K^e = \iint [B]^T D B \, dv \tag{4.41}$$

A submatrix K^e_{ij} linking the nodes i and j may be evaluated from the expression:

$$K^e_{ij} = \iint [B_i]^T D B_j t \det J \, d\xi d\eta \tag{4.42}$$

where t is the element thickness and,

$$dxdy = \det J \, d\xi d\eta \tag{4.43}$$

In the case of the plate bending element the interpolation functions $N_1,\ldots,N_8$ are identical to those used in the case of the plane stress/strain element. The geometry is also expressed using the same functions. However, the nodal displacement components here are: ω: the deflection, θ_x the average rotation about x-axis and θ_y the average rotation about y-axis. As in the case of the beam element θ_x and θ_y can be expressed in terms of $\frac{\partial \omega}{\partial x}$, $\frac{\partial \omega}{\partial y}$, ϕ_x and ϕ_y, where ϕ_x and ϕ_y are the average shear deformation in x and y directions respectively [Mindlin, 1951]. Thus we can write:

$$\delta = \begin{bmatrix} \omega \\ \theta_x \\ \theta_y \end{bmatrix} = \begin{bmatrix} \omega \\ \frac{\partial \omega}{\partial x} + \phi_x \\ \frac{\partial \omega}{\partial y} + \phi_y \end{bmatrix} = \sum_{i=1}^{8} N_i \delta_i \tag{4.44}$$

The strain matrix $B=[B_1,B_2,\ldots,B_8]$ is given by:

$$B_i = \begin{bmatrix} 0 & -\frac{\partial N_i}{\partial x} & 0 \\ 0 & 0 & -\frac{\partial N_i}{\partial y} \\ 0 & -\frac{\partial N_i}{\partial y} & -\frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial x} & -N_i & 0 \\ \frac{\partial N_i}{\partial y} & 0 & -N_i \end{bmatrix} . \tag{4.45}$$

$$D = \begin{bmatrix} \frac{Et^3}{12(1-\nu^2)} & \frac{Et^3}{12(1-\nu^2)} & 0 & 0 & 0 \\ \frac{\nu Et^3}{12(1-\nu^2)} & \frac{Et^3}{12(1-\nu^2)} & 0 & 0 & 0 \\ 0 & 0 & \frac{(1-\nu)}{2}\frac{Et^3}{12(1-\nu^2)} & 0 & 0 \\ 0 & 0 & 0 & \frac{Et}{2.4(1+\nu)} & 0 \\ 0 & 0 & 0 & 0 & \frac{Et}{2.4(1+\nu)} \end{bmatrix} \quad (4.46)$$

Thus the element stiffness matrix can be calculated in an identical manner to that of a plane stress/strain element.

4.7.2.2 Implementation Details

The ELASTIC package consists of two integrated programs: the ELASTIC pre-processor and the ELASTIC processor. The ELASTIC pre-processor is an interactive program which prompts the user for the input data. It validates these data and stores it in a file for subsequent processing by the ELASTIC processor. A general flowchart of the ELASTIC preprocessor is shown in Figure 4.22. The data required by the preprocessor is as follows:

(i) The menu options displayed by the pre-processor are:

1 = Beam

2 = Plain stress

3 = Plain strain

4 = Plate bending

(ii) The control data is:

- The problem title

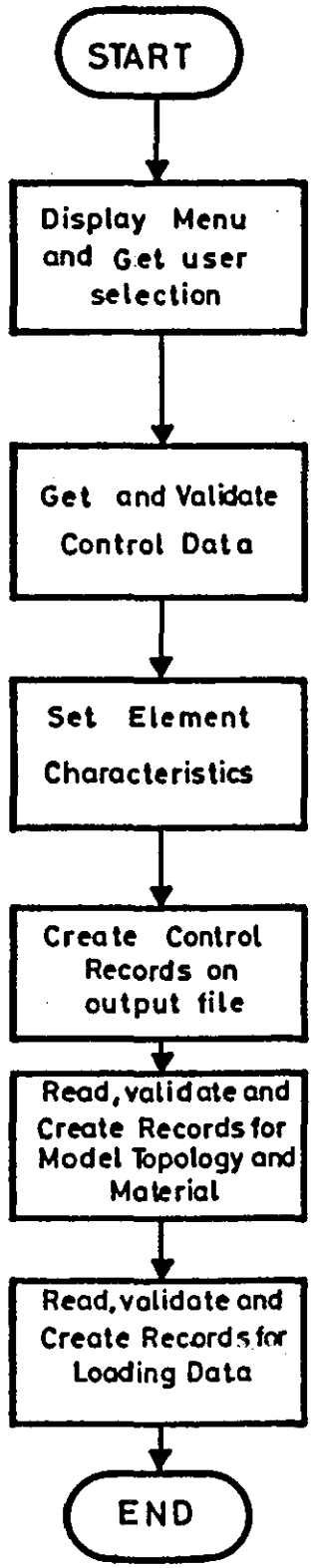


Figure: 4.22
A Block Flowchart For the ELASTIC Pre Processor

- Number of nodal points in the structure
- Number of elements in the structure
- Number of nodes where a known displacement is prescribed.
- Number of loading cases
- Number of different materials.

(iii) Elements characteristic parameters are set by the preprocessor according to the selection done at (i) as follows:

- Number of nodes per element:

  3 for beam elements

  8 otherwise

- Number of degrees of freedom per nodal point:

  2 for beam elements: u and θ

  2 for plane stress/strain elements: u,v

  3 for plate bending elements: $\omega, \theta_x, \theta_y$.

- Number of material parameters:

  3 for beam analysis

  5 for plane stress/strain

  4 for plate bending

- Number of coordinate components required to define each nodal point:

  1 for beam analysis

  2 otherwise.

- Number of independent stress components at any point:

  2 for beam analysis

  3 otherwise

The ELASTIC process reads the data file created by the preprocessor and perform the FE analysis. A general block diagram of this program is shown

in Figure 4.23. Four work files are used in this program in addition to the input file which is prepared by the preprocessor. The work files are referred to as unit numbers 1,2,3 and 4 in the source Fortran program. The input data file is number 5. File 1 is used to store the element stiffness matrices. File 3 is used to store the stress matrix and the Gauss point coordinates for the elements. Files 2 and 4 are used in the frontal subroutine: file 2 is used to store the reduced equations; while file 4 is used to hold the righthand sides of the equations. To facilitate the execution of this program, the following procedure has been designed:

```
1.  GOELASTIC Indata
2.  PURGE WORK1,WORK2,WORK3,WORK4
3.  Build WORK1, WORK3 on LDN 1
4.  Build WORK2, WORK4 on LDN 2
5.  FILE FTN01=WORK1,OLD
6.  FILE FTN02=WORK2,OLD
7.  FILE FTN03=WORK3,OLD
8.  FILE FTN04=WORK4,OLD
9.  FILE FTN05=!Indata,OLD
10. RUN ELASTIC
11. RESET FTN01,FTN02,FTN03,FTN04,FTN05
```

- Line 1 of this procedure specify the procedure name and the required argument. The name given to this procedure is GOELASTIC, the required argument is given the symbolic name Indata which is the file name of the input data that has been created by the pre-processor.

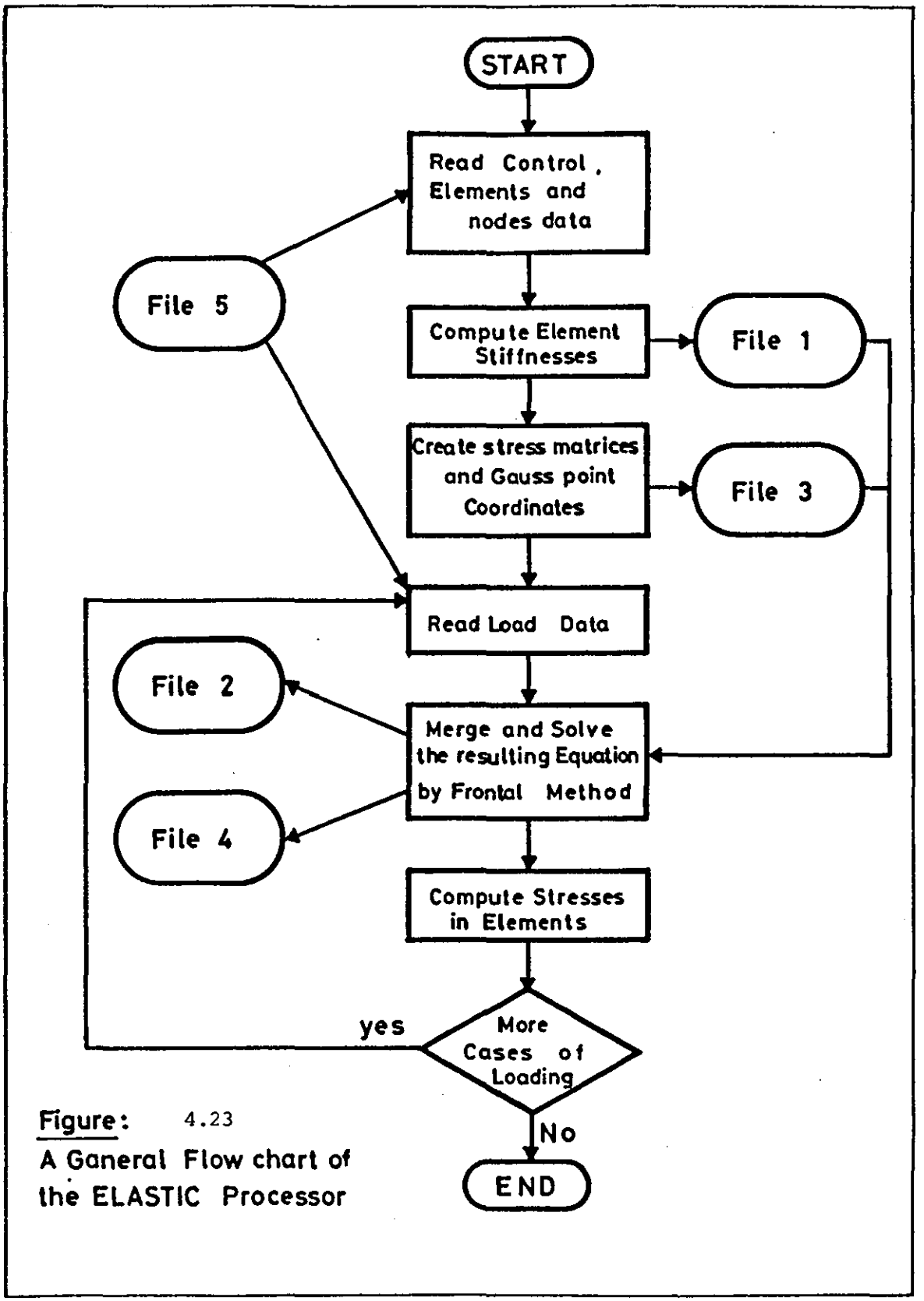


Figure: 4.23
A Ganeral Flow chart of the ELASTIC Processor

- Line 2 purges the four work files needed by this program. This is done so that any old files used in a previous problem will be purged out.
- Line 3 builds (creates) the new files work 1 and work 3 on a logical device number different from those of the other two work files work 2 and work 4. This "trick" is found to be useful in speeding up the turn around time of the run.
- Line 4 is the same but for building files work 2 and work 4.
- Lines 5 to 9 set the necessary file equations required by the operating system in order to allocate actual files to Fortran unit numbers.
- Line 10 is running the ELASTIC program.
- Line 11 is to cancel the file equations previously defined in lines 5 to 9.

4.7.2.3 The ELASTIC Program Structure

The ELASTIC program has a modular structure. It consists of a main segment which drives the whole subroutines in the program. The basic FE steps are performed by primary subroutines which in turn call auxiliary subroutines to carry out secondary operations. In addition to that, utility subroutines are added which do general utility operations. These routines and their functions are as follows:

INPUT: reads the input data which has been prepared by the preprocessor.

STIFB: calculates the stiffness and stress matrices for the beam element.

STIFPS: calculates the stiffness and stress matrices for the plane stress/strain element.

STIFPB: calculates the stiffness and stress matrices for the plate bending element.

LOADB: calculates the equivalent nodal loads for beam elements.

LOADPS: calculates the equivalent nodal loads for plane stress/strain elements.

LOADPB: calculates the equivalent nodal loads for plate bending elements.

FRONT: solves the FE equations by the frontal algorithm.

CPU TIME: a function to give the total CPU time consumed so far from the start of computation.

STREB: computes the stress components for the beam element.

STREPS: computes the stress components for the plane stress/strain element.

STREPB: computes the stress components for the plate bending elements.

NODEXY: calculates the coordinates of midside nodes in plane stress/strain or plate bending elements.

GAUSSQ: sets up the sampling point position and weighting constants for numerical integration by Gaussian quadrature.

MODB: calculates the elements of the D matrix for beam elements.

MODPS: calculates the elements of the D matrix for plane stress/strain elements.

MODPB: calculates the elements of the D matrix for plate bending elements.

SFR1: computes the shape functions and their derivatives in one dimension in natural coordinates for the beam element.

SFR2: computes the shape functions and their derivatives in two-dimensions in natural coordinates for plane stress/strain and plate bending elements.

JACOB 1: computes the Jacobian matrix, its inverse and the Cartesian derivatives of the shape functions for the one-dimensional case, i.e. for beam elements.

JACOB 2: computes the Jacobian matrix, its inverse and the Cartesian derivatives of the shape functions for the two-dimensional case i.e. for plane stress/strain and plate bending elements.

BMATB: computes the B matrix for beam-elements.

BMATPS: computes the B matrix for plane stress/strain elements.

BMATPB: computes the B matrix for plate bending elements.

DBE: performs the matrix multiplication DB.

ERRORMSG: writes error messages specified by their numbers.

To facilitate the communications between different segments of the program, common blocks are used. Three common blocks are used:

CONTROL: which contains the control parameters of the problem being solved as has been explained in the pre-processor earlier like number of elements, number of nodal points, etc. This common block is required in all subroutines (except the error messages one) and in the master segment as well.

LGDATA: which contains arrays needed to hold the FE topology, material, loads and boundary condition arrays.

WORK: which contains work arrays used at the element level e.g. D^e,B^e,... etc.

4.7.2.4 Numerical Tests

The aims of the following test problems are to verify the correctness of the ELASTIC package and to try to find out approximate values of the

CPU time spent in each phase of the solution process. In addition to that, the package is a useful educational and analysis tool for some of the standard structural mechanics problems. In otherwords, it can be used to solve a fairly wide class of problems in structural mechanics, structural analysis and elasticity.

Test Problem 1

This problem is taken from Hinton and Owen [1979] to check the validity of the ELASTIC package. The problem is shown in Figure (4.24) It is a simply supported beam of unit length subjected to a uniformly distributed load of intensity q=1.0. The material properties are: EI=1 and S=1000. The FE model is composed of two beam elements with 5 nodes. The results obtained are:

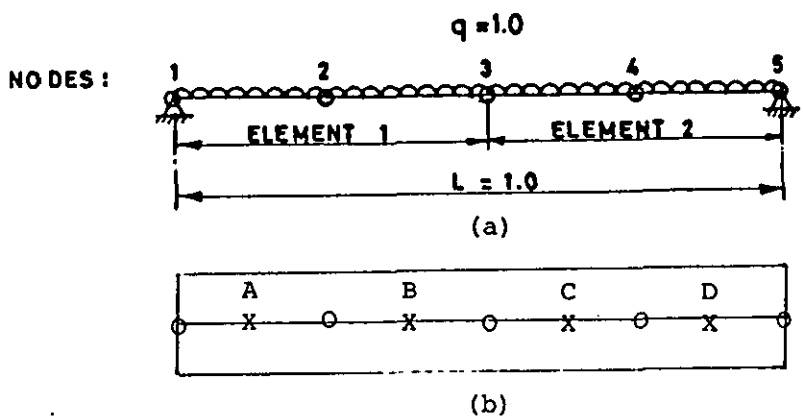


FIGURE 4.24: Test problem 1 for ELASTIC

(a) The beam problem
(b) The FE model:
2 Elements: 1,2
5 Nodes: 1,2,3,4,5
4 Gauss points: A,B,C,D

- Reactions at nodes 1 and 5 are -.5 and -.5.
- Displacements at different nodes are:

Node	Displacement	Rotation
1	0.0	.041667
2	.0091240	.02865
3	.013033	0.
4	.0091240	-.02865
5	0	-.041667

- Stresses are computed at Gauss points:

Point	Moment	Shearing Force
A	-.04725	-.3943
B	-.1194	-.10566
C	-.1194	.10566
D	-.04725	.3943

These results are almost identical to those quoted in the stated reference and are in excellent agreement with those predicted by simple beam theory.

The distribution of the CPU time required to solve this problem on the minicomputer HP3000 series II is as shown in Table 4.1.

Phase	CPU time in seconds	Percentage % to total time
Data entry and Validation	.43	24.4
Element stiffnesses	.16	9.0
Loading	.08	4.5
Frontal solution	.94	53.1
Stress computation	.16	9.0
TOTAL	1.77	100.0

TABLE 4.1: CPU time distribution for test Problem 1

It is clear from this table that more than 50% of the CPU time required to solve this problem is spent in the solution of the FE equations. This demonstrates the critical role played by equation solvers in FE analysis. The other interesting result is that about 25% of the total time is spent in data entry and validation. This in turn emphasises the importance of preprocessors.

However, since this problem is fairly simple, we consider a series of test problems that vary in the number of elements, nodes and dimensionality. We start by one-dimensional problems, i.e. beam elements. The test problems are all similar in structure but with different number of elements. We notice that in these problems the number of nodes n_n and the number of elements n_e are related by the following equation,

$$n_n = 2n_e + 1 \ , \tag{4.47}$$

The results are summarized in Table 4.2. These results show that more than half of the total CPU time is spent in the frontal solution algorithm irrespective of the number of nodes (and elements). The

stiffnesses computation takes relatively less ratio of the time that increases with the increase in the number of nodes. This is due to the simplicity of 1-D element stiffness. The Jacobian in this case is a scalar rather than a matrix. Also the number of Gauss points is two only in each element.

Phase	5 nodes		11 nodes		21 nodes		41 nodes	
	Time (sec.)	%	Time (sec.)	%	Time (sec.)	%	Time (sec.)	%
Data entry and validation	.43	24.4	.650	20.1	1.014	17.2	1.631	14.5
Element stiffness	.16	9.0	.397	12.3	.794	13.5	1.604	14.3
Loading	.08	4.5	.166	5.1	.343	5.8	.765	6.8
Frontal solution	.94	53.1	1.688	52.2	3.081	52.4	5.947	53
Stress computation	.16	9.0	.331	10.3	.653	11.1	1.264	11.4
TOTAL	1.77	100.0	3.232	100	5.885	100	11.211	100

TABLE 4.2: CPU time distribution for problems of one dimensional elements

Phase	13 nodes		28 nodes		53 nodes		103 nodes	
	Time (sec.)	%	Time (sec.)	%	Time (sec.)	%	Time (sec.)	%
Data entry and validation	.615	8.2	1.140	6.5	2.050	5.9	3.413	5.1
Element stiffness	2.870	38.3	7.414	42.1	14.933	43.1	29.314	43.6
Loading	.414	5.5	1.042	5.9	2.055	5.9	4.009	6.0
Frontal solution	2.683	35.8	5.972	33.9	11.485	33.2	22.536	33.5
Stress computation	.915	12.2	2.060	11.6	4.099	11.9	7.925	11.8
TOTAL	7.497	100	17.628	100	34.622	100	67.197	100

TABLE 4.3: CPU time distribution for problems of 2-D elements

In the case of two-dimensional problems there is no general equation that relates the number of elements to the number of nodes in general. However, in the considered problem of plane stress/strain the domain is assumed to be a rectangle [Figure 4.25] which is discretized systematically into smaller two-dimensional plane stress/strain elements. In this particular case, the number of nodes n_n and the number of elements n_e are related according to:

$$n_n = 5n_e + 3 \ . \tag{4.48}$$

The results of these test problems are summarized in Table 4.2. It indicates that the frontal solution's share of total time is no more dominant as the case in 1-D problems, but rather, the element stiffness consumes more CPU time. The ratio of the frontal solution is around 1/3 of the total time. The element stiffness consumes little more than 40% of the total time. The data entry and validation share is around 5% or 6% only. The main reason is that element stiffness in 2-D are much more complicated in comparison to that in 1-D. The Jacobian is no longer scalar but rather a matrix that needs to be inverted. Numerical integration is in two dimensions rather than one with 3 Gauss points instead of 2.

It is possible to conclude that most of the CPU time in 1-D problems is expected to be spent in the solution of the FE equations while in the 2-D problems both the solution of the FE equations and the element stiffness consume most of the time. It seems, however, that computation of element stiffness may need more research efforts to reach an optimal strategy with sufficient generality for implementation.

These results are shown graphically in Figures 4.26 to 4.39.

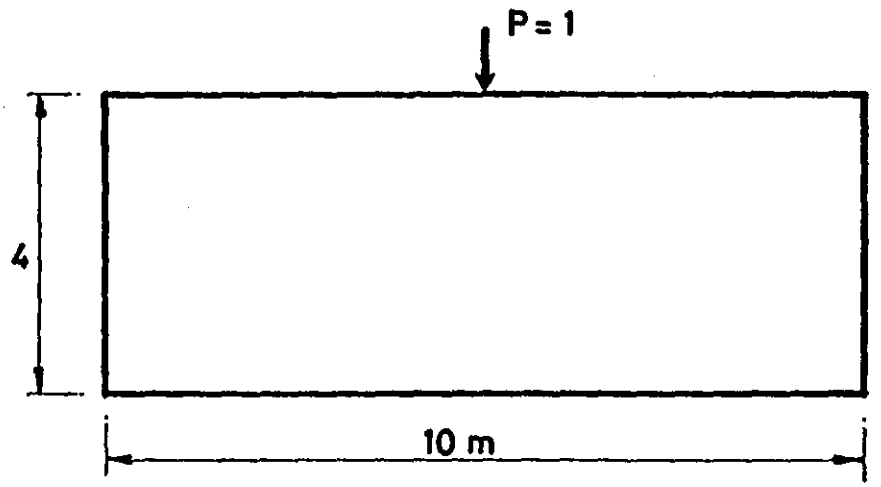


Figure 4.25 Test Problem # 2

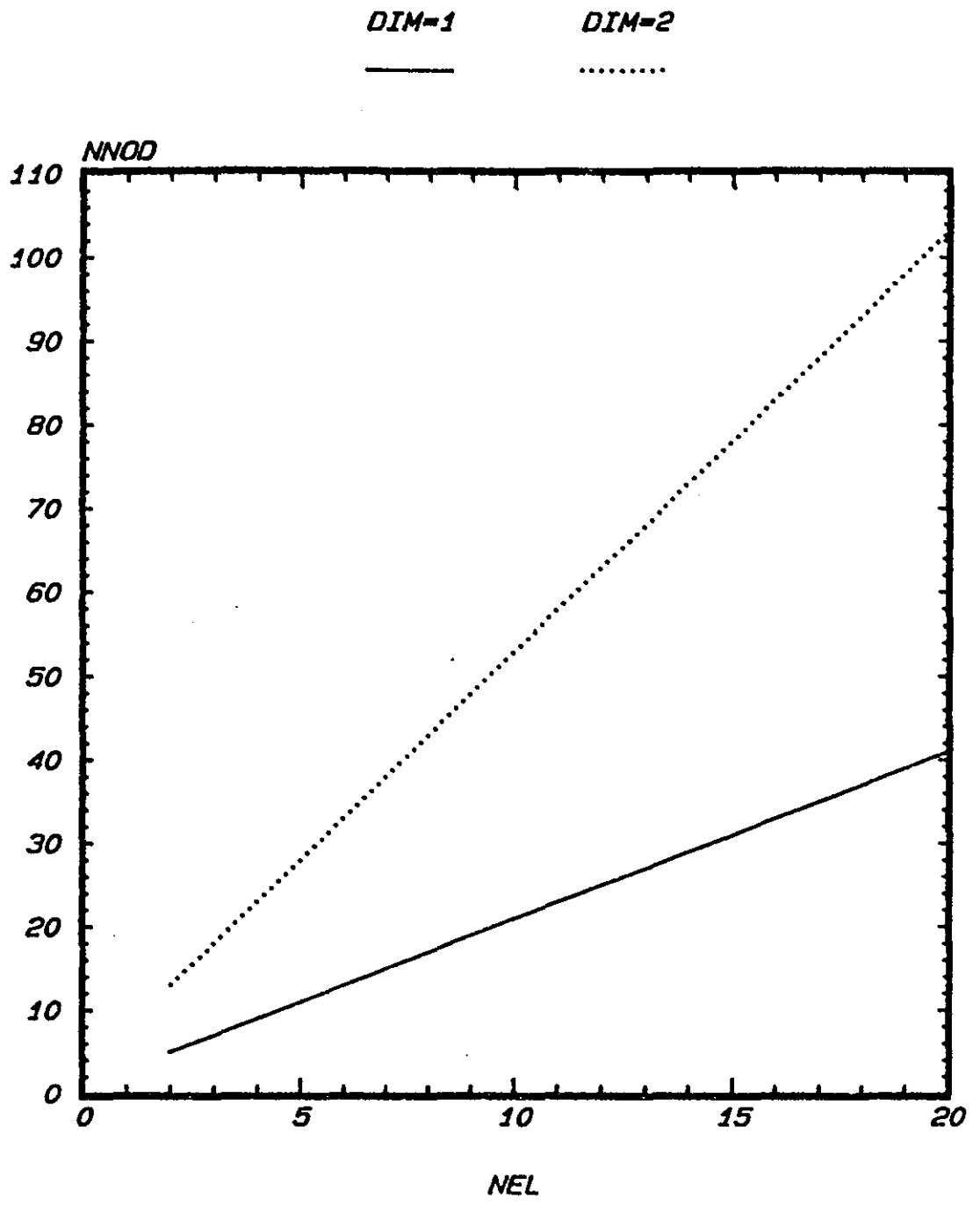


FIG. (4.26) RELATIONSHIP OF NEL - NNOD

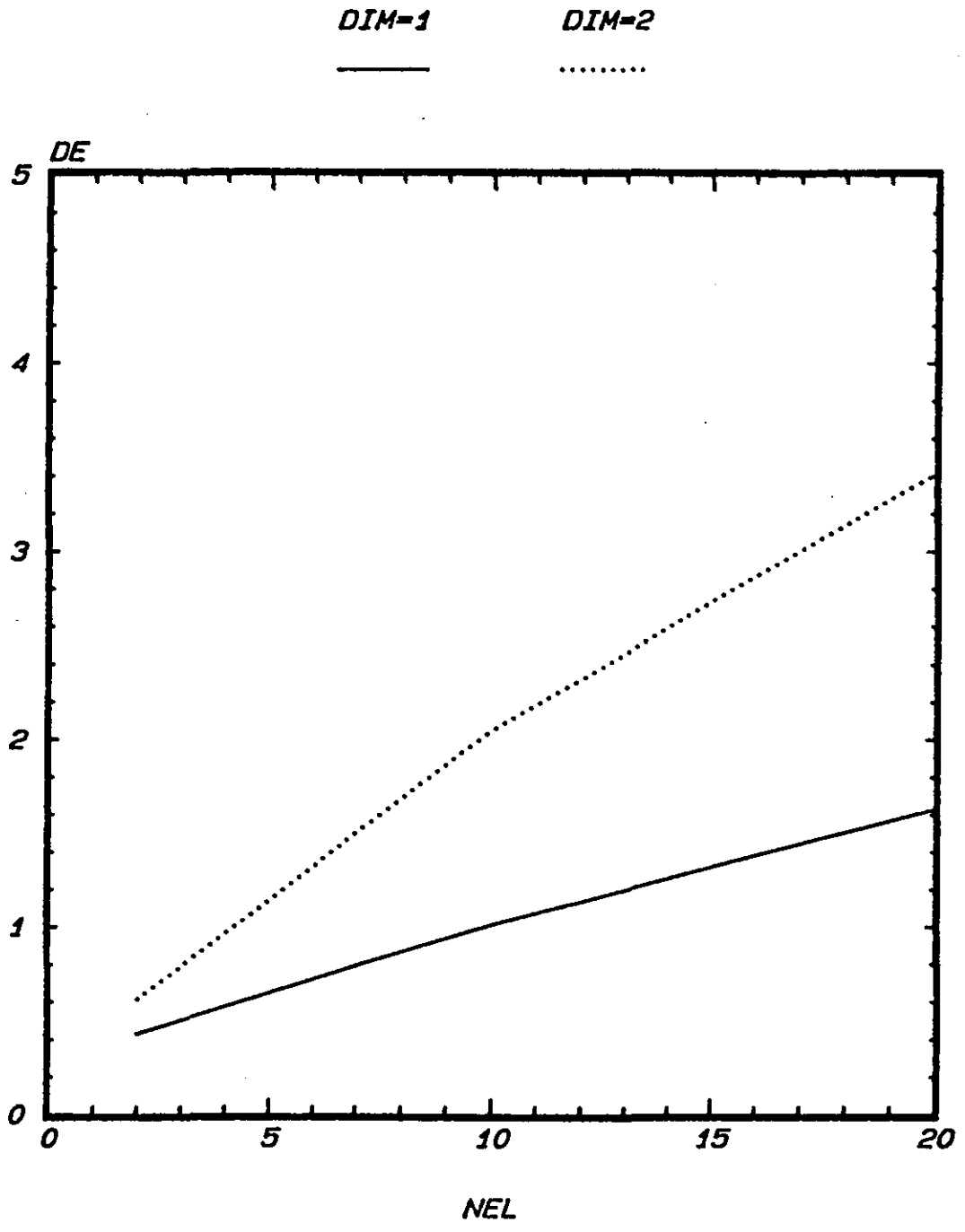


FIG. (4.27) EFFECT OF NEL ON DE TIME

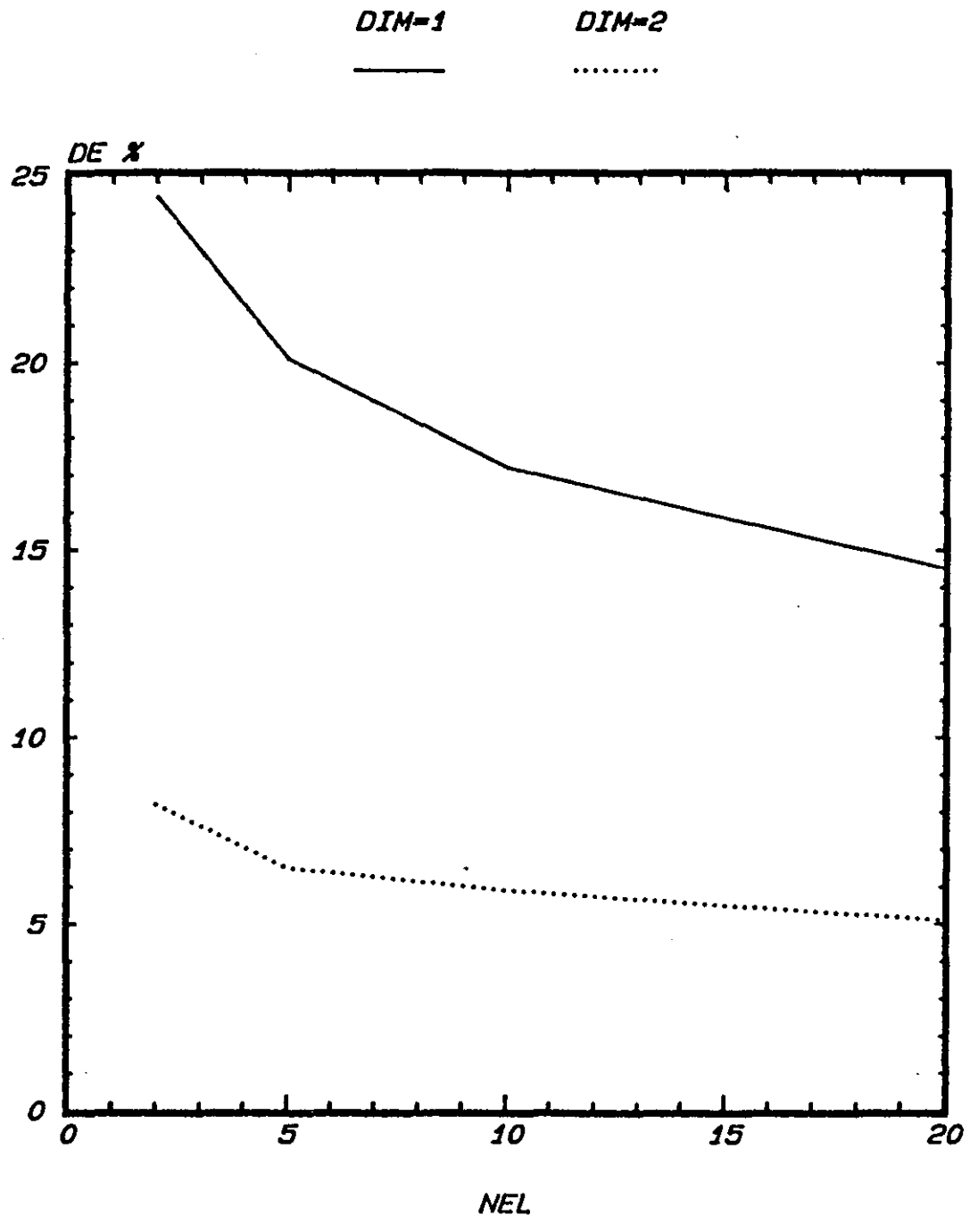


FIG. (4.28) EFFECT OF NEL ON DE % TIME

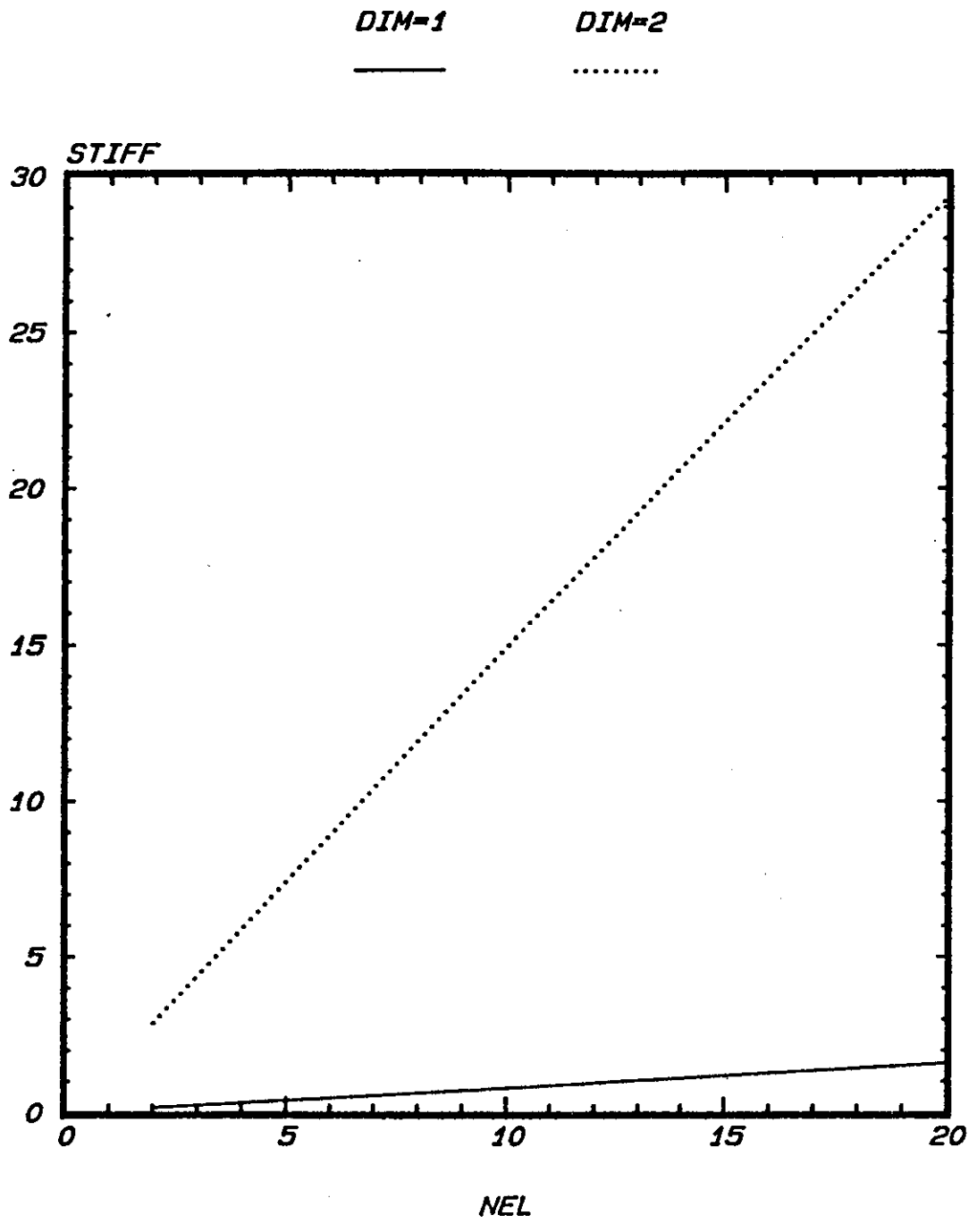


FIG. (4.29) EFFECT OF NEL ON STIFF TIME

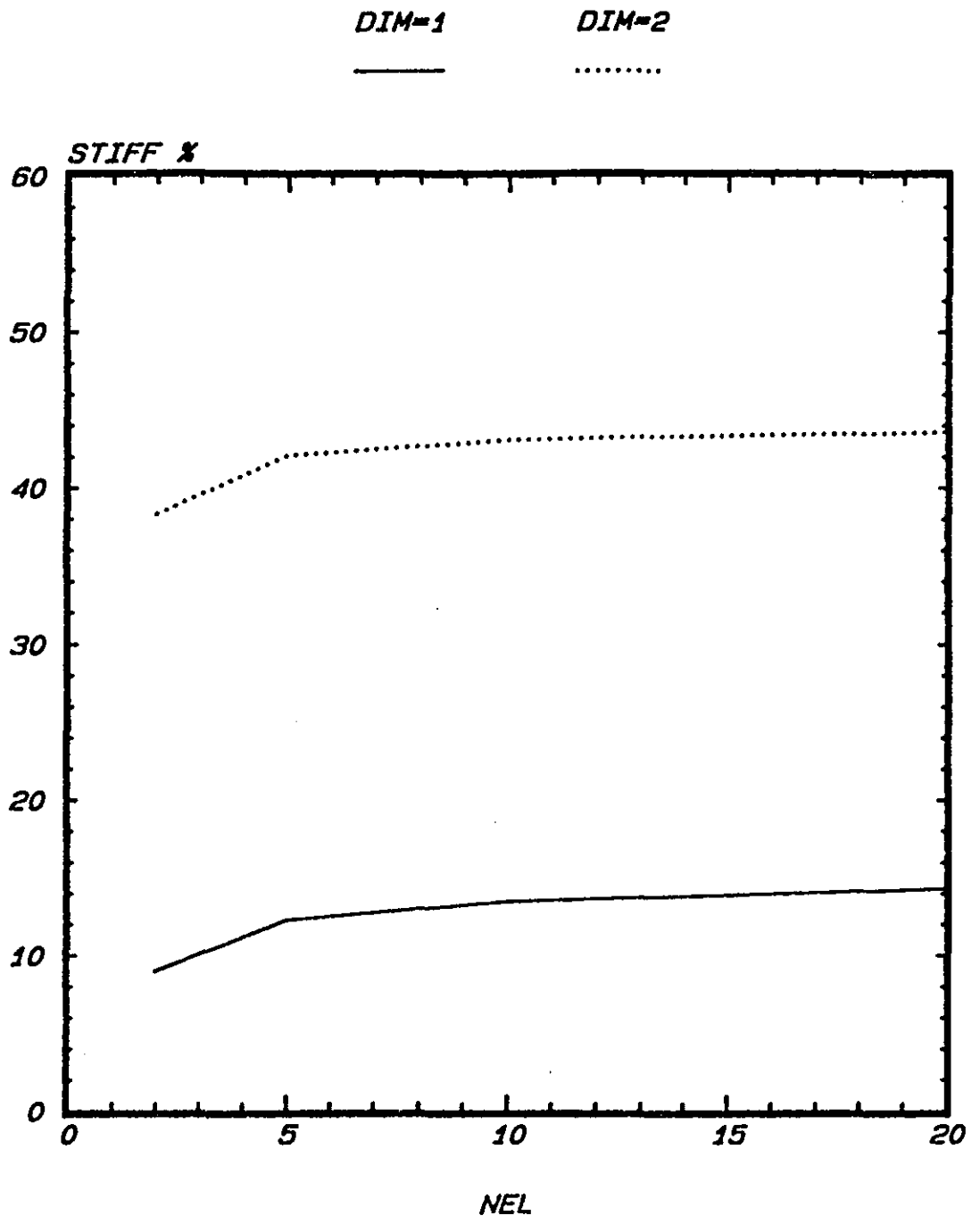


FIG. (4.30) EFFECT OF NEL ON STIFF % TIME

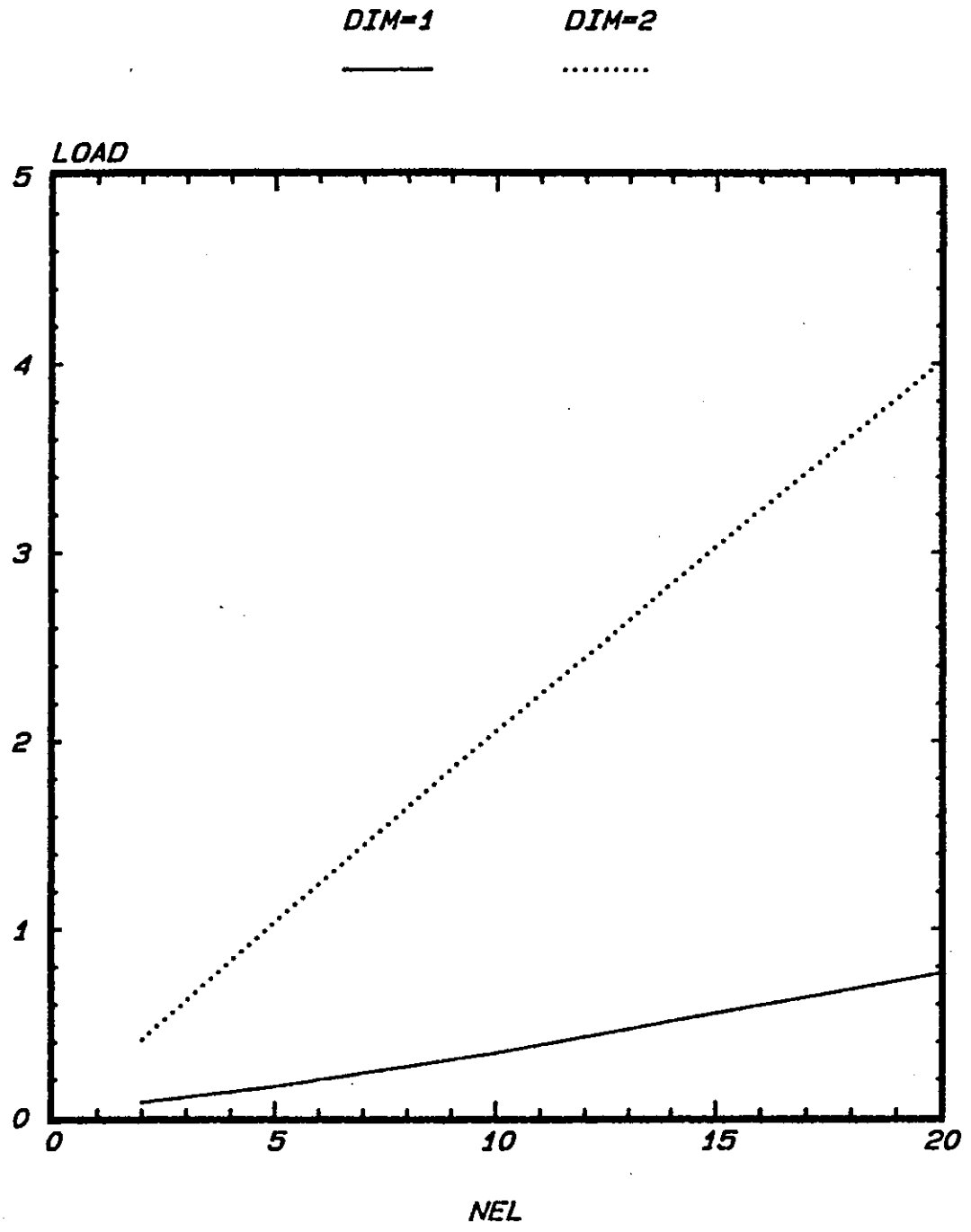


FIG. (4.31) EFFECT OF NEL ON LOAD TIME

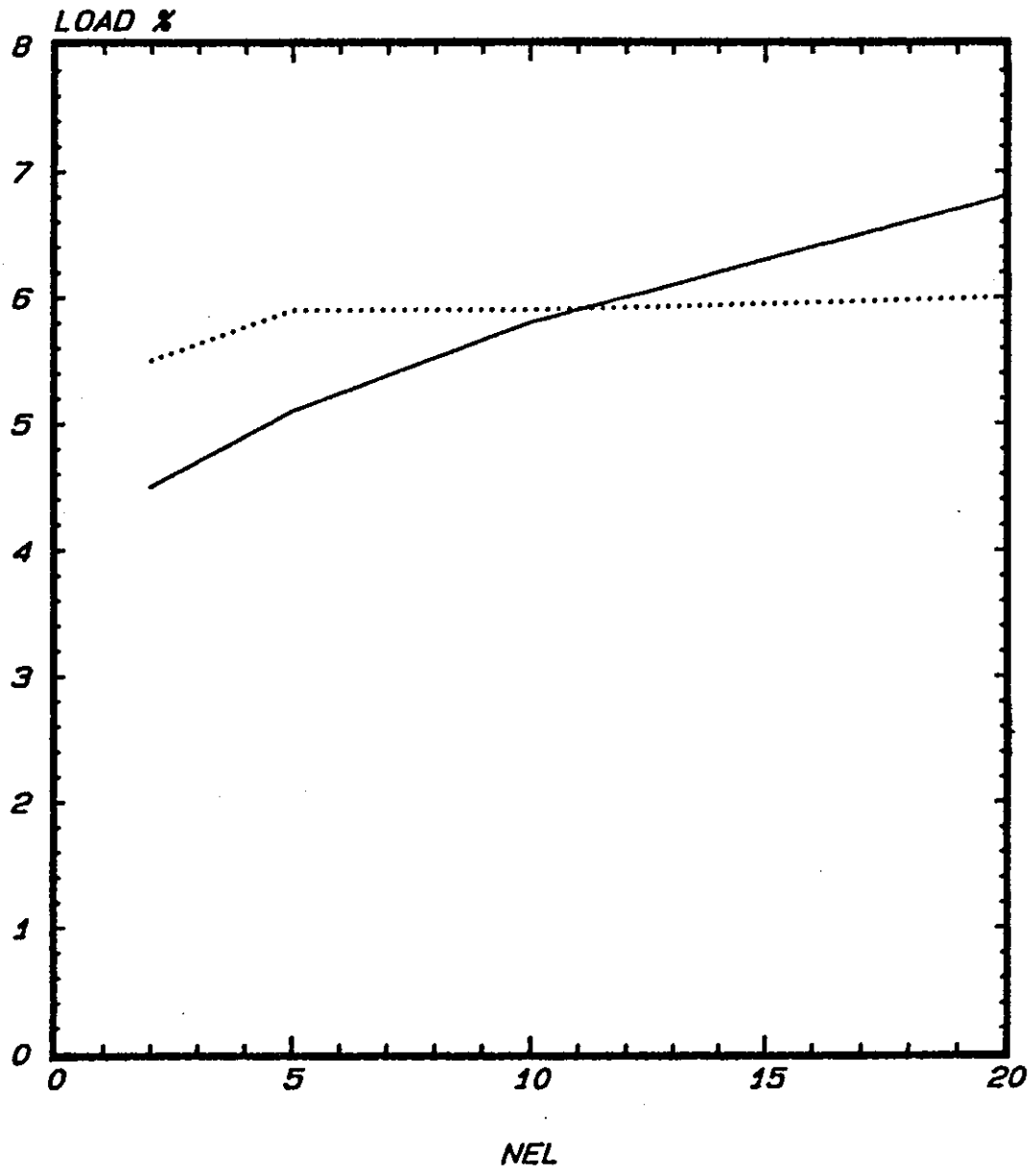


FIG. (4.32) EFFECT OF NEL ON LOAD % TIME

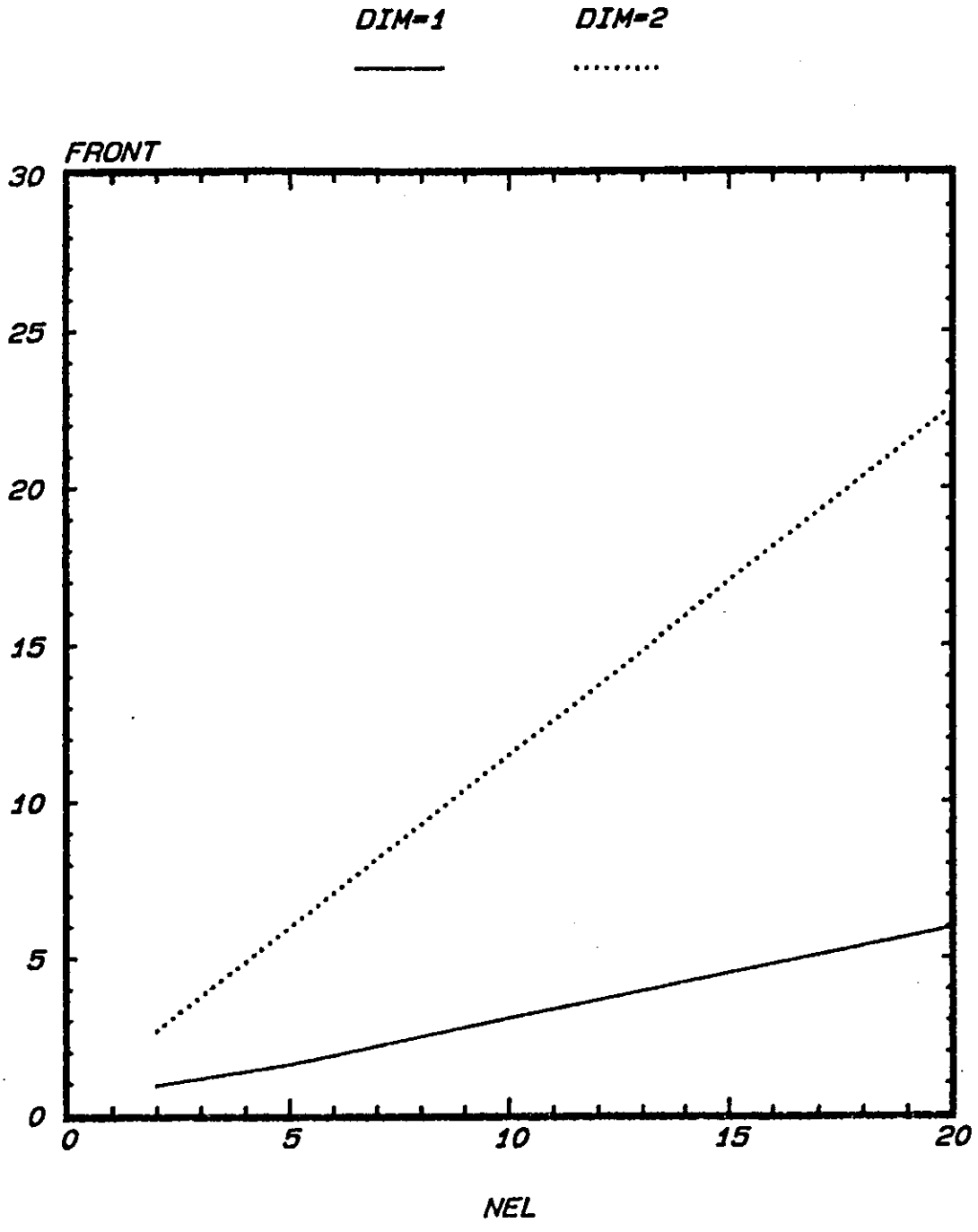


FIG. (4.33) EFFECT OF NEL ON FRONT TIME

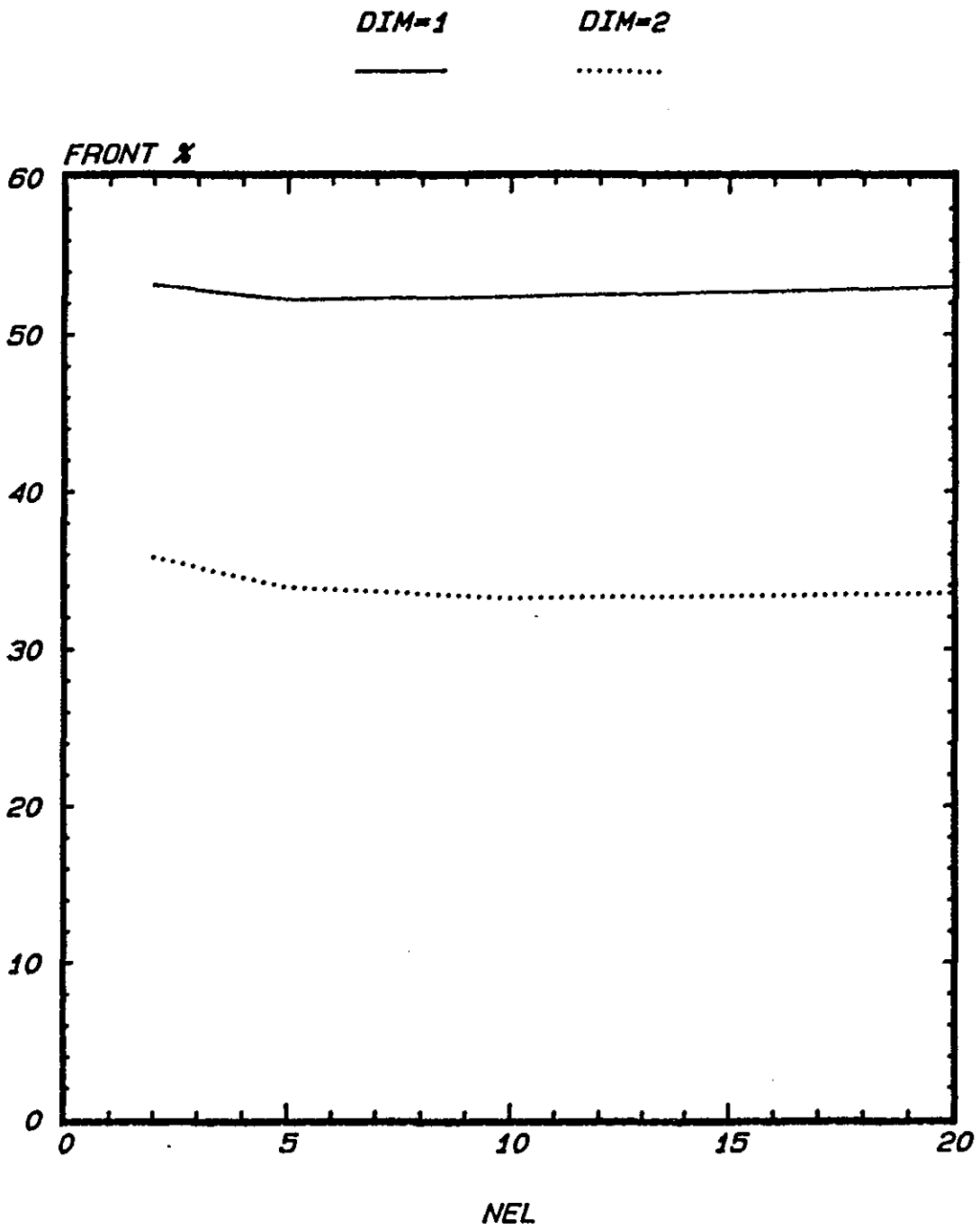


FIG. (4.34) EFFECT OF NEL ON FRONT % TIME

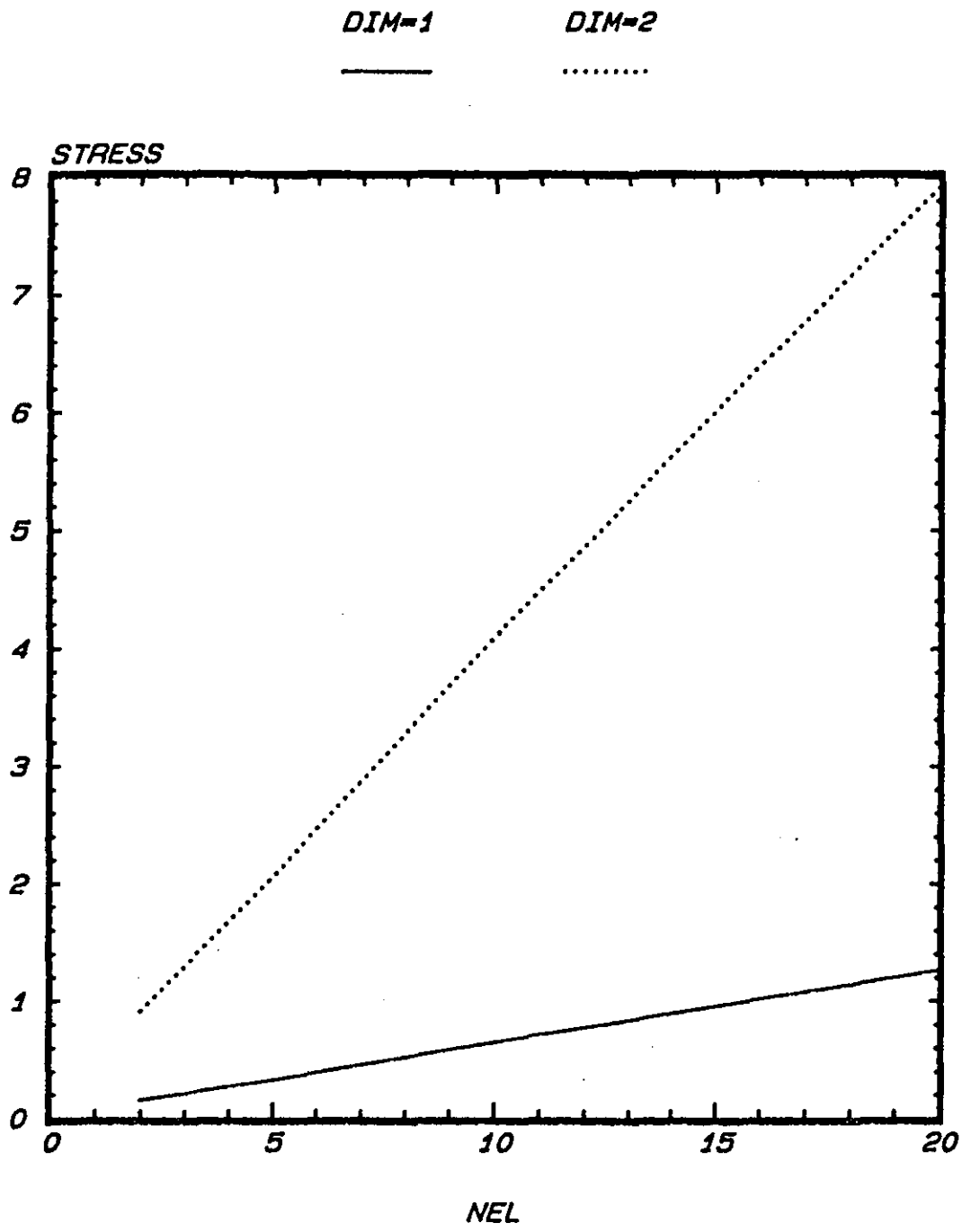


FIG. (4.35) EFFECT OF NEL ON STRESS TIME

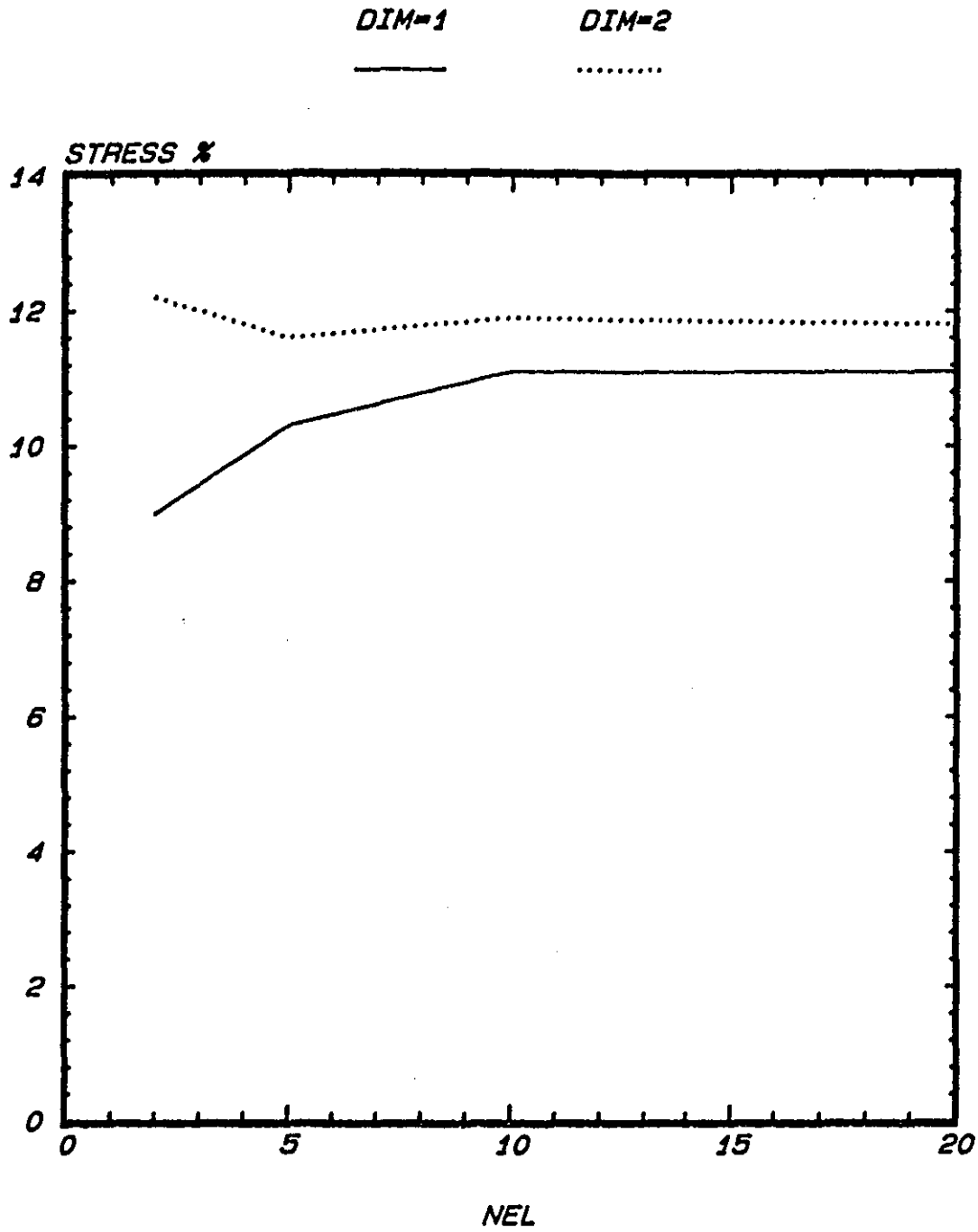


FIG. (4.36) EFFECT OF NEL ON STRESS % TIME

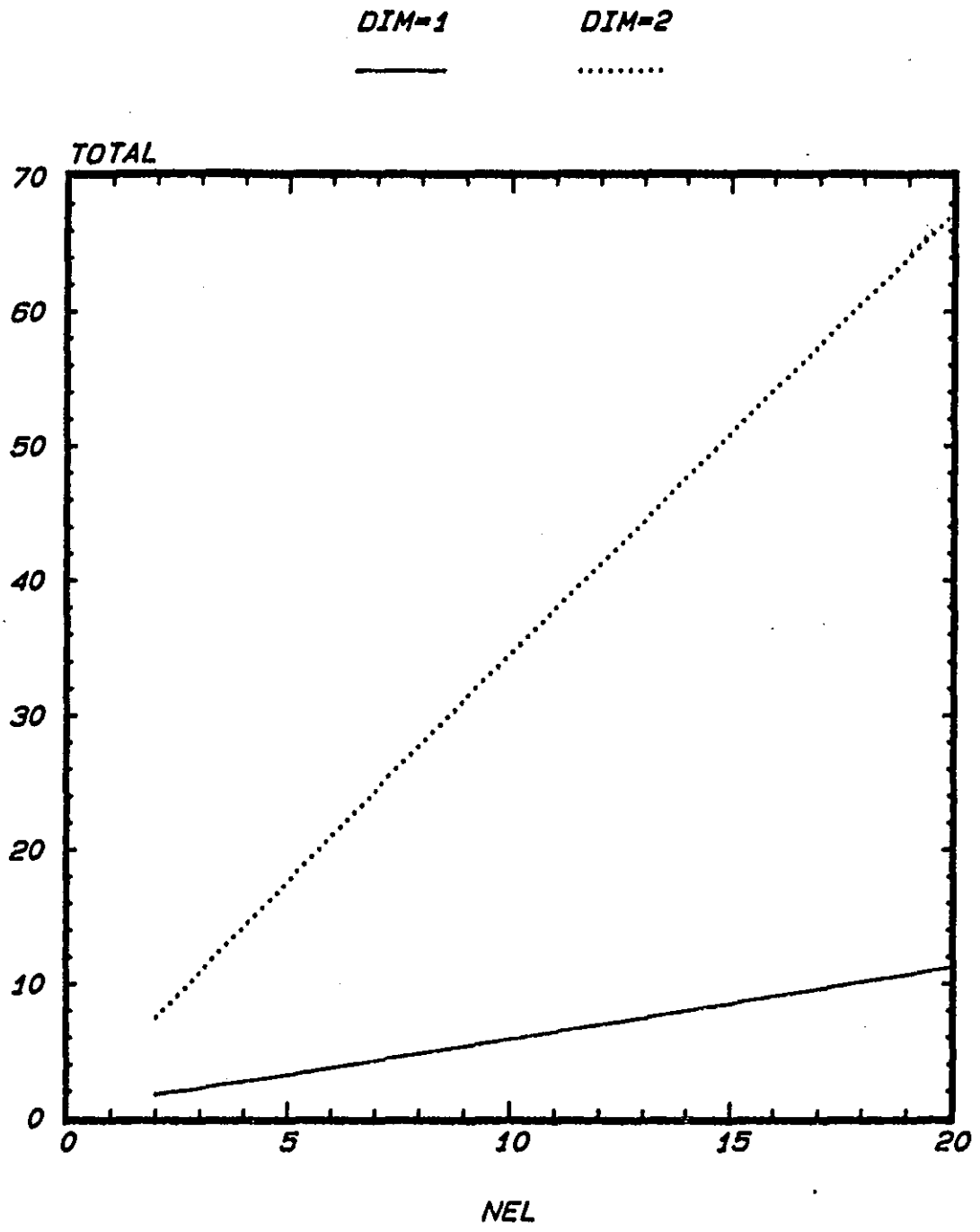


FIG. (4.37) EFFECT OF NEL ON TOTAL TIME

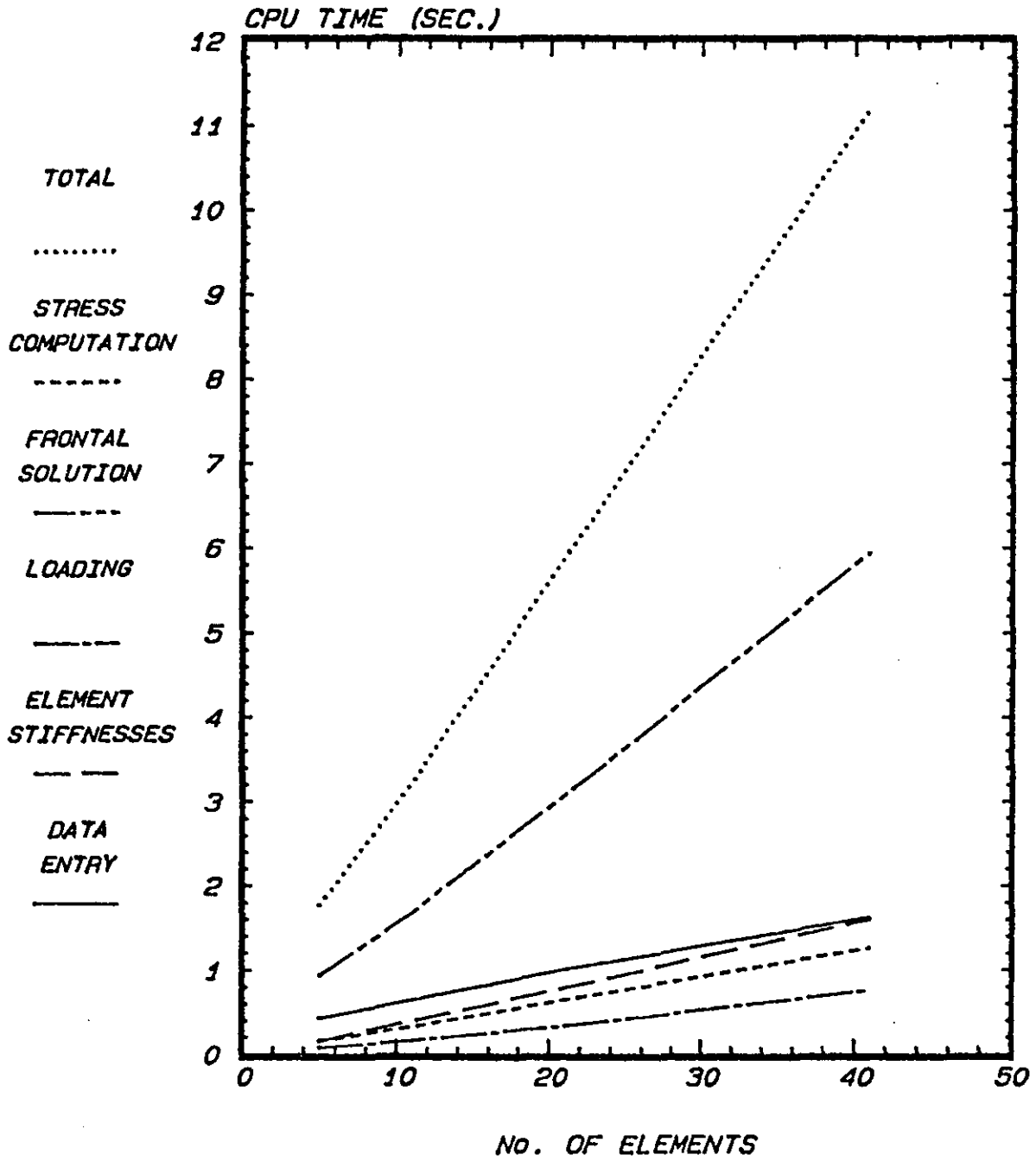


FIGURE 4.38: CPU TIME DISTRIBUTION FOR 1-D PROBLEMS.

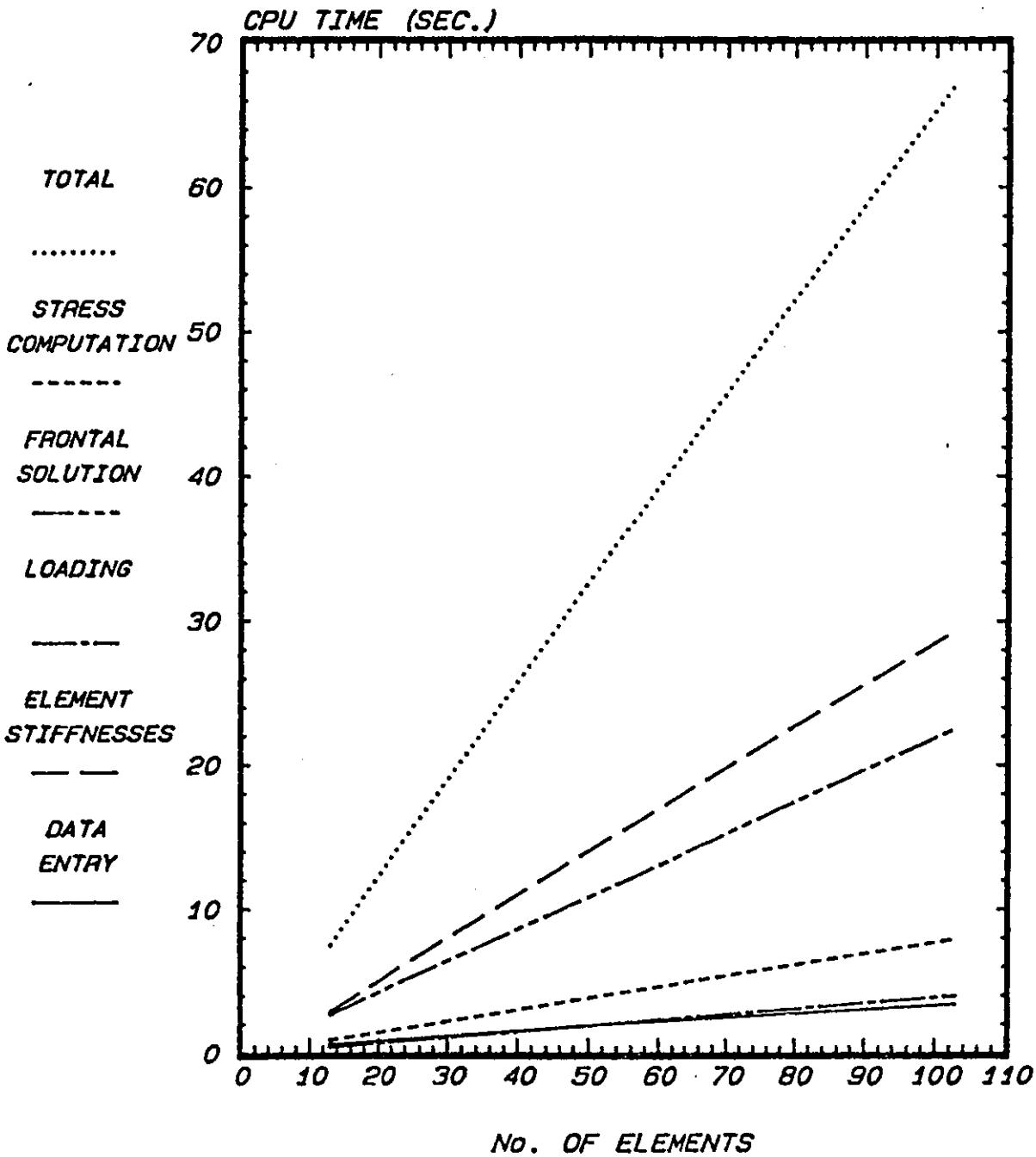


FIGURE 4.39: CPU TIME DISTRIBUTION FOR 2-D PROBLEMS.

Note that the following abbreviations are used:

NEL	number of elements
NNOD	number of nodes
DIM	number of dimensions: 1 or 2
DE	data entry CPU time
DE%	percentage of CPU time spent in data entry compared to total CPU time for the problem
STIFF	stiffness computation CPU time
STIFF%	percentage of CPU time spent in stiffness computation compared to total CPU time for the problem
LOAD	equivalent nodal loads computation CPU time
LOAD%	percentage of CPU time spent in load computation compared to total CPU time for the problem
FRONT	frontal solution CPU time
FRONT%	percentage of CPU time spent in frontal solution compared to total CPU time for the problem.
STRESS	stress computation CPU time
STRESS%	percentage of CPU time spent in stress computation compared to total CPU time for the problem
TOTAL	the total solution time for the problem

All times are in seconds.

A statistical analysis was performed using the well-known Statistical Analysis System - SAS [SAS, 1982] to correlate the CPU time in these problems to the other parameters.

Table 4.4 shows the correlation matrix between all the considered parameters. In this table, each cell (i,j) contains two numbers. The

upper one is the correlation coefficient between the two variables i and j. The lower number is the probability that this value of correlation is insignificant i.e. the null hypothesis is:

$$H_O: \ \rho_{i,j} = O \ , \tag{4.49}$$

where $\rho_{i,j}$ is the correlation coefficient between the two variables i and j.

The diagonal elements are, of course 1's for correlation coefficients $\rho_{i,j}$ and its associated probability is O. Examination of other elements in the correlation matrix shows that all correlation coefficients are positive and that all of these coefficients are highly significant.

This seems to be natural since the CPU time consumed in any phase of the solution phases will certainly increase as the number of elements or other parameters are increased. Perhaps the notable fact from this table is that the correlation coefficients with the number of nodes is bigger and more significant compared to the corresponding correlation coefficient with the number of elements.

On the other hand, Table (4.5) shows the correlation matrix between the number of nodes, number of elements and the percentage of the CPU. time spent in each step of the solution. Here about half of the correlation coefficients are -ve. However, considering only the significant correlation coefficients as those have a probability value $\lesssim$.10 we can conclude that:

(i) The percentage of time for data entry will decrease as the number of nodes are increased. The same is true with the percentage of CPU time for stiffness matrix computations.

(ii) The percentage of loading time is increasing with the increase in the number of elements. Here the correlation is much stronger with the number of elements.

(iii) The percentage of frontal time is increasing as the data entry time is increased. The same is true with the stiffness formulation and stress computation %.

Similar tables are done for each individual class of problems, namely: 1-D and 2-D. These results are shown in Tables (4.6) and (4.7) for 1-D problems and in Tables (4.8) and (4.9) for 2-D problems.

A regression analysis that relates the total CPU time in seconds to the main parameters shows that we can write:

$$T = .8966\ NNOD - 1.217\ NEL - .9306 \tag{4.50}$$

where:

T is the total CPU time

NNOD is the number of nodes

NEL is the number of elements.

When considering 1-D problems only, the correlation between CPU time and NEL can be:

$$T = .527\ NEL + .6524 \tag{4.51}$$

and with number of nodes:

$$T = .263\ NNOD + .3891 \tag{4.52}$$

In case of 2-D problems alone, the corresponding equations will be:

$$T = 3.3149\ NEL + 1.07296 \tag{4.53}$$

and

$$T = .663\ NNOD - .91599 \tag{4.54}$$

It is worth mentioning that an abstracted form of some of these results has been given in [Sharaf Eldin and Evans, 1987].

VARIABLE	N	MEAN	STD.DEV.	SUM	MINIMUM	MAXIMUM
NEL	8	9.25000000	7.30459738	74.00000000	2.00000000	20.00000000
NNOD	8	34.37500000	32.03987471	275.00000000	5.00000000	103.00000000
DE	8	1.36787500	0.98966047	10.94300000	0.43000000	3.41300000
STIFF	8	7.18575000	10.25263602	57.48600000	0.16000000	29.31400000
LOAD	8	1.10925000	1.33339585	8.87400000	0.08000000	4.00900000
FRONT	8	6.78050000	7.20259201	54.24400000	0.94000000	22.53600000
STRESS	8	2.17587500	2.64499692	17.40700000	0.16000000	7.92500000
TOTAL	8	18.63025000	22.29118928	149.04200000	1.77000000	67.19700000

TABLE 4.4: Correlation matrix of all parameters (1-D and 2-D)

	NEL	NNOD	DE	STIFF	LOAD	FRONT	STRESS	TOTAL
NEL NO. OF ELEMENTS	1.00000 0.00000	0.79795 0.0176	0.81689 0.0133	0.55127 0.1567	0.65127 0.0802	0.70236 0.0521	0.62398 0.0982	0.62943 0.0945
NNOD NO. OF NODES	0.79795 0.0176	1.00000 0.0000	0.99749 0.0001	0.94249 0.0005	0.97692 0.0001	0.98936 0.0001	0.96880 0.0001	0.97043 0.0001
DE DATA ENTRY TIME	0.81689 0.0133	0.99749 0.0001	1.0000 0.0000	0.92913 0.0008	0.96752 0.0001	0.98176 0.0001	0.95829 0.0002	0.96013 0.0002
STIFF STIFFNESS TIME	0.55127 0.1567	0.94249 0.0005	0.92913 0.0008	1.00000 0.0000	0.00202 0.0001	0.98098 0.0001	0.99587 0.0001	0.99529 0.0001
LOAD LOADING TIME	0.65127 0.0802	0.97692 0.0001	0.96752 0.0001	0.99202 0.0001	1.00000 0.0000	0.99750 0.0001	0.00028 0.0001	0.99952 0.0001
FRONT FRONTAL SOLUTION TIME	0.70236 0.0521	0.98936 0.0001	0.98176 0.0001	0.98098 0.0001	0.99750 0.0001	1.00000 0.0000	0.99443 0.0001	0.99515 0.0001
STRESS STRESS COMPUTATION TIME	0.62398 0.0982	0.96880 0.0001	0.95829 0.0002	0.99587 0.0001	0.99928 0.0001	0.99443 0.0001	1.00000 0.0000	0.99994 0.0001
TOTAL TOTAL CPU TIME	0.62943 0.0945	0.97043 0.0001	0.96013 0.0002	0.99529 0.0001	0.99952 0.0001	0.99515 0.0001	0.99994 0.0001	1.00000 0.0000

TABLE 4.4: Correlation coefficients - Prob > |R| Under HO:RHO=0 - N=8

VARIABLE	N	MEAN	STD.DEV.	SUM	MINIMUM	MAXIMUM
NEL	8	9.25000000	7.30459738	74.00000000	2.00000000	20.00000000
NNOD	8	34.37500000	32.03987471	275.00000000	5.00000000	103.00000000
DEP	8	12.73750000	7.34670723	101.90000000	5.10000000	24.40000000
SP	8	27.02500000	15.91986630	216.20000000	9.00000000	43.60000000
LP	8	5.68750000	0.67915389	45.50000000	4.50000000	6.80000000
FP	8	43.38750000	9.96242046	347.10000000	33.20000000	53.10000000
TP	8	11.12500000	1.04437268	89.00000000	9.00000000	12.20000000

TABLE 4.5: Correlation matrix of % of all parameters (1-D and 2-D)

	NEL	NNOD	DEP	SP	LP	FP	TP
NEL NO. OF ELEMENTS	1.00000 0.0000	0.79795 0.0176	-0.32310 0.4350	0.11148 0.7927	0.78110 0.0221	-0.03215 0.9398	0.31179 0.4522
NNOD NO. OF NODES	0.79795 0.0176	1.00000 0.0000	-0.65388 0.0786	0.57976 0.1320	0.54187 0.1654	-0.53427 0.1725	0.49791 0.2092
DEP DATA ENTRY %	-0.32310 0.4350	-0.65388 0.0786	1.00000 0.0000	-0.95651 0.0002	-0.57338 0.1373	0.92536 0.0010	-0.91805 0.0013
SP STIFFNESS TIME %	0.11148 0.7927	0.57976 0.1320	-0.95651 0.0002	1.00000 0.0000	0.31793 0.4428	-0.99555 0.0001	0.80797 0.0153
LP LOADING TIME %	0.78110 0.0221	0.54187 0.1654	-0.57338 0.1373	0.31793 0.4428	1.00000 0.0000	-0.22911 0.5852	0.65508 0.0779
FP FRONTAL SOLUTION TIME %	-0.03215 0.9398	-0.53427 0.1725	0.92536 0.0010	-0.99555 0.0001	-0.22911 0.5852	1.00000 0.0000	-0.76351 0.0275
TP STRESS COMPUTATION %	0.31179 0.4522	0.49791 0.2092	-0.91805 0.0013	0.80797 0.0153	0.65508 0.0779	-0.76351 0.0275	1.00000 0.0000

TABLE 4.5: Correlation coefficients - Prob > |R| Under HO:RHO=O - N=8

VARIABLE	N	MEAN	STD.DEV.	SUM	MINIMUM	MAXIMUM
NEL	4	9.25000000	7.88986692	37.00000000	2.00000000	20.00000000
NNOD	4	19.50000000	15.77973384	78.00000000	5.00000000	41.00000000
DE	4	0.93125000	0.52499230	3.72500000	0.43000000	1.63100000
STIFF	4	0.73875000	0.63336528	2.95500000	0.16000000	1.60400000
LOAD	4	0.33850000	0.30468618	1.35400000	0.08000000	0.76500000
FRONT	4	2.89200000	2.22473324	11.56800000	0.94000000	5.94700000
STRESS	4	0.60200000	0.48636406	2.40800000	0.16000000	1.26400000
TOTAL	4	5.52450000	4.15604255	22.09800000	1.77000000	11.21100000

TABLE 4.6: Correlation matrix of all parameters (1-D problems)

	NEL	NNOD	DE	STIFF	LOAD	FRONT	STRESS	TOTAL
NEL NO. OF ELEMENTS	1.00000 0.0000	1.00000 0.0000	0.99899 0.0010	0.99998 0.0001	0.99774 0.0023	0.99938 0.0006	0.99991 0.0001	0.99991 0.0001
NNOD NO. OF NODES	1.00000 0.0000	1.00000 0.0000	0.99899 0.0010	0.00008 0.0001	0.99774 0.0023	0.99938 0.0006	0.99991 0.0001	0.99991 0.0001
DE DATA ENTRY TIME	0.99899 0.0010	0.99899 0.0010	1.00000 0.0000	0.99871 0.0013	0.99381 0.0062	0.99751 0.0025	0.99897 0.0010	0.99847 0.0015
STIFF STIFFNESS TIME	0.99998 0.0001	0.99998 0.0001	0.99871 0.0013	1.00000 0.0000	0.99811 0.0019	0.99949 0.0005	0.99989 0.0001	0.99996 0.0001
LOAD LOADING TIME	0.99774 0.0023	0.99774 0.0023	0.99381 0.0062	0.99811 0.0019	1.00000 0.0000	0.99889 0.0011	0.99778 0.0022	0.99843 0.0016
FRONT FRONTAL SOLUTION TIME	0.99938 0.0006	0.99938 0.0006	0.99751 0.0025	0.99949 0.0005	0.99889 0.0011	1.00000 0.0000	0.99965 0.0003	0.99974 0.0003
STRESS STRESS COMPUTATION TIME	0.99991 0.0001	0.99991 0.0001	0.99897 0.0010	0.99989 0.0001	0.99778 0.0022	0.99965 0.0003	1.00000 0.0000	0.99993 0.0001
TOTAL TOTAL CPU TIME	0.99991 0.0001	0.99991 0.0001	0.99847 0.0015	0.99996 0.0001	0.99843 0.0016	0.99974 0.0003	0.99993 0.0001	1.00000 0.0000

TABLE 4.6: Correlation coefficients - Prob. > |R| Under HO:RHO=0 - N=4

VARIABLE	N	MEAN	STD.DEV.	SUM	MINIMUM	MAXIMUM
NEL	4	9.25000000	7.88986692	37.00000000	2.00000000	20.00000000
NNOD	4	19.50000000	15.77973384	78.00000000	5.00000000	41.00000000
DEP	4	19.05000000	4.23674403	76.20000000	14.50000000	24.40000000
SP	4	12.27500000	2.33291663	49.10000000	9.00000000	14.30000000
LP	4	5.55000000	0.98826447	22.20000000	4.50000000	6.80000000
FP	4	52.67500000	0.44253060	210.70000000	52.20000000	53.10000000
TP	4	10.37500000	0.99121138	41.5000000	9.00000000	11.10000000

TABLE 4.7: Correlation matrix of % of all parameters (1-D problems)

	NEL	NNOD	DEP	SP	LP	FP	TP
NEL NO. OF ELEMENTS	1.00000 0.0000	1.00000 0.0000	-0.93287 0.0671	0.83893 0.1611	0.98966 0.0103	0.21242 0.7876	0.79385 0.2061
NNOD NO. OF NODES	1.00000 0.0000	1.00000 0.0000	-0.92387 0.0671	0.83893 0.1611	0.98966 0.0103	0.21242 0.7876	0.79385 0.2061
DEP DATA ENTRY %	-0.93287 0.0671	-0.93287 0.0671	1.00000 0.0000	-0.97717 0.0228	-0.97444 0.0256	0.14312 0.8569	-0.95845 0.0416
SP STIFFNESS TIME %	0.83893 0.1611	0.83893 0.1611	-0.97717 0.0228	1.00000 0.0000	0.90579 0.0942	-0.34951 0.6505	0.98851 0.0115
LP LOADING TIME %	0.98966 0.0103	0.98966 0.0103	-0.97444 0.0256	0.90579 0.0942	1.00000 0.0000	0.08003 0.9200	0.87282 0.1272
FP FRONTAL SOLUTION TIME %	0.21242 0.7876	0.21242 0.7876	0.14312 0.8569	-0.34951 0.6505	0.08003 0.9200	1.00000 0.0000	-0.38946 0.6105
TP STRESS COMPUTATION %	0.79385 0.2061	0.79385 0.2061	-0.95845 0.0416	0.98851 0.0115	0.87282 0.1272	-0.38946 0.6105	1.00000 0.0000

TABLE 4.7: Correlation coefficients - Prob > |R| Under HO:RHO=0 - N=4

VARIABLE	N	MEAN	STD.DEV.	SUM	MINIMUM	MAXIMUM
NEL	4	9.25000000	7.88986692	37.00000000	2.00000000	20.00000000
NNOD	4	49.25000000	39.44933460	197.00000000	13.00000000	103.00000000
DE	4	1.80450000	1.22529085	7.21800000	0.61500000	3.41300000
STIFF	4	13.63275000	11.57730381	54.53100000	2.87000000	29.31400000
LOAD	4	1.88000000	1.57211810	7.52000000	0.41400000	4.00900000
FRONT	4	10.66900000	8.70497080	42.67600000	2.68300000	22.53600000
STRESS	4	3.74975000	3.07927214	14.99900000	0.91500000	7.92500000
TOTAL	4	31.73600000	26.15579160	126.94400000	7.49700000	67.19700000

TABLE 4.8: Correlation matrix of all parameters (2-D problems)

	NEL	NNOD	DE	STIFF	LOAD	FRONT	STRESS	TOTAL
NEL NO. OF ELEMENTS	1.00000 0.00000	1.00000 0.00000	0.99129 0.0027	0.99992 0.0001	0.99991 0.0001	1.00000 0.0001	0.99992 0.0001	0.99994 0.0001
NNOD NO. OF NODES	1.00000 0.0000	1.00000 0.0000	0.99729 0.0027	0.99992 0.0001	0.99991 0.0001	1.00000 0.0001	0.99992 0.0001	0.99994 0.0001
DE DATA ENTRY TIME	0.99729 0.0027	0.99729 0.0027	1.00000 0.0000	0.99813 0.0019	0.99813 0.0019	0.99720 0.0028	0.99800 0.0020	0.99802 0.0020
STIFF STIFFNESS TIME	0.99992 0.0001	0.99992 0.0001	0.99813 0.0019	1.00000 0.0000	1.00000 0.0001	0.99990 0.0001	0.99996 0.0001	1.00000 0.0001
LOAD LOADING TIME	0.99991 0.0001	0.99991 0.0001	0.99813 0.0019	1.00000 0.0001	1.00000 0.0000	0.99989 0.0001	0.99993 0.0001	0.99999 0.0001
FRONT FRONTAL SOLUTION TIME	1.00000 0.0001	1.00000 0.0001	0.99720 0.0028	0.99990 0.0001	0.99989 0.0001	1.00000 0.0000	0.99991 0.0001	0.99993 0.0001
STRESS STRESS COMPUTATION TIME	0.99992 0.0001	0.99992 0.0001	0.99800 0.0020	0.99996 0.0001	0.99993 0.0001	0.99991 0.0001	1.00000 0.0000	0.99998 0.0001
TOTAL TOTAL CPU TIME	0.99994 0.0001	0.99994 0.0001	0.99802 0.0020	1.00000 0.0001	0.99999 0.0001	0.99993 0.0001	0.99998 0.0001	1.00000 0.0000

TABLE 4.8: Correlation coefficients - Prob > |R| Under HO:RHO=0 - N=4

VARIABLE	N	MEAN	STD.DEV.	SUM	MINIMUM	MAXIMUM
NEL	4	9.25000000	7.88986692	37.00000000	2.00000000	20.00000000
NNOD	4	49.25000000	39.44933460	197.00000000	13.00000000	103.00000000
DEP	4	6.42500000	1.31497782	25.70000000	5.10000000	8.20000000
SP	4	41.77500000	2.39913179	167.10000000	38.30000000	43.60000000
LP	4	5.82500000	0.22173558	23.30000000	5.50000000	6.00000000
FP	4	34.10000000	1.16904519	136.40000000	33.20000000	35.80000000
TP	4	11.87500000	0.25000000	47.50000000	11.60000000	12.20000000

TABLE 4.9: Correlation matrix of % of all parameters (2-D problems)

	NEL	NNOD	DEP	SP	LP	FP	TP
NEL NO. OF ELEMENTS	1.00000 0.0000	1.00000 0.0000	-0.89398 0.1060	0.78232 0.2177	0.75737 0.2426	-0.67219 0.3278	-0.33376 0.6662
NNOD NO. OF NODES	1.00000 0.0000	1.00000 0.0000	-0.89398 0.1060	0.78232 0.2177	0.75737 0.2426	-0.67219 0.3278	-0.33376 0.6662
DEP DATA ENTRY %	-0.89398 0.1060	-0.89398 0.1060	1.00000 0.0000	-0.97814 0.0219	-0.96315 0.0368	0.92588 0.0741	0.65147 0.3485
SP STIFFNESS TIME %	0.78232 0.2177	0.78232 0.2177	-0.97814 0.0219	1.00000 0.0000	0.98533 -0.0147	-0.98169 0.0183	-0.73499 0.2650
LP LOADING TIME %	0.75737 0.2426	0.75737 0.2426	-0.96315 0.0368	0.98533 0.0147	1.00000 0.0000	-0.95158 0.0484	-0.82681 0.1732
FP FRONTAL SOLUTION TIME %	-0.67219 0.3278	-0.67219 0.3278	0.92588 0.0741	-0.98169 0.0183	-0.95158 0.0484	1.00000 0.0000	0.71854 0.2815
TP STRESS COMPUTATION %	-0.33376 0.6662	-0.33376 0.6662	0.65147 0.3485	-0.73499 0.2650	-0.82681 0.1732	0.71854 0.2815	1.00000 0.0000

TABLE 4.9: Correlation coefficients - Prob > |R| Under HO:RHO=0 - N=4

Although this package is of limited capabilities compared to the more advanced and complicated packages like NASTRAN or MSAP, it proved to be a cost effective solution for small to medium size FE problems which lie within the domain of applications covered by it. The development of the time log module in this package makes it suitable for research purposes in addition to the solution of practical structural mechanics problems.

4.7.3 The STRAP Program

This program was developed by the author as part of a joint project between the Civil Engineering Department and the Computer Centre at the College of Engineering, King Saud University [Turaby and Sharaf Eldin, 1978]. The aim is to solve classical skeletal structures such as continuous beams, frames and trusses. STRAP is a fully interactive program, the user need only "sketch" the problem on the screen of the terminal in the same way he would sketch it on paper. In addition, the output may also be plotted on a small x-y graph plotter together with the printed results. The system is also offered in batch mode. Although this program is designed to run on a mini-computer, it is readily available to run on a micro-computer. The presented program is for the solution of beam elements only. Despite these fairly simple elements, the main advantage of this program is the ease of data input. The user of this program is relieved from the burden of preparing his problem in the usual manner, i.e. numbering of nodes, elements, etc. A versatile plotting program is added to STRAP as a postprocessor which gives additional facilities to STRAP. The total memory required to run

STRAP is only 26K bytes. A disk file is used as a temporary storage to hold the values of bending moments and shearing forces at different points. The size of this work file is computed by STRAP.

4.7.3.1 STRAP Capabilities

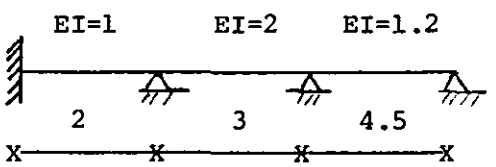
The major capabilities of the system can be summarized in the following:

1. Sketching

A continuous beam can be sketched on the alphanumeric screen using the standard keyboard with the following symbols:

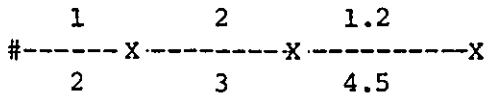
- # stands for fixed end where both deflection and rotation are prohibited.
- x stands for hinged supports where deflection is prohibited while rotation is permissible.
- ---- stands for beam centre line (spans).
- * stands for points of change of rigidity.

Thus a structure like this (Figure 4.40)



(a)

is sketched as:



(b)

FIGURE 4.40: Sketching facility in STRAP
(a) Original structure (b) Sketched model.

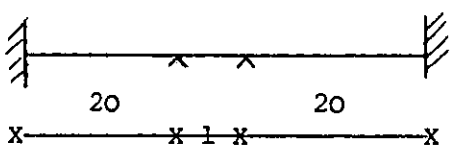
The values of rigidities are typed above each span, while the span length is typed under the centre line of the beam. There is no need for scaling or proportioning the input.

2. Interaction and Batches

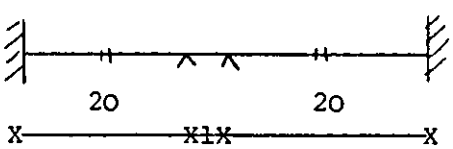
STRAP supports both interactive terminals and batch jobs. The system is designed so that the input could be done through the keyboard of a terminal by sketching or by preparing data in a batch file. Outputs can be obtained on the screen, printed and plotted.

3. Plotting

All the results together with the given structure itself can be plotted. STRAP checks the available size of plotting table together with the geometry of the structure. If "nice" readable graphs can be produced in the available plotter space, STRAP plots the graphs, otherwise the results are printed only. To illustrate this, consider the structure shown in Figure 4.41. The middle span is too short compared to the other spans. If the available plotting table is only 30×20 cms, then the middle span will be represented by less than 1 cm. which is not enough to dimension it. One solution to this situation is to plot parts of the long spans only.



(a) Original structure



(b) Not-to-scale plotting

FIGURE 4.41: A non-proportional structure
(a) Original structure. Plotting to scale is impossible.
(b) Not-to-scale plotting.

The following graphs can be produced:

(i) The input problem: geometry and loading

(ii) The bending moment diagram (B.M.D.) over the whole structure, stating the maximum and minimum values.

(iii) The shearing forces diagram (S.F.D.) over the whole structure, stating the maximum and minimum values.

(iv) Influence lines for bending moments and shearing forces at any section.

Special Problem Handling

STRAP is capable of handling some special problems in structural analysis. These are:

(1) Influence Lines

Influence lines (I.L.) at different sections for bending moments, shearing forces and reactions can be obtained. In the interactive version, the computer displays the structure sketch for the user. Then, the user marks the letter 'M' or 'Q' on the section for which the B.M.I.L. or S.F.I.L. is required.

(2) Members of Variable Moment of Inertia

In the case of a member of variable rigidity, it is split into several members of constant rigidity. This is done automatically by STRAP by placing an * at the points of variations.

(3) Settlements or Rotations of Supports

STRAP can handle supports with given settlements or rotations, employing the equivalent joint loads in the same way as external loads.

4.7.3.2 STRAP Structure

STRAP has a modular structure. It is coded in Fortran IV. The general routines for STRAP are:

(1) MSTFM: Formation of member stiffness matrix.

(2) RAMAP: Computation of the function which splits the overall structure stiffness matrix into submatrices corresponding to known and unknown joint displacements.

(3) SMSJ: Mapping the member stiffness matrix into the overall one.

(4) EJLDP: Computation of the equivalent nodal loads for a concentrated load acting at any distance on a member.

(5) EJLDW: Computation of the equivalent nodal loads for a distributed load of constant intensity running over part of the member.

(6) EJLDM: Computation of the equivalent nodal loads for a moment acting at any distance on a member.

(7) EJLDS: Computation of the equivalent nodal loads due to a given rotation or settlements at any support.

(8) AMAJ: Mapping the member action vector into the overall one.

(9) SJSJ: Re-arranging the overall stiffness matrix using the function computed by "RAMAP" in place.

(10) MTCPY: Copying a sub-matrix of a larger one.

(11) ZEROS: Zeroizing a sub-matrix of a larger one.

(12) MTADD: Adding two matrices.

(13) MTSUB: Subtracting two matrices.

(14) MATTRN: Transposition of a matrix in place.

(15) MTINV: Inversion of a sub-matrix in place.

(16) AMXIM: Finding the maximum value of a vector.

(17) AMNIM: Finding the minimum value of a vector.

(18) BMSF: Computation of B.M. and S.F. at any point of a member due to a given load.

(19) CLPLT: Plotting the centre line of the structure.

(20) SPPLT: Plotting a support on a centre line previously plotted by "CLPLT".

(21) LDWLT: Plotting a uniform load on a centre line previously plotted by "CLPLT".

(22) LDPLT: Plotting a concentrated load on a centre line previously plotted by "CLPLT".

(23) LDMLT: Plotting a moment load on a centre line, previously plotted by "CLPLT".

(24) LDDLT: Plotting B.M.D. and S.F.D. on a centre line, previously plotted by "CLPLT".

The user interface with STRAP is realized through the sketch analyzer routine in the interactive mode, while in the batch mode it is through a job processor routine.

4.8 MICROCOMPUTER IMPLEMENTATION

4.8.1 Background

Technological advances in the last few years especially in the large scale integration (LSI) and the very large scale integration (VLSI) has made it possible to have microcomputers with increasing power. The early microcomputers were built around 8-bits microprocessors (e.g. PDP 8). They had very limited capabilities that would not suit FE analysis. However, with the introduction of the 16 bit microprocessors more powerful microcomputers could be used for some of the FE analysis. With the more powerful new 32 bit micros, it seems that most of the classical FE analysis for small to medium size problems can be handled by such machines.

However, many problems should be resolved to have a successful FE implementation on a microcomputer. In what follows the limitations and problems encountered when using such systems for FE software are discussed with proposed solutions. The first published FE software on a microcomputer seems to be that quoted by Yamada et al [1980]. It is based on an 8-bit microprocessor. Its main use was for educational and experimental purposes rather than practical analysis. A more general implementation is SAP-80 by Wilson [1985] which runs on an IBM/PC. This program can handle medium size FE analysis for linear and some nonlinear structures. Blakely et al [1985] introduced their MSC/PAL program for relatively small size FE problems of about 300 nodes. The main feature of this program is its graphical package which could be used to display the FE model (preprocessing) and the deformed shape of the structure (post-processing). An interactive FE program for aquifer simulation which is developed by the author is also presented.

4.8.2 Finite Element Programming on Microcomputers

Problems and Solutions

There are several problems which should be solved to have a successful FE system on a microcomputer. It is possible to separate these problems into two main categories: those related to hardware and those related to software. Perhaps the first problem is the slow execution speed of a microcomputer. For a microcomputer system such as the IBM-PC, the execution time of a moderate size FE problem is measured in hours. To illustrate consider the sample problem quoted by Wilson [1985] which is a static analysis of a cooling tower which has 434 joints and 400 shell elements with a total number of unknowns of 2600. The solution time for this problem is reported to be 3 hours. On the relatively faster microcomputer IBM-PC/XT another problem was solved by Griffin et al [1983] for the solution of a non-linear system. The reported time is 46 hours.

One solution to the problem of the slow speed is the use of a co-processor for the floating point arithmetic. However, it seems that in the near future the promising solution can be in connecting some of those micros in a network and to have a FE software that utilizes the distributed power of these micros and synchronize their operations to solve fairly big FE problems in a reasonable time. Another problem of microcomputers is their limited main memory. An associated problem is also the limited address space. This problem with a proposed solution are explained in details in Chapter 5 of this thesis. A third problem is the peripherals that can be supported by a microcomputer. The size of disc drives that are supported is limited. The introduction of hard Winchester disc drives is one solution though the mean access

time is considerably more than those supported by bigger computers. The low speed of printers and plotters can be enhanced by increasing the buffer size to these peripherals. Another problem is the bus used in microcomputers. By the time this thesis is written, the most popular bus is the S-100 bus which is the IEEE-696 standard. One of the major drawbacks with this bus is that it cannot support full 32-bit operations.

The other category of problems is that related to software. Two major problems are considered. First, the Fortran supported on microcomputers and second, the available software libraries that can be found on these machines. To exemplify, most of the versions of Fortran run on microcomputers allow smaller range of integers than on a mainframe. This restriction is usually due to the word length. For a 16-bit microcomputer the maximum integer that can be represented is limited to 32767 only. Another problem is the depth of nesting. For example, some Fortran compilers will allow a nesting depth of five only. Thus, if an arithmetic expression, for example, contains more than five levels of parentheses, temporary storage locations must be used to lower the nesting depth. Another potential problem is the limited address space. Some microcomputers allow an address space of only 32767 words though the physical memory can be more than that. To overcome this obstacle it is possible to segment the data space into different data blocks each of which is less than the allowed maximum. However, some times it is not possible to split the data space as the case of the master matrix in FE programs. A complete solution to this problem is given in Chapter 5 of this thesis. Finally, the available software support on micros is still limited compared to that available for bigger machines.

For example, very limited mathematical and graphical libraries are available on these systems.

Another approach that can be adopted is to use a bigger computer in developing the FE program or use a code already developed on a bigger machine and move this code to the microcomputer for execution. However, one must realize the problems encountered in this process. To physically move a program from a bigger computer to a microcomputer we have to connect both computers through a communication line and have an emulator with file transfer capability to move the source code from the bigger to the smaller computer. The source code usually must be modified. Although Fortran has its standards like ANSI Fortran 77 (ANSI X3.9-1978) but most of the available compilers do not 100% adhere to these standards. Before doing the actual file transfer from the bigger computer to a microcomputer it is wise to check the portability of the code to be transferred and do most of the necessary modifications while the code is still in the bigger computer. The reason for that is simply because editing is much easier on bigger computers as compared to microcomputer editing. Here are the most important things to note with the solutions:

(1) Statement length:

A Fortran statement can span over several lines. The first one is the primary line while the others are continuation lines. In standard Fortran 19 continuation lines are allowed, i.e. statement length can be up to 20 lines. Most of the microcomputer Fortran compilers can allow for only 4 or 5 continuation lines. Thus before doing a file transfer, statements longer than that must be split to multiple shorter statements. Practically, it is not common to find Fortran statements

longer than 5 lines except in Format statements.

(2) Variable and subprogram names:

Names in Fortran are limited to 6 or 7 characters only. However, most of the available compilers, even on some microcomputers will allow more. If the target machine compiler allows only 6 characters as a maximum length of a variable, subprogram or common block name, then a change must be done to conform to this specification. Change can usually be done by issuing a single Edit command like:

Change "STIFFNESS" to "STIFF" IN ALL

which will change the characters STIFFNESS to STIFF in the whole program. However, care must be taken to ensure that erroneous situations will not arise like if we have the string STIFFNESS in a format statement or if the variable name is split on two lines.

(3) Data types:

Data types must match in calling subprograms and in comparison operations. For example, if two character variables are to be compared, the result of a comparison may be different on different computers. This problem is particularly noticed when comparing Arabic text or a language other than English where the collating sequence is not the same on different computers. These types of problem are usually discovered at a later stage at compilation or even at execution time on the target computer.

(4) Subroutines and functions:

When using dynamic dimensionality in subprograms, no arithmetic expressions should be used. For example:

```
SUBROUTINE ADD(A,B,C,N,M)
DIMENSION A(N,M)B(N+M,M),C(N*M)
```

should be avoided and the following code is followed:

```
SUBROUTINE ADD(A,B,C,N,M,NPLUSM,NM)
DIMENSION A(N,M)B(NPLUSM,M),C(NM)
```

where the variable NPLUSM is set to N+M in the calling segment and the variable NM is set to N*M. Entry statements are usually handled differently in different compilers and it is a good practice, therefore, to avoid them. A subprogram with multiple entry points can be substituted either by multiple subprograms or by adding a switching variable in the list of arguments that branches to the appropriate entry point.

(5) Common Blocks:

Mixing character and non-character variables in the same labelled common block may cause problems in many microcomputer compilers. It is a good practice in such systems to have character common variables in a separate common block. Length of named common blocks must be the same in each program unit. Failing to do that may result in compliation or linkage error.

(6) Format statements:

Non-standard format descriptions should be avoided. It is worthwhile to note that most Fortran compilers do not fully check the details of the format descriptors until execution time when a formatter routine is invoked to execute formatted read and/or write statements.

(7) System calls:

Sometimes it is desirable to call some of the service routines provided by the operating system. Examples are: calls to get CPU time, calendar, date and time, file handling procedure, etc. Since these calls are dependent on the operating system they must be fully documented and when moving a code containing these calls to another

machine they must be substituted by the equivalent calls.

(8) Special hardware devices:

Some input/output devices are handled in different versions of Fortran differently. Also, some Fortran compilers cannot support some devices. Usually in these cases a routine in another programming language; usually assembly, is written and the required Fortran interface is established. As an example, consider the use of tablets, light pens, mouse and soft keys of a terminal. All such devices are not accessible directly through standard Fortran but rather they require a particular sequence of instructions that differ from machine to machine. In these cases, it is a very good practice to isolate these parts in a few isolated routines and new routines are developed on the target computer to do the same function.

4.8.3 The Interactive Finite Element Program for Aquifer Simulation - IFEP

This interactive program is developed by the author [Sharaf Eldin, 1983a] for aquifer simulation. Although the program was originally developed and tested on a mini-computer HP3000, yet its small size and modularity makes it very adequate for microcomputer implementation [Sharaf Eldin, 1985a]. The considered problem is the steady two-dimensional flow in a confined aquifer. The aquifer may be anisotropic and non-homogeneous. The governing partial differential equation is [Bear, 1979]:

$$\frac{\partial}{\partial x}\left(-T_{xx}\frac{\partial h}{\partial x}\right) + \frac{\partial}{\partial y}\left(-T_{yy}\frac{\partial h}{\partial y}\right) = Q + S\frac{\partial h}{\partial t} \quad , \tag{4.55}$$

where T_{xx} and T_{yy} are the components of the acquifer transmissivity in the x and y directions, respectively [L^2/T];

h is the piezometric head [L];

S is the aquifer storage coefficient;

Q is the volumetric flux of discharge or recharge per unit surface area of the aquifer [L/T]; and

t is the time [T].

In the case of steady flow this equation becomes:

$$\frac{\partial}{\partial x}(-T_{xx}\frac{\partial h}{\partial x}) + \frac{\partial}{\partial y}(-T_{yy}\frac{\partial h}{\partial y}) = Q \qquad (4.56)$$

Three types of boundary conditions can be specified:

(a) Dirichlet condition (also called, essential, 1st. type, or geometric) where the head is known on some boundary, i.e. $h=h_o(x,y)$ on S of the boundary.

(b) Neumann condition (also called force, 2nd. type, or natural) where the flow across some boundary is known, i.e.,

$$q_o(x,y) = -(T_{xx}\frac{\partial h}{\partial x}n_x + T_{yy}\frac{\partial h}{\partial y}n_y)$$

on ∂S of the boundary where n is the unit vector normal to the boundary.

(c) Mixed conditions of both types (also called 3rd type). The FE formulation of this problem will be detailed in Chapter 7 of this thesis and therefore will not be repeated here but the major aspects of the computer implementation of this problem will be given.

When designing this program the following objectives are considered:

(i) Simplicity:

- Friendly user interface
- Interactive ask-and-answer mode of operation

- Contains a pre- and a post-processor
- Oriented towards microcomputers of small size

(ii) Portability:

- Programmed in ANSI Fortran
- No special hardware or software is required to run the program. However, the use of a small x-y plotter is necessary to get a hard copy of plots.
- Minimal use of external storage (diskettes) for the larger problems. The smaller problems can be processed in the computer memory.

(iii) Maintainability:

- Structured programming
- Hierarchy input, process, output (HIPO)
- Self documented code

(iv) Accuracy:

- Use of proved techniques for the solution of the resulting equations.
- On-line validation of the input data

(v) Efficiency:

- Utilization of the banded nature of the master matrix.
- Resolution for new well parameters is due without re-computation of the master matrix.

4.8.3.1 Program Structure

The program contains three major parts: the input, the process and the output modules. A brief description of each is given below:

(1) The Input Module:

This module is responsible for reading all the data needed for the problem like, problem title, job specifications, number of elements, number of nodes, number of different soil zones, number of known heads and number of known boundary flows. Geometry specifications are either given in detail one by one or prepared by the pre-processor if the domain is a rectangle. However, the (x,y) values of all the nodal points, element-material map for all elements and soil properties must be given. In addition to that, the boundary conditions must also be given for both types I and II. In case of type I conditions, the head value of the discharge or recharge and the element side number must be given. The sources and sinks data are given interactively. All the inputs are echoed in the final printout for checking and documentation purposes. All inputs are also validated for type and range errors. Some logical errors can be detected by the program.

(2) The Process Modules:

There are three modules:

(i) Checking that the problem can be solved within the current program capacity. It computes the maximum bandwidth of the master matrix. This value is a direct function of the number of nodes and their numbering sequence. In fact, the semibandwidth is the maximum value between any two node numbers in the same element. If the data was prepared by the pre-processor, the semibandwidth will be automatically minimized. However, to keep the program as simple as possible no inherent re-numbering is done if the data was given manually.

(ii) Setting up the master matrix in a compact storage mode. Since

the master matrix is known to be symmetric and banded, its storage mode is as well.

(iii) Solution of equation using a banded Gauss elimination method.

(3) The Output Module:

This module prints all the computed and prescribed nodal heads and calculates the pressure heads as well. It is also possible to use the post-processor to plot the FE model.

Figure (4.42) shows a block flowchart for the IFEP program.

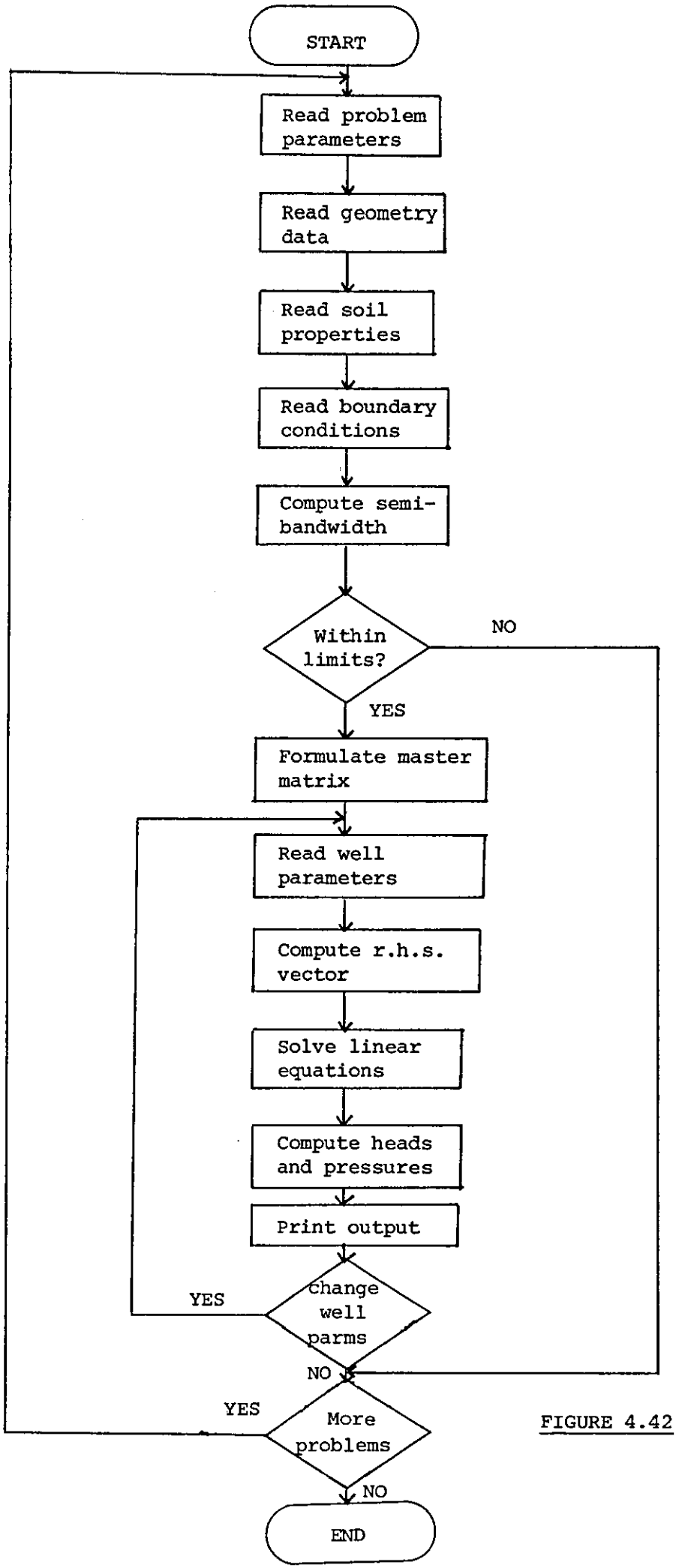


FIGURE 4.42: IFEP Block flowchart

4.9 PRE-PROCESSORS FOR FINITE ELEMENT PROGRAMS

4.9.1 Introduction

The term pre-processor has no definite definition when used in computer studies. This term is being used with different meanings in different fields of computer studies. When used in Finite Element programs it means usually a subprogram, a program or a suite of programs that are used to aid in data preparation which is required for a F.E. program. When used in digital image processing applications, pre-processing means filtering of digital pictures, normalizing grey levels and/or thinning to skeletonize objects in a picture prior to recognition which is the main processor in this field of computer studies. However, since the main processor of different applications are not the same, a reasonable general definition of a pre-processor may be: it is a part of the programming system used to do preliminary tasks to relieve the main processor and to achieve a better global efficiency for the whole process. The main reasons to split these tasks from the main processor, in my opinion, is that these tasks could be implemented on another computer rather than that used for the main processor. Usually this machine is a smaller one but usually with a better man-machine interface. For example, it is possible to have a pre-processor to automate FE mesh generation and then use this data as an input to a formatter which uses it to create the input data set for a FE processor like ASKA or MSAP. Of course, the formatter must support as many formats for the different FE systems it supports. Another reason may be to have the preprocessing functions independent of the processor itself. For example, it is possible to have a general preprocessing which is used mainly for solid modelling with the resulting model to be analysed by

finite elements or other techniques. However, it should be noted that the existing preprocessors are used exclusively with one method of analysis only. It is a good idea indeed to have a general preprocessor for solid modelling that can format the data to different standard packages in finite elements or even other techniques like boundary elements or finite differences according to the user choice.

In the literature, in most cases FE preprocessors are used to assist a user of a FE program in data preparation and/or entry. In practice, roughly around 80% of the total effort spent in the analysis of a FE model is spent in the data preparation and input. Since most of the data input in FE programs are the geometry topology, most of these pre-processors offer a means for mesh generation. However, in order to have a more precise determination of the different functions of FE pre-processors, the following list is proposed [Sharaf Eldin, 1983b]:

1. Automatic or semi-automatic mesh generation:

   This means that the user has only to describe the FE region and let the preprocessor generate the necessary nodes and elements, i.e. discretize the domain into finite elements. Although there are a great number of such mesh generators, yet, the employed methods are different, as explained later and there is no truly complete reliable automatic mesh generator that can handle a general arbitrary region and generate a variety of element types with a reasonable amount of computer memory and execution time [Durocher and Gasper, 1979]. However, it should be mentioned that the range of degree of mesh automation is very wide. At one extreme the fully automatic methods which require a minimum of user input and

in which the mesh generator itself determines regions of high and low element densities. While at the other extreme, we find the simple methods which require the user to completely define the element mesh, while the "mesh generator" itself performs only minor operations like check for consistency, error detection and simple "numerical book-keeping" functions. However, an "inbetween" mesh generator is expected to be able to:

- Handle multiply connected regions (e.g. different soils in an aquifer).
- Handle a variable density of discretization (e.g. more triangulation near wells).
- Allow the user to edit the generated mesh.

(2) A good node numbering algorithm:

It is well known that the node numbering scheme is very important in determining the memory requirements for the master matrix in any finite element model. Only for simple problems, hand labelling by a careful inspection of the FE model topology can lead to an optimum bandwidth. Generally, for large size problems and/or complex topology, hand labelling becomes tedious and most probably the optimum or near optimum bandwidth is not guaranteed.

(3) Data formatting for the main FE processor:

The pre-processor itself is only a means to prepare the data efficiently for the actual FE processor. So, one of the most important functions of a preprocessor is to format the input data as required by the host FE processor. It seems from the literature review that this function has not been given the necessary attention. This may be due to the fact that each FE processor needs its data

in a different format from others. Most of the existing preprocessors are "tailored" for particular FE processors.

An ideal preprocessor is one that enables the user of generating the required data by the FE processor with the least amount of effort. A user friendly preprocessor should have the following basic features [Pesquera, et al, 1983]:

(1) Flexibility: The user should be able to follow any logical path he wishes in the definition of the FE model. For example, once the geometry has been defined, the preprocessor should be flexible enough to allow the user to choose the order he prefers to specify element and nodes attributes. The user should be able to control the element density in any part of the region to be modelled.

(2) Previous status recovery: An incorrect user action may have undesirable, even drastic, effects on the execution of the preprocessor. A good feature of an ideal preprocessor is to allow the recovery of the status prior to the last action done by the user. If a fatal error was detected by the preprocessor then it is desirable to be able to use the data and the computations accomplished so far prior to the breakdown of the preprocessor execution.

(3) Flexible means for data entry: Data entry is usually tedious and prone to errors. An ideal preprocessor should allow the most efficient way for data entry. For example, if a digitizer is used in FE modelling then data entry through a stylus and a calculator-type keypad may be more appropriate.

(4) Visual feedback: A powerful preprocessor should provide the facility for visual checking of the FE modelling using a graphic terminal or a plotter.

(5) Transportability: An ideal preprocessor should be portable as much as possible. This is usually very difficult to satisfy.

(6) Command duplication: The ability to duplicate certain commands without re-entering them is very desirable in a preprocessor. Using this facility it becomes possible to assign the same attribute to several different parts of the FE model.

(7) On-line help: As in most interactive systems, the availability of on-line help is an asset for an ideal preprocessor.

(8) Modularity and expandability: As any other software, modular structure for a preprocessor is a required feature to faciliate its maintenance. Expandability features allows the updating of the preprocessor to incorporate new algorithms or to support new hardware and/or software.

(9) Economy: An ideal preprocessor should be economical in terms of computer resources. It should also produce a model which can be processed by the FE processor economically. For example, if the FE processor uses a banded algorithm technique to solve the resulting FE equations, then, the preprocessor should number the nodes in order to minimize the bandwidth. On the other hand, if the FE processor utilizes a frontal solution scheme, then, the pre-processor should ensure minimum front width.

The information that an ideal preprocessor should produce can be grouped into:

(i) Model topology: this is usually the major task of a preprocessor and its main product. It usually consumes most of its processing time. It consists of nodes, elements and connectivity data.

(ii) Attributes and boundary conditions: this includes the properties of elements, for example, in an aquifer FE model the transmisivity, aquifer storage coefficient, etc. While in a structural analysis problem the attributes may be the cross-sectional area of elements, Poisson's ratio, Young's modulus, etc. Boundary conditions are also included in the preprocessor outputs. For example in a structural analysis problem specifying a fixed end will result in producing the boundary conditions of no vertical, horizontal deflections or rotations at this node. Loading data should also be included in the preprocessor outputs.

(iii) General information regarding the considered problem like: problem title, date and time of run, total CPU time used in preprocessing, control parameters used in preprocessing, etc.

4.9.2 Methods of Mesh Generation

Over the last 10 years many mesh generators have been developed. Most of them are oriented to structural analysis and solid modelling problems. Nevertheless, the main concepts can be used in other field problems like groundwater flow problems. In order to use a mesh generator the user must specify a global mesh to the preprocessor and the mesh generator, starting from this coarse mesh can refine recursively the mesh to the desired degree. Mesh generation is usually done in two steps: (i) Generation of nodes and (ii) Connecting nodes to form elements. The first attempts of mesh generation were fairly simple. Among the earliest methods is that used by the SAP IV program and its brother program MSAP which has been presented in this chapter. In this program a straight line interpolation technique is used for node

generation. A little further step is the division of the region into equal rectangles for regions which can be approximated by a rectangle. This method was used by the author in the preprocessor of the IFEP program [Sharaf Eldin, 1983b]. However, more sophisticated mesh generation schemes have been developed for regions of general shapes. Before presenting these techniques it is important to emphasize that each of the methods used in mesh generation has its merits and disadvantages and usually a particular scheme may be the best for certain applications and geometries only. Haber et al [1981] proposed a list of the criteria that can be used to evaluate the relative merits of different mesh generators as follows:

1) Precise modelling of boundaries: The mesh generator should place the boundary nodes on the boundary precisely. Failing to do so may result in more severe discretization error in addition to the "by-definition" discretization error inherent in the FE modelling itself.
2) Grading capability: It is usually desirable to have a user control on the grading of the mesh produced by a mesh generator. This ability can save unnecessary refinement of the whole mesh.
3) Minimal input efforts: Initial triangulation necessary to describe the FE region and the grading control information should be as minimum as possible. This saves time and effort and minimizes probable errors in data input.
4) Editing facility: The capability of editing the generated FE model produced by the mesh generator will allow the "human touches" to be added to the FE model. Engineering judgement cannot be totally substituted by the computer in many cases.

5) Broad range of applications: A mesh generator that covers a wider spectrum of applications is considered better than that suitable for particular applications.

6) Generality: General topology should be supported by a good mesh generator. Automatic generation of elements should be done without user intervention.

7) Favourable element shapes: The elements produced by the mesh generator should have favourate shapes to avoid ill-conditions that may arise in the FE model.

8) Computational efficiency: The preprocessor should number the nodes and elements automatically so that the solution of the FE equation will be efficient. Usually minimization of bandwidth or front-width is required according to the method employed in the FE processor. The method of mesh generation itself should be economical and efficient.

Now, let us explain some of the most general and efficient methods for mesh generation.

4.9.2.1 Mapping Techniques

Mesh generation by mapping is a technique where the element topology is simplified to a square or triangular grid system which is then mapped into the actual shape of the domain. Several mapping techniques can be used. Among the earliest and the easiest methods is the isoparametric mapping [Zienkiewicz and Phillips, 1971]. The main idea behind this method is to use an isoparametric curvilinear mapping of quadrilaterals. A typical parabolic quadrilateral has been shown in Figure (4.21). However, for convenience we repeat the same element

definition here. As shown in Figure (4.43) the x and y coordinates of a typical point within this element is related to the eight pairs of nodal coordinates by the equations,

$$x = \sum_{i=1}^{8} N_i(\xi,\eta)x_i \tag{4.57}$$

$$y = \sum_{i=1}^{8} N_i(\xi,\eta)y_i \tag{4.58}$$

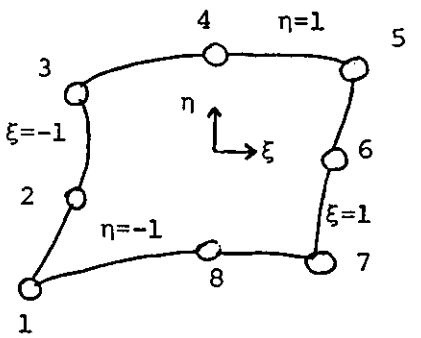


FIGURE 4.43: A typical parabolic quadrilateral

Where N_i is the shape function associated with each node and is defined in terms of the curvi-linear coordinates system (ξ,η). These functions have been determined before and will not be repeated here. Thus, if a region is defined by eight sets of nodal coordinates (x_i,y_i) for i=1 to 8; it is easy to generate a mesh to any desired size by sub-dividing the region in the (ξ,η) with equidistance lines in both directions and computing the x,y coordinates of the generated points using the shape functions N_i as in equations (4.57) and (4.58). In a multiply-connected region, a mesh of superelements is specified together with the required mesh size inside each superelement. One of the major restrictions of this method is that it requires each of the opposite sides to have the same number of node points which sometimes may be

undesirable. It is possible to generate element types other than the parabolic quadrilateral provided that the appropriate shape functions are used as has been demonstrated by Durocher and Gasper [1979].

The other two popular mapping techniques which have been used for mesh generation are the discrete transfinite mappings [Haber and Abel, 1982] and Laplacian mapping [Akin, 1982].

4.9.2.2 Mesh Generation by Direct Subdivision

In these methods a coarse mesh is defined by the user and then the computer starts partitioning the region into non-overlapping elements. Rivara [1984] described two general algorithms for the construction of triangular grids by iterative bisection of the longest side of each triangle. A similar idea is also implemented in the TWODEPEP package which will be described later. The algorithm used to subdivide any initial conforming triangulation D_o to a conforming triangulation by iterative bisection can be stated as follows:

(1) Define $D_o=\{T_i^o | T_i^o \in D_o\}$ where T_i^o are the initial triangles.

(2) Bisect every T_i^o by its longest side. Generally we obtain a non-conforming triangulation except in the special case where all sides are on the boundary. Let k=1, we denote the triangulation thus obtained by,

$$D_k = \{T_i^k | T_i^k \in D_1\}$$

(3) Consider the subset $S \in D_k$ of triangles where one of its sides contains a non-conforming node.

(4) Bisect triangles of S by its longest side.

(5) If the obtained triangulation is conforming then stop else increment k by 1 and goto (3).

The different steps when applying this algorithm to an initial triangulation is shown in Figure (4.44).

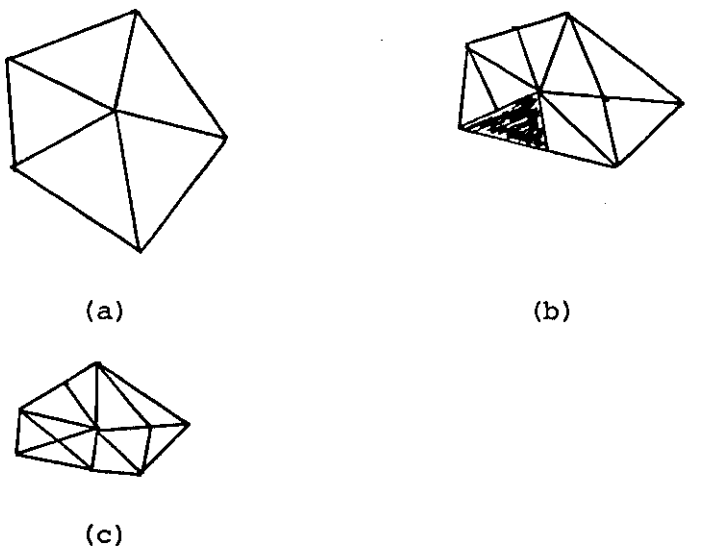


FIGURE 4.44: Subdivision of conforming triangles

(a) Initial triangulation D_0

(b) D_1 triangulation. The shaded triangle contains a non-conforming node

(c) D_2 final conforming triangulation.

A simpler version of this algorithm can be as follows:

(1) Bisect every T_i^0 by its longest side.

(2) Find the subset S of triangles generated in step 1 and contain a non-conforming node at one of its sides.

(3) For each triangle $T \in S$ join its non-conforming node with the opposite vertex. This will produce a conforming triangulation.

Of course the same procedure may be applied to every triangle of the new conforming triangles until the desired mesh refinement is achieved.

4.9.2.3 Mesh Generation by Quad Trees

A new approach for mesh generation by direct subdivision is the use of a modified quad tree [Shephard and Yerry, 1983]. Quad trees were used in digital image processing techniques and computer graphics as a convenient way for encoding 2-D objects in an integer tree structure which can be easily and efficiently manipulated by a computer. In the original quad tree encoding, the two-dimensional coherence is exploited by recursively decomposing the image into square areas until a sufficient homogeneous square is reached. Thus the whole image is the root of the tree and each square is a node in the tree. These nodes are leaves when the quadrants they represent are homogeneous otherwise, more refinement is done by decomposition. Operations on quad tree encoded images are much faster compared to other encoding techniques as was demonstrated by Oliver and Wiseman [1983]. A tutorial survey on quad trees and their applications in digital image processing is found in Samet [1984]. In order to apply these ideas for finite element meshes, the whole region of interest is considered as the full image and is placed in a square defined in the integer domain. The square is then divided into four quadrants. In digital image processing we usually consider a quadrant as homogeneous if its grey level is sufficiently homogeneous. In an analogous manner, in mesh generation, it is possible to consider a quadrant as homogeneous if it is fully inside the region or empty, i.e. fully outside the region. For a partially filled quadrant further decomposition is done. A further point which can be added here is that if the material properties within a full quadrant is not homogeneous then it must be divided again. The steps showing a quad tree representation of a circle are shown in

Figure (4.45) where one quadrant of the circle is shown in detail. One of the advantages of this technique is that boolean operations and translations or rotations can be handled with the simple integer operations of addition and multiplications by two. It is worthwhile

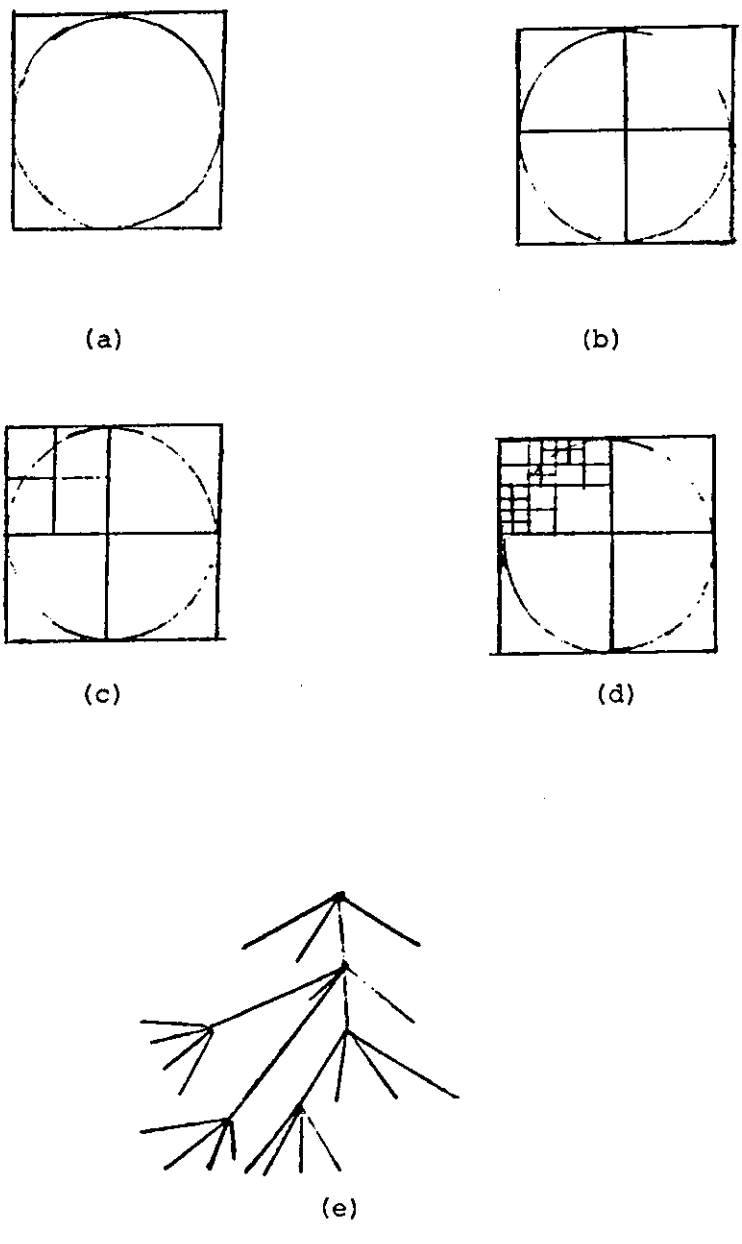


FIGURE 4.45: Quad tree representation of a circle (only one quadrant is shown)

to mention that since the tree structure for all objects is similar, it is possible for it to be implemented by special hardware. Operations with integer numerics is much faster than with floating point numerics. However, the direct application of quad tree as such for mesh generation is not suitable. First, the interior of the region may be represented by a small number of elements of large size. Secondly, two adjacent quadrants may be divided to different degrees thus creating non-conforming nodes. For most irregular boundaries we need too many subdivisions in order to represent the boundary. Among the solutions to these problems is to set a maximum size for any quadrant and if a quadrant is homogeneous and still larger than the stated threshold it is further divided. In order to solve the problem of non-conforming nodes it is possible to allow a corner cut-off for a quadrant. This modification to the quad tree will increase the status of a quadrant from three to four: full, empty, partially filled without cut and, the new case, partially filled with cut. The data of the cut edge is saved in another array maintaining the two end points of the line segment that cuts the quadrant. A transition mesh configuration could be used to eliminate non-conforming nodes. Graded meshes can be generated by specifying different levels of modified quad tree representation in various portions of the region.

4.9.2.4 Duplicate Nodes in Automatic Mesh Generators

It was noticed [Wu, 1982] that some automatic mesh generators may produce duplicate nodes i.e. more than one node number is generated for the same node. This is believed to happen in multiply connected regions where in the initial triangulation common nodes are shared by several super-elements (more than 4). In such cases it was noticed that some of

the generated nodes with identical coordinates are assigned different node numbers. Of course using this wrong information in the FE processor will have drastic effects on its execution. In a structural analysis application, for example, an overflow will occur due to the existence of a member of zero length connecting the duplicate nodes. The presented technique is quite simple and will eliminate duplicate nodes. The strategy is divided into three steps:

(i) Renumber the initial triangulation (super-elements) in a manner that minimizes the probability of having duplicate nodes. Practically, it is noticed that this can be achieved if super-elements sharing a common node are numbered in a continuous sequence. This step is done before using the mesh generator.

(ii) Check the existence of duplicate nodes after the execution of the mesh generator. This can be done by scanning the generated nodes to check for duplication.

(iii) If there are duplicate nodes, this step is done otherwise it can be bypassed. Here node numbers are compressed to eliminate any false nodes. To avoid unnecessary effort, the elements which contain a false node number are only corrected while other elements and nodes are kept unchanged.

4.9.3 Data Input for Finite Element Programs

Despite the method by which a FE mesh is prepared there should be some means to feed the data into the computer. In case full automatic mesh generation is used, the input data is usually prepared by the pre-processor. However, in other cases several methods are used for data input which can be classified as follows.

4.9.3.1 Interactive Ask-and-Answer

In this method the user sketches on a piece of paper the FE model. The user determines, based on his experience, the nodes and elements of the whole domain. Moreover, he assigns numbers to nodes and elements. Typically, an interactive program is developed which will be run using a terminal or a micro-computer. The computer starts asking the user, step by step, for the necessary data and the user answers the computer queries. The program formats the user inputs in a file according to the FE processor for which the preprocessor is designed. Sometimes plotting of the FE model is provided to check the correctness of the input data. One of the advantages of this approach is that data is checked at the time it is entered in the computer. Another advantage is that these programs are easy to program and maintain. The main disadvantage is that this method is time-consuming for large volumes of data. To overcome this difficulty it is possible to add a mesh generator within the program. An example of using this method is the pre-processor program for the MSAP program; PREMSAP [Kalidjian, 1976]. PREMSAP is a format free interactive program to prepare the input data for MSAP. This approach is also adopted by the author for the program ELASTIC which has been presented earlier. It is also used in the pre-processor for the IFEP program. This pre-processor, named PREIFEP is designed to facilitate the data preparation for the IFEP [Sharaf Eldin, 1983b]. The typical steps to use PREIFEP are as follows:

- The user should define his problem in sketches on paper including dimensioning and numbering. These data are: indicative title of the problem, job specifications, soil characteristics, geometry and boundary conditions.

- If the domain is a rectangle then an automatic mesh generator could be used to divide it into equal sized rectangles of that given dimension.
- If the domain is irregular, then, no automatic mesh generation could be done and the user should submit the details of his model topology as asked by the computer in step-by-step fashion.

All the data are given in free format, mixed modes are allowed, i.e. integer and real numbers can be mixed together. However, in the problem title the character "/" should be avoided because it is sensed by the operating system as the end of record delimiter. Similarly, the strings:EOD and :EOJ should be avoided for similar reasons. The process of requesting information by the computer and the user's responses to it continue till the last bit of information necessary to complete the input data required by the IFEP.

Since the IFEP is basically designed to run on a mini- or micro-computer, no external work files are used and thus all data can be stored in memory for immediate use by the IFEP or stored into a disc file in the same format as specified by the IFEP program for subsequent processing. Although the mesh generation in PREIFEP is fairly simple, it seems to be practically useful for some of the aquifer management problems particularly if no sources or sinks exist.

4.9.3.2 Special Definition Language

A special language is designed to describe the FE model. In this approach the user draws his FE model and fully describes it using a command language to define nodes, elements, materials, etc. This method can be implemented in batch mode as well as interactively. The

main drawback is, however, the need to learn some commands. Usually these languages are of free format nature. Examples of this method is that used in a preprocessor for the finite element program SAP IV proposed by Haugeneder et al [1981]. All the commands of this language follow the structure of:

(action): (descriptive values): (location);

The action part of the command indicates the type of action to be taken; while the descriptive values represent the values of attributes required by the specified action. The location part of the command specifies the location within the FE mesh on which the action is to be done. Three groups of commands are available:

(a) Commands to define general information for the problem to be solved. Among these are: the head command which is used to assign a title for the problem, the MC (master control) command which defines the global mesh parameters like number of nodes,... etc.

(b) Commands to describe the attributes of the nodal points and the incidence of the nodal points in the mesh like degrees of freedom, coordinates and temperature. Among these commands the coordinates which are used to define the coordinates of a node.

(c) Commands to describe the elements and applied loads. Amongst which is the PARAMETER command for the description of concentrated load/ mass data. The total number of commands offered by this language is 22. To show its versatility consider the following command:

X:10.3:10,14,17-27,120-220/10;

This command specifies the value of 10.3 as the X-coordinate of the nodal points: 10,14,17 to 27 inclusive and 120 to 220 in steps of 10, i.e. 120,130,...,220.

4.9.3.3 Direct Data Input Through Digitization

In these methods a digitizer or a graphic tablet is used to feed the topology of a FE mesh drawn to scale on paper directly into the computer memory. Digitizers are usually operated as a stand-alone system where a large size digitizing table is controlled by a micro-computer and the read data are recorded on a magnetic tape which is used subsequently as an input to a FE processor. On the other hand, a graphic tablet is connected to a computer directly and is operated on-line. A description of the Calcomp digitizer/edit system (DIGED) with different modes of operation can be found in [Sharaf Eldin, 1983d]. There are four types of digitization: point by point, grid, tracking and incremental digitizing. In the point by point digitizing the coordinates x,y pairs of the point to be digitized are transferred to a buffer when pressing the cursor's button. In the grid digitizing mode, a grid of equal Dx and Dy sizes in the x and y directions is prepared. Each point is approximated to the nearest grid intersection point. In the tracking mode; the user can specify the speed of data sampling i.e. points are digitized every Dt time interval. In the incremental digitizing mode incremental steps Dx and Dy in both x and y directions are specified and sampling takes place only if the absolute difference between two consecutive points is >Dx and/or Dy. In other words, sampling is based on distance criteria rather than time criteria as in the tracking mode.

In practice to use a digitizer, the FE model is drawn on paper to scale and fixed firmly to the digitizer table. The driving software requires the digitization of three points which do not lie on a straight line and the corresponding coordinates in user units. This will establish

the scale of the mesh and the orientation of the axis w.r.t. the table axis. Then the actual digitization of nodes begins node by node.

One of the known FE systems that contains a module that can be used to digitize a FE model is the GIFTS system [Kamel and Navabi, 1980]. In this system a digitizing tablet is used to enter key point positions. A graphical feedback of what is being digitized is displayed on the terminal. The output of this step is an input file that can be used by the BULKM mesh generator which is one of the GIFTS modules. It is not evident from the available literature that the full capabilities of a digitizer are utilized in automatic mesh generation. Perhaps using mixed modes of operations can be a better approach to utilize these capabilities. For example, using the point by point mode for the very critical points in the FE mesh and using the grid mode to define a rectangular mesh. Although this means seems to be attractive, yet some restrictions do exist. First, drawing the FE model precisely on a drawing paper which may not be convenient. Secondly, the digitization process itself has its own errors due to human error and the A/D conversion. In case the digitizing hardware is not available another method must be thought of.

4.9.4 Numbering Algorithms

For a given FE model the structure of the master matrix depends not only on the chosen interpolation functions and the element types, but also on the scheme used in numbering the nodes or elements of the FE model. Generally, for large size problems and/or complex topology, hand labelling for nodes is tedious and reaching the optimum node numbering (or element numbering) is usually not guaranteed. We must

indicate, first of all, that the criteria for assessing optimality of numbering is not unique. For example, an optimal numbering can be the one which minimizes the main storage requirements. Another one is that which gives the minimum solution time. Even when considering the minimum main storage as the governing criteria it is possible to consider the algorithms which minimizes the fill-in's, maximum bandwidth or a profile. From a practical point of view, we usually seek an algorithm which minimizes the maximum bandwidth if a band solver is to be used and an algorithm which minimizes the profile size if a profile technique is to be utilized, and finally a frontwidth minimization if a frontal solution is thought of. The problem of optimum numbering in FE models is known to be NP-complete. Accordingly, there is no efficient algorithm that guarantees the absolute optimum numbering of a general FE model. The only alternative is, therefore, heuristics. There are two general approaches for direct solution of FE equations: those based on banded algorithms and those based on frontal solutions. The maximum bandwidth is a direct function of node numbering while in the frontal algorithm the element numbering is the governing parameter. Since many of these algorithms are based on a graph theoretic approach, it is convenient to brief some of the basic graph concepts relevant to bandwidth minimization as follows:

Definition 4.1

A graph G consists of a set of nodes (vertices) V together with a set B, whose members are unordered pairs of nodes, the edges (branches) of G. A directed graph is one where the pairs in the set B are ordered. Two nodes of a graph are said to be adjacent if there is an edge between them.

Let $V = \{v_1, v_2, \ldots\}$ (4.59)

$E = \{(u,v) \mid u,v \in V\}$ (4.60)

Then, $G = (V,E)$. (4.61)

Definition 4.2

The adjacency set of a node u is defined as:

$$Adj(u) = \{v \in (V-U) \mid (v,u) \in E\} \quad (4.62)$$

where U={u}.

Definition 4.3

The degree of a node u is the cardinality of its adjacent list, i.e. $|Adj(u)|$. A path from one vertex to another is formed by a sequence of L edges, $(v_0,v_1),(v_1,v_2),\ldots,(v_{L-1},v_L)$, in which each edge is used only once. L is the path length. The vertices with which a path begins and ends are called the terminal vertices.

Definition 4.4

The distance between two terminal vertices u and v, denoted as d(u,v) is the length of the shortest path joining them.

Definition 4.5

The span of a subset U of V is defined as:

$$Span(U) = \{v \in V \mid \exists \text{ a path between v and u, } u \in U\}. \quad (4.63)$$

If G=(V,E) is connected, then the eccentricity of a vertex v is defined to be the longest distance from v as:

$$e(v) = \max\{d(u,v) \mid u \in V\} \quad (4.64)$$

The diameter is the maximum of all eccentricities:

$$D(G) = \max\{e(v) \mid v \in V\} \quad (4.65)$$

A vertex v such that e(v)=D(G) is called a peripheral vertex.

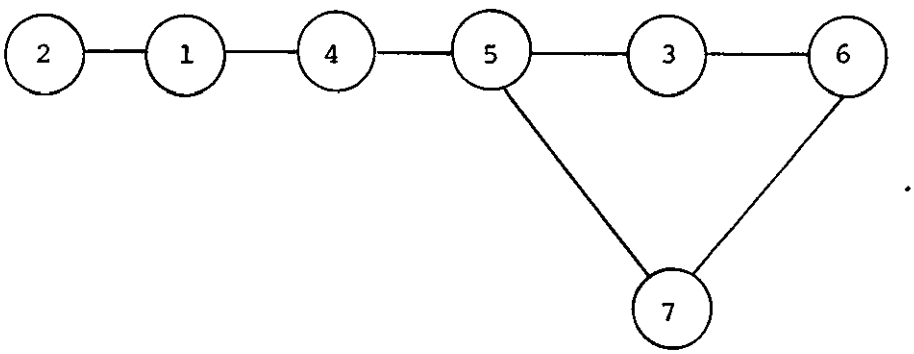
Definition 4.6

For a graph G(A) corresponding to the matrix A we will have n

nodes labelled 1,2,...,n. For each non-zero element a_{ij}, i<j of A there will be an edge connecting nodes i and j. From that graph it is possible to determine the location of all off-diagonal non-zero elements of A. To illustrate consider the matrix.

$$A = \begin{bmatrix} X & & & & & & \\ X & X & & & & & \\ O & O & X & & \text{Symmetric} & & \\ X & O & O & X & & & \\ O & O & X & X & X & & \\ O & O & X & O & O & X & \\ O & O & O & O & X & X & X \end{bmatrix}$$

Its associated labelled graph G will be:



It should be noted that we can number the nodes of the system in n! ways. The absolute minimum bandwidth that can be reached is m-1, where m is the largest degree of the nodes in the system. If the original matrix A is permuted using a permutation matrix P then the permuted system PAP^T will be represented by a graph $G'=G(PAP^T)$ which is structurally identical to A but the node labels will be different according to P.

4.9.4.1 Algorithms for Minimizing Bandwidth

Algorithms for minimizing matrix bandwidth became of great interest in the early sixties. The early methods were time consuming and impractical. For example, Alway and Martin [1965] proposed a search of permutations in order to find one which can be used to permute rows and corresponding columns of the original matrix to minimize the bandwidth. However, the first very successful and still very widely used, is the Cuthill and McKee (CM) algorithm [1969] for the reduction of the bandwidth of sparse symmetric matrices. Later, many numbering algorithms for minimization of bandwidth and/or profile of a matrix were developed which have practical value. In what follows the most well-known methods are presented. Since the numbering algorithm employed in the FE software used in this thesis is based on the reverse Cuthill-McKee (RCM) algorithm it is convenient to start with the CM algorithm.

1. The Cuthill-McKee Algorithm

The Cuthill-McKee (CM) algorithm was primarily designed to reduce the bandwidth of a sparse symmetric matrix. The scheme makes use of a simple observation: if x_i is a labelled node; then in order to minimize the bandwidth of the row containing x_i it is clear that all nodes connected to x_i and not yet labelled should be labelled as soon as possible after x_i.

Thus the CM algorithm may be considered as a method that reduces the bandwidth of a matrix via a local minimization of the row's bandwidths. The first node to be numbered is called the starting node. The choice of a good starting node in the CM and many other similar algorithms is very critical and affects the bandwidth and profile

reduction. Based on substantial experience it was discovered that if the starting node is a peripheral node the better bandwidth can be achieved. However, since the search of a peripheral node is generally expensive for a non-trivial graph, it is usually a pseudo-peripheral node which is searched for as a starting node [George, and Liu, 1981]. The CM algorithm can be summarized as follows:

(1) Determine a starting node and assign it the number 1.

(2) Main loop:

For i=1 to N;

Find all the un-numbered neighbours of the node x_i and number them in increasing order of their degree until all the nodes are numbered.

To illustrate the CM algorithm consider the FE model assuming one unknown is associated with each node as shown in Figure 4.46.

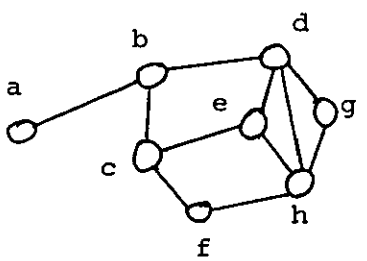


FIGURE 4.46: A FE model to be numbered

This model is numbered using "a" as a starting node. The step-by-step numbering is shown in Table 4.10.

Node Label	Node Degree	Assigned Number	Un-numbered neighbours sorted by order of degree (ascending)
a	1	1	b
b	3	2	c,d
c	3	3	f,e
d	4	4	g,h
f	2	5	-
e	3	6	-
g	2	7	-
h	4	8	-

TABLE 4.10: CM numbering using node "a" as a starting node

The numbered model is shown in Figure (4.47).

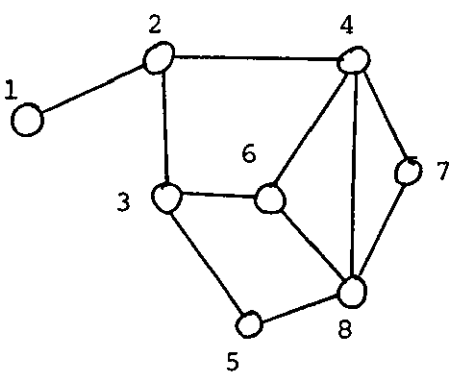


FIGURE 4.47: Numbering using "a" as a starting node

The associated matrix structure is:

$$\begin{bmatrix}
* & & & & & & & \\
* & * & & & & & & \\
o & * & * & & & & & \\
o & * & o & * & & & & \\
o & o & * & o & * & & & \\
o & o & * & * & o & * & & \\
o & o & o & * & o & o & * & \\
o & o & o & * & * & * & * & *
\end{bmatrix}$$

The profile size in this case is 16 and the semi-bandwidth is 4.

Solving the same example using "e" as a starting node, we reach a semi-bandwidth = 4 also but an increased profile to 19 as shown in Figure (4.48) and Table 4.11.

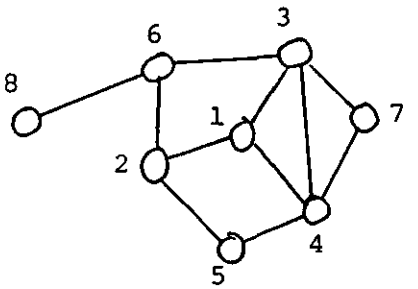


FIGURE 4.48: Numbering using "e" as a starting node

Node Label	Node Degree	Assigned Number	Un-numbered neighbours sorted by order of degree (ascending)
e	3	1	c,d,h
c	3	2	f,b
d	4	3	g
h	4	4	-
f	2	5	-
b	3	6	a
g	2	7	-
a	1	8	

The associated matrix structure is:

$$\begin{bmatrix}
* & & & & & & & \\
* & * & & & & & & \\
* & o & * & & & & & \\
* & o & * & * & & & & \\
o & * & o & * & * & & & \\
o & * & * & o & o & * & & \\
o & o & * & * & o & o & * & \\
o & o & o & o & o & * & o & *
\end{bmatrix}$$

2. The Reverse Cuthill-McKee (RCM) Algorithm

As storage schemes for linear equations moved from banded to profile, the objective of renumbering methods has changed to minimize the matrix profile. George [1971] has discovered that the ordering by reversing the Cuthill-McKee (CM) ordering often turns out to give a better profile as compared to the original CM algorithm. He called this the reverse Cuthill-McKee ordering (RCM). Reversing is done by assigning new node numbers $y_i = x_{N-i+1}$ where N is the total number of nodes and the x's are the node numbers obtained by the CM algorithm. To illustrate consider the previous example shown in Figure (4.46), the numbering using the RCM algorithm will result in the numbering shown in Figure (4.49) and Table 4.12.

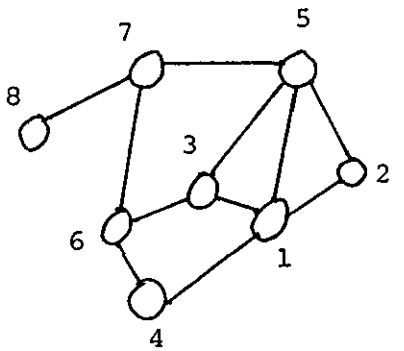


FIGURE 4.49: RCM numbering using "a" as a starting node

TABLE 4.12: RCM numbering using "a" as a starting node for the CM

Node Label	CM Numbering	RCM Numbering
a	1	8
b	2	7
c	3	6
d	4	5
f	5	4
e	6	3
g	7	2
h	8	1

The associated matrix is:

$$\begin{bmatrix} * & & & & & & & \\ * & * & & & & & & \\ * & o & * & & & & & \\ * & o & o & * & & & & \\ * & * & * & o & * & & & \\ o & o & * & * & o & * & & \\ o & o & o & o & * & * & * & \\ o & o & o & o & o & o & * & * \end{bmatrix}$$

The profile size is 16.

When considering the other numbering scheme using node "e" as a starting node in the CM algorithm, we get the following results illustrated in Figure (4.50) and Table 4.13 with associated matrix:

$$\begin{bmatrix} * & & & & & & & \\ o & * & & & & & & \\ * & o & * & & & & & \\ o & o & o & * & & & & \\ o & * & o & * & * & & & \\ o & * & * & o & * & * & & \\ o & o & * & * & o & o & * & \\ o & o & o & o & * & * & * & * \end{bmatrix}$$

and profile size of 16.

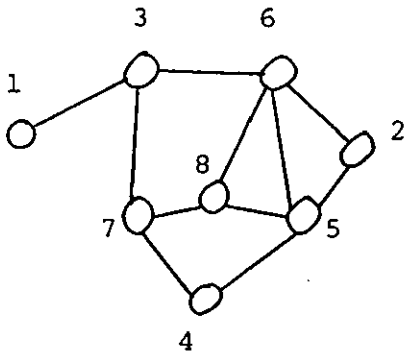


FIGURE 4.50: RCM numbering using "e" as a starting node.

Node Label	CM Numbering	RCM Numbering
e	1	8
c	2	7
d	3	6
h	4	5
f	5	4
b	6	3
g	7	2
a	8	1

TABLE 4.13: The RCM numbering using node "e" as a starting node in the CM

It is possible to conclude that:

(i) Choosing a good starting node can affect the semi-bandwidth and/or the profile size in the CM algorithm.

(ii) The RCM does not reduce the bandwidth obtained by the CM.

(iii) The RCM often reduces the profile size obtained by the CM but can never increase it.

3. The Gibbs-Poole-Stockmeyer [GPS] Method

This method, presented by Gibbs et al [1976], is based on searching

a spanning tree of the graph of a matrix. This method can be outlined as follows:

(i) Determine the end points u and v of distance k of a pseudo-diameter. This is done iteratively to determine a vertex which is at a maximum distance away from a given vertex.

(ii) Partition the set of other vertices into levels $L_1, L_2, \ldots, L_k$ such that adjacent vertices in the graph G are in the same or adjacent levels and such that $\max_i |L_i|$ is nearly minimized.

(iii) Number vertices of the graph G level by level beginning with the end point of the pseudo-diameter of least degree and according to the increasing degree of each node in the same level.

Solving the previous example shown in Figure (4.46) again using the GPS method with a and h as a pseudo-diameter nodes will result in the following partition: {a,b},{c,d} and {e,f,g,h}. The numbering is, therefore, a=1, b=2, c=3, d=4, g=5, f=6, e=7 and h=8. This is shown in Figure (4.51).

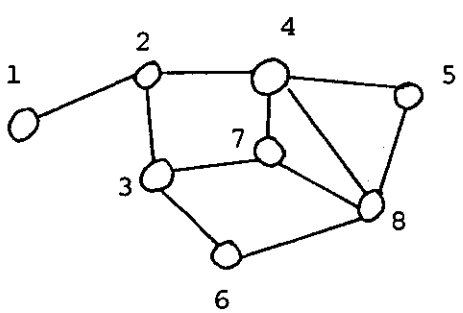


FIGURE 4.51: The GPS numbering

The associated matrix is:

$$\begin{bmatrix} * & & & & & & & \\ * & * & & & & & & \\ o & * & * & & & & & \\ o & * & o & * & & & & \\ o & o & o & * & * & & & \\ o & o & * & o & o & * & & \\ o & o & * & * & o & o & * & \\ o & o & o & * & * & * & * & * \end{bmatrix}$$

This gives a semibandwidth of 4 and a profile size of 16 also.

4. The Span-Ponderation-Sum (SPS) Method

This method is an iterative one developed by Akhras and Dhatt [1976]. The main idea is that an optimally ordered FE model possesses three properties which they called sum, ponderation and span properties. Thus the SPS method tries to respect these three criteria iteratively. There is no proof that this method will give minimum bandwidth nor that it will converge. However, as reported by the authors, it is tested numerically and proved to be efficient and reliable. To explain what is meant by span and ponderation consider the following FE model which is optimally numbered (Figure 4.52).

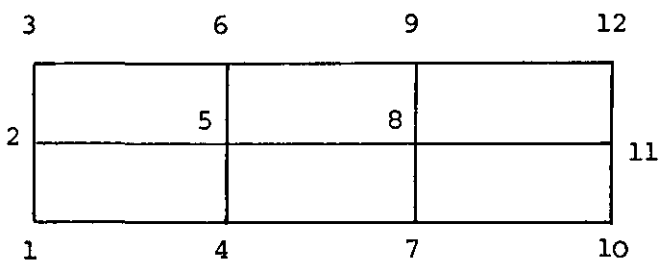


FIGURE 4.52: FE model optimally numbered

We form the connectivity matrix C of each node as shown in Table 4.14. Note that the last column in this table is the number of nodes connected to the considered node.

Node	Connectivity Matrix	Number of Nodes
1	1,2,5,4	4
2	2,5,4,1,3,6	6
3	3,6,5,2	4
4	4,1,2,5,8,7	6
5	5,6,9,8,7,4,2,3,1	9
6	6,9,8,5,2,3	6
7	7,8,11,10,4,5	6
8	8,9,12,11,10,7,5,6,4	9
9	9,12,11,8,5,6	6
10	10,7,8,11	4
11	11,8,9,12,10,7	6
12	12,11,8,9	4

TABLE 4.14: The connectivity matrix of the model problem

Note that the common nodes in the connectivity matrix are not duplicated. For example, for node 2, the node number 5 is considered only once. Note also that the node number of the considered node is included. The three terms: sum, ponderation and span are defined as follows:

(i) Sum of node i = sum of node numbers in row i in the connectivity matrix C. For example sum(1)=12.

(ii) Ponderation of node i = sum of node i ÷ number of nodes connected to node i.

(iii) Span of node i = the sum of the highest node number and the lowest node number in the ith row in the connectivity matrix.

These values are computed for the considered problem as shown in Table 4.15.

Node	Number of Nodes	Sum	Ponderation	Span
1	4	12	3	6
2	6	21	3.5	7
3	4	16	4	8
4	6	27	4.5	9
5	9	45	5	10
6	6	33	5.5	11
7	6	45	7.5	15
8	9	72	8	16
9	6	51	8.5	17
10	4	36	9	18
11	6	57	9.5	19
12	4	40	10	20

TABLE 4.15: The SPS of the model problem

The three noticed properties are:

(i) The sum of nodes having the same number of nodes in the connectivity matrix is arranged in an ascending order. For example, the nodes which have 4 neighbours are: 1,3,10 and 12; their sums are: 12,16,36 and 40.

(ii) The ponderation of higher numbered node is > that of lower numbered, i.e. the elements of the ponderation vector are arranged in increasing order.

(iii) The elements of the span vector are arranged in an increasing order.

Thus the SPS algorithm is actually a two phase one where in the first phase the nodes are renumbered based upon the span criterion and the ponderation criterion, respectively. This is done iteratively until no further decrease in bandwidth can be achieved. In the second phase the ponderation and the sum criterion are satisfied simultaneously in a similar iterative procedure. Usually, the first phase is re-executed for a possible further reduction. It should be emphasized, however, that there is no proof of the convergence of this procedure and for that reason if at any iterative step number n we find that the bandwidth is the same as in the (n-1),(n-2) a minimum value is assumed. Despite the lack of proof for this method some numerical experiments have proved its validity. However, to illustrate that these three criteria may be valid without reaching the minimum bandwidth; the same test problem used in the previous methods will be numbered in a non-optimal way as shown in Figure (4.53).

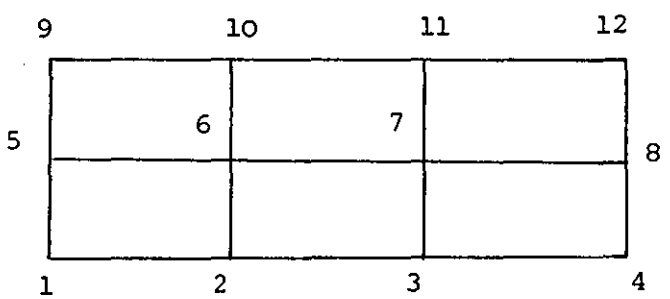


FIGURE 4.53: Test problem for the SPS method

The corresponding SPS data for this problem is shown in Table 4.16. Although the three criteria are satisfied in this numbering, we are sure it is not the optimal, or even near optimal numbering.

Node	Connectivity Matrix	No.of Elements	Sum	Ponderation	Span
1	1,5,6,2	4	14	3.5	7
2	2,6,7,3,1,5	6	24	4	8
3	3,7,8,4,2,6	6	30	5	10
4	4,3,7,8	4	22	5.5	11
5	5,6,2,1,9,10	6	33	5.5	11
6	6,7,3,2,1,5,9,10,11	9	54	6	12
7	7,8,4,3,2,6,10,11,12	9	63	7	14
8	8,4,3,7,11,12	6	45	7.5	15
9	9,10,6,5	4	30	7.5	15
10	10,11,7,6,5,9	6	48	8	16
11	11,12,8,7,6,10	6	54	9	18
12	12,8,7,11	4	38	9.5	19

TABLE 4.16: The SPS data for the test problem

The following modifications may lead to better results for the SPS method:

(i) We notice that the maximum semi-bandwidth is a direct function of the difference between highest and lowest node numbers connected together. Note also that in the first row of the C matrix, the node number 1 is always present. Our aim is to have C_1max to be as close to 1 as possible. This implies that the smaller is their sum the better row bandwidth we obtain, i.e. it is better to have a lesser span for row number 1. On

the contrary, for the last row in C, we notice that node number n is always present in this row and we are sure it is the highest node numbered in this row. Thus our goal is to have the minimum node number in this row to be close to n as much as possible.

This implies that the bigger their span the better row bandwidth is obtained. So respecting the span criteria only is not enough to get better minimum bandwidth. If the stated condition is applied, it is very probable that a better bandwidth may be obtained. By the same argument, we can conclude that for the lower sum of the first row and the bigger sum for the last row better results can be obtained.

(ii) To detect that possible reduction may be obtained we notice that there are repeated entries in the span vector (11 and 15 in our example). There is some sense in considering that repeated entries may be a sign of a non-optimal numbering. To illustrate assume that two rows in the C matrix have the same span and assume the first row is $\{a_1,a_2,\ldots,a_r\}$ with a_1 as the minimum node number and a_r the maximum one within that row. The second row is assumed to be, in a similar manner $\{b_1,b_2,\ldots,b_r\}$. If b_1 is assumed to be $>a_1$ and since the span is the same, it implies that $a_r>b_r$. This implies that a-row bandwidth is larger than b-row bandwidth.

The validity of these conditions can be visualized by considering Tables 4.15 and 4.16. The first row span and sum in the first table which is for optimal numbering, are 6 and 12, respectively. The same row in the other table, which is for non-optimal numbering, are 7 and 14, respectively. This is evidence that our modifications may improve the SPS algorithm. The same is for the last row where the sum and span in the first table are 40 and 20, respectively, while in the other table

they are 38 and 19, respectively. It is also noticed that no duplicate entries exist in the span vector in the first table.

However, it should be emphasized again that this algorithm and the proposed modifications do not have mathematical proof.

5. Hoit and Wilson Profile-Front Minimization

This is a direct method by Hoit and Wilson [1983]. The main idea behind it is to renumber the nodes in a manner identical to the equation solving sequence used in the well-known frontal method. After renumbering, the equations are stored in a profile form whence a profile or frontal solver may be used for the actual solution of the equations. They called this method profile-front minimization (PFM). The algorithm can be summarized in the following:

(i) Select a starting node.

(ii) Add all the nodes of the elements connected to this node to the front.

(iii) Number all completed (in the frontal solver sense) nodes sequentially using the next available number and remove them from the front.

(iv) Choose the next element to add to the front. The choice is based on a frontal sense. In other words, we pick the element that completes most of the nodes in the front but adds the minimum number of new nodes to the front. In case more than one element satisfies this criteria we choose the one of least weighted element degree [to be defined later].

(v) Repeat steps (iii) and (iv) until all the nodes are numbered.

Two points must be discussed; the weighted element degree and the choice of a starting node. The degree of a node is the number of its adjacent

nodes. Element degree is defined to be the sum of its node degrees. To introduce the effect of neighbours of an element or a node a weighted degree is introduced. The weighted node degree is the sum of the degrees of nodes adjacent to it. The weighted degree of an element is the sum of the weighted degrees of its nodes. It is clear that this weighting procedure reflects the connectivity of nodes and elements on a global basis rather than a local one. The same process can be repeated further i.e. a weighted weighted element degree and so on. Thus if multiple passes are made through the weighting process, the hyper-weighting values begin to reflect the degree on a global structure level rather than the local level as most of the current numbering algorithms.

As in most of the numbering techniques, the choice of a good starting node is important irrespective of the numbering technique itself. Usually the problem of finding a suitable starting node is treated separately from the numbering problem itself. There is no algorithm that can give the best starting node directly. Usually multiple nodes are tried in order to have a better chance of finding an optimum numbering. Trying all nodes is, of course, prohibitive and will certainly take time much more than which is needed to solve the original problem itself. In the PFM method three starting nodes are tried: (a) the node with minimum global nodal degree. If more than one satisfies this condition, we choose the one whose element has the minimum global degree. If still more than one exist, the last one found is chosen. This node is used as the root to form a level structure or a spanning tree L_1. (b) The second node is chosen from the last level of the level structure L_1. It is chosen as the one with minimum nodal and element global degrees. This node is chosen in a similar manner to that

used in the previous one. A level structure L_2 is formed using this node as a root. (c) The third node is chosen from the level structure L_2 in a similar manner to that used when choosing the previous two nodes but from the highest level to contain nodes that were also in the last level of the primary level structure L_1. These three nodes are used as starting nodes, the numbering is performed and the one which results in a minimum profile is considered. If more than one, the one which produces minimum bandwidth is considered.

To illustrate this method, the previous example is considered again. The FE model is shown in Figure (4.54).

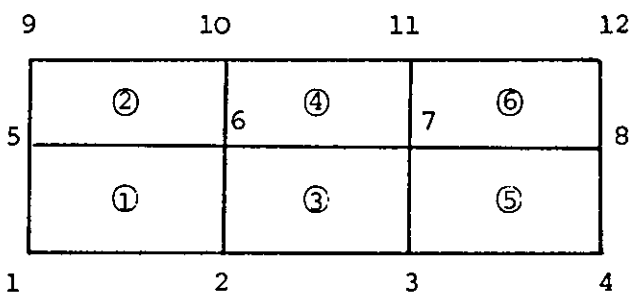


FIGURE 4.54: A sample FE model

Tables 4.17 and 4.18 are for the nodal and element degrees, respectively.

Node	Adjacent Nodes	Nodal Degree	Weighted Nodal Degree
1	2,5,6	3	18
2	1,5,6,3,7	5	29
3	2,6,7,8,4	5	29
4	3,7,8	3	18
5	9,10,6,2,1	5	24
6	1,5,9,10,11,7,3,2	8	39
7	6,10,11,12,8,4,3,2	8	39
8	11,12,4,3,7	5	24
9	10,6,5	3	18
10	9,11,7,6,5	5	29
11	10,12,8,7,6	5	29
12	11,8,7	3	18

TABLE 4.17: Nodal degrees

Element Number	Its Nodes	Element Degree	Weighted Element Degree
1	1,2,5,6	21	110
2	5,9,10,6	21	110
3	2,6,7,3	26	136
4	6,10,11,7	26	136
5	3,7,8,4	21	110
6	7,11,12,18	21	110

TABLE 4.18: Element degrees

We notice that the different nodes degrees were three: 3,5 and 8. However, weighting nodal degrees give four different degrees: 18, 24, 29 and 39. The level structure L_1 will be as shown in Figure (4.55).

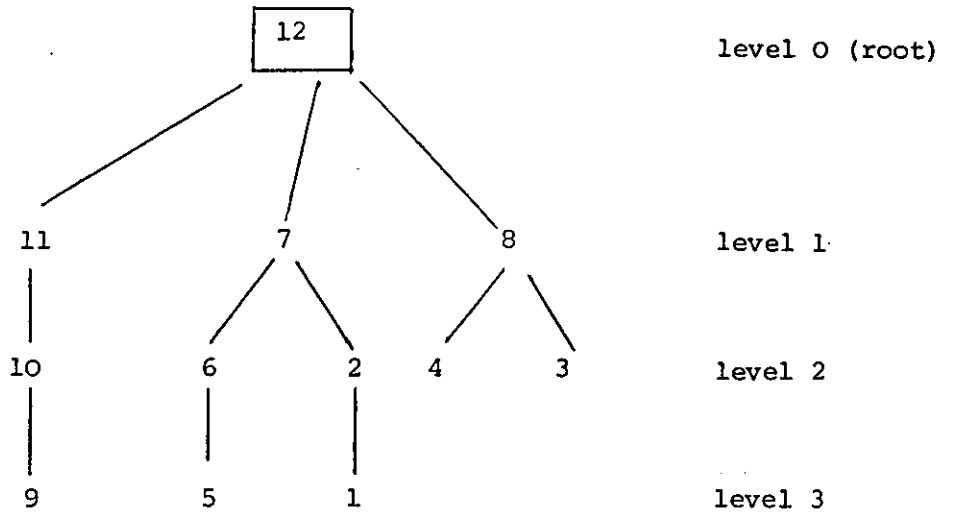


FIGURE 4.55: Level Structure L_1

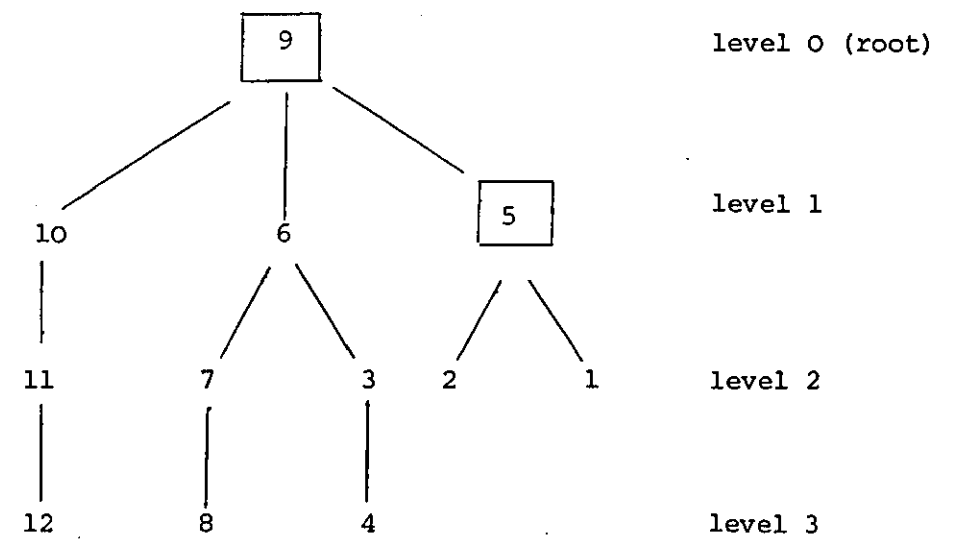


FIGURE 4.56: Spanning tree L_2

The three starting nodes are, therefore: 12,9 and 5.

It is clear that using 12 as a starting node and applying the PFM method in this example will lead to the optimum numbering.

6. Comments on Node Numbering Techniques

Practical node numbering techniques are based on heuristics and intuition rather than formal methods. This is due to the fact that techniques which are guaranteed to give the absolute minimum for NP-complete problems are prohibitive in terms of computer time and the time spent to get such solutions is certainly much more than any savings which might be gained. Nevertheless, there are some general points which should be considered in such methods and can be summarized in the following:

(i) Separation between the two problems of node resequencing and selection of the starting node as most of the current techniques do, is not, in my opinion, the most suitable approach. The reason for this separation may be due to the complexity of the problem and this approach breakdown the renumbering problem into two "independent" problems. However, the question of independency is still an open one.

(ii) Most of the current techniques are based on local minimization. Except the method of PFM which tries to take the global nature of the problem into account by recursive weighting of nodes and elements.

(iii) The criteria of what is the best renumbering technique is not unique. Although most of the methods consider the minimization of the width of the band or profile as a criteria, other criteria should be considered as well, like minimization of the fill-in's or the minimization of fast memory requirements or the minimization of the I/O operations with backing storage devices. Since the nodes are numbered prior to solving a set of FE equations by a Gaussian

elimination or a similar technique it is wise to consider the solution method when designing a numbering algorithm. It is also important to consider the nature of the problem to be solved. Time-dependent and non-linear problems may impose requirements that may differ from linear analysis. As important is the topology of the domain. For example, it is known that the Cuthill-McKee (CM) algorithm is not suitable for tree structures [Everstine and Cuthill, 1983].

(iv) The problem can, therefore, be considered as a decision-making one and it is a good idea to include multiple methods incorporated in FE programs and using a starting module to determine which method is most suitable for the considered problem.

(v) In some simple cases it is usually easy to number the nodes optimally by inspection. For example, a rectangular region should be numbered along the shortest side while a connected region or a circular should be numbered alternately from right and left of the chosen starting node.

4.9.4.2 Algorithms for Minimization of Frontwidth

It seems that not many algorithms have been developed to minimize the frontwidth in a frontal solver for FE analysis. This is perhaps due to the fact that the frontal algorithm is basically an "out-of-core" technique which does not require a large fast memory as compared to banded algorithms for example. The problem of determining the assembly sequence of elements in a frontal solution which result in minimum frontwidth during the solution process is very difficult and there is no algorithm known which guarantees absolute minimum frontwidth. It is

possible to classify the techniques for frontal minimization into two major approaches: (i) reducing frontwidth through node renumbering and (ii) reducing frontwidth through direct element renumbering. Three different methods may be considered as the most popular techniques for frontwidth minimization and represent different approaches. The first one is that by Bykat [1977] who used a technique very similar to the Cuthill-Mckee method for reducing bandwidth of symmetric sparse matrices which have been explained earlier. This approach seems to be logical in a sense that at step i during the assembly process we choose the element which has a greater number of neighbouring elements already assembled, thus minimizing the increase in frontwidth. As in the CM method, the starting element is chosen as the one with the smallest number of neighbours.

Razzaque [1980] develops a method based on node renumbering to reduce the bandwidth then the elements are resequenced according to their least numbered node in an ascending order. To illustrate consider the example shown in Figure (4.57a) which represents a FE model which is numbered arbitrarily. Figure (4.57b) represents the same mesh after

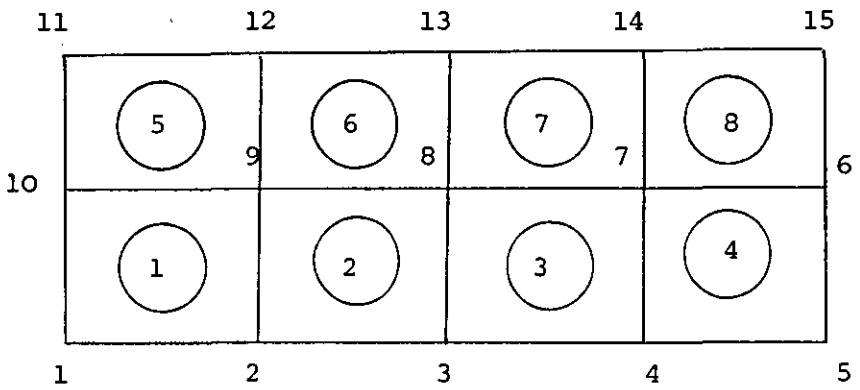


FIGURE 4.57a: Original numbering

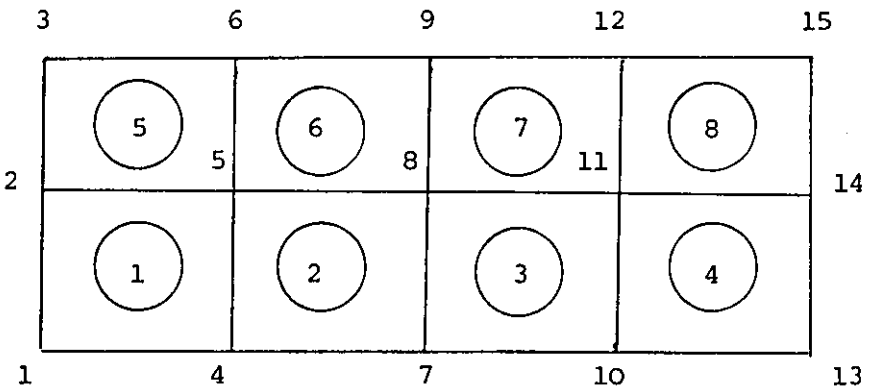


FIGURE 4.57b: Minimum bandwidth numbering

doing bandwidth minimization using CM algorithm (say). Note that element numbers are unchanged while node numbers are now changed. Now we re-number the elements according to the least numbered "new" nodes. To do so it is convenient to establish Table 4.19. The new element numbers are shown in the same table. The final element numbering is shown in

Element Number	Least Numbered Node	New Element Number
1	1	1
2	4	3
3	7	5
4	10	7
5	2	2
6	5	4
7	8	6
8	11	8

TABLE 4.19: Element renumbering

Figure 4.57c. Note that original node numbers are retained.

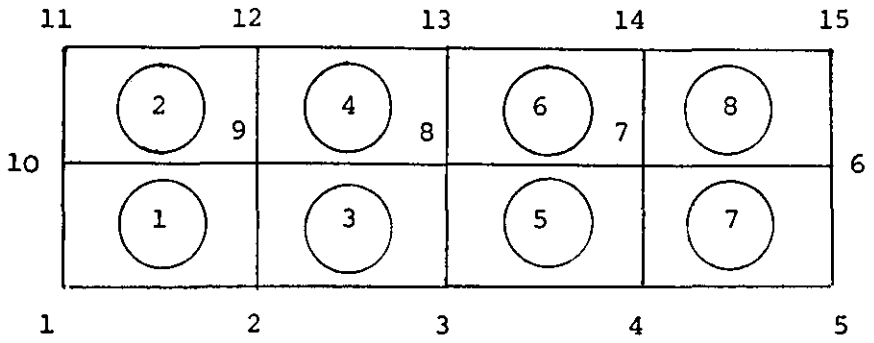


FIGURE 4.57c: Final element numbering

The method developed by Pina [1981] employs the concept of node degree in order to choose the next elements to be assembled. In this method the nodes in the front are scanned and those of minimum current degree are considered. If more than one node is found we use the current weighted degree of nodes to choose the one of minimum weighted degree. If still more than one exist, the last one is arbitrarily chosen. The elements to be assembled are all elements which have this node amongst their nodes. The starting node is selected on the same basis, i.e. the one with smallest degree. If more than one exists the one with minimum weighted degree is chosen.

4.10 POST-PROCESSORS FOR FE PROGRAMS

4.10.1 Introduction

Three major problems normally face the sophisticated user of the FEM. These are: the availability of an adequate computer system (hardware and software), an efficient means to prepare the input data and feed it into the computer and finally, an efficient way to extract the useful information out of the numerous computer outputs. For the first point, normally, the user needs a mainframe or a super mini-computer to be able to run realistic FE problems, together with the necessary software. It should be pointed out that only very few FE systems are available on small mini- or micro-computers. The feeding of input data into the computer is another problem. To solve this problem, preprocessors are used as explained earlier. The outputs of a FE system are normally huge. It is not uncommon to get thousands of printed lines as an output of a FE analysis. In practice, the analyst or the engineer, skips hundreds of values of the computer outputs and concentrates on only a relatively few figures that are of interest to him and govern his design. This is a waste of computer resources, waste of paper and waste of the engineer's time. Moreover, usually the FE models satisfy some of the physical problem quantities while the other are not satisfied. To exemplify, usually the displacement field is satisfied at nodes only while the resulting stress field is normally not satisfied. Several methods like stress averaging are proposed to solve this problem. The above mentioned problems were the motivation behind the evolution of post-processors.

4.10.2 The Functions of Post-Processors

In the literature, the term post-processor is used as a synonym to a plotting or graphical package. This situation has been explained earlier for preprocessors. It is the author's opinion that the post-processor definition should be extended to allow for more logical functions. The following list is proposed for an ideal post-processor for FE programs [Sharaf Eldin, 1983c]:

(i) Plotting the FE model as described by the user or as a result of a preprocessor. This function could also be a part of the pre-processor but, in most cases it is incorporated into the post-processor. This plotting is useful for the checking of the input specially in complicated FE models. One mistake in input data could result in the solution to a completely different problem. Such mistakes could be easily detected by eye if the FE model is plotted. This plotting is also useful for FE model editing and for documentation purposes.

(ii) Making simple refinement of the computed data, if possible. As explained earlier, the existing FE models usually satisfy only some criterion of the physical system. The other criteria are only approximately satisfied. Although, in practice, in many cases this does not represent a serious problem, it becomes sometimes very critical.

(iii) Printing of the relevant results only and as required by the user. It does not make sense to print the displacements at each node in the region. The nodes, except in a few positions, have no physical meaning. A user of a groundwater analysis program may be interested to know the head-drop rather than the actual head

and at a relatively few nodes. He may be interested in knowing the flow among a specified boundary and its direction. So, one of the most important functions of a post-processor is to print results only at selected points (and not necessarily the nodal points).

(iv) Computing of other relevant quantities as required by the user. In a structural mechanics problem, the basic unknown solved by a FE system is normally the displacements. Usually the more important quantity is the stresses. The computation of other quantities based on the obtained solution should, therefore, be a post-processor function.

(v) Plotting of some of the results. Usually a graphical representation makes the results easier to be understood and interpreted. Plotting can be done on a graphics terminal or plotter. In case these facilities are not available, printer plotting can be used instead.

In the following sections the concepts and facilities required to fulfill each of these functions will be discussed.

4.10.3 Stress Smoothing Methods

It seems that this point has not been widely considered in most of the well-known FE systems. In fact, none of the 36 well known FE programs presented by Noor [1981] has a postprocessor that tries to smooth the resulting stress field. It is known that, the stress field σ is usually discontinuous across interelement boundaries in an FE analysis based on the displacement method. Recall the fundamental FE equation:

$$Kd = f \ , \tag{4.66}$$

where K is the structure stiffness matrix, d is the unknown displacement vector and f the vector of nodal loads. The computed displacements d are then used to compute the stress vector σ. In order to have a continuous stress field, many methods have been proposed. The oldest of these methods was based on simple averaging technique [Wilson, 1963]. The most successful method, and yet easiest to implement and of a general nature, seems to be that of Loubignac and is known as Loubignac iterative procedure [Loubignac, et al, 1977 and Cook, 1982]. This method can be summarized as follows:

(i) The average nodal stresses are computed by simple averaging:

$$\sigma_i^{(0)} = \frac{1}{m} \sum_{e=1}^{m} \sigma_i^e \tag{4.67}$$

where m is the number of elements sharing node i, $\sigma_i^{(0)}$ is the average nodal stresses and σ_i^e are the stress values computed from the solved nodal displacement by FEM. σ_i^e is related to the nodal displacements d^e by:

$$\sigma_i^e = DBd^e \ , \tag{4.68}$$

where D is the matrix of elastic properties in the constitutive law and B is the transformation matrix between strains and nodal displacements.

(ii) A continuous stress field can be obtained within an element by:

$$\bar{\sigma}_j^{(0)} = N\sigma_i^{(0)} \ , \tag{4.69}$$

where $\bar{\sigma}_j^{(0)}$ is the continuous stress field in element j and N are the usual shape functions used in the displacement analysis.

(iii) The nodal forces q corresponding to the continuous stress field are:

$$q^{(0)} = \sum_{\text{elements}} \int_v B^T \bar{\sigma}^{(0)} \, dv \tag{4.70}$$

(iv) If the $q^{(0)}$ is equal to f, the external loads, then the computed stresses are accepted and the iteration is stopped, otherwise, the load imbalance $q^{(0)}-f$ is used to iteratively improve the solution as explained in the following steps.

(v) Denote $d^{(0)}$ to be:

$$d^{(0)} = K^{-1}f \tag{4.71}$$

i.e. the displacement vector without any corrections.

(vi) The difference between $q^{(0)}$ and f; i.e. the load imbalance is denoted by $\Delta q^{(0)}$, i.e.,

$$\Delta q^{(0)} = f-q^{(0)} \tag{4.72}$$

(vii) Compute a correction displacement vector $\Delta d^{(1)}$ that corresponds to this load imbalance:

$$\Delta d^{(1)} = k^{-1}\{\Delta q^{(0)}\} \tag{4.73}$$

or

$$\begin{aligned}\Delta d^{(1)} &= k^{-1}\{f-q^{(0)}\}\\ &= d^{0}-k^{-1}q^{(0)}\end{aligned}$$

Since $d^{(1)} = d^{(0)}+\Delta d^{(1)}$

then:

$$\begin{aligned}d^{(1)} &= 2d^{(0)}-k^{-1}q^{(0)}\\ &= 2k^{-1}f-k^{-1}q^{(0)}\end{aligned}$$

$$d^{(1)} = k^{-1}\{2f-q^{(0)}\} \tag{4.74}$$

(viii) Similarly $d^{(2)}, d^{(3)}, \ldots$, etc. can be computed. For example, $d^{(2)}$ will be:

$$d^{(2)} = k^{-1}\{3f-q^{(0)}-q^{(1)}\} \tag{4.75}$$

(ix) If the load difference in two consecutive steps is sufficiently small the computed values for displacements, stresses and loads are retained and the iterations are stopped.

It should be noted that k is formed and reduced only once while

new load vectors are repeatedly formed and operated upon which makes this method computationally attractive.

Although this method was developed for elastostatic problems, the main idea can be applied to other similar problems. It is a good idea also to incorporate such a module in any of the well-known post-processors, so that, the computed stress field will retain continuity in cases it has to be so.

4.10.4 Hardware for Interactive Graphical Postprocessors

There are many different devices which can be used for graphical postprocessors. They differ in their complexity and cost. In its simplest form where no special hardware is available the use of a printer for alphanumeric "plotting" can be used to represent the FE model or for contour plotting. However, this type of "plotting" is useful only for fast checking of the FE model at low cost. Many problems are encountered in this type of plotting. First, the scaling of most printers is not the same in horizontal and vertical directions. Usually 6 lines per inch are printed in the vertical direction while 10 characters per inch could be printed in the horizontal direction which should be considered when scaling the x-y values. A second problem is the element sides. Since continuous lines are not possible on printers, the resulting drawing usually suffers distortions in boundaries which are not parallel to the x or y directions. If element numbers are to be printed inside centroids of elements, the resulting drawing will usually be too crowded. An example of an alphanumeric FE mesh plotting is that by Felippa [1972]. A similar routine is adopted by the TWODEPEP package [IMSL, 1983]. To have better plotting capabilities in FE post

processors, we need plotters, graphics terminal or workstations. These are briefly reviewed in the following subsections.

4.10.4.1 Graphics Terminals

It is a fact that the computer graphics terminals in use today use the cathode ray tube - CRT - as their displaying device. The main idea is that a heated cathode emits a high-speed electron beam into a phosphor-coated glass screen. The electrons energize the phosphor coating, causing it to glow at the points where the beam makes contact. So, by changing the beam contact points, its intensity and focusing, it is possible to generate pictures on the CRT screen. Two basic techniques are used to generate these pictures. They are known as (i) stroke writing or vector writing, and (ii) raster scan. In the vector writing technique the electron beam is operated much like a pencil to create a line segment on the CRT screen. The whole picture can be constructed out of a sufficient sequence of these straight line segments. In the raster scan technique, on the other hand, the screen is divided into a large number of discrete phosphor picture elements (pixels or pels). The matrix of pixels constitutes the raster. The intensity of a CRT screen usually ranges from the low resolution of 256×240 to the high resolution screens of 1024×2048 pixels. A refresh rate of 30 to 60 per second is usually used in these terminals. Three types of graphics terminals seem to be the most widely used: (1) Directed-beam refresh, (2) Direct-view storage tube (DVST) and (3) Raster scan (digital TV). A brief discussion of these types follows [Groover and Zimmers, 1983].

(1) Directed-Beam Refresh: These utilize the stroke-writing approach

to generate the image on the CRT screen. The image is regenerated many times per second in order to avoid noticeable flicker of the image. Because the image is being continually refreshed, selective erasure and alteration of the image is easily done. It is also possible to provide animation of the image with a refresh tube. This technology is considered as the oldest of the modern graphics terminals. They are referred to as vector refresh or stroke-writing refresh systems.

(2) Direct-View Storage Tube (DVST): In these displays an electron flood gun is directed at the phosphor coated screen which keeps the phosphor elements illuminated once they have been energized by the stroke-writing electron beam. The resulting image, is therefore, flicker free without the need to continuously rewrite the image. The main disadvantages of this technology is that erasing particular parts of the image is not possible unless the whole picture is regenerated. It also lacks animation, colour and the ability to use a light pen as a data entry device. The main advantage of this technology is the low price compared to other technologies and the ability of displaying large amounts of data without any flickering.

(3) Raster Scan Terminals: due to the recent developments in memory technology the raster scan terminals become a reality. To illustrate, consider a low resolution raster scan terminal of 256×256 pixels. This requires 64K bits for the memory if each pixel is either on or off i.e. 2-state pixels or black and white display. This memory, known as frame buffer or refresh buffer also, would be 1 megabits for a higher resolution screen of 1024×1024 pixels. If colour is required then more than 1 bit/pixel would be needed to store the grey level of each pixel. Usually 8-24 bits are required to have continuous grey level in coloured raster scan displays.

Table 4.20 shows the main characteristics of these three types of graphics terminal techniques.

Attribute	Display Type		
	Directed-beam refresh	DVST	Raster scan
Method of generation	Vector writing	Vector writing	Raster scan
Picture continuity	Excellent	Excellent	Good to excellent according to the number of pixels
Flickering	May occur for high data content	None	None
Grey levels	Many	Binary	Many
Colour capability	Yes	No	Yes
Animation capability	Yes	No	Yes
Selective Erase	Yes	No	Yes

TABLE 4.20: Characteristics of graphical displays

4.10.4.2 Input Devices for Interactive Graphical Postprocessors

Following our definition of the functions of a FE postprocessor, input devices are most required at the preparation stage of a FE model. As explained earlier, in order to construct a FE model and prepare input data sets for a FE processor, interactive techniques may be used. On the other hand, it is often desirable to be able to edit produced plots of a postprocessor before final output. Input devices are thus provided for the convenient communication between the user and the system.

In addition to the classical keyboard which exists with almost all types of interactive graphical terminals, other input devices are more convenient for the user-computer interaction in graphics. Keyboards are more convenient for commands and entering alphanumeric data, while picture handling is better done using other devices. Two main types of interactive input devices are in use: (1) cursor control devices and (2) digitizers.

(1) Cursor control devices (CCD): these devices are used to control the location of the cursor on the CRT. A typical use of CCD is to input a FE model interactively using a light pen, for instance. A more advanced application is to select an element from a menu of elements already displayed on the screen or printed on an electronic tablet. CCD include: light pen, joysticks, tracker ball (mouse), thumbwheels, electronic tablet/pen.

(2) Digitizers: these are large, smooth boards with an electronic tracking device. In fact digitizer use has been explained earlier and no need to be repeated here.

4.10.4.3 Output Devices for Interactive Graphical Postprocessors

Since the plots displayed on the CRT are "soft" plots it is usually desirable to have "hard" copy plots. This can be achieved by the use of pen plotters, hard-copy units, electrostatic plotters and computer output microfilm (COM) units among the most well-known output devices for graphics.

(1) Pen plotters: There are various types of pen plotters that differ in accuracy, repeatability, tolerance, pen thickness, pen types and precision. Two major types of pen plotters are the drum type plotters

and the x-y flat-bed plotters. In general, a plot produced by pen plotter is better than that displayed on the CRT which is limited by resolution and physical size. In the drum plotters, a round drum to which the paper is attached is rotated while the pen is sliding along a slide part. The relative motion between pen and paper is achieved by coordinating the rotation of the drum with the motion of the slide. Drum plotters are faster and virtually an unlimited length of paper can be used. In addition to that, they are less expensive compared to flat bed plotters. Flat-bed plotters use a flat drawing surface to which the drawing paper is attached. Parallel tracks are located on two sides of the flat surface. A bridge is driven along these tracks to provide the x-direction motion. Attached to the bridge is another track, on which rides a writing mechanism. Movement of the writing mechanism relative to the bridge produces the y-direction motion. These plotters are more expensive but more accurate compared to drum plotters. They are, however, limited in paper size in both directions: length and width.

(2) Hard-copy units: these machines are used to have an instant screen-image on paper. Sometimes it is called snap-shot of the screen. Most hard-copy units are dry silver copiers that use light-sensitive paper exposed through a narrow CRT window inside the copier. The main advantage of these machines are low price and speed. It is a matter of seconds to get a snap-shot of the graphical CRT. However, the produced plots are of poorer quality compared to those of pen plotters.

(3) Electrostatic plotters: they represent a compromise between hard-copy units and pen plotters. They are faster than pen plotters but slower than hard-copy units. They are also of less accuracy compared to

pen plotters but little better than hard-copy devices. A virtually unlimited length of paper can be used in electrostatic plotters as in the case of drum plotters. A limitation of the electrostatic plotters is that the data must be in the raster format in order to be readily converted to hard-copy using the electrostatic method. If the data is not in raster format, a type conversion is required which is called banded vector to raster converter (BVRC). This is a combination of hardware and software. Electrostatic copier consists of a series of wire stylii mounted on a bar which spans the width of the change-sensitive paper. The stylii have a density which differs according to the type of the copier. It can go up to 200 per inch. The paper is gradually moved past the bar and certain stylii are activated to place dots on the paper. The plot is, therefore, generated by coordinating the generation of the dots with the paper movement (advance).

(4) Computer-output-microfilm (COM) units: these units are used to produce the plots on microfilm rather than as a full-size drawing on paper. They are expensive and usually used in production-type CAD (Computer-Aided Design) systems.

4.10.5 Software for Interactive Graphical Postprocessors

In the previous part the hardware elements of interactive computer graphical postprocessors have been presented. In this part the software elements of these systems are discussed. The prime function of graphics software is to transfer data between the applications program and the display hardware. The FE processor produces, for example, the stress values at various nodes within the FE model. An application program may be used to produce the contour lines of constant stress. This

application program is usually designed to output data by issuing commands or by creating some sort of data structures that are used by the graphics software for the physical generation of the plots on the hardware devices. The applications-dependent part of the graphics software accepts commands from the applications program, or scans a data structure produced by the applications program and generates a description in two-dimensional space. Thus, in fact, there are three elements required to produce a graphical plot as a postprocessing function of a FE processor: (i) the graphics software, (ii) the applications program and (iii) the hardware for graphics. When designing a graphical software there are some general rules that should be considered [Newman and Sproull, 1982]:

1. Simplicity: The graphics software should be easy to use.
2. Consistency: The package should be operated in a consistant and predictable way.
3. Completeness: There should be no inconvenient omissions in the set of graphics functions.
4. Robustness: The graphics system should be tolerant of minor instances of misuse by the user.
5. Performance: Graphics programs should be efficient and the speed of response should be as fast as possible within the limitations imposed by hardware.
6. Economy: Graphics programs should not be so large or expensive as to make their use prohibitive.

In the following subsections a brief discussion of some of the most important problems in computer graphics software is given.

4.10.5.1 Representation of Graphical Entities

There are two approaches in representing graphical entities in graphics software. These are known as (1) display procedures and (2) graphical structures. In the display procedures approach, the application program generates the plot by issuing specific display commands. A full graph is generated by issuing a nested display procedure. To exemplify, assume that a FE postprocessor has to plot the FE model itself. In the display procedures approach it is then possible to define the various elements that have been used in the modelling. For example, it is possible to define a parabolic isoparametric element, a brick element or a shell element. The definition of any of these elements is actually some calls to locate points and connecting lines. Using these entities it is possible to build higher order entities using the original primitives in addition to the previously defined elements and so on. It is even possible to define complete substructures that can be used to produce a new structure. The main drawback of this approach is that it requires either language extensions or a special callable software library of graphical procedures. This in fact will not give clear line cuts between the application program and the graphical software, in other words, they will be intermixed. The other alternative of handling graphical entities is to represent them by data structures rather than procedures. In this case, the data structure is built using the facilities provided by the programming language. For example, a linked list can be used to represent graphical entities in a FE postprocessor. The actual display or plotting occurs by calling a system-display routine which scans the data structure. It is a common practice to store the graphical structures as display files. The main difficulty in this

approach is that the data structure used for graphics is limited by the available data structures supported by the host programming language. Since most FE postprocessors are programmed in Fortran which lacks the variety of data structures it implies that the use of display procedures is usually more appropriate in this case. However, if the Fortran extensions proposed earlier this chapter are adopted it is probable that using graphical structures may be much easier.

4.10.5.2 Programming Languages for Interactive Computer Graphics

There are three approaches used for programming languages in interactive computer graphics: (1) the use of a standard high-level language and a plotting library, (2) the use of a totally new programming language for graphics, and (3) the use of language extensions to a high-level programming language. In the first approach, the application program is written in a standard high-level language like Fortran, Pascal or Algol. The program calls a series of subroutines and functions using the standard calling procedures supported by the programming language in order to do the actual graphical outputs. This approach seems to be the one most widely used. This may be due to its simplicity and there is no need to modify the host language compiler. Certain graphical libraries are built which are callable from Fortran and Basic programs. These libraries are referred to as Host Computer Basic Software (HCBS). They are usually machine dependent. The second approach, building a totally new programming language, is rarely used. It can only be used to do the most primitive functions in graphics and is usually machine-dependent. A compromise between these two approaches is the use of language extensions. In this case the compiler of the

host language must be modified to take care of the graphics extensions. An example of the first type is the plot 21 package which is a library of Fortran subroutines used to drive Hewlett-Packard plotters. An example of the second approach is that by Kulsrud [1968] who devised a general purpose graphic language. An example of the third approach is that by Hurwitz et al [1967] who described graphical extensions to Fortran. They called it GRAF. In addition to these approaches there are several language-independent graphics packages that can be used for plotting of pre-prepared values in flat files. Although these packages do not possess the greater flexibility of the other approaches, they proved to be useful for many direct applications. An example of these packages is the GINO-F package developed at the Computer Design Centre at Cambridge [1976]. The main functions that should be supported by a graphics package are:

(1) Generation of graphic elements: dots, lines, etc.

(2) Transformations: to reposition a graphic element in different locations and orientations.

(3) Display control and windowing functions: to enable the user of zooming parts of the FE model, removing hidden lines, etc.

(4) Segmenting functions: to selectively replace, delete or modify portions of the model only.

(5) User input functions: to enable the user defining other parameters through input devices like giving the material properties in structural mechanics applications or issuing commands to the package itself.

It is possible to set an analogy between the hierarchy of programming languages and graphics software as shown in Table 4.21.

Programming Language	Graphics Software
Machine language	Graphics commands in hexdecimal codes
Assembly language	Mnemonic graphics commands
High level languages	Callable graphics libraries
Fourth generation languages	Graphics packages

TABLE 4.21: Analogy between programming languages and graphics software

4.10.5.3 Geometry Modelling

Geometry modelling is used to build a FE model of a problem interactively utilizing graphics terminal capabilities. It is concerned with the computer-compatible mathematical description of the geometry of the domain to be modelled. This mathematical description allows the displaying and the editing of the model on a graphics terminal easily. Typically, there are three levels of commands used in geometry modelling. The first level allows the creation of the basic elements e.g. points, lines and circles. The second one is used to handle these elements using standard utility operations like scaling or transformations in general. The third level is used to build the total model from the basic and the transformed elements. The wire frames method is usually used to represent the model. In this method the model is displayed and plotted by connecting lines. If the original model is a two-dimensional one the resulting wire frame model will be two-dimensional too. On the other hand three-dimensional geometries are represented by 3D wire frames. However, for complicated shapes wire frame models may not be adequate as such. This is mainly due to the excessive line interconnections. To enhance this, hidden lines may be removed. A more

recent and advanced technique is to use solid modelling to represent the object to be modelled.

Geometry modelling can be enhanced utilizing multi-coloured images. For example, a FE postprocessor can display the original structure in a colour superimposed by the deformed shape in another colour. If the boundary conditions of the problem are not the same then it is possible to display each type in a different colour. Different element types can also be displayed in different colours for ease of visualization. Colour graphics displays allow, no doubt, more information to be clearly given to the user. In a typical colour CRT three electron beams and a triad of coloured dots on the phosphor screen are used to provide the basic three colours: blue, green and red. By combining these three colours at different intensity levels a variety of colours can be obtained.

Another feature that can enhance FE postprocessing is computer animation. This is particularly useful in problems like dynamical analysis of structures, linkage mechanisms and many mechanical systems. A major requirement for computer animation is that redrawing must be done fast enough. Usually a workstation powered by local CPU is used in order to do most of the processing locally to relieve the host processor from this overhead.

4.10.5.4 Removal of Hidden Surfaces

As explained earlier, geometry modelling can be greatly enhanced if hidden parts are removed. Removal of hidden lines in wire frame models and of hidden surfaces in solid ones is one of the most difficult and time consuming tasks in graphical postprocessors and in computer

graphics in general. Many algorithms have been designed to do this task. They differ in complexity, speed and success. However, they share a common characteristic. They all use some sort of geometric sorting to distinguish visible parts from hidden ones. By geometric sorting is meant locating objects that lie near to the viewer and those that lie far away. The former will be visible and should be kept while the latter will be hidden and should be removed. There are two approaches for hidden surface elimination: object space oriented and image space oriented. In the object space oriented algorithms, concern is with the geometrical relationships amongst the objects in the model in order to determine which parts are to be removed. On the other hand, the image space oriented algorithms concentrate on the final image and determine which pixels will be visible. While object oriented algorithms are primarily used for hidden line removal, image space oriented ones are for hidden surface removal. The main difficulty with image space oriented algorithms is to handle all the pixels in the screen which might be as big as 2,000,000 in high resolution terminals. A survey of some of these algorithms can be found in [Sutherland et al, 1974]. Here two algorithms are presented one for hidden surface removal and the other for hidden line removal.

(i) The depth-buffer algorithm:

This is the simplest one among image space oriented algorithms for the removal of hidden surfaces. It can be considered as an extension to the normal frame buffer of an image where depth is considered and hence the name depth-buffer. The main idea is to keep a record of the depth of the object within the pixel that lies closest to the viewer. The intensity of each pixel is also determined to be used when displaying

the object. This algorithm can be summarised as follows [Newman and Sproull, 1982]:

(1) Set data structures: Int and Depth arrays each is array of n×m elements where n and m are the screen dimensions in pixels. For example a 2048×1024 pixels CRT will require the two arrays to be set each of that size.

(2) For every pixel on the screen (x,y) the elements: Depth (x,y) is set to 1 and Int (x,y) is set to the background value.

(3) For each polygon in the scene, find all pixels (x,y) that lie within the boundaries of the polygon when projected onto the screen.

(4) For each of these pixels do.

(5) Calculate the depth z of the polygon at (x,y).

(6) If z < Depth (x,y); this polygon is closer to the viewer than others already recorded for this pixel. In this case, set Depth (x,y) to z and Int (x,y) to a value corresponding to the polygon's shading. If z > Depth (x,y), the polygon already recorded at (x,y) lies closer to the viewer than does this new polygon, and no action is taken.

(7) Repeat steps 3 through 6 for all polygons in the scene. The solution is in the Int array.

The main drawback of this algorithm is the great amount of storage required to hold the two arrays "Depth" and "Int". To reduce this excessive amount of memory, image segmentation may be used which is more or less a substructuring technique. In this case, the screen can be divided into smaller windows and each one is processed in sequence. However, the processing time will be increased since each polygon will be processed several times. Despite that, coherence of neighbouring pixels and scan lines can be utilized to greatly speed up the processing.

Examples of algorithms which utilize such a property are: the scan-line coherence [Watkins, 1970] and area-coherence algorithm [Warnock, 1969].

(ii) Janssen's hidden line algorithm:

This simple and efficient hidden line removal algorithm was developed by Janssen [1983]. Its main advantages are the minimal computer storage required and being easily incorporated in general FE graphical postprocessors. The basic idea is to compare each unique line to be plotted against potential planar surfaces which may hide all or a portion of the line. Each planar element is decomposed into triangles. Triangular area coordinates are then used to determine the intersection point of the line and the triangle. Using area coordinates, no angle determination or trigonometric functions evaluation is required. This results in speed of execution. The algorithm can be summarized as follows:

(1) Preliminary operations:

Transform all nodes to be plotted in the FE model from the (x,y,z) Cartesian coordinate system to the plotter coordinate system (R,S,T) where the picture plane is the (S,T) plane and the viewpoint is on the positive R-axis. Depending on the form of this transformation a perspective or orthographic view can be obtained. These two coordinate systems are shown in Figure (4.58).

(2) Loop on each line in the mathematical model.

(3) If the line is $\perp$ to the view plane goto (2) directly to consider the next line.

(4) Loop on each polygon in the mathematical model.

(5) If the polygon $\perp$ the view plane goto (4) directly to consider the next polygon.

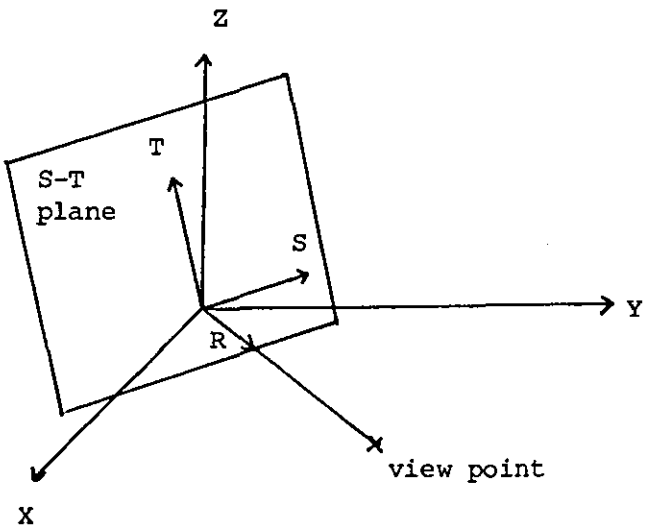


FIGURE 4.58: Cartesian and plotter coordinates

(6) If the line is totally in front of the polygon or totally in one side of the polygon (above, below, right or left) goto (4).

(7) The line is interesecting the polygon. If the polygon is quadrilateral, it is divided to two triangles and for each one the hidden portion of the line is determined.

(8) If the line is totally covered goto (2) else goto (4).

(9) After all polygons are considered, plot the visible portion of the line and goto (2).

4.10.6 Design of User Interface in Graphical Postprocessors

Success of FE graphical postprocessors is highly dependent on the user interface. Software acceptability is, in fact, a function of its

user interface. Bad user interface not only makes software operation difficult, but also may lead to inefficient, erroneous and unreliable results. Moreover, user training becomes more difficult in case of bad user interface. It is, therefore, very important to pay careful attention to the design of interactive user interfaces.

It is possible to divide the user interface into four major components [Newman and Sproull, 1982]:

(i) The user's model;

(ii) The command language;

(iii) The information display; and

(iv) The feedback.

In what follows a short discussion of each of these components is given.

4.10.6.1 The User's Model

In order to build a good user interface it is very important to have a conceptual understanding of the software and to convey this understanding to the user. Very short notices to the user, a thing which is frequently noticed in bad user interfaces, tend to discourage the user from developing any understanding of the software he is using. In designing the user's model it is important to consider:

(i) The user's model is a mental model and acts as a framework for the development of strategies for operating the software;

(ii) The user's model should employ familiar concepts to the user. For example, for a structural engineer concepts like: beam, column, slab,... etc. are among his daily "language". The model should use these concepts in order to be more intuitive and easier to learn and use.

(iii) There are some constraints imposed by the available hardware and software that must be taken into consideration.

It is customary to represent a user's model as a set of objects and a set of actions which the user can apply to the objects. Each object is an item of information over which the user has some control. For example, a set of objects can be an element, node or support in a structural FE postprocessor. A set of actions can be: delete, move, copy or rotate. Objects can be classified into two classes: those which are intrinsic to the application and those whose purpose is to assist in the control of the program.

4.10.6.2 The Command Language

In contrast to batch-oriented command languages, graphical interaction is much complicated. It involves the use of different devices, many functions must be supported and the objects are generally a mixture of alphanumeric and graphical data. A good command language should have a clear syntax and semantics which reflect the user's model. The key issues which must be considered in designing a command language for graphical postprocessors are:

(i) Command modes: It is highly desirable to have one command mode only. If this is not possible, then keeping these modes to the absolute minimum should be tried. To exemplify, it is customary to find in text editors two modes: - insert mode and edit mode. It is also possible to have the same command abbreviation with more than one meaning according to the mode into which it was issued.

(ii) Selection sequence: Command operands must be specified either

before or after the specification of the command. In each case, the resulting sequence should not be ambiguous.

(iii) Command abort mechanism: Some commands may be containing other commands e.g. macro commands contain several commands. If for some reason the macro-command was aborted, then a correct and clear mechanism must be available to handle this action and provide the user with clear information about the current status of the program in this situation.

(iv) Error handling: The command language must handle erroneous data or meaningless commands supplied by the user. The user must be informed if such a situation occurs and clear error messages with possible correction actions should be displayed.

The design and structure of the command language depends on the hardware to be used in the user dialogue. For example, a keyboard-based dialogue will normally be done through the use of short alphanumeric commands typed on the keyboard. It is usually started by a computer prompt to accept the user command, validate it and then do the required actions accordingly. An example of this approach may be the program development system - PDS by the author [Sharaf Eldin, 1985b]. For simple graphical interaction it is possible to use the function keys which are usually accessed and can be modified by the user. For more sophisticated operations menu-driven command languages can be used.

4.10.6.3 Information Display

The main advantage of using graphics terminals are their abilities in displaying information graphically on the CRT. The power of the graphical display is not only in displaying the results on the screen

but also in its flexibility and speed. Many facilities must be supported in order to have flexible information display. Among these are the primitive parameter selection like: line type, colour, position of text, font type for alphanumeric texts,... etc. More advanced operations involved in information display may be zooming and pan. A careful layout of the CRT which can be controlled by the user is imperative in order to have a successful display of information.

4.10.6.4 Feedback

The purpose of feedback is to supplement the response provided by information display so as to permit more effective interaction. It is possible to classify the feedback to three components or types:

(i) Feedback received from the command interpreter. This will inform the user whether his command has been accepted or not. If it is accepted, then it should give the user an indication about the execution progress or any error conditions that might occur. Although printed text messages on the screen are the standard feedback received from the command interpreter, it can be enhanced by audio effects (beeps) or cursor blinking at the erroneous place or both.

(ii) Feedback from the application database, principally for selection feedback. This usually is not easy to implement. Moreover, it may take a relatively long time in order to be done. For example if the user selects a particular element and ordered the system, by command language commands, to place it at a particular place, it may take some time before being fed back by the new image on the screen.

(iii) Feedback from the system which is cursor feedback or character echoing. This is very useful and very easy to implement. Echoing the commands, in particular critical commands, for user verification is very effective in perfecting user interface. To exemplify, if a user starts modifying a FE model on the screen and then he issued a quit command, then a good feedback is to repeat the same command and a confirmation should be requested from the user. Modifying the cursor position must have immediate feedback with the cursor moved to the new location.

4.10.7 Examples of FE Postprocessors

There are many types of FE postprocessors. They are primarily graphical postprocessors and the vast majority of them are for structural analysis and mechanics applications. They differ in their hardware/ software requirements. Most of them are based on either mainframe or super mini-computers although some new programs are available on micro-computers. In what follows some of these programs are briefly presented. They are chosen to represent a mainframe, mini-computer and micro-computer implementation of FE postprocessors.

(i) Kaldjian [1977], developed MSAPLOT which is a postprocessor for the MSAP FE processor which has been presented before. It is a graphical postprocessor that can be used to display the initial FE model, view the deformed structure and producing ink drawings of both on a plotter. It is a stand-alone program that utilizes the same inputs as MSAP itself. The outputs of MSAP are passed to MSAPLOT to plot the deformed shape. MSAPLOT is implemented on a mainframe computer and requires graphic display terminals. It utilizes two software plotting libraries developed at Michigan University in USA. The screen is divided into two display

regions. Region 1 is used as a reference picture region where the first display is drawn. Region 2 is the main display area where the user can blow-up part of the structure or rotate the structure to obtain a different view with region 1 coordinate axes as reference. After display of the initial FE model, commands can be issued to allow the zooming of particular regions of the structure, rotate the view angle to get a 3-D plotting if required, obtain hard copy plots of the displayed pictures and so some other utility operations like filing of plots.

(ii) Kalkani [1976] presented a contour plot program for stresses in 2-D slopes. This program is one of the rare postprocessors for geotechnical problems where the stresses calculated by a 2-D FE analysis for stresses in rock slope are plotted prior to locating critical regions for the slope stability. This is particularly important for highways design and construction. Input data to this program are the FE model topology and the computed principal stresses at centroids of the FE elements. The program is mini-computer-based and produces hard copy plots only.

(iii) Kamel and McCabe [1976] developed the GIFS system which is a graphics oriented interactive finite element package for time-sharing. This comprehensive system is, in fact, a pre- and post-processor for FE analysis. Moreover, it can be used for solving relatively small size FE problems in structural analysis and mechanics. It is mini-computer-based that utilizes overlaying techniques to minimize the main memory requirements. It supports graphics terminals in addition to hard copy plotters. The GIFTS system has been upgraded several times since then. Utilization of digitizing techniques has been previously discussed [Kamel and Navabi, 1980]. More recently, a smaller version which can

be implemented on a micro-computer has been released [Kamel et al 1985].

(iv) Stafford [1983] utilizes a simple technique for complementing the FE postprocessing of the large-scale program ADINA. This program, ADINA, is a mainframe FE processor for non-linear analysis of structures. A low-cost means to make simple postprocessing of the outputs of ADINA (and similar programs) is to move parts of their outputs to a micro-computer and use one of the many standard spread sheets available for minimal cost. Stafford used the well-known Visicalc software for tabulation and simple graphical representations of the results. The hierarchy of computations can be summarized as follows:

(1) A mini-computer is used for solid modelling and preprocessing of the FE model. The model data are then transferred to a mainframe computer for processing.

(2) A mainframe computer is used to do the FE analysis. Parts of the outputs are downloaded to the micro-computer storage devices (floppy diskettes or hard disk).

(3) The micro-computer is used for low resolution graphics and data tabulation.

As many graphics software are now available on micro-computers it seems that the more advanced graphics software like Auto-CAD will be more useful than spread sheets.

(v) Sharaf Eldin [1983c] developed IFEPLOT which is a postprocessor for the IFEP FE processor which has been presented earlier. It is a graphical postprocessor which can be used to plot the original FE model and the contours of the hydraulic head for flow in aquifer problems. It is programmed in Fortran and uses the PLOT21 basic plotting software to support the operation of the small size x-y plotter. Linear interpolation

is used and 10 contour lines can be plotted.

4.10.8 Recent Trends in Graphical Postprocessors

Although computer graphics were first recognised as a man-machine interface in the mid-60's, their applications in FE postprocessing can be dated to the mid-70's, where general graphical postprocessors started to exist. Computer graphics became increasingly sophisticated, allowing more and more postprocessing of FE outputs. However, the needs of engineers and scientists still exceed the abilities of current computer graphics systems. To exemplify, an aeronautical laboratory may take several days of supercomputer processing time to solve hypercomplex problems and many months to interpret the generated data. The requirements of more powerful computer graphics are particularly noticed in areas of: man-machine interface, computational geometry, animation, workstation hardware and more important: standardization and integration. In this section some of the recent advances in FE postprocessing and computer graphics in general are reviewed. The treatment follows the arguments of Plunkett [1985] and Kirk [1986].

(i) The display of FE model of curved surfaces in 3-D interactively involves the 3-D transformations, shading, hidden surface removal among many other computations. These computations are heavy and traditionally they have to be done on the host computer which is a mainframe or a super-minicomputer in order to have an acceptable turn-around time. In earlier graphics systems, it was a common practice to let the user interactively create and manipulate the model as a wireframe and then have the host computer to make a solid image in the background processing mode for later displaying. In the new generation of graphics workstations,

the display controller has to remove the hidden surfaces and shade the surface polygonal patches thus relieving the host to do other tasks of graphics display that are beyond the capabilities of the display controller like decomposition of the surface into polygonal patches. As microprocessor technologies improve, more and more tasks can be performed locally by the display controller. Ultimately, a display controller will be in the future capable of handling all display-related processing.

(ii) Many new software systems and hardware devices for graphics are introduced every year. There is no comprehensive general standards for graphics. This has several consequences. First, the software developed for one hardware device cannot be moved easily to another one without extensive modifications and sometimes rewriting the software becomes the only alternative. In otherwords, there is a lack in graphics software portability. Second, the same argument applies to programmers. It is a fact that graphics programmers spend more time in learning new graphics systems rather than developing applications. Most graphics programmers are still computer specialists rather than specialists in other fields. Thus introducing graphics standards will not only ensure program portability but also programmers portability. Third, different devices of similar capabilities usually do not display the same drawing in an identical manner, i.e. there is device dependence for the produced graphics.

The first step towards graphics standardization was realized during the late 70's where the graphics problem was separated into two distinct sections: modelling the problem and drawing that model. Later on two major efforts towards drawing standardization emerged. The first is the core graphics system by the Graphics Standards Planning Committee (GSPC)

of the ACM special interest group on computer graphics in the USA. The second is that by the West German standards organization which is called the Graphical Kernel System (GKS) which has been submitted to the International Standards Organization (ISO).

At present, there are five "general standards" on graphics:

(1) Graphical Kernel System (GKS) which is a set of software subroutines that describes the interface between an application program and a set of graphics devices.

(2) Programmer's Hierarchical Interactive Graphics Standard (PHIGS) which provides some additional capabilities not covered by GKS. PHIGS is concerned solely with 3-D data and allows more interactivity in viewing dynamic changes in a model. It provides for a hierarchical description of separate components of a single object, which can be useful in the construction of complicated drawings.

(3) Virtual Device Interface (VDI) specifies the interface between device-independent software and device-dependent code. It specifies an interface to a virtual ideal device. Conversion to a real device is then done either in the host computer via a device driver or in the firmware of the device itself. The VDI is scheduled for adoption as a standard during 1987 [Plunkett, 1985].

(4) Computer Graphics Metafile (CGM) which can be considered as the counterpart of VDI. It allows drawing descriptions to be stored in a device-independent manner.

(5) Initial Graphics Exchange Specification (INGES) which was originated for standardization of Computer-Aided Design/Manufacturing (CAD/CAM) databases which allows the transfer of CAD/CAM files between different systems.

4.11 SPECIAL TOPICS IN COMPUTER IMPLEMENTATION OF FE

In this section some special topics in the computer implementation of FE will be discussed. The discussion is by no means complete but the main reason of adding this section is to complete the whole picture which has been presented within this chapter. It also reflects the author's views of some of the areas that will be of active research in the coming years in the computer implementation of FE.

4.11.1 FE on Parallel Computing Systems

Due to the limited rate of speedup of serial processors from technology alone and obtaining better cost performance, parallel computing systems have recently emerged. In its most general sense, the term parallelism refers to the simultaneous execution of two or more operations by a computer system. So, in fact, parallel computing systems try to solve the bottlenecks found in classical Von Neumann systems. In Von Neumann computer systems the programs are stored with data in the memory unit. One instruction at a time can be executed in a sequential fashion. In parallel systems this is solved by different architectures. A frequently used classification of parallel processors is that shown in Table (4.22).

Instruction Stream / Data Stream	Single	Multiple
Single	SISD	MISD
Multiple	SIMD	MIMD

TABLE 4.22: Classification of Parallel Systems

where the terms:

SISD stands for Single Instruction Single Data Stream which is the classical Von Neumann architecture.

SIMD stands for Single Instruction Multiple Data streams

MISD stands for Multiple Instructions Single Data Stream

MIMD stands for Multiple Instructions Multiple Data Streams.

Figures 4.59 to 4.62 show the structure of these different architectures. Note that a processor (p) stands for central processing unit (CPU), i.e. the arithmetic logic unit and the control unit.

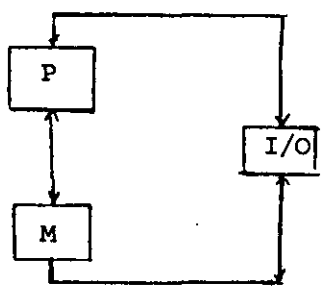


FIGURE 4.59: Von Neumann Model

1 Memory Unit (M)
1 CPU (P)

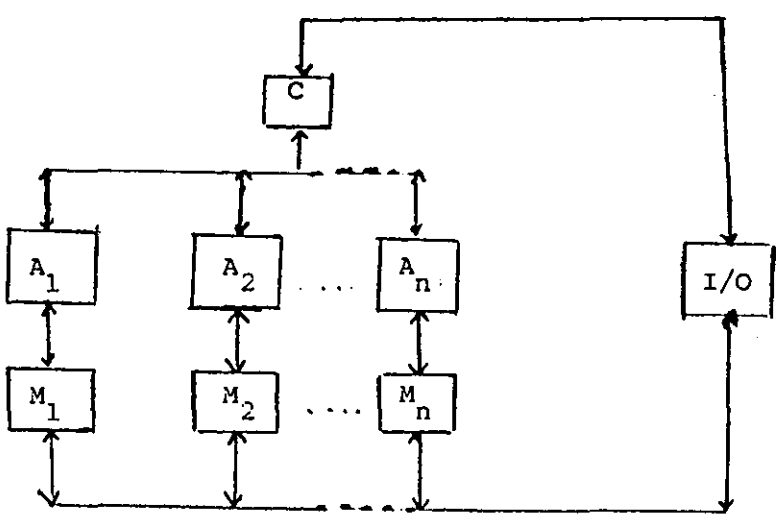


FIGURE 4.60: SIMD Model

n Arithmetic logic units (A)
n Memory units (M)
1 Control unit (C)

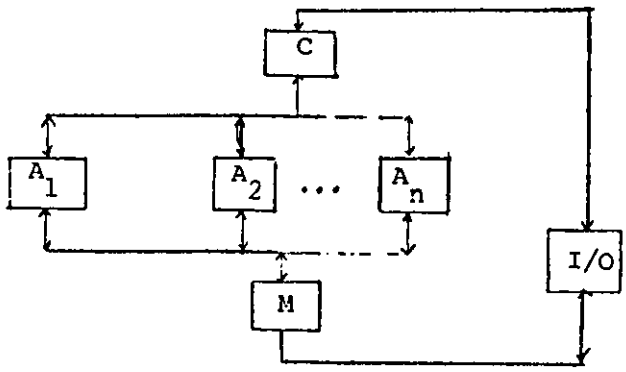


FIGURE 4.61: MISD

n Arithmetic logic units (A)
1 Memory unit (M)
1 Control unit (C)

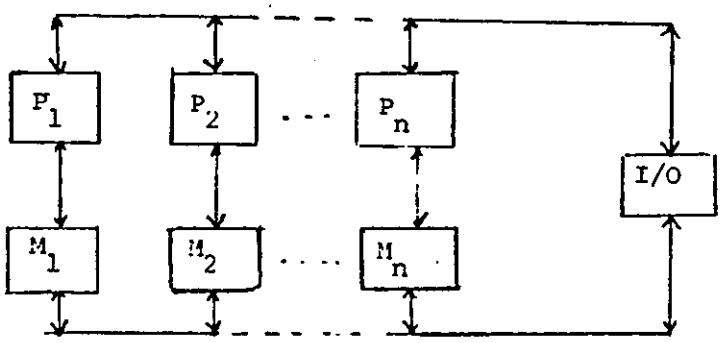


FIGURE 4.62: MIMD

n Processors (P)
n Memory units (M)

SIMD systems were designed in the early 70's to achieve high computing speed without the need to replicate the relatively expensive

control units [Vemuri and Karplus, 1981]. With the introduction of the low-cost mini- and micro-computers and the drop of cost of hardware, MIMD systems have been realized. In pipeline processing, a sequential computational procedure is broken down into stages, the separate hardware units are provided for carrying out the computations of each stage. When the pipeline is full, each hardware unit is engaged in processing different data, and as each computes its task, it passes the results of its computation to the next unit.

Since the FEM is a technique which depends on computers for the solution and which needs heavily computations for large-scale non-linear problems, it is natural to utilize these new architectures in the computer implementation of FE. In fact, the need to solve large-scale computational problems was behind the motivation for the development of parallel computing systems. In what follows, some of the most well known efforts in this context are presented.

(i) Jordan and Sawyer [1979] proposed a multi-microprocessor system for the solution of structural analysis problems by finite elements. They called this architecture "the finite element machine". This machine is a MIMD system which utilizes the current single chip microprocessor technology. Each processor in the machine represents a node in the FE model. The processors are arranged in an array and the FE model is mapped onto the hardware. The processors will communicate data by explicit transmissions through a multiplexed bus connecting all processors. Thus processor i, for example, can communicate to processor j in the array. The major steps in the FE analysis are the formation of the stiffness matrix K and the solution of the resulting equations. Therefore, new techniques must be used to exploit parallelism in their

computations. Indeed many efforts have been done for the solution of linear equations on parallel systems e.g. Evans [1982]. It seems, however, that less attention has been paid to the parallel algorithms for stiffness matrix computations. In the finite element machine the computational work is partitioned into the processors and the element stiffnesses are performed in a way isomorphic to the interconnection of nodes by elements of the structure. Each individual processor is assumed to be 16 bit, single chip microprocessor with hardware multiply and divide, coupled to a read/write memory, RAM for programs and data and a ROM for the startup algorithm. Careful mapping of these processors on the nodes of the FE model can result in minimal use of the connecting bus which has a great influence in the total system performance.

(ii) Zave and Cole [1983] presented an experimental implementation of a design for the adaptive parallel finite element system. The implementation was used to simulate the performance of this design on several microprocessor-based multiprocessor architectures. Thus this implementation is essentially a MIMD architecture. Each processor is assigned a process and the various processes of a system can communicate only by sending messages. This process-level parallelism is completely independent of instruction-level parallelism which is used by pipeline processors, for example. However, it is possible to exploit at the process-level the instruction level parallelism. Experiments were conducted using the FEARS (Finite Element Adaptive Research Solver] system of the University of Maryland, U.S.A. However, since the machine itself was not built, they used a technique to evaluate the anticipated time on the proposed architecture. This technique is to measure each

task time run on the available mainframe computer - a Univac machine - then multiply this time by a constant factor to reflect the relative slowness of the microprocessor. These constant factors were computed by counting and grouping machine instruction and determine conversion factors for each group and then computing a weighted average of the conversion constant. Of course, these factors are only rough approximations but there seems no other alternative except the actual running of the code on the target machine itself. The presented results show a speedup factor that ranges from 4.89 to about 1. In some test problems the speed factor was even around .99 (i.e. negative gain). The reason for these modest speedups was probably due to the computation and waiting times of the processors.

(iii) Strohkorb and Noor [1984] used a different approach based on a mini-computer-array processor system for the non-linear FE analysis of structures. The idea behind this choice is that its elements are readily commercially available and the system can be built by much less cost as compared to the class of supercomputers like Cray or CDC Cyber 205, for example. A Prime 750 is used as the host computer with AP array processor as an attached processor. A software simulator residing on the Prime is employed to assess the performance of the floating point system AP array processor. Array processors are high-speed special-purpose computational devices which can perform repetitive computations on well-structured data sets at effective speeds far beyond those achieved by current minicomputers. The array processor used in this study is AP-120B. The main features of which are [Karplus and Cohen, 1981]:

(1) It is designed and accessed as a peripheral to the host minicomputer.

(2) It achieves high performance through both parallelism and pipelining.

Its arithmetic unit contains one floating point adder and has two stages. It also has one three-stage pipeline multiplier. Each can perform up to 6 million floating point operations per second (MFLOPS). Practically a maximum speed of about 4 MFLOPS only can be realized as compared to a theoretical maximum of 12 MFLOPS if both pipelines are always full.

(3) It can be programmed in Fortran or assembly languages to perform many computational tasks. Parallelism on the AP-120B can be exploited only by the software. Thus, the programmer (or the compiler) must explicitly code all parallelism into the program. However, extensive libraries of micro-coded routines are supplied to support software development on this device.

The host computer, Prime 750, is a supermini-computer with 8M bytes of central memory. In this machine resides two distinct operating systems: the Primos operating system of the host computer and the APEX which the array processor executive uses to drive the AP-120B.

Two benchmark problems were used to assess this architecture. The execution speed using simulation was 5.2 and 9.9 times faster for the two problems compared to the sequential processing. The major hardware characteristics of the system that affect the speed are:

(1) Virtual memory on the host computer.

(2) Pipeline processing in the AP

(3) Parallel Processing achieved inside the AP

(4) Distributed processing where computationally heavy portions of the program are allocated to the AP, while other tasks like data management, AP control and user interface are allocated to the host.

On the other hand, the major software characteristics that must be noticed

to get benefits of this hardware are:

(1) Proper selection of the computational algorithm.

(2) Minimization of I/O operations on the host and between the host and the AP. Techniques like buffering and merging smaller arrays into larger ones can be useful in this context.

(3) Vectorization of the chosen numerical algorithm.

(iv) Romo and Burns [1986] used the Alliant FX/8 parallel processing mini-supercomputer system. In this system up to 8 processors used for computational work and called computational elements (CE) can be connected. In addition to that, up to 12 interactive processors (IP) can be used for handling user jobs, system I/O and other operating systems functions. Synchronization is achieved by built-in hardware concurrency control bus. A CE simultaneously supports up to five instructions in different execution stages in its pipeline. It can reach a speed of up to 11.8 MFLOPS. The FX/Fortran compiler examines Fortran-77 programs and compiles them to work with up to 8 general purpose vector processors operating in parallel. It also does code vectorization. One of the best features of the FX/Fortran compiler is the handling of nested loops which is normally a common bottleneck on sequential machines. The compiler translates these loops such that the innermost loop runs in vector mode and the next outer loop runs in concurrent mode. This is termed as Concurrent Outer, Vector Inner (COVI) mode. For example, in the following code, the J loop iterations run concurrently while the I loop iterations within each J loop are vectorized:

```
      DO 100 J=1,M
      DO 50 I=1,N
 50   A(I,J)=A(I,J)+P
100   CONTINUE
```

COVI implementation for nested loops is of great advantage when solving simultaneous linear equations which in turn are among the most time consuming tasks in FE analysis. The COVI mode operates most quickly when the number of iterations of the inner loop is a multiple of 32 which is the maximum length of a vector on one CE and the number of iterations of the outer loop is a multiple of the number of the available CE's.

Some experiments have been performed to test this architecture. The Abaques FE analysis program was chosen and modified to take advantage of the FX/8 hardware. Also matrix factorization problems were considered. The experiments were done using different number of CE's. They concluded that for an eight CE's with only 10% serial code, the best that can be expected is a five-fold increase in processing speed.

4.11.2 Data Base Technology for FE Software

The concepts of database management systems (DMS) has grown during the 70's in business-oriented applications. These concepts, however, were not realized in scientific computation except in the late 70's. This may be due to the sheer growth of large-scale codes, the appearance of integrated program networks that share a common project database and the introduction of interactive CAD/CAM systems. It is a fact that many large-scale computational software, in general, and FE software in particular have a good deal of data management. As a FE program increases in size and its capabilities, the data management portion becomes relatively more important in relation to the original processing

functions. A large-scale FE software is usually developed by a group of people working together. It usually contains many modules and independent programs that need to be interfaced. This interfacing can be done using any of the following approaches or a combination of them [Felippa, 1979]:

(i) Direct link approach:

In this approach programs communicate directly and there is one interface only for every possible connection path.

(ii) Super-executive approach:

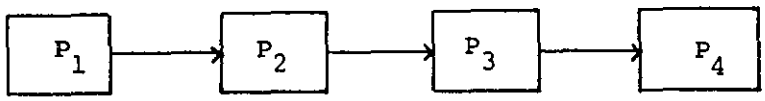
The components are linked to an executive program that functions as an executive or supervisory control program. This organization generally results in a tightly coupled program network.

(iii) Database approach:

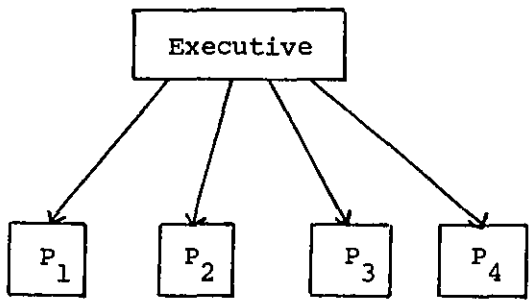
The components communicate through a database management system. Streams of data flowing between components go in and out of the database through the DBMS. This organization generally results in a loosely coupled programs network as each one can retain his own structure and control.

These three approaches are shown schematically in Figure 4.63.

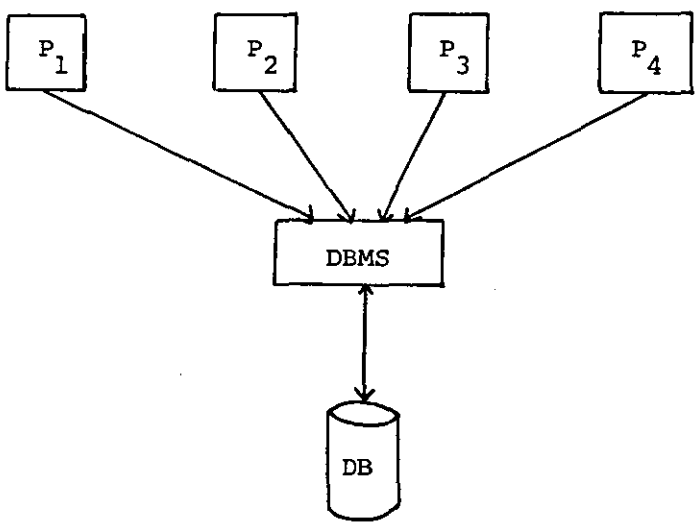
It seems that the current trend in large-scale FE software is being moved towards the database approach. However, since many well established software had been developed in the past and is still in use in different places, many of the current software will continue to survive despite their inflexible interfaces. There are 3 different approaches for organizing the data model in a DB system. They are: the hierarchical approach, the network approach, and the relational approach. Principles and details of these approaches can be found in



(a) Direct link structure



(b) Executive-based structure



(c) Database structure

FIGURE 4.63: Interfacing programs network

standard textbooks, e.g. [Date, 1982]. One of the first engineering analysis systems which uses the DBMS approach is the ICES [Christ, 1982] which is "Integrated Civil Engineering System". The system was originally

developed at the MIT during the mid-60's as a structural analysis program. Many developments and enhancements occurred since then and now this system is one amongst the most comprehensive and popular systems. ICES is a supervisor program for its subsystems. The user communicates with a subsystem through a problem oriented language composed of words and phrases based on the normal engineering terminology. ICES uses fully integrated super-elements [Jacobsen, 1983].

As the relational approach in DBMS is getting more popular some new relational-based systems are being developed. It is possible, for example, to define a FE data model in the following sets of relations:

Relation nodes≡{node number, x,y,z}

Relation elements≡{element number, element type, material type ID, node i, node j, node k}

Relation materials≡{material type ID, E,ν,α}

Relation constraints≡{node number, prescribed x-disp, prescribed y-disp, prescribed z-disp., prescribed x-rotation, prescribed y-rotation, prescribed z-rotation}

Relation loads≡{node number, load value, load type ID, load direction}.

One of the first projects to assess the use of a relational engineering data management system by integrating various application programs is the Pride system developed at the NASA Langley Research Centre in the U.S.A. [Blackburn et al, 1982]. The system can do FE analysis with pre- and post-processing. Since the engineering analysis/design task is usually an iterative one where a preliminary design is proposed, then analysed and then the design is modified and a re-analysis is done and so on, it is important to provide the engineer with a private database to do the initial work and at the end of the design these data are then

moved to the global database. In the Pride system the private data areas reside in various mini-computers and are generally associated with specific functions, and often specific engineers. The global shared area contains the data necessary to form a model (e.g. of an entire aircraft).

4.11.3 Standardization for FE Software

Because of the long time needed and the high cost required to develop FE programs, many FE users try to use available packages to solve their problems. However, available packages are developed by several organizations and are of varying reliability. The lack of standards for FE codes makes the FE user in the hands of the software developer. Most of the test problems in many cases which have been used to check the commercially available FE packages are more or less "trivial"problems and they are usually supplied by the developer himself in the so-called "Automatic Test Problems - ATP". Up till now there have been no FE standards that have to be followed by FE developers. However, some trials are initiated and may lead to set specifications for FE software. Of course some people are against the FE standards, while the others are supportive of these standards. Their arguments can be summarized in the following.

(i) Against:

- Standards may impede research and development.
- Standards are difficult to set and to maintain.
- Standards could cause great hardship if adopted as law and used to avoid a meaningful model validation effort.

(ii) With:

- No FE code is bug free and it is not possible for every user to check the code validity in all aspects.

- Standards can help in training and maintaining of software.
- Standards will improve communications among the FE community and unify the used terminology.

As in other similar cases, the key to successful FE standards is to determine general sets of rules to be followed, determine standard benchmark problems that are to be passed by the candidate code and to define certain rules for terminology and documentations. Meanwhile, the standards must be flexible enough to allow the greatest possible freedom and initiation within the specified framework. Fong [1984] surveyed the attempts done to set standards for FE codes. The major noticed efforts are those by the American Institute of Aeronautics and Astronautics (AIAA) in the USA, the National Agency for Finite Element Methods and Standards (NAFEMS) in the U.K. and the Japan Society of Mechanical Engineers (JSME) in Japan. These efforts are briefly reviewed as follows:

(i) The AIAA program was initiated during 1982 as a sub-committee of the AIAA. The members of this sub-committee started their work by proposing developing standards for linear general-purpose codes. Some FE benchmark problems were developed. This set includes: straight cantilever beam, curved beam, twisted beam, rectangular plate, thick-walled cylinder and spherical shell. The patch tests are performed for different element types and the set of problems was tested with these element types for particular cases of loading. The beam, membrane plate, bending plate, shell and solid elements were chosen. These problem-element combinations were tested on the well known FE program MSC/NASTRAN. The results show that these elements do not perform well in every test.

(ii) The NAFEMS was formed by the U.K. Department of Trade and Industry in 1983. The aims of the NAFEMS are to:

(1) Set FE standards and testing procedures.

(2) Coordinate and sponsor evaluations and studies of codes.

(3) Create a forum for users, developers, universities, government, and research organizations.

(4) Establish a central office. This has been located at the National Engineering Laboratory in East Kilbride, Glasgow.

(5) Develop and maintain databases for FE systems and users. This will help users in searching these databases for information on code capabilities.

(6) Consider the legal implications of using FE codes.

(7) Publicize requirements for education and training.

Documents that are near completion or already produced according to Fong include: Guildelines to finite element practice; NAFEMS Benchmark Tests and Finite Element Primer.

(iii) The JSME participated in two benchmark tests to compare nonlinear general and special-purpose FE as well as finite difference (FD) codes. Each problem was tested against 10 of the well-known FE and FD codes. The results of these tests showed that for impact problems under step loading, explicit time integration scheme is more popular than implicit schemes. An important result of these tests is that a great discrepancy in results occurs when the FE or FD model composed of axisymmetric body of several materials with quite different Young's moduli and more efficient time-marching schemes are needed.

Despite these efforts, it seems that the best chance of success for FE standards are those for linear static problems as a starting

point. Upgrading to other problems may be done after community acceptance of the first standards. However, it seems that it will not be before several years that the first complete standards for FE codes will be born.

4.12 SELECTION OF FINITE ELEMENTS SOFTWARE

4.12.1 Introduction

Many finite element packages (FEP) are now available in the marketplace which can be used to solve a fairly wide spectrum of applications. Moreover, many new computer codes are developed every year. The sophisticated user of FE software is faced with a flood of packages amongst which he tries to select the one that best fits his requirements, subject to some constraints. Since the cost of such software is usually high, it is wise to do a careful examination of the alternative FEP to choose the best of them. Such a complex decision problem can involve multiple conflicting objectives. It is often true that no dominant alternative will exist that is better than all other alternatives in terms of all of these objectives. Consequently, we make use of decision methods to evaluate the relative ranking of the available alternatives amongst each other. There are several methods used for decision making. It is possible to classify these methods into three major categories: (1) Deterministic approaches, (2) Probabilistic approaches; and (3) Fuzzy set and multi-attribute utility approaches.

In the deterministic approach, numerical ratings are assigned to the considered variables and a final "measure of merit" is formulated for each alternative. Such a "measure of merit" can be a simple average or a weighted average among other measures. In order to measure the sensitivity of the final measure of merit to each variable, the assigned value of variables are changed one at a time and the effect of each individual variable on the final result is studied.

In the probabilistic approach, probabilities are assigned to each variable which reflects the relative certainty of giving the

estimated outcome. Usually simulation and Monte Carlo techniques are used for complex problems. A more detailed treatment of these methods can be found in [Raiffa, 1968 and Cornell, 1980].

More recently, two new approaches have been introduced in decision making, the fuzzy sets approach and the multi-attribute utility analysis. Although the theoretical foundations of the fuzzy sets was introduced by Zadeh [1965], its application in engineering analysis is very recent [Brown and Yao, 1983]. The same is true for the multi-attribute utility analysis.

In the following sections a novel approach based on simple matrix operations in addition to the fuzzy sets and multi-attribute utility approaches are utilized in order to give a framework for selecting the 'best fit' finite element software based on quantitative approaches. These ideas in fact have been introduced in [Sharaf Eldin and Evans,1986].

4.12.2 Attributes of Finite Element Packages (FEP)

The definition of the suitable attributes of a FEP and their relative importance depend on the function for which the package is to be used. To illustrate, consider the availability of the source code of the FEP. This is considered much more important if the FEP is to be used for research and educational purposes. However, its importance will be less if the FEP is to be used in a business environment. Here we list in Table (4.23) the most relevant attributes and sub-attributes for a general purpose FE package used in the practical analysis of structures. We notice that all attributes are decomposed into sub-attributes, however, it is not necessary to breakdown all attributes to sub-attributes.

CRITERIA	SUBCRITERIA
1. Package global capabilities	1.1 Element library 1.2 Material library 1.3 Procedure library 1.4 Loading library
2. Solution strategy	2.1 FE formulation 2.2 Solution of FE equations 2.3 Numerical integration 2.4 Error estimates
3. Preprocessors	3.1 Methods of data input 3.2 Mesh generation capabilities 3.3 Availability of an interface from a general purpose pre-processor 3.4 Nodes or elements re-ordering for minimizing storage requirements
4. Post processing	4.1 Methods of presentation of output 4.2 Stress smoothing 4.3 Plotting and graphics capabilities 4.4 Availability of an interface to a general purpose post-processor
5. Cost	5.1 Initial cost 5.2 Running cost 5.3 Upgrading cost
6. Maintenance	6.1 Availability 6.2 Cost

continued.....

CRITERIA	SUBCRITERIA
7. User support	7.1 Methods of support
	7.2 Cost of support
	7.3 Documentation
	7.4 Training courses
	7.5 Test problems
	7.6 Source code availability
	7.7 On-site problem solving
8. Vendor credibility	8.1 Vendor past experience
	8.2 Financial position
	8.3 User group support
	8.4 Software reliability
9. Package requirements	9.1 Hardware requirements
	9.2 Software requirements
10. Special features	10.1 Error recovery
	10.2 Re-start capabilities
	10.3 Substructuring
	10.4 Portability
	10.5 Error checking run

TABLE 4.23: List of criteria and subcriteria for FEP

4.12.3 The Simple Matrix Method

This fairly simple method is essentially a normalized form of a weighted average method. Weights are given to each attribute and to all its subattributes. The different FEP to be compared and the alternatives are given numerical values for each subattribute according

to their relative merits. Assume that we have n attributes and that the number of subattributes for attribute i is m_i. Assume further that the number of alternatives to be compared is q. Let us define the following:

- The vector T of n elements which represents the relative weights given to each attribute. To have normalized values the first element of T, i.e. T(1), is always assigned the value of 1 and all other elements of T are given values ≥ 1. In other words, the attributes are arranged in ascending order according to their relative importance.
- The vector S for the subattributes is similar to T. The size of this vector is $k = \sum_{i=1}^{n} m_i$. The number of subattributes per attribute should be the same whenever possible.
- The overall vector of relative importance U of k elements is computed as follows:
- Consider attribute i, the elements of U which correspond to this attribute are computed as:

$$u_j = t_i * s_j / \Sigma s_j \quad \forall\, j \in i \,. \tag{4.76}$$

- Consider the alternatives we assign relative weights among them for every subattribute which results in a matrix W of $k \times q$ elements. To be normalised we must keep $\sum_{j=1}^{q} w(i,j)$ the same for $\forall i$.
- Compute $R=U^T W$ which gives the final rating vector R. The best FEP is the element of the maximum value in R.

Although this method is very simple it gives good rough results and can be calculated by hand. However, for more accurate results it is recommended to use other methods for verification.

4.12.4 The Multi-Attribute Utility Theory (MAU)

Multi-attribute utility (MAU) is based on the idea that complex decision alternatives can be decomposed into structurally-related parts (attributes and subattributes), the values (utilities) of these attributes are assessed and the assessments recomposed to obtain the overall weighted utility of each of the alternatives. The utility of a consequence is a quantitative measure of a person's subjective feelings about the consequence. A consequence with a higher utility is preferred to one with a lower utility. A detailed study of the theoretical basis of the MAU theory can be found in [Keeney and Raiffa, 1976]. Here we give only a brief outline of the technique as follows:

(1) The specification of assessments is done in a hierarchical fashion: categories, attributes and subattributes. For each of these, three estimates of "cost" are considered: best, worst and certainty equivalence (CE). For example, the total cost for a FEP is estimated to be at best £20,000 and at worst £50,000. While the most probable estimate is £30,000, respectively. Thus the three estimates for this attribute are 20,000, 50,000 and 30,000, respectively. In case of subjective attributes, we use relative numerical values. For example, the element library of a FEP could be given estimates as 1 for the best estimate, 0.2 for the worst and 0.5 for the CE.

(2) The utility function model is assumed to be:

$$u(x) = a+be^{cx} \quad , \tag{4.77}$$

where u(x) is the utility of attribute at value x. a,b and c are constants that should be determined. To determine these constants, we utilize the three estimates done in the previous step (1) bearing in mind that u(best)=1, u(worst)=0 and u(CE)=0.5.

Assuming that the three values of x are x_1 for the best estimate, x_2 for the worst and x_3 for the CE and substituting in the utility function (4.77), then,

$$e^{c(x_1-x_2)}(u_2-u_3) + e^{c(x_3-x_2)}(u_1-u_2) = u_1-u_3 \qquad (4.78)$$

The trivial solution C=0 is excluded and this equation is solved by a Newton-Raphson iterative procedure for c whence a and b are determined by back substitution. Practically, 3 or 4 iterations usually give good results. The computation starts by determining the utility function of each subattribute, attribute and category. We denote categories by Z; attributes by Y and subattributes by X.

(3) The values of the alternatives corresponding to each subattribute are mapped onto a utility value using the appropriate utility function.

(4) According to personal preference and experience one of the X's is chosen as the reference subattribute within each attribute. This is used as a reference entity. For example, if for an attribute there are 3 subattributes, then we choose the most important one among them, say, subattribute 1 as the reference entity in this case. This reference subattribute is given a "k" value which is a utility <0,1>. The other k values correspond to other subattributes which belong to the same attribute are calculated from:

$$k_{ij\ell} = k_{ijm}.U_{ijm}(x) \qquad (4.79)$$

where,

$k_{ij\ell}$ is the required k value of the subattribute ℓ of attribute j of category i.

k_{ijm} is the chosen value for subattribute m, the reference subattribute, of attribute j of category i.

u_{ijm} is the utility function of the reference subattribute m of attribute j of category i.

x is the value of the subattribute ℓ of attribute j of category i.

(5) To compute the utility values of all alternatives of attributes from those calculated subattributes, we do the following for each attribute y.

(6) Compute

$$k_{ij} = \sum_{\ell=1}^{n_s} k_{ij\ell} \tag{4.80}$$

where n_s is the number of subattributes in attribute j of category i.

(7) If $k_{ij} \cong 1$, the utilities of the attributes are computed from:

$$u_{ij} = \sum_{\ell=1}^{n_s} k_{ij\ell} . u_{ij\ell} \quad \forall \text{ alternatives} \tag{4.81}$$

(8) If $k_{ij} \neq 1$ the following equation is solved iteratively for Q:

$$1+Q = \prod_{\ell=1}^{n_s} (1+Qk_{ij\ell}) \tag{4.82}$$

This is solved by simple iteration. The initial guess Q_0 is in the interval <0,1> if k_{ij}<1 otherwise an initial guess <-1,0> is chosen. After the value of Q is determined the utility value of attribute $u(y_{ij})$ is computed from:

$$1+Qu(y_{ij}) = \prod_{\ell=1}^{n_s} (1+Qk_{ij\ell} . u_{ij\ell}) \tag{4.83}$$

(9) If one of the y's has no subattributes, these values are mapped to the utility domain directly.

(10) To compute the Z utilities we repeat the same procedures used to compute the utilities of attributes from their subattributes.

(11) The final utilities, the Z utilities, are sorted in descending order (high to low) and the best alternative is the one with highest utility value.

A computer package for the analysis of the MAU has been developed

which can be used for general decision making (Sharaf Eldin, et al, 1986). This package consists of two programs: the UTILITY program and the UTILPLOT program. Both programs are implemented on a small-sized mini-computer and can be easily run on micro-computers. The UTILITY program is the main processor which performs the actual MAU analysis, while the UTILPLOT program is a graphical postprocessor which plots the utility functions if so desired. The UTILITY program is an interactive one where the user is prompted step by step for each bit of information required to do the MAU analysis. On-line validations of all inputs are done and thus the validity of input data is ensured. All data are input in free format.

There are four types of data which are:

(i) Problem definition parameters:

These are the main parameters of the problem:

- number of categories,
- number of attributes in each category;

and number of subattributes in each attribute.

(ii) Basic utility values: These are the three values that correspond to the best, worst and CE values. These data are read for every category, attribute and subattribute.

(iii) Alternative values: The value of each alternative is given. These values are checked to be $\geq$ worst estimate and $\leq$ best one.

(iv) K analysis: Choosing one of the subattributes of an attribute as the reference one and its value which must be <0,1> and getting values of other subattributes. These values must be in the range <worst,best>. The same is repeated in the higher level, i.e. from attributes to categories.

The outputs of the MAU package are comprehensive. It includes the echo of all types of inputs plus:

(1) The utility function coefficients for all subattributes, attributes and categories.

(2) The k-analysis values and results.

(3) The utilities of all alternatives.

4.12.5 The Multi-Attribute Fuzzy Decision Analysis (MFDA)

One of the difficult tasks in the MAU method is the determination of the estimates for the subjective attributes. To overcome this in the multi-attribute fuzzy decision analysis (MFDA) we use verbal ratings to the attributes and subattributes. In fuzzy sets an event x_i is a member of the fuzzy set according to its membership function $\mu_A(x_i)$ which is in the interval <0,1>. To utilize this technique in the considered problem we have to assign weights to attributes and subattributes and ratings to alternatives all based on membership functions. We assume the following membership functions for different weights and ratings:

(i) Weights for Attributes

VI≡Very important≡{(.8,0),(.95,1),(1,0)}

I≡Important≡{(.6,0),(.8,1),(1,0)}

MI≡Moderately important≡{(.3,0),(.5,1),(.7,0)}

UI≡Unimportant≡{(0,0),(.2,1),(.4,0)}

VUI≡Very unimportant≡{(0,0),(.05,1),(.2,0)}

(ii) Ratings for Alternatives

E≡Excellent≡{(.8,0),(.95,1),(1,0)}

VG≡Very good≡{(.7,0),(.9,1),(1,0)}

G≡Good≡{(.6,0),(.8,1),(1,0)}

F≡Fair≡{(.3,0),(.5,1),(.7,0)}

P≡Poor≡{(0,0),(.2,1),(.4,0)}

VP≡Very poor≡{(0,0),(.05,1),(.2,0)}

The three steps in the MFDA are: (1) assign weights to all attributes and subattributes which reflect their relative importance, (2) assign ratings for each alternative with respect to each criteria and sub-criteria alone, independently of all other alternatives; and (3) calculate the global membership function of each alternative starting from subattributes to attributes and obtain the final ranking of alternatives. More details of the technique can be found in [Zadeh,1976].

4.12.6 A Case Study

In [Noor, 1981] 36 FEP for non-linear structural analysis were reviewed. We choose four of them to illustrate the stated techniques without any reference to their names, some missed data are assumed. Tables (4.24),(4.25) and (4.26) show the steps of the solution using the simple matrix, the MAU and the MFDA methods. Note that only parts of the attributes and subattributes are shown.

The methods described in this section give three quantitative approaches for the selection of finite element software. Obviously, the personal attitude and the experience of the user of sophisticated FE software affect the preference structure and the values assigned to attributes which influence the final result. It is also of great importance to select the attributes and subattributes which are relevant to the thought application.

Attributes		Subattributes		Alternatives			
Name	Rating	Name	Rating	A	B	C	D
Vendor Credibility	1	Vendor experience	2.5	5	2	1	2
		Financial Pos.	2.5	3	3	2	2
		User group support	1	4	1	5	0
		Software reliabil.	4	5	1	2	2
		$\Sigma S_1 =$	10				
Cost	2	Initial cost	2	3	1	2	4
		Running cost	2	2	3	1	4
		Upgrading cost	1	4	2	1	3
		$\Sigma S_2 =$	5				
Package Capabilities	5	Element lib.	2	2	2	3	3
		Material lib.	2	2	2	3	3
		Procedure lib.	3	2	3	2	3
		Loading lib.	1	1	2	3	4
		$\Sigma S_3 =$	8				

$T=\begin{Bmatrix}1\\2\\5\end{Bmatrix}$; $S^T=[2.5,2.5,1,4,2,2,1,2,2,3,1]$

$U^T=[.25,.25,.1,.4,.8,.8,.4,1.25,1.25,1.875,.625]$

$U^TW=[19.125,17.625,17.975,25.025]$

The final ranking of FEP is: D,A,C then B.

TABLE 4.24: The simple matrix method

Subattribute	Best Value	Worst Value	CE Value
Initial Cost	8	44	18
Running Cost	2	8	5
Upgrading Cost	.6	3	1.5

(a) Estimates of subattributes

Subattribute	Values of Alternatives			
	A	B	C	D
Initial Cost	9.604	43.24	42.98	9.36
Running Cost	4.46	3.51	6.298	3.2
Upgrading Cost	2.69	1.69	.77	2.04

(b) Values of alternatives

	Subattribute	Utility values of alternatives			
		A	B	C	D
reference → entity given a wt. of .5	Initial Cost	1.0	.5	.5	1.0
	Running Cost	.5897	.7482	.2833	.7998
	Upgrading Cost	.0784	.4172	.8899	.2816

(c) Utility values of alternatives/subattributes

Final utility considering all subattributes = .7496, .7255, .6997 and .8572. The ranking considering this attribute only = {D,A,B,C}

TABLE 4.25: The MAU method

Attribute	wt.	Subattribute	wt.	Alternatives			
				A	B	C	D
Cost	VI	Initial Cost	VI	VG	VP	VP	VG
		Running Cost	I	F	FG	P	G
		Upgrading Cost	MI	P	G	VG	FP

The ranking membership function of each alternative for the attribute cost is:

[.9018, .641, .6175, 1.0]

The ranking is {D,A,B,C}.

TABLE 4.26: The MFDA method

CHAPTER 5

THE VIRTUAL STACK FACILITY

TABLE OF CONTENTS

5.1 INTRODUCTION

It is clear that FEM requires extensive computer resources. One of the many restrictions when implementing these methods on a computer is the limited size of the random access memory (RAM). Although it is not unusual to find a microcomputer of 0.5 Megabytes RAM, yet for moderate size FE systems, this size is not as big as it might be thought. Moreover, most of the available microcomputers and many of the mini-computers have a limited address space to only about 64K bytes which could only be used for fairly small FE systems. Even for medium mini-computers, like the HP3000 series, the address space for the data stack is limited to 1 binary word of 16 bits i.e. 32,767 words [HP3000, 1979]. Many of the micro- and mini-computers operating in a time-sharing environment possess a stack architecture. In these computers there is a separation between the code and data. Each process is allocated a variable length code segment, which may be shareable, and a private data segment. Although FE packages have normally large code size the actual challenge is in the very large size of the data segment. This could be due to the modularity of programming in such systems. For that reason it is enough to have a code segment size that can fit the largest module in the FE package. Overlaying techniques are then used to store the mostly used modules in the RAM while swapping other modules to a DASD [Direct Access Storage Device]. Since the code is not modified during execution, we need not restore an overlayed module again onto the DASD. However, for the data segment the problem is not that simple. First, the segmentation of the data stack is not straightforward. To illustrate, assume that we have a double precision array of 200×200 elements and assume that a double precision variable occupies 8 bytes,

then we need about 320,000 bytes just to store this array. If the available RAM is only 64K bytes then we cannot store the whole array in it. Meanwhile, the program is dealing with the array's elements as one entity. So we cannot split the array into parts at random and store only one or two rows in the RAM. Moreover, since data could be changed due to the execution of the code, it implies that parts to be swapped cannot be overwritten before being copied to a backing storage media like DASD.

In this chapter we present a solution for this problem of limited stack size. This is done through the evolvement of a virtual stack facility (VSF) to the computer system. By the use of VSF, arrays (or data segments in general) of practically unlimited size can be manipulated though the computer will still have the hardware limitation of the maximum stack size. The VSF is a software aid which uses a stack management scheme in a similar sense to that of virtual storage already adopted in most of the main-frame computers. The computer used in this study is the HP3000. The choice is due to the availability of this machine and because the HP3000 is considered as a mini-computer which has the attributes of a stack computer [Baer, 1980]. A special language is designed to declare the VSF. Consequently a translator is designed that maps this source language into a host one for compilation and execution. Different algorithms for the execution of the VSF are studied in order to choose those which best fit FE packages. Some numerical experiments are done in order to measure the effect of the VSF on the speed of execution of particular operations frequently required in FE analysis.

5.2 THE STACK ARCHITECTURE

Before presenting the details of the stack architecture for the considered model computer, the HP3000; let us give first some preliminary definitions of the terminology used:

(a) Process:

A process is the unique execution of a program by a particular user at a particular time. Thus, a process is an execution of a program. The process is the basic executable entity.

(b) Code and Data Segments:

Any program could be divided into code and data segments. A code segment consists of the information that is not to change during program execution. This includes the program instructions and constants. The data segment contains the values, variables and arrays used by the program. The code and data are maintained in strictly separate domains. The data for each process is organised into a data stack. The code segment could be shared between more than one process, while the data segment is private as shown in Fig.(5.1).

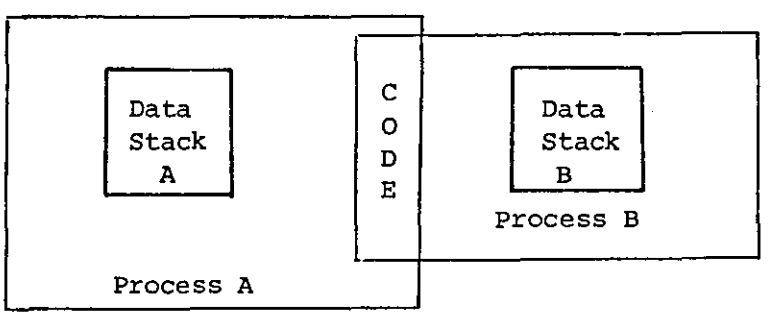


FIGURE 5.1: Code Sharing and Data Privacy

(c) Stack

In general, a stack is a data structure where the last item stored in is the first item to be taken out. The stack structure provides an efficient mechanism for parameter passing, dynamic allocation of temporary storage and efficient evaluation of arithmetic expressions and recursive subprogram calls.

The HP3000 instruction set (more than 200 instructions) is biased towards stack rather than general register operations. All the features of the stack including: automatic transferring of data to and from the CPU registers, checking for stack bounds (overflow and underflow) and the top of stack manipulation are implemented in the hardware.

The general structure of a data stack is shown in Fig.(5.2). The beginning of data is referred to as DB. The stack pointer is referred to as S. The address of DB and S are retained in dedicated CPU registers. The data in the DB location is the oldest element on the stack. The data in the S location is the most current element. The area from S+1 to Z is available for adding more elements to the stack. The Q register separates the data of a calling program or subprogram from the data of a called subprogram. The top four elements of the stack are the most frequently used. Therefore, four CPU registers (RA, RB, RC and RD) are dedicated to them. The use of CPU registers in this way increases the execution speed of stack operations by reducing the number of memory references needed when manipulating data at or near the top of the stack (TOS). The maximum size of the data stack is limited to 64K bytes. As shown in Appendix, A1 sample Fortran programs are given to demonstrate the existing problem of limited stack size and three different cases are illustrated. These are when the integer constant defining array limits

in a DIMENSION statement exceeds 32767, a single array requires more than this amount of storage or when all arrays require more than that, the compiler detects these errors and is aborted.

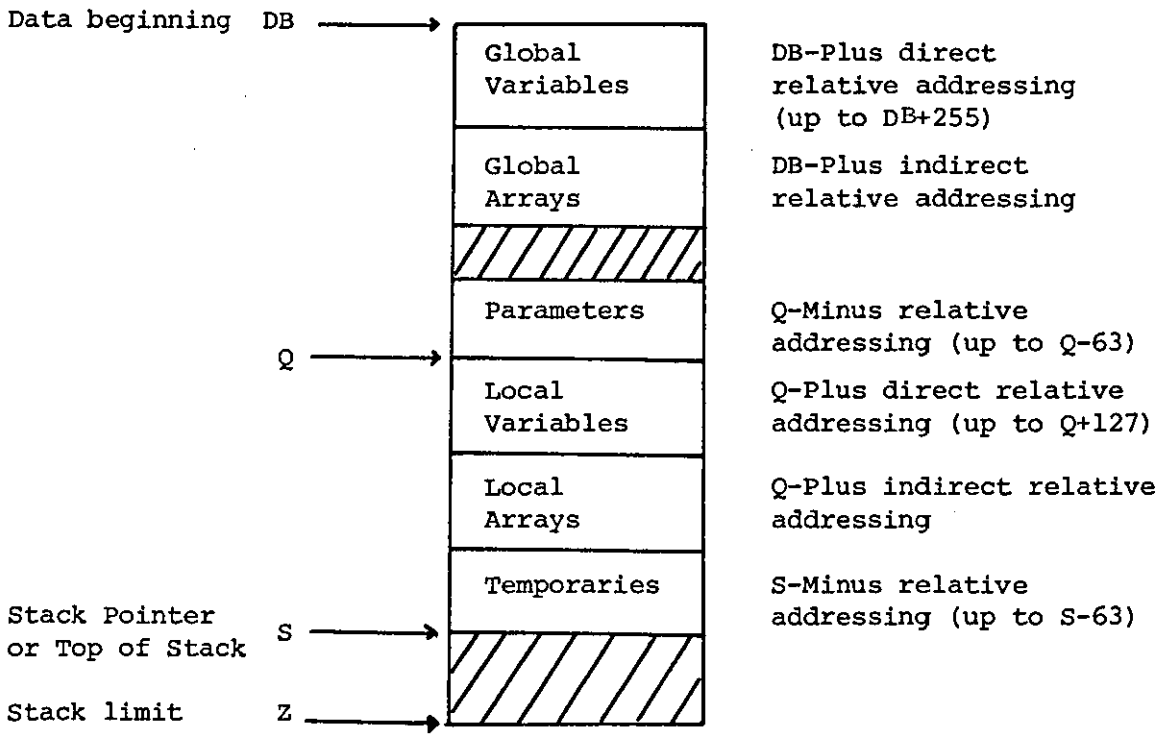


FIGURE 5.2: The HP3000 Data Stack

5.3 THE VSF SOURCE LANGUAGE

5.3.1 The Declaration Block

In order to declare the use of the VSF within a program, a special simple language is designed that must be used to write special statements and commands. This language is very similar to FORTRAN type statements in order to be easily remembered and compiled. It is a fact that most of the FE packages are programmed in FORTRAN and this is another reason why the VSF source language is FORTRAN-like. The part of the program which declares the VSF is called the VSF declaration block (VSFDB). The VSFDB is thus a block of statements and commands written in a specific syntax to declare that the VSF is required and to define the arrays to be manipulated in the virtual stack. These arrays are called virtual arrays. The VSFDB consists of two types of instructions: the VSF compiler commands and the VSF statements. Both of them must be coded from column 1 of the source record. It must be the first segment in the source program. In order to have a formal definition, the VSFDB is defined using the Backus-Naur form (BNF). The BNF notation consists of a number of productions each of which has the form:

<entity>::=<expression>

where the syntactic entity on the left hand side is defined by or may be replaced by the syntactic expression on the right hand side. The expression may be a sequence of syntactic terms or several of these sequences separated by the symbol "/" . Thus, the object "digit" may be defined as,

<digit>::=<0/1/2/3/4/5/6/7/8/9> .

Note that when more than one sequence appears, it means that the entity

may be replaced by one, and only one, of the sequence of syntactic terms. In addition to the '/', the symbol * is used to indicate the repetition of the symbol many times (including o). Thus, an integer could be defined as:

<integer>::=<digit,digit*>

The maximum number that symbol can be repeated is usually explained in English rather than in the grammatical notation.

<VSFDB>::=<VSF compiler commands><VSF statement>

<VSF compiler command>::= $<compiler keyword>

<compiler keyword>::= LIST/NOLIST/MAP/ARRAY/CROSS REF/COMMENT/
INITIALIZE/END

<VSF statement>::=<VSF keyword><virtual array definition/integer>

<VSF keyword>::= INTEGER/REAL/DOUBLE/SIZE

<virtual array definition>::=<virtual array name><dimension specification>

<virtual array name>::=<alphabetic><alphanumeric*>

<dimension specification>::=<(,integer, ',integer'*,)>

<alphabetic>::= A/B/C/D.../Z

<alphanumeric>::=<alphabetic><digit>

<digit>::= 0/1/2.../9

It should be noticed that the manifest data type used in FORTRAN [the IJKLMN rule] is not applicable for the virtual arrays.

5.3.2 The VSF Compiler Commands

The VSF compiler commands (VSFCC) are used to direct the VSF compiler to choose some of its options. Each command is one keyword preceded by a $. No blanks are allowed inbetween.

The available VSFCC's and their effect are as follows:

(a) $ARRAY

This command must be the first statement in the VSFDB and consequently in the source program. It tells the compiler that the next statements are the VSFDB body coded according to the rules of the VSF source language rather than FORTRAN. If the VSF compiler does not find the $ARRAY command as the first record it will be aborted.

(b) $LIST

This command is used to get a list of the VSFDB until the end of the VSFDB unless the list is suppressed by a $NOLIST command. The $LIST command can be inserted anywhere in the VSFDB. If neither $LIST nor $NOLIST is specified, the VSF compiler will default to $LIST.

(c) $NOLIST

This command is used to suppress the listing of the VSFDB. This command can be inserted anywhere in the VSFDB. Any combination of the $LIST and $NOLIST commands can be used within the VSFDB as required.

(d) $MAP

This command is used to produce a map of the VSFDB. The map contains a full description of all the virtual arrays declared, type, size, bounds and total size in bytes. The default of the VSF compiler is 'NOMAP'.

(e) $COMMENT

This command is used to insert comments in the VSFDB. Any text can be inserted after the keyword $COMMENT. If multi-line comments are required, the $COMMENT must be repeated in each line.

(f) $CROSSREF

This command is used to get a cross reference of the virtual

arrays as referenced in the main FORTRAN program and its subprograms. The default is NO CROSS RFFERENCE.

(g) $INITIALIZE

This command is used to set the declared virtual arrays initially to zeros. The actual value initialized in each array is compatible with its type. Initializing virtual arrays through the $INITIALIZE command is faster than using nested DO loops in the program. No initialization will take place if $INITIALIZE is not specified.

(h) $END

This must be the last statement in the VSFDB. It tells the VSF compiler that the VSFDB is finished and the main FORTRAN program will start.

5.3.3 The VSF Statements

VSF statements are very similar to the standard FORTRAN IV type statements. All these statements must start from column 1 with a keyword followed by at least a single space. No continuation lines are allowed in order to conform with the VSF coding rules. The VSF statements are used to define the virtual arrays used in the FORTRAN program.

A virtual array name must be of six alphanumeric characters as maximum. The first of which must be an alphabetic as in standard FORTRAN IV. No special characters or symbols are allowed within virtual array names. The name must not be a "reserved" FORTRAN or VSFDB keyword. The number of dimensions is limited to three only. The virtual array bounds must be positive and a dimension must be given to allocate storage. A maximum of 100 virtual arrays could be delcared. However, the total number of declared arrays (virtual or real) should

not exceed 255 arrays. The following are the allowable VSF statements:

(a) INTEGER

Syntax:

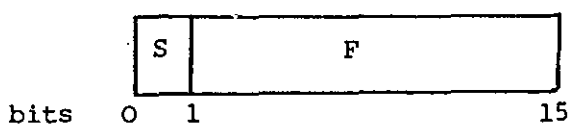
INTEGER name1,name2,...

where name1,name2,... are the virtual array names and dimensions. The names must conform to naming rules. The maximum number of elements of a virtual array in one dimension is 2^{32}-1 i.e. 2,147,483,647.

Example:

INTEGER IM(30000),X(100000)

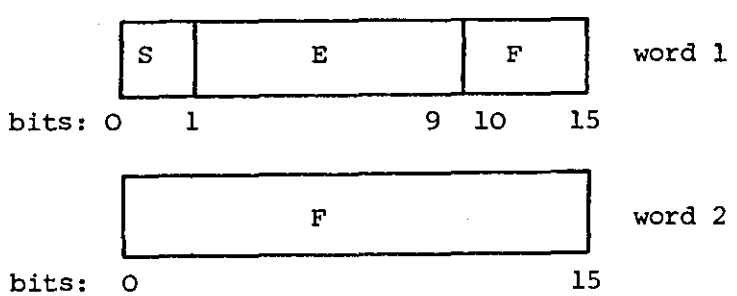
Note that although IM starts with the letter 'I', it will not default to integer. Note also that each element occupies one computer word of 16 bits. The range of integer values is from -32768 to 32767. The internal representation of integers is shown in Fig.5.3.



S is the sign bit

S is 0 for positive numbers and 1 for negative ones

FIGURE 5.3: Internal representation of an integer



S is the sign bit: 0 for positive

1 for negative

E is the exponent value

F is the fixed number

FIGURE 5.4: Internal representation of a real

(b) REAL

Syntax:

REAL name1,name2,...

where name1,name2,... are the virtual array names and dimensions.

Example:

REAL YA(40000)

Each real element occupies two computer words (4 bytes) as shown in Fig.5.4. The range of a real number is from $.863617\times10^{-77}$ to $.1157921\times10^{78}$. The decimal value of a real number is

$$(-1)^S * 2^{(E-256)} * F$$

(c) DOUBLE

Syntax:

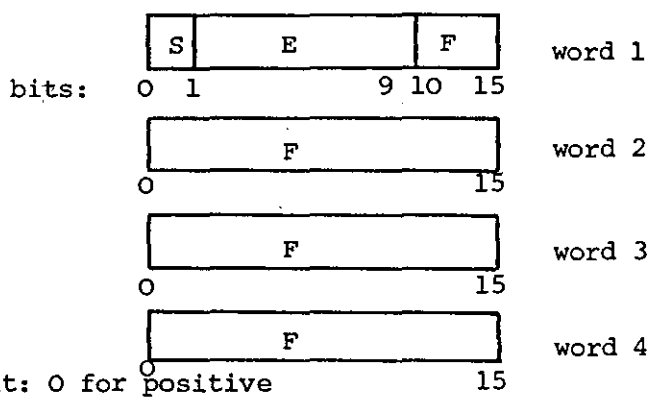
DOUBLE name1,name2,...

where name1,name2,... are the virtual array names and dimensions.

Each element occupies four computer words as shown in Fig.5.5. The range of a double number is from: $.863616855509445\times10^{-77}$ to $.1157920892373162\times10^{78}$.

Example:

DOUBLE Z(10000),MAT(30000)



S is the sign bit: 0 for positive
1 for negative

E is the exponent value and F is the fixed number

FIGURE 5.5: Internal representation of a Double

(d) SIZE

Syntax:

SIZE integer

where integer is the available real stack size in binary words. The minimum value is 128 binary words (256 bytes) which is the size of one sector of the DASD. The maximum value is 32767 minus the size occupied by real arrays in the main program. The default value is 128. The VSF compiler will build a buffer in the real stack with the defined size to store in it the active parts of the virtual arrays.

5.3.4 Use of Virtual Arrays

There are some considerations and restrictions in the use of virtual arrays. These can be concluded in the following:

(1) Each virtual array declared in the VSFDB must be declared in the main FORTRAN program and its units as a type statement with the same number of dimensions but with one element only. The following example illustrates the above mentioned rule:

```
$ARRAY
$MAP
  INTEGER MKATI(30000)
  REAL STIFF(40000,40)
  DOUBLE SEED(100,100,100)
$END
PROGRAM MAIN1
INTEGER MKATI(1)
REAL STIFF(1,1)
DOUBLE PRECISION SEED(1,1,1)
      ...
      ...
      ...
END
```

```
      SUBROUTINE XYZ(A,B,C,D)
C     A subprogram referencing any of the virtual arrays
      INTEGER MKATI(1)
      REAL STIFF(1,1)
      DOUBLE PRECISION SEED(1,1,1)
                ...
                ...
      END
```

(2) Virtual arrays are available to all subprograms provided that they are declared as indicated in (1).

(3) Virtual arrays could appear in simple assignment statements only. This is a VSF compiler constraint to keep its design as simple as possible. To overcome this constraint, temporary storage locations could be used. The following examples clarify this:

(a) I/O statements:

```
      READ(5,11)A(1,10000)
```

is converted to

```
      READ(5,11)X
      A(1,10000)=X
```

(b) IF,GOTO,DO statements:

```
      IF(A(1,5000).LT.A(I,I)) GOTO 1000
```

is converted to

```
      X=A(1,5000)
      Y=A(I,I)
      IF(X.LT.Y) GOTO 1000
```

Similar conversions could be used for GOTO,DO,... statements.

(c) Using subroutines and functions

Since all virtual arrays are available to all subprograms (as if they were common), virtual arrays are not passed as parameters to subroutines or functions.

```
CALL COMPT(A,B,X)
```

where A is a virtual array converted in the VSF Fortran to:

```
CALL COMPT(B,X)
```

(4) File numbers 98 and 99 (unit numbers 98 and 99) in VSF Fortran programs should not be used. These unit numbers are reserved for virtual arrays only.

5.4 THE VSF COMPILER

5.4.1 Implementation Considerations

There are three methods of language implementation in use. These are: compilation, interpretation and pre-processing into another language. Interpretation is not convenient for the present work since the VSFDB is an extension to a FORTRAN program and FORTRAN is usually compiled rather than interpreted. Efficiency of execution is another reason to rule out the interpretation method. Writing a full compiler for the VSFDB is time consuming and not related to other parts of the thesis. So, pre-processing seems to be the logical alternative. In fact, the pre-processing technique could be defined as a mapping procedure from the source language, VSFDB, to another source language - FORTRAN, for which a compiler already exists. This approach is sometimes referred to as cascading [Brown, 1981]. The major advantage of this approach is that portability is ensured and the produced programs are portable as FORTRAN. On the other hand, full syntax and type checking should be performed by the pre-processor otherwise errors will be detected by the FORTRAN compiler. For that reason, the VSF compiler requires that the FORTRAN program must be error free.

Another consideration is the language in which this pre-processor is coded. There are three basic different approaches in encoding a translator [Brown, 1981]:

(a) Writing it in the assembly language of the machine on which it is to be run.

(b) Writing it in a high-level language.

(c) Writing it using some special compiler-building tool or compiler-generators.

Approach (b) is chosen in this work. The reasons for this are:

(i) Ease of implementation, debugging and maintainance.

(ii) Portability is ensured through high level languages for which standards exist.

(iii) Since our compiler is a pre-processor rather than 'true' compiler and the VSFDB is fairly simple, it seems to be logical to use the same high level language of the target code i.e. FORTRAN.

5.4.2 The VSF Compiler Structure

It is well known [Bornat, 1979] that the compilation process can be divided into a sequence of phases. Each phase carries out one of the compilation sub-tasks. If the compiler goes through all the phases for each part of the program it is called a single-pass compiler. On the other hand, if the compiler goes through the entire program for each phase before starting the next phase, it is termed a multi-pass compiler.

The phases of the VSF compiler are schematically shown in Figure 5.6. In the first phase, a record of the source program is read and a lexical analysis is performed to partition the input character stream into tokens. The resulting tokens are then passed and syntactically checked. The result of this phase of the compilation is to create a symbol table showing the virtual array names with their attributes: type, number of dimensions and bounds. At the end of this phase, if any errors were flagged then the compiler will be stopped to correct these errors. If no errors were encountered, then the compiler will start scanning the FORTRAN program to identify the virtual array names that are used within the main body of the program or in one of the subprograms.

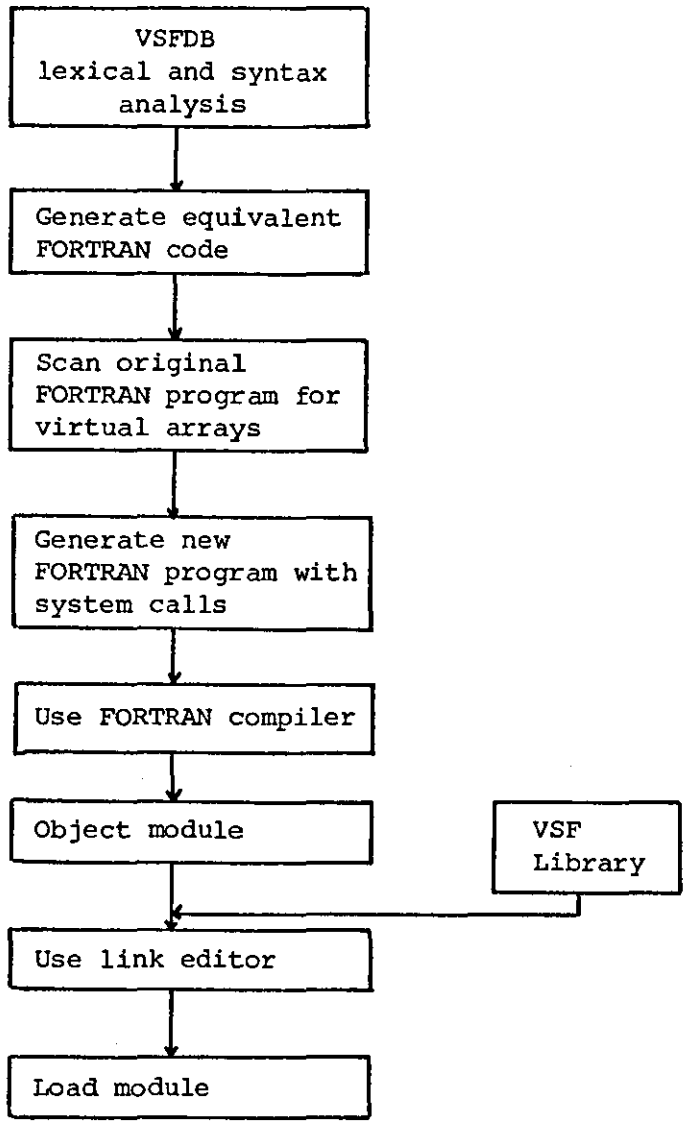


FIGURE 5.6: Processing of VSF FORTRAN

If a virtual array name is found in statements other than assignment, errors are flagged. The necessary FORTRAN code is generated for assignment statements as explained later. All of these phases are done in one-pass. The final produced FORTRAN program is then compiled using the standard FORTRAN compiler and an object code is produced. The binding of this object code with standard FORTRAN Library and the VSF Library

is done using the standard link editor to produce a load module (executable code).

5.4.3 Compiler Dictionaries

During the execution of the VSF compiler, it uses two types of dictionaries; static and dynamic. Both types of dictionaries consist of a sequence of entries each consisting of the entry name together with its attributes. The static dictionaries contain the commands and statements keywords like INTEGER, MAP,... . The dynamic dictionary is usually called the symbol table. It contains the user invented names for virtual arrays. For each entry, the following items of data are maintained: six characters for the virtual array (VA) name, 2 bits for its type, 2 bits for the number of dimensions and four double words (integer*4) three for array bounds in each dimension and the fourth to mark the initial record number in the DASD at which the elements of this array will be stored which is automatically computed by the VSF compiler.

The look-up procedure within these dictionaries is done by sequential search. Although other types of searches like binary search and hash addressing [Knuth, 1975] are usually used in practice for their efficiency as compared to linear search, but in the case of the VSF compiler since we have a very limited number of keywords and on the other side usually a fairly small number of virtual arrays are used in a FE system [probably only the master or the global stiffness matrix need to be defined in the VA domain] which justifies the simple linear search.

5.4.4 Translating the Assignment Statement

A simple way to recognise an assignment statement [Hopgood, 1974]

is that it has no zero-level commas on the righthand side of a zero level "=" sign. Virtual arrays when appearing on the righthand side of an assignment statement are substituted by a function call with the subscripts passed as arguments in the function call. To illustrate consider the following assignment statement in which the virtual array STIFF appears on the righthand side,

```
X = A + B * STIFF(I,J) + C
```

This FORTRAN statement will be translated to,

```
X = A + B * GETVSFELEMENT(array name, line no., start
      address, type, Ndim,bound1,bound2,bound3,I,J,J) + C
```

where,

* the function GETVSFELEMENT is a library function that will retrieve a particular virtual array element. Fetching a virtual array element may result in a DADSD reading if it was not already in the real stack.
* array name is a character variable containing the virtual array name.
* line number is the source line number of this statement. The array name and the line number are used for error reporting.
* start address is the record number that identifies the start of the virtual array in the backing storage file (DASD). This is a double word integer.
* type is an integer that identifies the type of the array. An integer is of type 1, a real is given type 2 and a double precision is given type 3.
* Ndim is the number of dimensions of the considered virtual array.
* bound1, bound2 and bound3 are the virtual array bounds. These parameters are passed in order to compute the actual offset of any element from the first one. It is also useful to check that the

referenced element is within the array bounds, i.e. to check that bounds in each dimension are not violated. Note that I,J are passed to identify the particular element to reference. Note also that a "0" is inserted to have compatibility when using the GETVSFELEMENT function in case a three dimensional array is referenced. It is clear that the length of the assignment statement will be increased and this is the reason why assignment statements containing virtual arrays are limited to one line length only.

If a virtual array appeared in the lefthand side of an assignment statement it is translated as follows:

The lefthand side is replaced by a system variable VSF VAR 01 and a new CALL statement is inserted immediately after the assignment statement to store the computed value in the actual referenced element. To illustrate consider the following statement which assigns a value to an element of the virtual array STIFF:

```
STIFF(I,J) = 100.0 * EI
```

This will be translated to:

```
VSF VAR 01 = 100.0 * EI
```

CALL SAVE VSFELEMENT(VSFVAR01, array name, line no., start addres, type, Ndim, bound1, bound2,0, I,J,0), where the arguments are similar to those used in the GETVSFELEMENT.

5.5 THE RUN-TIME LIBRARY

Two main library subprograms are used namely: The GETVSFELEMENT function and the SAVEVSFELEMENT subroutine. The syntax and the use of these two subprograms was given in the previous section. The internal structure of both of them is very similar. While the GETVSFELEMENT fetches a virtual array element, the SAVEVSFELEMENT does the opposite and stores a virtual array element. To access the (I,J,K) element of a three dimensional array these subprograms in the first instance will check that:

$$1 \leqslant I \leqslant \text{bound1}$$

$$1 \leqslant J \leqslant \text{bound2}$$

$$1 \leqslant K \leqslant \text{bound3}$$

Although the formal definition of the FORTRAN prohibits only accessing elements that are outside the whole space allocated to an array, it is felt that making bounds checking for each individual dimension is very useful particularly in large computer programs like those of FE systems.

If the subscripts I,J and K are within the declared bounds then the actual element location is calculated as follows:

$$S = [(K-1) * d1 * d2 + (J-1) * d1 + I] * F \qquad (5.1)$$

where d1, d2 and d3 are the dimensions of the virtual array as declared in the VSFDB, and F is 1 if the array is integer, F=2 for real arrays and F=4 for double arrays.

The sector number that contains this particular element is computed according to:

$$r = \left|\frac{s-1}{128}\right| + v \qquad (5.2)$$

where, r is the sector number

s is the element displacement relative to the first one in the considered array. s is computed as in (5.1).

v is the sector number of the first element in the array which is determined by the VSF compiler.

Note that the sector length is 128 words.

The offset of the element within the sector is given by:

$$f = \frac{s-(r-v)*128}{F} \quad , \tag{5.3}$$

where: f is the offset in words

s,r,v and F as before.

At last, the location of this element is checked for existence in the real stack. If it is there then the required operation is performed (storing or fetching). Otherwise, the element is retrieved from the virtual stack [i.e. from the DASD] and placed in real stack. If there is no room in the real stack for the referenced element, then parts of the real stack must be swapped to leave space for the required element. The replacement algorithms will be explained in a later section. A library routine which is the error handler will be explained in the next section. For the implementation of a new proposed replacement algorithm a library routine is explained in Section 5.7.

There are three versions of the main library subprograms: GETVSFELEMENT and SAVEVSFELEMENT. The returned value could be integer, real or double precision according to the type of the VA.

When initialization is required for VA, a library routine ZERIOZEVSF is used to initialize the sectors on the DASD which correspond to a virtual array with the appropriate zeros according to the type of the VA, i.e. either a 0 or 0.0E0 or 0.0D0 is moved to all the sectors allocated for the VA to be initialized.

5.6 THE VSF ERROR MESSAGES

Since programs are man-made we expect errors. It is very important to have good error checking with suitable error reporting to ensure that the generated code is correct. Three types of errors could be classified: syntax, semantics and execution. In the following we consider the VSF errors only. The FORTRAN errors are detected by the FORTRAN compiler. Other run-time errors are detected either by the hardware (like arithmetic overflow) or by the operating system (like addressing a location outside the user data domain). The VSF syntax errors are flagged if a token cannot be recognised. To illustrate, if the $MAP command was misspelled as $MOP then a syntax error is flagged as the compiler cannot identify the keyword. Another type of error that can be found is the declaration of an array twice. A third type is that of referencing an element outside the array bounds. The error messages of the VSF are given in the Appendix.

The main concepts in designing the error messages could be summarized in the following:

(i) The whole error messages are saved on a file which could be accessed directly and concurrently by many users. A special routine is written than accesses that file. This routine is called the VSFERRORMESSAGE. The arguments of this routine are the start record number and the end record number of the records that contain the relevant error messages. To illustrate, if an error message is stored in the sixth and seventh records of the error message file, then, the call statement will be:

```
CALL VSFERRORMESSAGE(6,7)
```

(ii) Error messages are written in an eye-catching fashion to be easily

found. Moreover, an audio-alarm (Bell) is produced on the terminal to get the users attention.

(iii) Run-time errors like the reference of an element outside array bounds (i.e., bounds violation) are supplemented by run-time information giving the current values of the relevant parameters.

5.7 REPLACEMENT ALGORITHMS

If an element of a virtual array is referenced, the VSF library subroutines will check that this element is in the real stack. If it is not the case, then the sector of the DASD that contains that element will be read and stored in the real stack. If there is no room in the real stack, then one of the existing sectors should be removed from the real stack and swapped to the virtual stack (if necessary) to leave space for the new one. The policy by which a sector is chosen to be removed from the real stack is called the replacement algorithm. Different replacement algorithms are already known and implemented in mainframe computers. Perhaps the four most well known algorithms are, [Belady, 1966 and Coffman, 1973]:

(a) First-in-First-out: The FIFO algorithm, i.e. the oldest pages are replaced.

(b) Least recently used item replacement (LRU)

(c) Last-in-First-out: The LIFO algorithm, i.e. the newest pages are replaced.

(d) Working set method: only locations which are not part of current working sets may be replaced. A working set of pages referenced by a program during the previous t seconds.

However for micro-computers, it is only recently that similar ideas have evolved. Odgen (1979) described the memory mapping implemented in four microprocessors. The mapper is implemented in hardware and the logical address is limited to 16 bits in the Intel 8086 while the physical address can be as big as 1MB using 20 bits. The concept of virtual memory for microcomputers was introduced later by Schmitt [1983]. Using memory-management units (MMU's) designed to do the hardware functions

required for a virtual memory computer system it will be possible to build a virtual memory system based on a microprocessor like the MC68000 [Motorola, 1981] or the Intel 8086 [Intel, 1981].

The hardware implementation of the VSF is not possible within the course of this work. First, it needs resources that are not available, i.e. like the memory management unit which are not available for the HP3000 computer used in this work. Second, it is a time-consuming process. The other alternative is, therefore, to use software to implement such algorithms, although the exeuction will be much slower.

The optimal replacement algorithm is only possible if the computer can look ahead at the future references and throw out the pages which are referenced furthest in the future [Martin, 1977]. This cannot be achieved in practice. However, the situation may be different between mainframe computers and mini and micro-computers. In a mainframe computer, many processes are running concurrently and doing different applications. So, thinking of a replacement algorithm for such big computers will be independent of the running applications since they do not have common features. On the other hand, for small computers the running applications are either one at a time in single user machines or related to each other in multi-user systems. To illustrate, consider an engineering design office with a mini-computer installed. It is very probable to find two or three processes running concurrently and all of them use a finite element package like MSAP to solve different problems. For that reason it may be logical to think of a replacement algorithm that best suits the running application rather than use one of the well-known general replacement algorithms which do not consider the running applications altogether. To illustrate, if a process is doing matrix

addition then it is preferable to keep the corresponding rows in the operands of the matrix addition operation in the real stack. After the addition is done we do not need these rows in real stack any more. On the other hand for matrix multiplication it is desired to keep a row of the first operand of the matrix multiplication and a column from the second operand. In a Gaussian elimination operation it is suitable to keep the pivot row and column in real stack.

To implement such ideas two approaches could be considered. First, it is possible to use some modified version of the working set method, where the set of sectors referenced by the program during the previous t time units are not considered for replacement. The other approach is more simple and seems that it may give better results but it needs some programming skills. Let us call it the "Keep-Release Technique" (K-R method). In this method two system calls are used to keep portions of virtual arrays in the real stack and vice versa. The calls have the format:

```
KEEP REAL id1,id2,...
```

and

```
RELEASE REAL id1,id2,...
```

where id1, id2,... are virtual array identifiers as shown in the following examples.

The effect of these statements is to set or unset a bit for each sector in the real stack to identify whether it is to be kept in the real stack or it may be replaced.

Example 5.1: Matrix addition

```
C = A + B
```

where A,B and C are virtual arrays. The VSF FORTRAN code could be:

```
      DO 20 I=1,N
      KEEP REAL A(I,*),B(I,*),C(I,*)
      DO 10 J=1,N
10    C(I,J)=A(I,J)+B(I,J)
      RELEASE REAL A(I,*),B(I,*),C(I,*)
20    CONTINUE
```

Example 5.2: Matrix Multiplication

```
      DO 100 I=1,N
      DO 100 J=1,P
      C(I,J)=0
      KEEP REAL C(I,J),A(I,*),B(*,J)
      DO 50 K=1,M
50    C(I,J)=C(I,J)+A(I,K)*B(K,J)
      RELEASE REAL C(I,J),A(I,*),B(*,J)
100   CONTINUE
```

Example 5.3: Simple Gauss-Jordan method

The coefficients are in an N×M array

```
      ...
      DO 100 I=1,N
      PIVOT=X(I,I)
      KEEP REAL X(I,*),X(*,I)
      DO 10 J=I,M
10    X(I,J)=X(I,J)/PIVOT
      DO 20 K=1,N
      PIVOT=X(K,I)
      IF(I.EQ.K) GOTO 20
      DO 15 J=1,M
15    X(K,J)=X(K,J)-PIVOT*X(I,J)
20    CONTINUE
      RELEASE REAL X(*,I),X(I,*)
100   CONTINUE
```

The actual implementation of the above mentioned methods requires the modification of a Fortran compiler. However, since the source code

of FORTRAN/3000; the HP3000 FORTRAN compiler; is a proprietary code and not available a simulation is used to study the effectiveness of the K-R method compared to a FIFO algorithm as shown in 5.9.

Four tables are kept in a common block called VSF TABLES in order to implement the VSF. These tables are:

(a) Real Stack which contains the values kept in the computer memory. The size of this table is dependent on the real stack size defined in the SIZE statement within the VSFDB. Since the physical size of a sector on the DASD used in this work is 128 words and since we need three words for each sector to hold its address and two flags to indicate whether it has been modified and to indicate whether it is kept in real stack, it follows that the number of sectors that could be kept in real stack is given by,

$$n = \left| \frac{\text{size}}{131} \right| \tag{5.4}$$

Read and write of sectors are done using unformatted binary transfer to achieve faster execution, e.g. READ(99@record no.)BUF where record no. is the address of the sector to be read and BUF is a real array of 64 elements which occupies that sector.

(b) Sector table which contains the sector addresses of those sectors stored in the real stack. The length of this integer vector is n as given by (5.4).

(c) Modification bit which is a vector of flags used to indicate whether a sector has been modified or not. The importance of this flag is that when replacing a sector from the real stack if this flag was set, then, the sector to be replaced is copied again to the virtual stack before being overwritten.

(d) Replacement bit table which is used for the K-R algorithm to indicate

whether a sector is to be kept in real stack or is released.

In addition to these tables, other common pointers are used to indicate the current and maximum size of the sector table; the type of replacement algorithm and the sector to be replaced in the FIFO algorithm.

Figure (5.7) gives the steps used to store a virtual array element. Assume that element I of virtual array x is to be assigned the value y which is in real stack, i.e. executing the statement X(I)=y. Note that y may be any arithmetic expression. A similar process is used to fetch a VA element i.e. to execute a statement like y=X(I) where X is a virtual array.

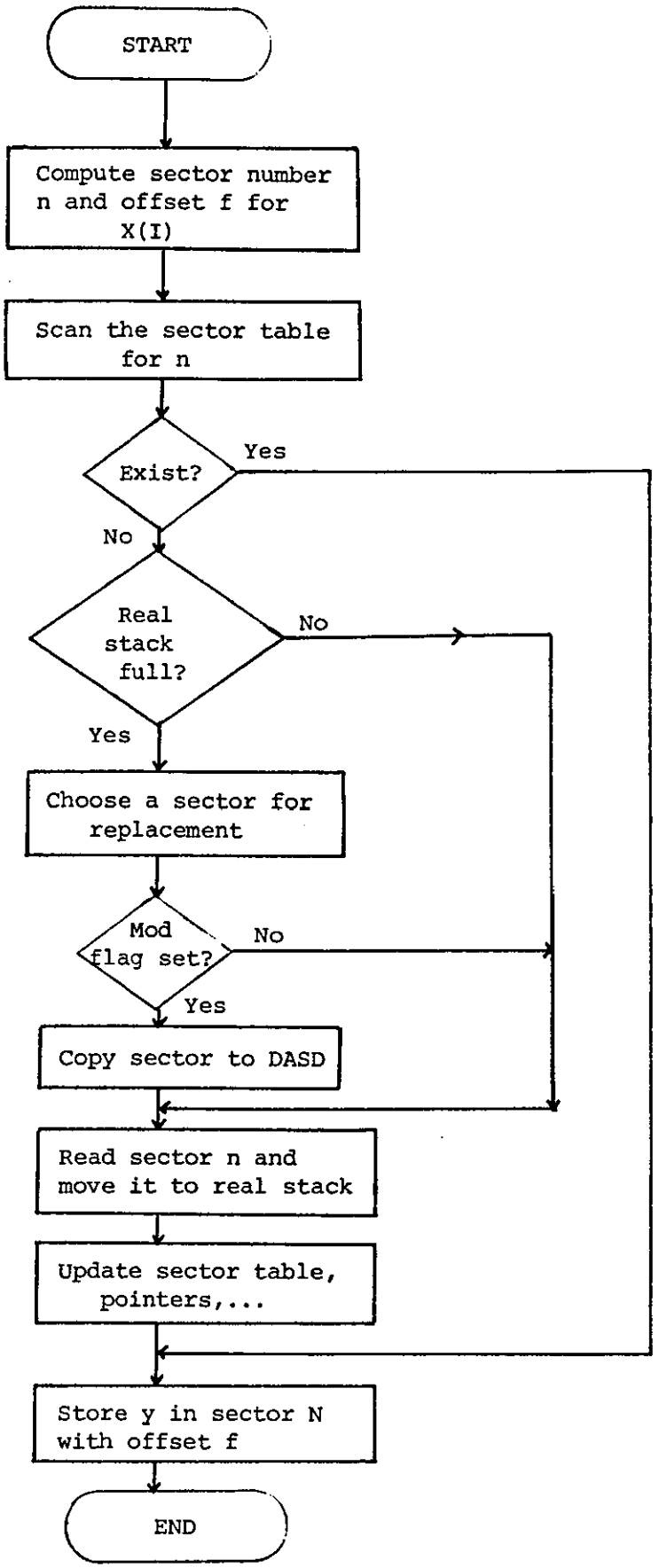


FIGURE 5.7: Storing a VA element

5.8 VSF PROCEDURE

The VSF compiler is saved as a load module in the computer system library under the name VSF.PUB.SYS. Every user is to get the execute-access to that program. The VSF library routines (problem independent subprograms) are saved in compiled form under the name VSFLIB and read-access is granted to all users. The error messages file is saved in ASCII form under the name VSFERROR.PUB.SYS and all users are given the read-access to that file. The file which contains the VA is a temporary one built in the user file domain under the name VSFARRAY and its size is computed by the VSF compiler.

In order to facilitate the invokation of the VSF compiler, a procedure is defined and added to the system procedures. Since the name given to the standard Fortran compilation, link and go procedure is FORTGO we used a similar name to the VSF procedure: VSFGO.

5.9 TEST PROBLEMS

In order to get some benchmarks of the speed of the VSF, numerical experiments were conducted. Three test problems are considered as follows:

(a) Addition of two matrices each of 64×64 elements.

(b) Addition of two matrices each of 128×128 elements.

(c) Multiplication of two matrices each of 64×64 elements.

In each problem different cases are considered: different real stack sizes, different replacement algorithms and different orders of execution. We considered real stack sizes range from 0 to a maximum value of 25600 words which corresponds to the size of 200 sectors on the disc. Three replacement algorithms are considered: FIFO, LIFO and K-R. The first two problems of matrix addition are solved in two different orders: row by row and column by column addition.

Test Problem (a)

Two matrices A and B are added and the result is stored in a third matrix C. All matrices are of type real. Before giving the results of this test problem we notice that:

(1) It is possible to have all the three matrices in real stack and thus one of the runs will be with all the three matrices in real stack.

(2) Since virtual and real arrays are stored columnwise it is expected that execution time will be less if the program is coded such that matrix addition is done column by column rather than row by row. In other words code X will be executed faster than code Y as follows:

Code X:

```
          DO 100 J=1,N
          DO 100 I=1,N
100       C(I,J)=A(I,J)+B(I,J)
```

Code Y:

```
          DO 100 I=1,N
          DO 100 J=1,N
100       C(I,J)=A(I,J)+B(I,J)
```

where any or all the arrays is virtual.

(3) Real stack allocated, using the VSF size statement, is an important factor. Various stack sizes are tested as follows: 0, 128, 256, 384, 1280, 2048, 2560, 4096, 8192, 16384, 20480, 24576 and 25600.

(4) Three different replacement algorithms are tested: FIFO, LIFO and K-R. However, since the K-R algorithm requires the allocation of a column of each A,B and C in real stack and since each column requires 128 words; it implies that the K-R method requires 384 words at least as a real stack size.

Table 5.1 gives the results of this test problem. These results show that the processing sequence has a very significant effect on execution time particularly for small sizes of stack. It is also evident that the K-R algorithm gives better results than LIFO and FIFO provided that processing is done column by column. No gain in execution time is noticed for larger stack sizes in the case of the K-R method.

It is also clear that the FIFO algorithm is faster than LIFO. The time required to do the matrix addition without real stack is more in the case of using row-wise addition as compared to column-wise ordering. This is due to the fact that matrices are stored column-wise. This implies that accessing the elements of the same column of any of

the considered arrays will be done without the repositioning of disc heads since elements of the same column are stored in the same sector. In case of row-wise computation the time required to access elements of the same row consists of head positioning plus latency because these elements are stored in different sectors.

Test Problem (b)

This is a similar problem to the one just considered. The only difference is that matrices of 128×128 elements are considered rather than 64×64 elements. These matrices cannot be saved in real stack together. In otherwords, this problem cannot be executed on the considered class of computers. Consequently, no timing data is available when the three arrays are kept in real stack. The results of this test problem are shown in Table 5.2. It is noticed that the superiority of the K-R method compared to other techniques is evident. It is also clear that since in matrix addition the last-in sector is that of C elements which are always modified then the LIFO algorithm will take more time than FIFO.

Test Problem (c)

This problem considers matrix multiplication. In this case two matrices A and B of 64×64 elements are multiplied to give a third matrix C. The elements of C are computed one by one in column-wise. form. Different stack sizes are considered. The three replacement algorithms are considered: LIFO, FIFO and K-R. However, since matrix multiplication requires a row and a column from both A and B and since A and B are stored column-wise it is evident that getting a row from A

is very expensive due to the many readings to be done. Keeping a row of A will result in reading all sectors of A. Since A is a 64×64 array and a real element occupies two words, then each column of A needs 128 words, i.e. one sector on the disc. It is most convenient in this case to keep the whole of A in real stack to avoid "thrashing". To keep the two columns of B and C we need only two sectors. The results of this problem are shown in Table 5.3. Note that we considered the case of real stack size = 8448 words which is the size necessary to keep A in full, a column of B and a column of C. Again, the K-R algorithm gives better results than the two others.

Real Stack Size (words)	Col. by Col.			Row by Row		
	FIFO	LIFO	K-R	FIFO	LIFO	K-R
0	185.1	187.2	NA	227.2	227.2	NA
128	192.5	187.8	NA	227.4	227.3	NA
256	179.7	188.7	NA	231.2	226.8	NA
384	6.7	182.6	5.6	231.2	226.0	224.8
512	6.8	185.0	5.6	231.3	224.2	224.2
640	6.9	183.1	5.6	191.1	182.0	223.5
768	6.9	181.6	5.6	190.9	181.4	180.5
896	6.9	180.9	5.6	187.7	182.2	181.9
1024	7.1	180.5	5.6	186.3	179.5	179.7
1152	7.0	176.1	5.6	186.7	179.0	177.8
1280	7.2	179.9	5.6	189.3	179.0	180.0
1408	7.1	181.6	5.6	187.0	177.0	177.1
1536	7.2	177.3	5.6	192.9	176.7	175.5
2048	7.4	175.3	5.6	188.0	174.9	171.6
2560	7.5	171.7	5.6	189.8	169.4	169.6
4096	8.4	162.5	5.6	190.6	159.4	159.5
8192	10.0	133.9	5.6	196.0	131.8	132.2
16384	12.1	77.3	5.6	205.7	76.2	76.7
20480	13.9	48.6	5.6	210.3	50.9	47.8
24576	16.7	16.7	5.6	17.0	17.0	16.9
25600	16.7	16.7	5.6	17.0	16.8	17.0

TABLE 5.1: Time of adding two matrices 64×64 in seconds

Time in case of using real stack only = 1.2 seconds

Real Stack Size in words	FIFO	LIFO	K-R
0	903.1	909.2	NA
128	910.3	909.5	NA
256	836.3	908.6	NA
384	28.7	905.6	NA
512	29.0	902.7	NA
640	29.3	901.8	NA
768	29.4	898.8	24.2
896	29.7	895.7	24.2
1024	30.0	894.9	24.2
1152	30.1	892.1	24.2
1280	30.4	888.9	24.2
1408	30.7	888.0	24.2
1536	30.8	884.9	24.2
2048	31.8	874.8	24.2
2560	32.7	866.9	24.2
4096	35.6	838.9	24.2
8192	43.0	761.4	24.2
16384	57.7	607.2	24.2
20480	63.8	526.3	24.2
24576	61.0	446.6	24.2
25600	63.7	418.8	24.2

TABLE 5.2: Time of adding two matrices 128×128 in seconds . Processing is done column by column

Real Stack Size in words	FIFO	LIFO	K-R
0	4743.6	4742.6	
128	4765.2	4751.4	
256	4703.6	4710.6	
384	4248.1	4693.4	
512	3073.4	4220.9	
640	2956.2	4153.6	
768	2932.1	4001.3	
896	2864.8	4088.4	NA
1024	2823.2	3913.5	
1152	2834.0	3934.3	
1280	2771.2	3882.3	
1408	2804.8	3732.4	
1536	2761.9	3818.9	
2048	2721.9	3504.8	
2560	2796.8	3237.4	
4096	2718.2	2480.9	
8448	2764.4	684.8	402.4
16384	719.1	603.4	441.5
20480	781.1	596.8	492.3
24576	561.4	559.9	563.9
25600	559.8	561.0	563.6

TABLE 5.3: Time of multiplication of two 64×64 matrices in seconds
Time in case of using real stack only = 57.5 seconds

5.10 CONCLUSION

A solution is presented for the problem of limited stack size in mini- and micro-computers. A virtual stack facility (VSF) is developed. No additional special hardware is required for the implementation of the VSF. A new simple replacement algorithm is given which requires some programming skills but gives better results for the considered test problems than classical replacement algorithms. The VSF could be used in FE packages without excessive housekeeping procedures. Further studies could be done in this direction. Among them is the development of new algorithms to implement the VSF, using hardware or firmware to perform the VSF functions which will speed up its operation by an order of magnitude and finally, the use of dedicated microprocessors to perform VSF thus relieving the operating system from the burden of VSF overheads.

CHAPTER 6

GENERAL PURPOSE MATHEMATICAL SOFTWARE

FOR THE FINITE ELEMENT METHOD

TABLE OF CONTENTS

6.1 INTRODUCTION

It has been shown in Chapter 4 that FE software can be classified into two main categories: Engineering and Mathematical. In Chapter 4 some FE programs of engineering type have been developed and presented and used to solve some engineering problems. In this chapter, a general purpose mathematical software program for the FEM is presented. This package will be used to solve some groundwater problems in the next chapter. In mathematical FE software the starting point is the partial differential equation (PDE) itself with the boundary conditions specified on a 2-dimensional domain. The early mathematical software for PDE was based on finite differences. However, after the FEM became more popular, much FE-based mathematical software was developed. Among these is the TWODEPEP package which is now available as one of the IMSL products [IMSL, 1983]. This package was originally developed by G. Sewell. The original version was capable of solving a single linear elliptic equation in a polygonal region with simple boundary conditions. Since then, many enhancements have taken place. The capabilities of the TWODEPEP preprocessor will be detailed in a later section but to mention, the package is now capable of handling nonlinear, time dependent problems, more than one equation can be solved simultaneously, irregular and curved boundaries are accommodated and even more involved boundary conditions can be considered.

6.2 REQUIREMENTS OF A GENERAL PURPOSE MATHEMATICAL FE SOFTWARE PACKAGE

It has been shown in Chapter 4 that FE software attributes are numerous and their relative importance are dependent on the purpose for which they will be used. Despite this, there are general requirements which are desirable in any computer software in general, and for mathematical FE software in particular. These can be listed as follows:

(i) Generality: Since many of the physical phenomena can be modelled using the same PDE, it implies that the more general the mathematical FE software (MFES) is, the wider the range of applications which can be handled. Since it is impossible to have a single MFES that can handle all types of PDE's, a wise choice is to select the class of PDE's that covers many applications. Further it can be safely said that the class of second order PDE's satisfy this requirement and in fact the TWODEPEP program handles a wide class of these problems. However, it is possible that by reformulating the original model, problems that cannot be handled directly may be tackled by the MFES.

(ii) Easy user interface: One of the most critical points to the success of a MFES is its user interface. This can be realised through efficient easy-to-use pre- and post-processors. Since the main model to be solved by MFES is a DE, the input data sets should be better specified in equation-like form. This requirement is satisfied by the presented package as will be demonstrated when describing the method to specify the governing equation(s), its boundary conditions together with other relevant parameters.

(iii) Expandability: Software is a dynamic process which means that the developed MFES will normally be upgraded and extended to handle more problems which were not originally be handled. This requires that the

MFES be expandable and has an open-ended design. Features like modularity and top-down design are likely to allow for more expandability. Expandability can be realized also by allowing the user to program within the environment supplied by the MFES. In fact, the presented software do possess many features of expandability. It allows the user definition of Fortran subprograms within the TWODEPEP.

(iv) Portability: Due to the numerous hardware available, MFES must be portable so that it can be moved from one machine to another. Portability is usually realised through the use of a high level programming language like Fortran. However, since MFES are usually complicated, many of these packages require the accessing of some supporting libraries which may not be available on all machines. To limit the effect of this feature and other similar ones, the parts of codes which contain these particular features must be isolated. Some special features of some computers can be utilized to increase the efficiency of the MFES at the expense of portability. Such features should not be utilized unless the increase in efficiency is considerable.

(v) Efficiency: MFES is one of the heaviest computational software. It is not abnormal for an analysis of a time-dependent complex problem to consume several hours of processing time. Efficienct algorithms and the efficient implementation of them is thus a critical requirement for a MFES. It should be noted, however, that usually special purpose packages are more efficient for the class of problems they can handle as compared to general purpose ones. Efficiency can be increased also if special features of the hardware can be utilized. Again this is in conflict to portability.

(vi) Reliability: This very important requirement can be realized if

sufficient test problems have been tested against the considered MFES and the results are compared with other methods. This in turn indicates the importance of setting standards for FE software in general as explained in Chapter 4. Since any complicated software is normally not 100% error free, a reliable software must be capable of detecting error conditions and implement some error indicators inside it. Unfortunately, this capability is not implemented in most of the known MFES. The TWODEPEP is not an exception.

Considering the above-mentioned requirements it is clear that the TWODEPEP satisfies most of them. Two exceptions are noticed: the dependence of the TWOPLOT, which is the post processor of the TWODEPEP, on a plotting supporting library as will be indicated later, and the second is the lack of error measures of the computed solutions. However, we must notice that it seems that the vast majority of FE software have the same situations.

6.3 THE PROBLEM DEFINITION

The TWODEPEP is a small, easy-to-use finite element package which can be used to solve a large class of elliptic (steady-state), parabolic (time-dependent) and eigenvalue partial differential equations problems in general two-dimensional regions. Moreover, some hyperbolic equations can be solved provided that they possess smooth initial and boundary conditions. In fact, this package satisfies most of the requirements of the general MFES as illustrated earlier. It contains a pre-processor, a FE processor and a post-processor. Its structure will be explained later, but now the problem definition will be explained. Considering two PDE's in a general 2-D region it is possible to write these equations in its general form as:

$$C1(x,y,u,v,t)\frac{\partial u}{\partial t} = \frac{\partial}{\partial x} OXX(x,y,u_x,u_y,v_x,v_y,u,v,t) + \frac{\partial}{\partial y} OXY(x,y,u_x,u_y,v_x,v_y,u,v,t) + F_1(x,y,u_x,u_y,v_x,v_y,u,v,t) \quad (6.1)$$

and,

$$C2(x,y,u,v,t)\frac{\partial v}{\partial t} = \frac{\partial}{\partial x} OYX(x,y,u_x,u_y,v_x,v_y,u,v,t) + \frac{\partial}{\partial y} OYY(x,y,u_x,u_y,v_x,v_y,u,v,t) + F_2(x,y,u_x,u_y,v_x,v_y,u,v,t) \quad (6.2)$$

for (x,y) in the region R with:

$$u = FB1(s,t) \quad (6.3)$$

and

$$v = FB2(s,t) \quad (6.4)$$

for s on part of the boundary (∂R_1)

and

$$OXX.n_x + OXY.n_y = GB1(s,u,v,t) \quad (6.5)$$

and

$$OYX.n_x + OYY.n_y = GB2(s,u,v,t) \quad (6.6)$$

on (∂R_2) where (n_x, n_y) is the unit outward normal to the boundary and:

$$u = U_o(x,y) \ , \tag{6.7}$$

and

$$v = V_o(x,y) \tag{6.8}$$

for $t=T_o$.

The package can be used to solve up to five simultaneous equations per set of equations and up to five of these sets is allowed as will be explained later. For the sake of simplicity and readability, the case of two equations will be used in other parts of this chapter. Simpler forms of these equations are used in special cases, for example, in elliptic equations (steady-state) the derivatives with respect to t will vanish with the initial conditions expressed in (6.7) and (6.8). These will be explained later when solving steady-state equations.

6.4 DOMAIN OF APPLICATIONS

This general purpose MFES can be used to solve a wide range of applications in Applied Mathematics, Physics and Engineering. Some of these applications are listed here:

(i) Elasticity problems in two dimensional regions. The 2-D elasticity equations may be written as:

$$\frac{\partial}{\partial x}\sigma_x + \frac{\partial}{\partial y}\sigma_{xy} + F1(x,y,u,v) = 0 \tag{6.9}$$

and

$$\frac{\partial}{\partial x}\sigma_{xy} + \frac{\partial}{\partial y}\sigma_y + F2(x,y,u,v) = 0 \tag{6.10}$$

where (u,v) is the displacement vector, F1 and F2 is the body force vector (force per unit volume) and σ_x,σ_y and σ_{xy} are the tensile stresses in the x,y directions and the shear stress, respectively. Stresses can be related to strains as explained in Chapter 2. For completeness, in case of plane stress, the relationship between stresses and strains can be expressed as:

$$\sigma_x = E(\varepsilon_x+\nu\varepsilon_y)/(1-\nu^2) \tag{6.11}$$

$$\sigma_y = E(\varepsilon_y+\nu\varepsilon_x)/(1-\nu^2) \tag{6.12}$$

$$\sigma_{xy} = \frac{E}{2}\varepsilon_{xy}/(1+\nu) \tag{6.13}$$

where ν is the Poisson's ratio, E is the modulus of elasticity, ε_x, ε_y and ε_{xy} are the strains in x,y and shear strains, respectively. The boundary conditions may be specified in terms of boundary displacements on part of the boundary:

$$u = FB1(s) \tag{6.14}$$

$$v = FB2(s) \tag{6.15}$$

On the other part, boundary forces may be specified as:

$$\sigma_x n_x + \sigma_{xy} n_y = GB1(s,u,v) \tag{6.16}$$

and

$$\sigma_{xy} n_x + \sigma_y n_y = GB2(s,u,v) \tag{6.17}$$

where (GB1,GB2) is the boundary force vector (force per unit area).

It is worth mentioning that some 2-D elasticity problems have been solved using an engineering FE software; the ELASTIC package in Chapter 4.

(ii) Diffusion and Heat Conduction:

The diffusion equation may be written as:

$$\frac{\partial u}{\partial t} = \partial(-J_x)/\partial x + \partial(-J_y)/\partial y + F1(x,y,u,t) \qquad (6.18)$$

where u is the concentration, (J_x,J_y) is the flux vector and F1 is the generation rate for u due to sources and sinks. The flux may be a function of x,y,t,u and the gradient of u. In case of steady-state problems, the left-hand side of the differential equation becomes 0. On part of the boundary, the concentration may be given by:

$$u = FB1(s,t) \qquad (6.19)$$

and on the other part, the boundary flux may be given by:

$$-J_x n_x - J_y n_y = GB1(s,u,t) \qquad (6.20)$$

where GB1 is the inward boundary flux of u.

(iii) Minimal surface problem:

If U(x,y) is the vertical height of a surface above the point (x,y), then the surface which has height FB1(s) above the boundary and which minimizes its surface area satisfies:

$$\frac{\partial}{\partial x}\left[\frac{U_x}{\sqrt{1+U_x^2+U_y^2}}\right] + \frac{\partial}{\partial y}\left[\frac{U_y}{\sqrt{1+U_x^2+U_y^2}}\right] = 0 \qquad (6.29)$$

$$u = FB1(s)$$

In addition to these sample applications, many other applications can be solved using this package. Some groundwater flow problems will be solved in the next chapter using this package.

It should be noticed, however, that although this package has a wide range of applications, it has some limitations as the case with similar MFES. Some of the main limitations can be summarized as:

(i) Problems in 2-D can be solved but in general 3-D cannot except if they are axisymmetric whence they can be considered as 2-D problems in r-z geometry.

(ii) Models based on other equation-types like integral equations cannot be solved directly unless a transformation can be found to convert them to the standard TWODEPEP form.

(iii) If the boundary conditions are functions of u_x,u_y,v_x or v_y, then the package cannot solve the problem, i.e. the boundary conditions must not be a function of the derivatives of u or v to be handled by the package.

(iv) The boundary conditions on a particular arc may be of one type only but not mixed. This difficulty can be overcome by converting Dirichlet conditions into equivalent Neumann conditions on these arcs where the condition u=FB1(s,t), (say) can be written as:

$$OXX.n_x + OXY.n_y = -BETA*(u-FB1(s,t)) \qquad (6.22)$$

where BETA is a very large number 10^{20} say, which can be handled by the package.

(v) Only one type of triangle can be used throughout the whole region at a time. It is not possible to have higher order elements in parts of the region (p-refinement) but rather, element condensation (h-refinement) can be specified only.

6.5 THE PACKAGE STRUCTURE

The package consists of three major parts: the pre-processor, the FE processor and the post-processor. The pre-processor accepts the user input and generates a driving main program and some problem-dependent subprograms. These when compiled and linked with other problem-independent modules result in the FE processor which in fact includes some of the pre-processing functions like mesh generation and node numbering. So, it is actually more convenient, in my opinion, to call the pre-processor as an interpreter or preprocessor compiler since its function is not exactly as understood by classical FE pre-processors. However, the outputs of the FE analysis can be plotted using the TWOPLOT program which is a member of this package that can be used for plotting some of the results. In what follows, the structure of this package is detailed. Much of this material is derived directly or indirectly from the TWODEPEP manual [IMSL, 1983] or from the listing of the source modules.

6.5.1 The Pre-Processor

The pre-processor used in this package allows the user to write the problem definition in an easy readable manner. It is used to control the dimensions size so that the required core is allocated to the problem. This pre-processor is a Fortran program which reads the input supplied by the user in the specified format and creates a Fortran program which consists mainly of the main segment (the driver) and any user supplied functions plus some additional functions which are problem dependent. This new Fortran program has then to be compiled and linked with other general problem independent modules to form the final program

to solve the required problem. Thus, in fact this preprocessor is of the type used in Chapter 5 to compile the VSF programs. It is customary to call these type of "compilers" as: interpreters, pre-processors or cascade compilers as has been detailed in Chapter 5. The FE pre-processors functions, like mesh generation and node numbering are in fact an integral part inside the generated Fortran program.

6.5.2 The Mesh Generation

The user supplies an initial triangulation with only enough triangles to define the region and associated boundary conditions. The mesh consists of triangles which may be quadratic, cubic or quartics. Only one type can be used throughout the whole region. The user specifies the maximum number of triangles to be created and a particular optimal function to guide the refinement of this triangulation is explained later. Triangulation is done by dividing each triangle by a line from the midpoint of its longest side to the opposite vertex. If this side is not on the boundary, the triangle which shares that side must also be divided to avoid nonconforming elements and discontinuous basis functions.

The element library of this package consists of three different element types only: the standard six nodes triangle with quadratic basis functions, the 10-nodes cubic triangle and the 15-nodes quartic triangle. In all cases one of the edges may be curved when adjacent to a curved boundary. These elements have been described in Chapter 3 and will not be repeated here.

For the time dependent problems, either the implicit or Crank-Nicolson scheme may be used to do time discretization. Optionally, a

Richardson extrapolation may be done to double the order of convergence of the time discretization. All these techniques have been developed and presented in Chapter 3.

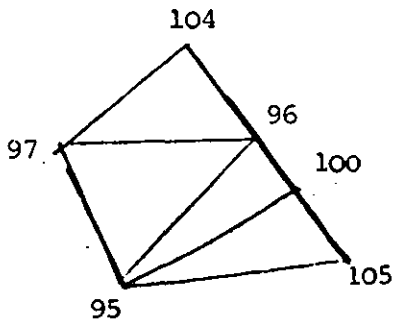
6.5.3 Node Numbering

As explained earlier, node numbering is very important in the FEM. Solution time of FE equations is highly dependent on the master matrix bandwidth which in turn is proportional to the maximum difference between two node numbers in the same element. The node numbering used in this package is a modified version of the well-known Cuthill-McKee (CM) strategy which has been presented before. The first step, is therefore, to number the nodes according to the CM algorithm. Assume that the Jacobian semi-bandwidth is M. Then, compute the local semi-bandwidth of each triangle K which is the maximum difference between any two node numbers inside each triangle. Assume that there are N triangles of local semi-bandwidth equal M. The algorithm proceeds then, as follows:

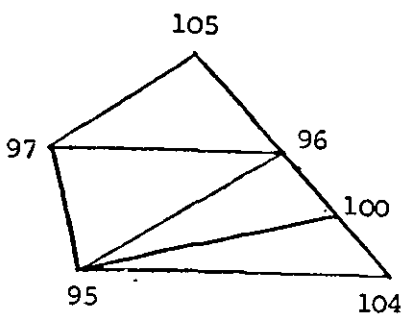
(1) Look at each of the N triangles. These "critical" triangles are considered since any decrease in their local semi-bandwidth may result in a similar decrease in the Jacobian semi-bandwidth. Consider the node with the highest number in each of these triangles and start switching this node number with the nodes which number are less than this by 1,2 and 3 and determine whether this will result in any advantage or not. By advantage we mean that either M will be decreased, or, the number N will be decreased, i.e. the number of critical elements will be decreased.

(2) The same procedure may be repeated in the opposite direction, i.e. look to the lowest node number in the critical triangles and consider

the switching with the nodes which numbers are +1, +2, and +3 to test whether it is advantageous or not. The following is an example for illustration of the node number switching concept: Consider the part of the FE mesh shown in Figure (6.1). Assume that the critical triangle is {95,100,105} i.e. the semi-bandwidth of the Jacobian=10 [Figure 6.1(a)]. Switching nodes 105 and 104 will result in decreasing the semi-bandwidth to 9 which is advantageous as shown in Figure 6.1(b). This heuristic usually can decrease the semi-bandwidth of the Jacobian by up to 20%.



(a) Original CM numbering, semi-bandwidth = 10



(b) Switch nodes 105 and 104, semi-bandwidth = 9

FIGURE 6.1: Node numbering switching

6.5.4 Solution Methods

The resulting algebraic equations, which may be linear or non-linear, are solved by Newton's method. One iteration is done. The system of linear equations which must be solved in each iteration cycle is solved directly by block Gaussian elimination. The core store required to solve the equations is given by (approximately),

$$Ndim = 12(ntf)^{1.5}(Neq)^{2}(Nsym)(Ndeg) \qquad (6.23)$$

where:

Ndim is the size of required core store,

ntf is the final number of triangles in the FE model,

Neq is the number of simultaneous PDE's being solved,

Nsym is a factor = 1 if the problem is symmetric and

= 2 otherwise,

Ndeg is a factor dependent upon the chosen triangles:

= 1 for quadratic triangles

= 5 for cubic triangles

= 16 for quartic triangles.

However, if the available core is not enough, then a standard frontal algorithm is used to efficiently solve the equations.

The core store required in this case is given by:

$$Ndim = 20(ntf)(Neq)^{2}(Nsym)(Ndeg) \qquad (6.24)$$

It should be noted that, assuming 100 triangles are created in the final triangulation, the ratio of core store required in the case of using an in-core solver to that of out-of-core is 6:1 which reflects the great savings in memory requirements. This saving is countered by more execution time and the use of auxiliary devices (disc files) for the frontal process. During the frontal solution process the triangles

are assembled in an order such that the minimum node number of a triangle K, written as MNN_k, is a non-decreasing sequence. For example, if we assemble the triangle {95,100,105}, then in the next step we assemble a triangle with minimum node number ≥ 95. Thus during the assembly of triangle k, then only those elements (I,J) of the Jacobian with $MNN_k \leq I \leq J \leq MNN_k + M$ need to be stored in core where M is the semi-bandwidth. This is obvious since that triangle makes contributions only to those elements assuming a symmetric Jacobian. Thus, only about $\frac{1}{2}M^2$ storage locations are required in main memory at any time. To illustrate, we assume that the last assembled triangle is {95,100,105}, and we assume that the next triangle to be assembled is {96,102,106}. It is clear that the elements of the Jacobian to be affected by the assembling of this triangle are: (96,96), (96,102), (96,106), (102,102), (102,106) and (106,106). After assembly of the triangle number k, the rows (MNN_k) to ($MNN_{k+1}-1$) are eliminated and written to a disk file. This is logical since no more contributions to these rows will occur due to the assembly of the remaining triangles. In fact, the package uses a buffer array to decrease the number of I/O actually needed by the frontal algorithm.

6.5.5 Subprograms

The package consists of a preprocessor which generates a Fortran program. This generated Fortran program is linked with a set of problem-independent routines to form the final program that is executed to solve the required problem. The main generated program is the subroutine TDPA which calls, directly or indirectly, most of the other subroutines. The subroutines and their functions are briefly given in Table 6.1.

NAME	FUNCTION
TDPA	Assigns default values, checks input parameters and does the major solution steps by calling other subroutines.
TDPB	Assembles and performs elimination on the Jacobian matrix by Newton's method. It also determines whether the Jacobian can be stored internally or not.
TDPC	Renumber the free nodes starting from a corner to reduce the Jacobian bandwidth using the algorithm described in Section 6.5.3.
TDPD	Calculates the parameters associated with the final triangulation.
TDPE	Splits a triangle K1, and also K2 if necessary to preserve conformity.
TDPF	Calculates some parameters associated with the initial triangulation.
TDPG	Evaluates the functions at the nodal points
TDPH	Plots, on the printer, the initial triangulation and the vertices and centers of the triangles in the final triangulation.
TDPI	Stores several rows of the Jacobian matrix and writes them together, or reads several lines together, to reduce the input/output counts and thus the computational cost.
TDPJ	Checks for non-symmetry and incorrect user-supplied partial derivatives.
TDPK	Calculates Ainv which is the inverse of the matrix A.

continued....

NAME	FUNCTION
TDPL	Copies a vector or sets the values of its entries to zeros according to the IJOB parameter.
TDPM	Updates the matrix A.
TDPN	Calculates the values and derivatives of the interpolation functions and isoparametric mapping functions.
TDPO	Calculates the next step size based on the user-supplied function DTINV.
TDPP	Calculates the integral of the function DTINV.
TDREAD	Reads data stored by a previous run of the package. This routine is called only by the user and thus need not be loaded unless referenced in the input data set.

TABLE 6.1: Package subroutines

6.5.6 The Postprocessor

A simple plotting program can be used to draw some of the outputs produced by the FE processor. The name of this program is TWOPLOT which is also programmed in Fortran. It requires Calcomp's host computer basic software - HCBS - which is a library of Fortran subroutines used to derive the plotter. TWOPLOT produces the required plots on a file with unit number equal 19 which is the plot file. This is usually a magnetic tape file with plotting commands which is used to produce the required plots. Specifications of the inputs to this program and the type of plots which can be produced will be given later.

6.6 INPUT DATA SETS

Input data are entered in free format except in function and parameters definitions where the line is divided to zones. The order of the lines is unimportant except as expressly specified. No continuation lines are required or allowed except in topology definition as will be explained. Arithmetic expressions are coded in Fortran style and the default type for Fortran variables (the IJKLMN rule) applies to all user defined variables except if the double precision version of the package is used where all real variables are defaulted to double precision. Comment lines are identified by **** in the first four columns. Lines can extend to column 72 as usual. The last line in the input must have 'END.' in the first four columns.

6.6.1 The Global Variables

The first line of input data is defining the global variables of the problem to be solved. There are 5 integers required and are given in free format. Those are in the following order:

NEQ: number of simultaneous PDE's to be solved.

NT: number of triangles in the initial triangulation.

NV: number of vertices in the initial triangulation. Note that each is counted only once.

NTF: number of triangles desired in the final triangulation.

NDIM: the storage reserved for the Jacobian matrix.

NDIM could be either an integer number giving the size of the matrix as defined by the user or it can be set to 1 or 2 to default to in-core evaluation or out-of-core as defined by equations (6.23) and (6.24). It is important to notice that the processing time is increased considerably

when using the out-of-core algorithm. However the total cost may still be less due to the fact that memory cost may sometimes exceed the processing time cost. The NEQ value may be 1 or 2, but for solving more equations NEQ is coded as will be explained in 6.7.2. In recent versions of the package (5 and later), the variables NT and NV are omitted from the first line and computed by the package automatically from topology data.

6.6.2 Specifying the Equations

The PDE's are described in a series of lines according to the following rules:

(i) A variable name, as given in TWODEPEP notation, starts from column 1.

(ii) In columns 9-72, the definition of that variable or function is coded in standard Fortran syntax.

(iii) All the functions or variables given below should be defined or defaulted.

(iv) The sequence of lines is unimportant.

(v) No continuation lines are allowed. If any function definition is too long to fit onto a single line, Fortran functions (subprograms) could be used within that definition. These functions are then defined as a subprogram after the 'ADD.' command.

(vi) If any function of the form A/B is defaulted, it means that the actual value will be computed by the package if $\partial A/\partial B$ is independent of B. This is denoted by * in the following table.

The definitions of these functions are:

NAME	DEFAULT	MEANING
OXX	0	OXX(X,Y,UX,UY,VX,VY,U,V,T)
OXX/UX	*	∂/∂(UX)*OXX(X,Y,UX,UY,VX,VY,U,V,T)
OXX/UY	*	∂/∂(UY)*OXX(X,Y,UX,UY,VX,VY,U,V,T)
OXX/VX	*	∂/∂(VX)*OXX(X,Y,UX,UY,VX,VY,U,V,T)
OXX/VY	*	∂/∂(VY)*OXX(X,Y,UX,UY,VX,VY,U,V,T)
OXX/U	*	∂/∂(U)*OXX(X,Y,UX,UY,VX,VY,U,V,T)
OXX/V	*	∂/∂(V)*OXX(X,Y,UX,UY,VX,VY,U,V,T)
OXY	0	OXY(X,Y,UX,UY,VX,VY,U,V,T)
OXY/UX	*	∂/∂(UX)*OXY(X,Y,UX,UY,VX,VY,U,V,T)
OXY/UY	*	∂/∂(UY)*OXY(X,Y,UX,UY,VX,VY,U,V,T)
OXY/VX	*	∂/∂(VX)*OXY(X,Y,UX,UY,VX,VY,U,V,T)
OXY/VY	*	∂/∂(VY)*OXY(X,Y,UX,UY,VX,VY,U,V,T)
OXY/U	*	∂/∂(U)*OXY(X,Y,UX,UY,VX,VY,U,V,T)
OXY/V	*	∂/∂(V)*OXY(X,Y,UX,UY,VX,VY,U,V,T)
OYX	0	OYX(X,Y,UX,UY,VX,VY,U,V,T)
OYX/UX	*	∂/∂(UX)*OYX(X,Y,UX,UY,VX,VY,U,V,T)
OYX/UY	*	∂/∂(UY)*OYX(X,Y,UX,UY,VX,VY,U,V,T)
OYX/VX	*	∂/∂(VX)*OYX(X,Y,UX,UY,VX,VY,U,V,T)
OYX/VY	*	∂/∂(VY)*OYX(X,Y,UX,UY,VX,VY,U,V,T)
OYX/U	*	∂/∂(U)*OYX(X,Y,UX,UY,VX,VY,U,V,T)
OYX/V	*	∂/∂(V)*OYX(X,Y,UX,UY,VX,VY,U,V,T)
OYY	0	OYY(X,Y,UX,UY,VX,VY,U,V,T)
OYY/UX	*	∂/∂(UX)*OYY(X,Y,UX,UY,VX,VY,U,V,T)
OYY/UY	*	∂/∂(UY)*OYY(X,Y,UX,UY,VX,VY,U,V,T)
OYY/VX	*	∂/∂(VX)*OYY(X,Y,UX,UY,VX,VY,U,V,T)
OYY/VY	*	∂/∂(VY)*OYY(X,Y,UX,UY,VX,VY,U,V,T)
OYY/U	*	∂/∂(U)*OYY(X,Y,UX,UY,VX,VY,U,V,T)
OYY/V	*	∂/∂(V)*OYY(X,Y,UX,UY,VX,VY,U,V,T)
F1	0	F1(X,Y,UX,UY,VX,VY,U,V,T)
F1/UX	*	∂/∂(UX)*F1(X,Y,UX,UY,VX,VY,U,V,T)
F1/UY	*	∂/∂(UY)*F1(X,Y,UX,UY,VX,VY,U,V,T)
F1/VX	*	∂/∂(VX)*F1(X,Y,UX,UY,VX,VY,U,V,T)
F1/VY	*	∂/∂(VY)*F1(X,Y,UX,UY,VX,VY,U,V,T)
F1/U	*	∂/∂(U)*F1(X,Y,UX,UY,VX,VY,U,V,T)
F1/V	*	∂/∂(V)*F1(X,Y,UX,UY,VX,VY,U,V,T)
F2	0	F2(X,Y,UX,UY,VX,VY,U,V,T)
F2/UX	*	∂/∂(UX)*F2(X,Y,UX,UY,VX,VY,U,V,T)
F2/UY	*	∂/∂(UY)*F2(X,Y,UX,UY,VX,VY,U,V,T)
F2/VX	*	∂/∂(VX)*F2(X,Y,UX,UY,VX,VY,U,V,T)
F2/VY	*	∂/∂(VY)*F2(X,Y,UX,UY,VX,VY,U,V,T)
F2/U	*	∂/∂(U)*F2(X,Y,UX,UY,VX,VY,U,V,T)
F2/V	*	∂/∂(V)*F2(X,Y,UX,UY,VX,VY,U,V,T)
C1	0	C1(X,Y,U,V,T)
C2	0	C2(X,Y,U,V,T)
U0	0	U0(X,Y)
V0	0	V0(X,Y)

Hint: (For the elliptic case, U0,V0 are used as initial values for Newton's method. They may be defaulted for linear elliptic problems.)

6.6.3 Specifying the Element Parameters

Three different parameters can be specified for the elements:

(i) D3EST which is a function of x,y that determines the distribution of the final triangulation over the region. This is defaulted to 1. More discussion will be given later. This function is specified by writing the keyword D3EST in the name field (columns 1-5) and the function definition in Fortran in columns 9-72.

(ii) The triangle ratio Δy (height) to width Δx. This ratio is defaulted to 1, however, it may be defined as a function of x,y using the SHAPE definition on a single line.

(iii) The default element type is the quadratic elements. To define other types, the keywords CUBICS or QUARTICS can be used.

6.6.4 Specifying Computational Parameters

These are some computational parameters and options that can be specified as follows:

(i) NUPDT: which is the number of time steps (iterations) between updates of the Jacobian matrix. The default is 1 which is normally required for elliptic problems. For eigenvalue problems, this parameter should be set to 0. If the problem is linear and all the functions are independent on time, then NUPDT is set to 0.

(ii) ALPHA: This is a variable used to specify the method to be used for handling the time variable. This can be set to 1 (as defaulted) if the problem is elliptic or if the implicit method is used to solve a parabolic problem. ALPHA is set to .5 if the Crank-Nicolson method is used to solve a parabolic problem. Although the Crank-Nicolson method has a higher accuracy, it can lead to oscillation in a few problems initially.

(iii) INTEGRAL: this can be a function of x,y,u,v,u_x,u_y,v_x,v_y and t whose integral over the region R is to be computed and output each NOUT steps. The default value is 0, i.e. no integration is to be done.
(iv) DTINV: this function can be used to define a variable time step at each time value, T. Thus, at time T, the step size will be $\frac{1}{DTINV(T)}$ The integral of DTINV from t_0 to t_f, i.e. over the whole time span must be finite. DTINV is defaulted to 1.

6.6.5 Specifying the Topology

To define the topology, initial triangulation information should be given that covers the whole domain without overlapping. The first step is, as usual, to divide the region into triangles (even if one side of the boundary is curved). Practically, one should proceed as follows to discretize the FE model and to define the topology in TWODEPEP notation.

(i) The boundary of the region R should be divided into distinct arcs, each of which is smooth with smooth boundary conditions. Consequently, at every corner or point at where the boundary conditions have a discontinuity or change type, a new boundary arc must begin.
(ii) Each arc is assigned an ID-number which should be an integer. This number must be positive if the boundary conditions applied on that arc are of the Neumann type i.e. if the arc belongs to ∂R_2 (i.e. free boundary conditions with GB1 and GB2 given along the arc). Arc number should be negative if a Dirichlet type boundary condition is specified on that arc, i.e. if the arc belongs to ∂R_1 (i.e. fixed boundary conditions with FB1 and FB2 given along the arc).

(iii) Each arc (if not a straight line) is described by a parameter S, varying from 0 to 1 to define its shape. The orientation of the arc is unimportant. For example, a circle can be specified by the parametric equations: $x=\cos(2\pi s)$ and $y=\sin(2\pi s)$.

(iv) Then an initial triangulation of R which possesses the following properties should be done:

- Each point where two of the boundary arcs meet should be considered a vertex in the triangulation.
- Triangles should meet at vertices only, i.e. conforming triangulation is done.
- No triangle may have all three vertices on the boundary.

(v) Now, to pass the above mentioned information to the package we specify the following three arrays:

(1) The vertices array VXY which contains the coordinates of all the vertices of the initial triangulation. Data are given in free format and if more than one line is needed to specify the data, the array name should be repeated on each continuation line. The vertices may be listed in any order, but that order will define the vertex numbers as referred to in the IABC array. The VXY array is coded as follows:

```
VXY     VX(1),VY(1),VX(2),VY(2),...VX(NV),YY(NV)
```

(2) The triangles array IABC which is coded as follows:

```
IABC    IA(1),IB(1),IC(1),IA(2),(B(2),IC(2),...
           IA(NT),IB(NT),IC(NT)
```

where IA(K), IB(K), IC(K) are the numbers (as listed in VXY) of the vertices A,B,C of the triangle K. A,B,C must be ordered counter-clockwise and such that C is not on the boundary.

(3) The arc-triangle array I which is coded as follows:

I I(1),I(2),...,I(NT),

where I(K) is the boundary arc number cut-off by the base AB of the triangle K. If the whole triangle K does not cut any arc, then I(K) is set to O.

(vi) If the region R is a rectangle, bounded by only 4 boundary arcs, X=X1, X=X2, Y=Y1 and Y=Y2, then the initial triangulation may alternatively be specified by superimposing a rectangular grid over R. TWODEPEP will automatically generate 4 equal area triangles in each grid cell. In this case, we define the following arrays:

XGRID	xgrid(1),...,xgrid(NT)
YGRID	ygrid(1),...,ygrid(NV)
IX	Ix(1),Ix(2) only
IY	Iy(1),Iy(2) only

In this case NT and NV on the first input line, will give the number of elements in arrays XGRID and YGRID, respectively, rather than the number of triangles and vertices in the initial triangulation. This is particularly useful in case of problems of only one space dimension. XGRID(1) is the X coordinate of the vertical grid line number i. Note that XGRID(1)=X1 and XGRID(NT)=X2. YGRID(1) is the Y coordinate of the horizontal grid line number i. Note that YGRID(1)=Y1 and YGRID(NV)=Y2.

Ix(1) and Ix(2) are the identifying integers of the boundary arcs X=X1 and X=X2. Similarly IY(1) and IY(2) are for the boundary arcs Y=Y1 and Y=Y2.

6.6.6 Specifying the Boundary Conditions

To define the boundary conditions of the considered problem, we should give the arc number on a separate line started by ARC=arc number and follow that line by lines that define the x,y,FB1,FB2,GB1,GB1/u,

GB1/v, GB2, GB2/u and GB2/v. The internal sequence of these lines is not important. Any of these boundary functions can be defaulted if this is applicable. These can be defined as follows:

NAME	DEFAULT	MEANING
X	line	X=X(S)} (0.LE.S.LE.1) parametric equations of arc
Y	line	Y=Y(S)} number I. Defined for curved arcs only.
FB1	0	FB1(S,X,Y,T) on arc number I (I negative)
FB2	0	FB2(S,X,Y,T) on arc number I (I negative)
GB1	0	GB1(S,X,Y,U,V,T) on arc number I (I positive)
GB1/U	*	∂/∂(U)*GB1(S,X,Y,U,V,T) on arc number I (I positive)
GB1/V	*	∂/∂(V)*GB1(S,X,Y,U,V,T) on arc number I (I positive)
GB2	0	GB2(S,X,Y,U,V,T) on arc number I (I positive)
GB2/U	*	∂/∂(U)*GB2(S,X,Y,U,V,T) on arc number I (I positive)
GB2/V	*	∂/∂(V)*GB2(S,X,Y,U,V,T) on arc number I (I positive)

6.6.7 Specifying the Outputs

Several types of data are required to fully describe the outputs. These are:

(i) Specification of output steps:

(1) For the time variable three parameters should be defined (or defaulted); TO, TF and DT where:

TO is the initial time value, default is 0

TF is the final time value, default is 1

DT is the average time step size, default is 1

For elliptic problems, Newton's method will be iterated (TF-TO)/DT times. Here, TO and DT are normally defaulted (i.e. TO to 0 and DT to 1). TF will be, in this case, the number of iterations desired. If convergence had occurred earlier, then iteration will be stopped unless DT is given

a value other than 1. Note that, however, if the elliptic problem is linear, then only 1 iteration will be needed. The output will be done each NOUT time steps. NOUT is defaulted to 1.

(2) For the x-y variables, the following parameters should be defined:

XA is the first value of x at which the solution is required.

YA same as XA but in the y direction.

The default values for XA and YA are X_{min} and Y_{min}.

NX is the number of points at which the solution is required after the first one.

NY same as NX but for the Y direction.

The default values for NX and NY is 4.

HX interval in X direction

HY interval in Y direction

HX is defaulted to $\frac{X_{max}-X_{min}}{NX}$

HY is defaulted to $\frac{Y_{max}-Y_{min}}{NY}$

Note that the solution will be given at the grid points:

$$X = XA + iHX, \quad i=0,1,2,\ldots,NX$$

$$Y = YA + jHY, \quad j=0,1,2,\ldots,NY$$

(ii) Specification of unit number:

The logical unit number for output is defaulted to 6 which is the line printer in most operating systems for Fortran programs. However, it may be changed if, for example, the outputs are to be kept on a disc file for subsequent postprocessing. This can be done using the variable MWR which can be coded as:

```
MWR   8,9
```

(iii) Optional printer plotting

It is possible to produce printer "plotting" for the initial triangulation (for input checking) and for the centres and vertices of the triangles in the final triangulation. This is specified by:

PLOT 1

The default is PLOT 0, i.e. no plotting.

(iv) Variables to be printed

By default the variables U,V,OXX,OXY,OYX and OYY are printed for each grid point at each time step. In other words for time $t=t_1$, all the values are printed for all the points required in the region and then, the time is incremented and printouts are produced for the second time step and so on. To override these printed values, we can define the following variables in terms of all variables and print them instead of the defaulted values. These are:

UPRINT	for U
VPRINT	for V
OXXPRINT	for OXX
OXYPRINT	for OXY
OYXPRINT	for OYX
OYYPRINT	for OYY

It is worthwhile mentioning that the package outputs consist of:

(1) A listing of the input data as specified by the user except that the first line is printed in a formatted form.

(2) Some boundary data to be used by the postprocessor.

(3) Initial triangulation plotted on the printer for checking purposes and triangle vertices and centres for the final triangulation provided that "PLOT 1" is coded as explained earlier.

(4) For every NOUT time step; the x,y of each point as specified earlier

together with the values of the function and its derivatives are printed unless other variables are specified for printing instead.

6.6.8 Inputs of the Postprocessor

The TWOPLOT postprocessor requires the initial data to be read from unit number 5 which is the standard input device in Fortran in the vast majority of operating systems. This data is 2 lines for each plot required while the last line in the input data set for this program is simply a blank line. The first line of the input contains 4 items in free format as follows:

IFILE which is the logical unit number of the file containing the TWODEPEP solution that is to be plotted by TWOPLOT. This file (unit number is 8 or 9) was defined in the TWODEPEP operation by the statement:

```
MWR   9
```

or

```
MWR   8.
```

Note that if we put IFILE preceded by a minus sign, then, the output of TWOPLOT will not be distorted i.e. an equal scale is adopted. Otherwise, if IFILE is positive and the X range is not equal to the Y range, then, the output will be distorted, i.e. scaling in X direction is different from that of the Y direction.

ISKIP. This command tells the TWOPLOT where to find the records which contain the solution to be plotted from the file IFILE. This is simply the number of output units (corresponding to each NOUT time step) that should be skipped to reach the requested solution.

(PLOT1,PLOT2) these are two values between parenthesis to define the variable to be plotted and the type of plot required. These can be any of the following symbols:

*,U,V,OXX,OXY,OYX,OYY,-U,-V,-OXX,-OXY,-OYX,-OYY and 3Dnn.

If (PLOT1,PLOT2) is =(*,*) it implies that plotting of the principle stresses are required. If (PLOT1,PLOT2)=(*,PLOT2) then vertical arrows will be plotted with magnitudes proportional to the scalar PLOT2 [similarly for (PLOT1,*)].

If (PLOT1,PLOT2)=(3Dnn,PLOT2), then, a three dimensional perspective plot of the scalar PLOT2 will be generated with longitudinal view angle of nn degrees. Note that nn must be between 10 and 80 - a suggested value is 45. The latitudinal view angle is always 45.

The second line of input data for the plot is its title written in the first 40 characters.

TWOPLOT produces the required plots on output unit 19 which is the plotfile. This is typically, a magnetic tape file with standard plot characteristics. This file is handed to the computer operator for off-line plotting. It should be noticed that this postprocessor is, in fact, of fairly limited capabilities compared to the general FE postprocessors as explained in Chapter 4.

6.6.9 Examples

Here some illustrative examples are given to demonstrate the simplicity of input data sets of the package in addition to its reliability.

Example 6.1

Solve,

$$(\omega\frac{\partial u}{\partial x})_x + (\omega\frac{\partial u}{\partial y})_y = 1 \qquad (6.25)$$

over a 0.5×0.5 square for the ω values: 1,10 and 100, with boundary condition u=0 along all sides.

Solution:

The initial triangulation is shown in Figure 6.2 which consists of 4 triangles,

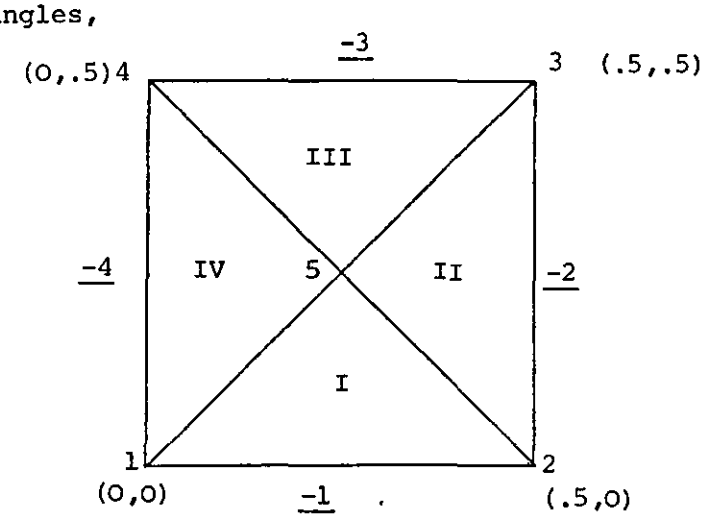


FIGURE 6.2: Initial triangulation for Example 6.1.

Note that nodes are numbered 1,2,3,4 and 5 while elements are numbered I, II, III, and IV. Arcs are given negative numbers as the boundary condition is a Dirichlet type. The input data set is:

```
1   4   5   50   1
OXX          UX
OXY          UY
OXX/UX       1
OXY/UY       1
F1           -1
NX           5
NY           5
XA           0.
HX           .1
YA           0.
HY           .1
PLOT         1
SYMMETRY     1
VXY    0,0    .5,0    .5,.5    0,.5    .25,.25
IABC  1 2 5  2 3 5    3 4 5       4 1 5
I       -1      -2      -3          -4
END.
```

The solution is given in Table 6.2 which is a direct output of the computer run for the case $\omega=1$. Figures 6.3 and 6.4 show the initial and final triangulation plots.

Example 6.2

The laminar flow around a circular obstable in a channel is governed by:

$$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = 0 \qquad (6.26)$$

Due to symmetry only one quarter of the region is considered, the boundary conditions and dimensions are shown in Figure 6.5.

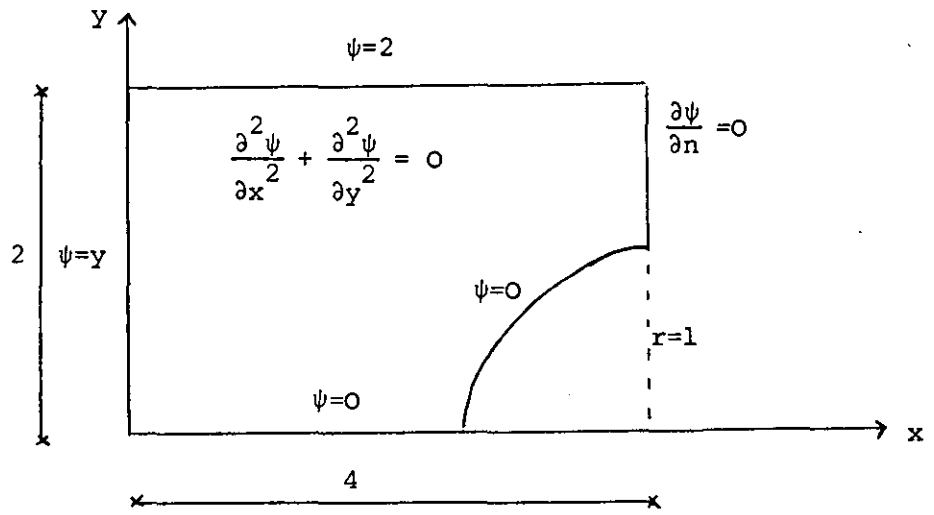


FIGURE 6.5: Laminar flow around a circular obstacle

The initial triangulation of this problem consists of 5 elements as shown in Figure 6.6.

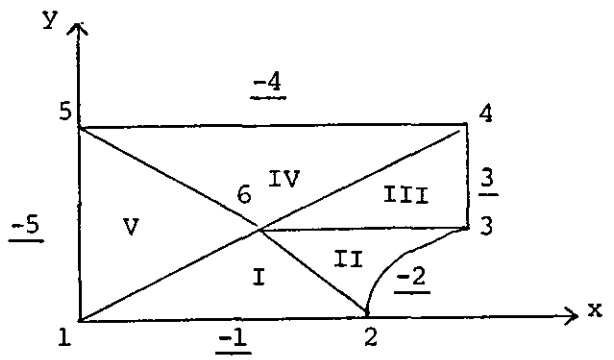


FIGURE 6.6: Initial triangulation of Example 6.2.

The input data set is as follows:

```
1      5     6   50    1
OXX            UX
OXY            UY
OXX/UX         1
OXY/UY         1
NX            10
NY            10
XA            0.0
HX             .4
YA            0.0
HY             .2
PLOT          1
CUBICS        1
ARC=-1
FB1           0
ARC=-2
X             4-COS(1.57079*S)
Y             SIN(1.57079*S)
FB1           0
ARC=3
GB1           0
ARC=-4
FB1           2
ARC=-5
FB1           Y
VXY           0.0, 3.,0.,4.,1.,4.,2.,0.,2.,2.,1.
IABC          1,2,6 2,3,6 3,4,6 4,5,6 5,1,6
I              -1     -2    3    -4    -5
END.
```

The solution of this problem is shown in Table 6.3. Plotting of initial and final triangulations are shown in Figures 6.7 and 6.8.

```
NEQ=     1 NT=     4 NV=     5 NTF=     50 NDIM=        1
OXX      =    UX
OXY      =    UY
OXX-UX   =    1
OXY-UY   =    1
F1       =    -1
NX       =    5
NY       =    5
XA       =    0.
HX       =    .1
YA       =    0.
HY       =    .1
PLOT     =    1
SYMMETRY=    1
VXY      =    0,0    .5,0    .5,.5    0,.5    .25,.25
IABC     =    1 2 5     2 3 5      3 4 5      4 1 5
I        =      -1        -2          -3         -4
END.
BOUNDARY DATA FOR TWOPLOT
 0.0         0.500D-01 0.100D+00 0.150D+00 0.200D+00 0.250D+00 0.300D+00 0.350D+00 0.400D+00 0.450D+00 0.500D+00
 0.0         0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0
 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00
 0.0         0.500D-01 0.100D+00 0.150D+00 0.200D+00 0.250D+00 0.300D+00 0.350D+00 0.400D+00 0.450D+00 0.500D+00
 0.500D+00 0.450D+00 0.400D+00 0.350D+00 0.300D+00 0.250D+00 0.200D+00 0.150D+00 0.100D+00 0.500D-01 0.0
 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00 0.500D+00
 0.0         0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0
 0.500D+00 0.450D+00 0.400D+00 0.350D+00 0.300D+00 0.250D+00 0.200D+00 0.150D+00 0.100D+00 0.500D-01 0.0
```

TABLE 6.2(a)

X	Y	U	V	UXX	UXY	UYX	UYY	
0.0	0.0	0.0	0.0	-0.312500D-01	0.0	0.0	0.0	
0.100000D+00	0.0	0.0	0.0	0.0	-0.121760D+00	0.0	0.0	
0.200000D+00	0.0	0.0	0.0	0.0	-0.159520D+00	0.0	0.0	
0.300000D+00	0.0	0.0	0.0	0.0	-0.160130D+00	0.0	0.0	
0.400000D+00	0.0	0.0	0.0	0.0	-0.121420D+00	0.0	0.0	
0.500000D+00	0.0	0.0	0.0	0.0	-0.312500D-01	0.0	0.0	NOT IN R
0.0	0.100000D+00	0.0	0.0	-0.121770D+00	0.0	0.0	0.0	
0.100000D+00	0.100000D+00	-0.866610D-02	0.0	-0.582910D-01	-0.501020D-01	0.0	0.0	
0.200000D+00	0.100000D+00	-0.120900D-01	0.0	-0.170960D-01	-0.801550D-01	0.0	0.0	
0.300000D+00	0.100000D+00	-0.120400D-01	0.0	0.149670D-01	-0.819750D-01	0.0	0.0	
0.400000D+00	0.100000D+00	-0.863940D-02	0.0	0.500780D-01	-0.540470D-01	0.0	0.0	
0.500000D+00	0.100000D+00	0.144740D-07	0.0	0.121420D+00	0.107490D-06	0.0	0.0	NOT IN R
0.0	0.200000D+00	0.0	0.0	-0.159530D+00	0.0	0.0	0.0	
0.100000D+00	0.200000D+00	-0.120810D-01	0.0	-0.902600D-01	-0.171640D-01	0.0	0.0	
0.200000D+00	0.200000D+00	-0.171640D-01	0.0	-0.249870D-01	-0.246300D-01	0.0	0.0	
0.300000D+00	0.200000D+00	-0.171790D-01	0.0	0.242960D-01	-0.246130D-01	0.0	0.0	
0.400000D+00	0.200000D+00	-0.120750D-01	0.0	0.918150D-01	-0.147360D-01	0.0	0.0	
0.500000D+00	0.200000D+00	0.190940D-07	0.0	0.160170D+00	0.257720D-07	0.0	0.0	NOT IN R
0.0	0.300000D+00	0.0	0.0	-0.159820D+00	0.0	0.0	0.0	
0.100000D+00	0.300000D+00	-0.121050D-01	0.0	-0.865730D-01	0.172550D-01	0.0	0.0	
0.200000D+00	0.300000D+00	-0.171290D-01	0.0	-0.227630D-01	0.237860D-01	0.0	0.0	
0.300000D+00	0.300000D+00	-0.171370D-01	0.0	0.230470D-01	0.235290D-01	0.0	0.0	
0.400000D+00	0.300000D+00	-0.120900D-01	0.0	0.921250D-01	0.145920D-01	0.0	0.0	
0.500000D+00	0.300000D+00	0.190960D-07	0.0	0.160190D+00	-0.257210D-07	0.0	0.0	NOT IN R
0.0	0.400000D+00	0.0	0.0	-0.121710D+00	0.0	0.0	0.0	
0.100000D+00	0.400000D+00	-0.865350D-02	0.0	-0.581430D-01	0.497910D-01	0.0	0.0	
0.200000D+00	0.400000D+00	-0.121030D-01	0.0	-0.172160D-01	0.805290D-01	0.0	0.0	
0.300000D+00	0.400000D+00	-0.121020D-01	0.0	0.173050D-01	0.805240D-01	0.0	0.0	
0.400000D+00	0.400000D+00	-0.864070D-02	0.0	0.581730D-01	0.496140D-01	0.0	0.0	
0.500000D+00	0.400000D+00	0.144790D-07	0.0	0.121460D+00	-0.107540D-06	0.0	0.0	NOT IN R
0.0	0.500000D+00	0.0	0.0	-0.312500D-01	0.0	0.0	0.0	NOT IN R
0.100000D+00	0.500000D+00	0.145080D-07	0.0	0.107830D-06	0.121700D+00	0.0	0.0	NOT IN R
0.200000D+00	0.500000D+00	0.190490D-07	0.0	0.359620D-07	0.159800D+00	0.0	0.0	NOT IN R
0.300000D+00	0.500000D+00	0.190490D-07	0.0	-0.358450D-07	0.159800D+00	0.0	0.0	NOT IN R
0.400000D+00	0.500000D+00	0.144980D-07	0.0	-0.107730D-06	0.121620D+00	0.0	0.0	NOT IN R
0.500000D+00	0.500000D+00	0.372530D-09	0.0	0.312500D-01	-0.107540D-06	0.0	0.0	NOT IN R
								****

TABLE 6.2(b): Solution of Example 6.1

```
NEQ=     1 NT=     5 NV=     6 NTF=     50 NDIM=        1
OXX      =    UX
OXY      =    UY
OXX-UX   =    1
OXY-UY   =    1
NX       =   10
NY       =   10
XA       =    0.
HX       =    .4
YA       =    0.
HY       =    .2
PLOT     =    1
CUBICS   =    1

ARC=-1
FB1      =    0

ARC=-2
X        =    4-DCOS(1.5707900*S)
Y        =    DSIN(1.5707900*S)
FB1      =    0

ARC=3
GB1      =    0

ARC=-4
FB1      =    2

ARC=-5
FB1      =    Y
VXY      =    0,0   3.,0    4,1  4,2  0,2  2,1
IABC     =    1 2 6   2 3 6  3 4 6  4 5 6   5 1 6
I        =      -1      -2     3      -4     -5
END.
BOUNDARY DATA FOR TWOPLOT
 0.0        0.3000+00 0.6000+00 0.9000+00 0.1200+01 0.1500+01 0.1800+01 0.2100+01 0.2400+01 0.2700+01 0.3000+01
 0.0        0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0
 0.3000+01 0.3010+01 0.3050+01 0.3110+01 0.3190+01 0.3290+01 0.3410+01 0.3550+01 0.3690+01 0.3840+01 0.4000+01
 0.0        0.1560+00 0.3090+00 0.4540+00 0.5880+00 0.7070+00 0.8090+00 0.8910+00 0.9510+00 0.9880+00 0.1000+01
 0.4000+01 0.4000+01 0.4000+01 0.4000+01 0.4000+01 0.4000+01 0.4000+01 0.4000+01 0.4000+01 0.4000+01 0.4000+01
 0.1000+01 0.1100+01 0.1200+01 0.1300+01 0.1400+01 0.1500+01 0.1600+01 0.1700+01 0.1800+01 0.1900+01 0.2000+01
 0.4000+01 0.3600+01 0.3200+01 0.2800+01 0.2400+01 0.2000+01 0.1600+01 0.1200+01 0.8000+00 0.4000+00 0.0
 0.2000+01 0.2000+01 0.2000+01 0.2000+01 0.2000+01 0.2000+01 0.2000+01 0.2000+01 0.2000+01 0.2000+01 0.2000+01
 0.0        0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0
 0.2000+01 0.1800+01 0.1600+01 0.1400+01 0.1200+01 0.1000+01 0.8000+00 0.6000+00 0.4000+00 0.2000+00 0.0
```

TABLE 6.3(a)

49 TRIANGLES, NUM COULD BE DECREASED TO 19772, TIME= 0.100000D+01

X	Y	J	V	DXX	DXY	DYX	JYY	
0.0	0.0	0.310599-13	0.0	-0.522468D-04	0.100000D+01	0.0	0.0	
0.400000D+00	0.0	0.0	0.0	0.0	0.992240+00	0.0	0.0	
0.800000D+00	0.0	0.0	0.0	0.0	0.991500+00	0.0	0.0	
0.120000D+01	0.0	0.0	0.0	0.0	0.991550+00	0.0	0.0	
0.160000D+01	0.0	0.0	0.0	0.0	0.992760+00	0.0	0.0	
0.200000D+01	0.0	0.0	0.0	0.0	0.851990+00	0.0	0.0	
0.240000D+01	0.0	0.0	0.0	0.0	0.713200+00	0.0	0.0	
0.280000D+01	0.0	0.0	0.0	0.0	0.334370+00	0.0	0.0	
0.320000D+01	0.0	0.0	0.0	0.0	-0.344980+00	0.0	0.0	NOT IN R
0.360000D+01	0.0	0.0	0.0	0.0	-0.132450+01	0.0	0.0	NOT IN R
0.400000D+01	0.0	-0.372350+01	0.0	-0.642980+00	0.525530+01	0.0	0.0	NOT IN R
0.0	0.200000+00	0.200000+00	0.0	-0.377930-02	0.100000+01	0.0	0.0	
0.400000D+00	0.200000+00	0.198470+00	0.0	-0.429040-02	0.992440+00	0.0	0.0	
0.800000D+00	0.200000+00	0.196310+00	0.0	-0.694980-02	0.992140+00	0.0	0.0	
0.120000D+01	0.200000+00	0.192550+00	0.0	-0.126110-01	0.964310+00	0.0	0.0	
0.160000D+01	0.200000+00	0.186520+00	0.0	-0.241080-01	0.930410+00	0.0	0.0	
0.200000D+01	0.200000+00	0.171570+00	0.0	-0.483420-01	0.865160+00	0.0	0.0	
0.240000D+01	0.200000+00	0.143370+00	0.0	-0.942030-01	0.726630+00	0.0	0.0	
0.280000D+01	0.200000+00	0.756460-01	0.0	-0.244410+00	0.428240+00	0.0	0.0	
0.320000D+01	0.200000+00	0.320530-02	0.0	0.445990-01	-0.358510+00	0.0	0.0	NOT IN R
0.360000D+01	0.200000+00	-0.108870+01	0.0	-0.155130+01	0.154600+01	0.0	0.0	NOT IN R
0.400000D+01	0.200000+00	-0.274450+01	0.0	-0.493760+00	0.454880+01	0.0	0.0	NOT IN R
0.0	0.400000+00	0.400000+00	0.0	-0.679980-02	0.100000+01	0.0	0.0	
0.400000D+00	0.400000+00	0.397100+00	0.0	-0.312320-02	0.993350+00	0.0	0.0	
0.800000D+00	0.400000+00	0.392990+00	0.0	-0.131260-01	0.985050+00	0.0	0.0	
0.120000D+01	0.400000+00	0.385920+00	0.0	-0.235960-01	0.969910+00	0.0	0.0	
0.160000D+01	0.400000+00	0.372710+00	0.0	-0.449760-01	0.942370+00	0.0	0.0	
0.200000D+01	0.400000+00	0.347000+00	0.0	-0.889270-01	0.890880+00	0.0	0.0	
0.240000D+01	0.400000+00	0.294650+00	0.0	-0.187980+00	0.792470+00	0.0	0.0	
0.280000D+01	0.400000+00	0.178590+00	0.0	-0.430750+00	0.606110+00	0.0	0.0	
0.320000D+01	0.400000+00	-0.161920+00	0.0	-0.166190+01	0.858500+00	0.0	0.0	NOT IN R
0.360000D+01	0.400000+00	-0.808710+00	0.0	-0.153550+01	0.130620+01	0.0	0.0	NOT IN R
0.400000D+01	0.400000+00	-0.189370+01	0.0	-0.363680+00	0.392270+01	0.0	0.0	NOT IN R
0.0	0.600000+00	0.600000+00	0.0	-0.911390-02	0.100000+01	0.0	0.0	
0.400000D+00	0.600000+00	0.596020+00	0.0	-0.112620-01	0.995460+00	0.0	0.0	
0.800000D+00	0.600000+00	0.590380+00	0.0	-0.178860-01	0.988010+00	0.0	0.0	
0.120000D+01	0.600000+00	0.580710+00	0.0	-0.319280-01	0.978070+00	0.0	0.0	
0.160000D+01	0.600000+00	0.562900+00	0.0	-0.607060-01	0.959330+00	0.0	0.0	
0.200000D+01	0.600000+00	0.528450+00	0.0	-0.118100+00	0.925040+00	0.0	0.0	
0.240000D+01	0.600000+00	0.460250+00	0.0	-0.235970+00	0.866130+00	0.0	0.0	
0.280000D+01	0.600000+00	0.318200+00	0.0	-0.511390+00	0.790260+00	0.0	0.0	
0.320000D+01	0.600000+00	0.566390-03	0.0	-0.120160+01	0.908410+00	0.0	0.0	
0.360000D+01	0.600000+00	-0.551770+00	0.0	-0.152210+01	0.138070+01	0.0	0.0	NOT IN R
0.400000D+01	0.600000+00	-0.117000+01	0.0	-0.252730+00	0.337700+01	0.0	0.0	NOT IN R
0.0	0.800000+00	0.800000+00	0.0	-0.107210-01	0.100000+01	0.0	0.0	
0.400000D+00	0.800000+00	0.795330+00	0.0	-0.131640-01	0.997710+00	0.0	0.0	
0.800000D+00	0.800000+00	0.788720+00	0.0	-0.263530-01	0.994460+00	0.0	0.0	
0.120000D+01	0.800000+00	0.777470+00	0.0	-0.569310-01	0.989500+00	0.0	0.0	
0.160000D+01	0.800000+00	0.756510+00	0.0	-0.499700-01	0.980760+00	0.0	0.0	
0.200000D+01	0.800000+00	0.717560+00	0.0	-0.134560+00	0.957400+00	0.0	0.0	
0.240000D+01	0.800000+00	0.643180+00	0.0	-0.252720+00	0.904800+00	0.0	0.0	
0.280000D+01	0.800000+00	0.474810+00	0.0	-0.496720+00	0.981330+00	0.0	0.0	
0.320000D+01	0.800000+00	0.213290+00	0.0	-0.979620+00	0.122290+01	0.0	0.0	
0.360000D+01	0.800000+00	-0.243290+00	0.0	-0.143650+01	0.179500+01	0.0	0.0	NOT IN R
0.400000D+01	0.800000+00	-0.542500+00	0.0	-0.160320+00	0.291170+01	0.0	0.0	NOT IN R
0.0	0.100000+01	0.100000+01	0.0	-0.116220-01	0.100000+01	0.0	0.0	
0.400000D+00	0.100000+01	0.995100+00	0.0	-0.137980-01	0.100010+01	0.0	0.0	
0.800000D+00	0.100000+01	0.983210+00	0.0	-0.217590-01	0.100030+01	0.0	0.0	
0.120000D+01	0.100000+01	0.976500+00	0.0	-0.396660-01	0.100060+01	0.0	0.0	
0.160000D+01	0.100000+01	0.955200+00	0.0	-0.712320-01	0.100300+01	0.0	0.0	
0.200000D+01	0.100000+01	0.915960+00	0.0	-0.130480+00	0.101170+01	0.0	0.0	
0.240000D+01	0.100000+01	0.842300+00	0.0	-0.245040+00	0.103670+01	0.0	0.0	
0.280000D+01	0.100000+01	0.708610+00	0.0	-0.437530+00	0.115640+01	0.0	0.0	
0.320000D+01	0.100000+01	0.484770+00	0.0	-0.731070+00	0.151790+01	0.0	0.0	
0.360000D+01	0.100000+01	0.171620+00	0.0	-0.720320+00	0.179410+01	0.0	0.0	
0.400000D+01	0.100000+01	0.221510-09	0.0	-0.234360-01	0.555790-01	0.0	0.0	NOT IN R
0.0	0.120000+01	0.120000+01	0.0	-0.108180-01	0.100300+01	0.0	0.0	
0.400000D+00	0.120000+01	0.119540+01	0.0	-0.139800-01	0.100260+01	0.0	0.0	
0.800000D+00	0.120000+01	0.118380+01	0.0	-0.207000-01	0.100550+01	0.0	0.0	
0.120000D+01	0.120000+01	0.117780+01	0.0	-0.363840-01	0.101130+01	0.0	0.0	
0.160000D+01	0.120000+01	0.115300+01	0.0	-0.656650-01	0.102440+01	0.0	0.0	
0.200000D+01	0.120000+01	0.112190+01	0.0	-0.113370+00	0.104200+01	0.0	0.0	
0.240000D+01	0.120000+01	0.105590+01	0.0	-0.211750+00	0.110660+01	0.0	0.0	

continued....

TABLE 6.3(b): Solution of Example 6.2

0.280000D+01	0.120000D+01	0.345120D+00	0.0	-0.359240D+00	0.123140D+01	0.0	0.0
0.320000D+01	0.120000D+01	0.775680D+00	0.0	-0.496230D+00	0.149320D+01	0.0	0.0
0.360000D+01	0.120000D+01	0.573410D+00	0.0	-0.443550D+00	0.135600D+01	0.0	0.0
0.400000D+01	0.120000D+01	0.473560D+00	0.0	-0.346870D-01	0.222230D+01	0.0	0.0
0.0	0.140000D+01	0.140000D+01	0.0	-0.930630D-02	0.100000D+01	0.0	0.0
0.400000D+00	0.140000D+01	0.139610D+01	0.0	-0.111780D-01	0.100440D+01	0.0	0.0
0.800000D+00	0.140000D+01	0.139050D+01	0.0	-0.174130D-01	0.101110D+01	0.0	0.0
0.120000D+01	0.140000D+01	0.133130D+01	0.0	-0.362780D-01	0.102240D+01	0.0	0.0
0.160000D+01	0.140000D+01	0.136470D+01	0.0	-0.559630D-01	0.104250D+01	0.0	0.0
0.200000D+01	0.140000D+01	0.133500D+01	0.0	-0.966720D-01	0.108070D+01	0.0	0.0
0.240000D+01	0.140000D+01	0.129330D+01	0.0	-0.166110D+00	0.115470D+01	0.0	0.0
0.280000D+01	0.140000D+01	0.119590D+01	0.0	-0.257630D+00	0.129180D+01	0.0	0.0
0.320000D+01	0.140000D+01	0.103780D+01	0.0	-0.331770D+00	0.153020D+01	0.0	0.0
0.360000D+01	0.140000D+01	0.951710D+00	0.0	-0.270160D+00	0.193760D+01	0.0	0.0
0.400000D+01	0.140000D+01	0.894250D+00	0.0	-0.273570D-03	0.199910D+01	0.0	0.0
0.0	0.160000D+01	0.160000D+01	0.0	-0.493390D-02	0.100000D+01	0.0	0.0
0.400000D+00	0.160000D+01	0.159710D+01	0.0	-0.902330D-02	0.100620D+01	0.0	0.0
0.800000D+00	0.160000D+01	0.159310D+01	0.0	-0.125810D-01	0.101490D+01	0.0	0.0
0.120000D+01	0.160000D+01	0.158650D+01	0.0	-0.217590D-01	0.102930D+01	0.0	0.0
0.160000D+01	0.160000D+01	0.157470D+01	0.0	-0.359500D-01	0.105660D+01	0.0	0.0
0.200000D+01	0.160000D+01	0.155360D+01	0.0	-0.678180D-01	0.110370D+01	0.0	0.0
0.240000D+01	0.160000D+01	0.151770D+01	0.0	-0.113270D+00	0.118760D+01	0.0	0.0
0.280000D+01	0.160000D+01	0.146150D+01	0.0	-0.167920D+00	0.133130D+01	0.0	0.0
0.320000D+01	0.160000D+01	0.138540D+01	0.0	-0.202030D+00	0.153380D+01	0.0	0.0
0.360000D+01	0.160000D+01	0.131020D+01	0.0	-0.162840D+00	0.175180D+01	0.0	0.0
0.400000D+01	0.160000D+01	0.127320D+01	0.0	0.150060D-01	0.195440D+01	0.0	0.0
0.0	0.180000D+01	0.180000D+01	0.0	-0.369200D-02	0.100000D+01	0.0	0.0
0.400000D+00	0.180000D+01	0.179350D+01	0.0	-0.423380D-02	0.100740D+01	0.0	0.0
0.800000D+00	0.180000D+01	0.179640D+01	0.0	-0.454780D-02	0.101750D+01	0.0	0.0
0.120000D+01	0.180000D+01	0.179230D+01	0.0	-0.113000D-01	0.103430D+01	0.0	0.0
0.160000D+01	0.180000D+01	0.178670D+01	0.0	-0.201230D-01	0.106410D+01	0.0	0.0
0.200000D+01	0.180000D+01	0.177590D+01	0.0	-0.353550D-01	0.111760D+01	0.0	0.0
0.240000D+01	0.180000D+01	0.175760D+01	0.0	-0.571570D-01	0.120900D+01	0.0	0.0
0.280000D+01	0.180000D+01	0.172950D+01	0.0	-0.828040D-01	0.134940D+01	0.0	0.0
0.320000D+01	0.180000D+01	0.169270D+01	0.0	-0.984420D-01	0.153740D+01	0.0	0.0
0.360000D+01	0.180000D+01	0.165750D+01	0.0	-0.736420D-01	0.173050D+01	0.0	0.0
0.400000D+01	0.180000D+01	0.164140D+01	0.0	0.111520D-01	0.179110D+01	0.0	0.0
0.0	0.200000D+01	0.200000D+01	0.0	0.410150D-03	0.100000D+01	0.0	0.0
0.400000D+00	0.200000D+01	0.200000D+01	0.0	-0.273160D-08	0.100750D+01	0.0	0.0
0.800000D+00	0.200000D+01	0.200000D+01	0.0	-0.386270D-08	0.101850D+01	0.0	0.0
0.120000D+01	0.200000D+01	0.200000D+01	0.0	-0.564670D-08	0.103540D+01	0.0	0.0
0.160000D+01	0.200000D+01	0.200000D+01	0.0	-0.127380D-07	0.106730D+01	0.0	0.0
0.200000D+01	0.200000D+01	0.200000D+01	0.0	-0.631790D-03	0.112220D+01	0.0	0.0
0.240000D+01	0.200000D+01	0.200000D+01	0.0	-0.344280D-07	0.121360D+01	0.0	0.0
0.280000D+01	0.200000D+01	0.200000D+01	0.0	-0.493610D-07	0.135420D+01	0.0	0.0
0.320000D+01	0.200000D+01	0.200000D+01	0.0	-0.598350D-07	0.154340D+01	0.0	0.0
0.360000D+01	0.200000D+01	0.200000D+01	0.0	-0.359370D-07	0.170410D+01	0.0	0.0
0.400000D+01	0.200000D+01	0.200000D+01	0.0	-0.118350D-01	0.190820D+01	0.0	0.0

....

TABLE 6.3(b): continued

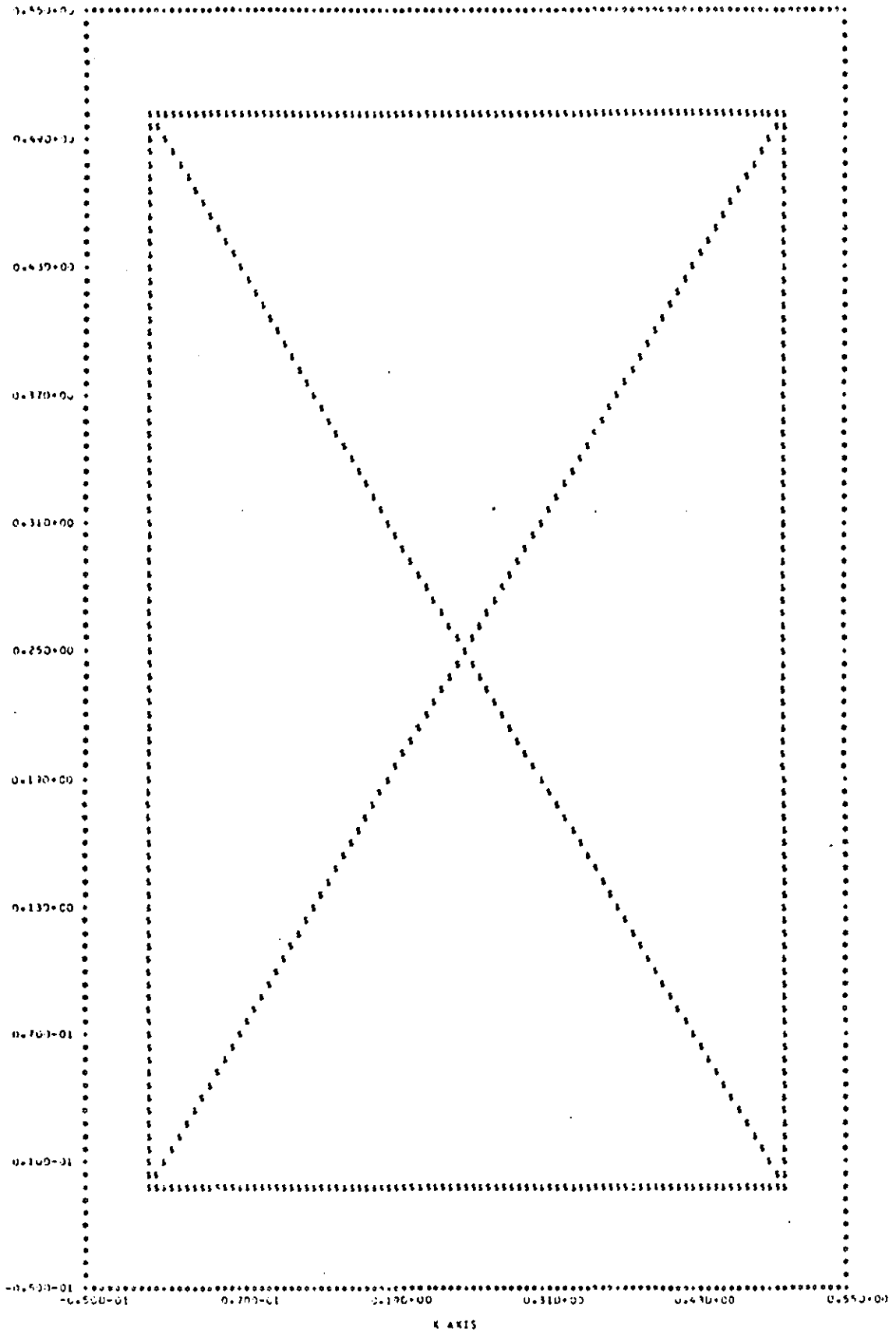


FIGURE 6.3: Initial triangulation of Example 6.1

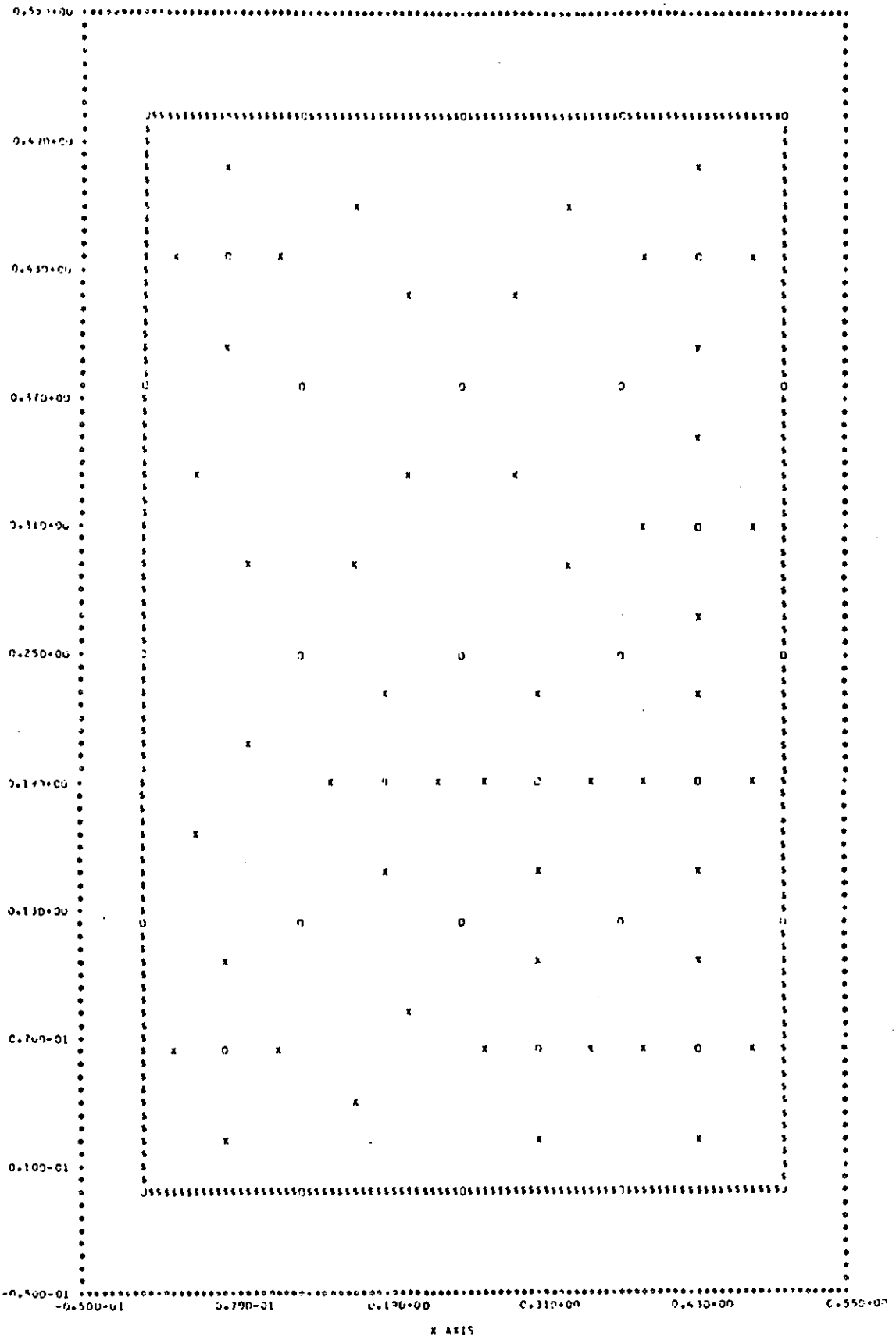


FIGURE 6.4: Vertices (O) and Centers (x) in final triangulation of Example 6.1

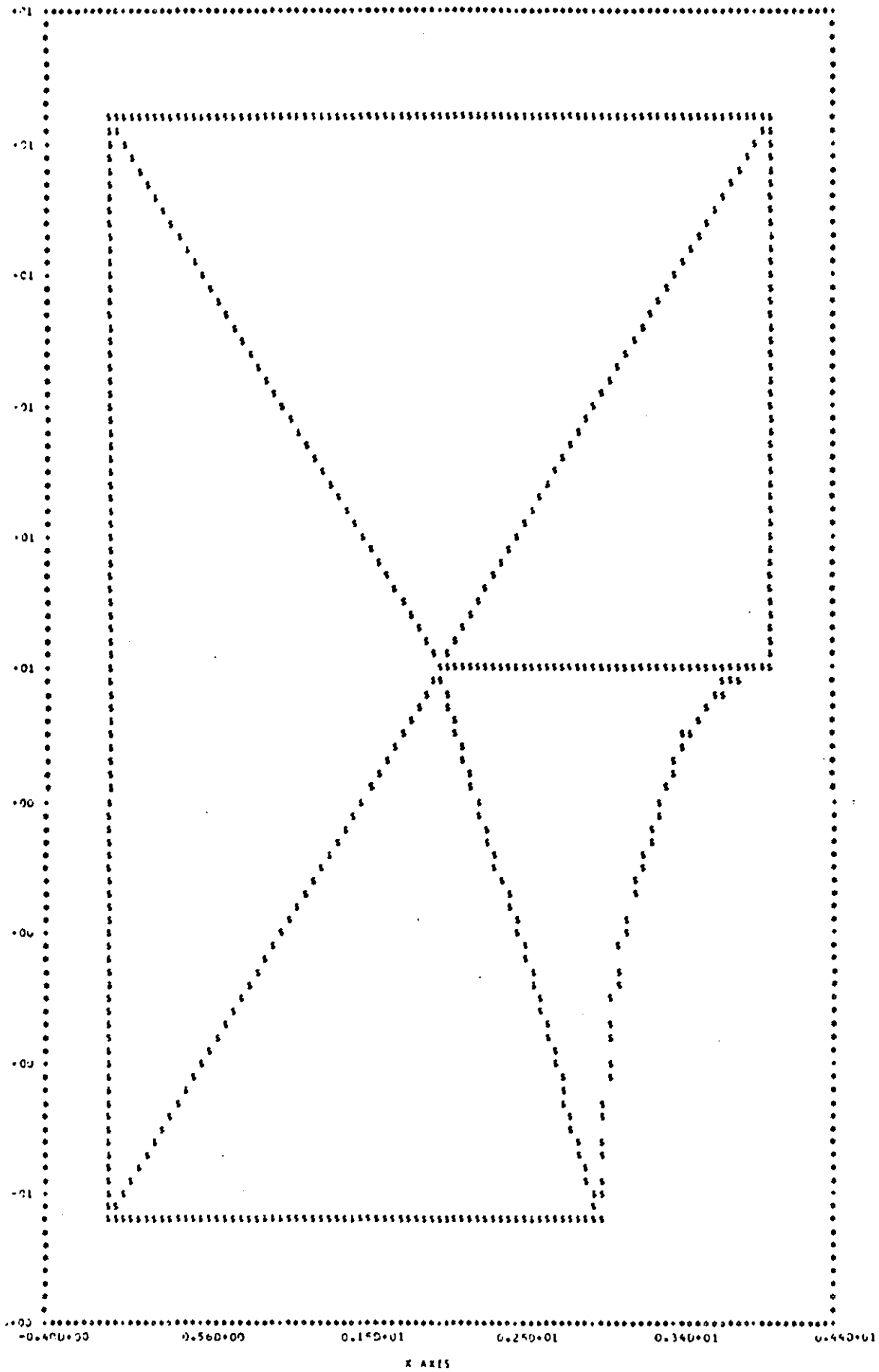


FIGURE 6.7: Initial triangulation of Example 6.2

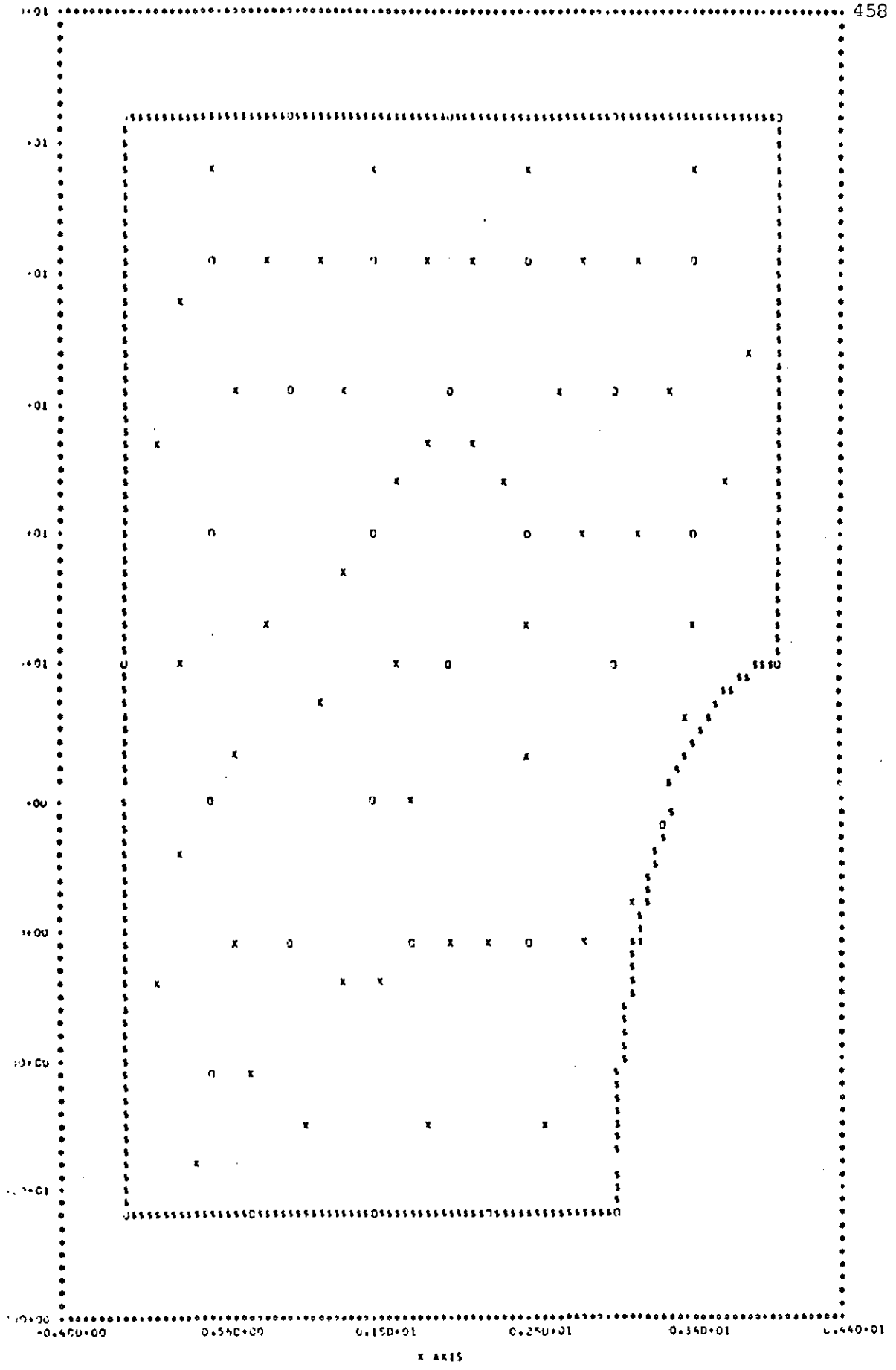


FIGURE 6.8: Vertices (O) and Centers (x) in final triangulation of Example 6.2

6.7 SPECIAL TECHNIQUES

6.7.1 Utilization of Symmetry

If the following two matrices are symmetric at every point, then, the storage required for the Jacobian will be halved since elements above the diagonal need not be stored. To tell the package that symmetry exists, a statement: SYMMETRY is coded. The matrices are:

-F1/U	-F1/UX	-F1/UY	-F1/V	-F1/VX	-F1/VY
OXX/U	OXX/UX	OXX/UY	OXX/V	OXX/VX	OXX/VY
OXY/U	OXY/UX	OXY/UY	OXY/V	OXY/VX	OXY/VY
-F2/U	-F2/UX	-F2/UY	-F2/V	-F2/VX	-F2/VY
OYX/U	OYX/UX	OYX/UY	OYX/V	OYX/VX	OYX/VY
OYY/U	OYY/UX	OYY/UY	OYY/V	OYY/VX	OYY/VY

and GB1/U GB1/V

GB2/U GB2/V ,

where A/B, as usual, means $\frac{\partial A}{\partial B}$.

6.7.2 Mixed Type Boundary Conditions

This package does not, in general, allow the specification of mixed types of boundary conditions on the same arc. However, if these boundary conditions can be written in the form:

$$u = FB1(s,t) \tag{6.27}$$

and

$$v = FB2(s,t) \tag{6.28}$$

then, it is possible to convert them to Neumann type boundary condition approximately by:

$$OXX*n_x + OXY*n_y = -BETA*(u-FB1(s,t)) \tag{6.29}$$

and

$$OYX*n_x + OYY*n_y = -BETA*(v-FB2(s,t)) \tag{6.30}$$

where BETA is a very large number (computer infinity). This will

force the terms u-FB1(s,t) and v-FB2(s,t) to be zeros which satisfies the boundary conditions (6.27) and (6.28) approximately.

6.7.3 Solving a Single Equation

It is possible to solve a single equation, simply, by ignoring the second equation in specifying the input data set as demonstrated in previous examples. However, if the problem considered is a one-dimensional problem, then, the execution time of the package can be considerably decreased if it is handled as follows:

(1) Let the region be $x_1 \leq x \leq x_2$. We assume that R is in two dimensions with $0 \leq y \leq 1$ and the normal derivatives of all unknowns are 0 at y=0 and at y=1.

(2) The grid option is used to define the initial and final triangulation with only two grid lines in the y direction i.e. ygrid(1)=0 and ygrid(2)=1. Grids in the x-direction are done as usual. Adopting this technique, the execution time is considerably decreased compared to that required if grid lines in both directions were done.

6.7.4 Solving Several Simultaneous Equations

The package can solve up to 5 equations per set. In this case, the variables and their derivatives are called U1,U2,...,U5,V1,V2,...,V5, U1X,..., etc. If we have more equations and we can divide them into sets with maximum number of sets of 5 and maximum number of equations per set is 5 and such that the coupling between sets is low, i.e. such that the unknowns in each set are not highly dependent on the unknowns outside that set, then, the TWODEPEP package can handle these sets.

To define a set the keyword 'SET=n' is coded at the top of each set where n is a sequence number i.e. 1,2,...,5. Then, inputs of each set are coded immediately after the 'SET=' statement. In the first line of the whole input data, the NEQ parameter will be coded as an integer each digit of which gives the number of equations in each set starting from the most significant digit as for set number 1 and so on, e.g. 234 means 3 sets with 2 equations in the first, 3 in the second and 4 in the third. The package handles each set independently of the others and substitutes the most updated values for computation iteratively.

6.7.5 Non-Uniform Distribution of Elements

The package distributes the number of triangles evenly over the region. In some applications it is desired, to get correct solutions, to have element condensation or a finer mesh in particular parts of R. Examples can be a crack tip problem or flow around an obstacle. It is also desired to have the elements properly graded, i.e. gradually graded. This can be done by specifying in the input data a function with the keyword D3EST as a function of (x,y). The package will use this function to distribute D3EST(x,y)**(2/3)*A(j) evenly over the final triangulation where A(j) is the area of triangle j. Practically, we choose D3EST to be largest where we want the triangulation most dense.

6.7.6 Updating the Master Matrix

If the user is in some doubt whether the master matrix should be updated or not, he can make a simple run with only a few elements and constant time step and default NUPDT to 1, i.e. updating the master matrix each time step. If the message "NOTE - Jacobian updated

unnecessarily" is printed, then, he can run the actual run with the true number of elements with NUPDT=0. This will result in a big saving in processing time.

6.7.7 Accessing the Solution Stored by an Earlier Run

If the problem to be solved requires the solution of a previous problem which has been obtained by the package in an earlier run and stored by the package using the MWR parameter, then, it is possible to access and obtain that solution by the calling of the TDREAD subroutine as follows:

```
CALL TDREAD(Ifile,Iskip,ICl,WSTO,X,Y,W,WX,WY)
```

where:

- Ifile is the logical unit number of the file containing the solution.
- Iskip is the number of output units that must be skipped from the start of the file to reach the desired stored solution. One output unit = (NX+1)*(NY+1) solution lines.
- ICl if ICl=1 then the values of U, UX and UY are to be returned. if ICl=2 then V,VX and VY are returned.
- WSTO is a real 2D array of at least {(NX+1),(NY+1)} elements. WSTO(1,1) must be initialized to -75.75 by a data statement.

TDREAD will store the returned values of U or V in WSTO. WSTO should not be altered by the user between successive calls of the TDREAD subroutine.

- X and Y are the coordinates of the interpolation point at which the solution is required. The output of that routine is W,WX and WY where W will hold U(x,y) or V(x,y) [according to ICl=1 or =2, respectively].

WX is $\frac{\partial}{\partial x}U(x,y)$ or $\frac{\partial}{\partial x}V(x,y)$

WY is $\frac{\partial}{\partial y}U(x,y)$ or $\frac{\partial}{\partial y}V(x,y)$.

TDREAD performs a quadratic interpolation scheme to compute W, WX and WY.

6.8 COMPUTER IMPLEMENTATION

6.8.1 Implementation on a Mainframe Computer

Since this package, as most of the similar ones, is a CPU bound job, it is wise to be run in the background as a batch job for medium size and large problems. It has been modified, as will be described later, to run on the mainframe computer IBM 3033 running under the operating system MVS. Moreover, it has been modified to run on the small size mini-computer HP3000 for small size jobs. The main outlines for implementing this package on the IBM machine is as follows:

(i) The pre-processor program is compiled and linked to produce a load module, i.e. an executable program. Since two versions: single precision and double precision are available, two programs are produced. They have been given the names: PRESNGL and PREDBLE.

(ii) The problem independent subroutines are compiled only and saved in another separate file.

(iii) The input data set for the problem to be solved is run as an input to the pre-processor program. In other words, either the program PRESNGL or PREDBLE is run with input the PDE description and data in TWODEPEP notation.

(iv) The output of the previous run (logical unit number 4) is the problem dependent Fortran subroutines plus the main program driver.

(v) This file is compiled and linked with the file containing the problem-independent routines into an executable program which will, in fact, solve the problem and print the required results and store them, if so desired, on a file for postprocessing or a subsequent run of the package.

(vi) During the execution of the program some work files may be needed with logical unit numbers 12, 13, 14 and 15.

(vii) Since the package is used within this Ph.D. programme as a research tool, it is desired to be able to fetch the generated Fortran program; this is done by reading the problem-independent routines and appending them to the source program file generated by the pre-processor.

(viii) A procedure has been designed and implemented to do the whole job by one statement only. The details of which are as follows:

```
//TDPSNGL   PROC

//* PROC-NAME:      TDPSNGL                 MAINT.-RESP.: A. SHARAF ELDIN
//* ==========                              LAST UPDATE : 09/05/1983
//*-------------------------------------------------------------------
//* DESCRIPT.:      THE TWODEPEP PACKAGE IN SINGLE PRECISION
//* ==========
//*                 SPECIFY YOUR INPUT DATA ON PRESNGL.FT05F001
//*                 THIS IS THE SINGLE PRECISION VERSION


//*===================================================================
//PRESNGL   EXEC  PGM=PRESNGL
//STEPLIB   DD    DSN=RUC.TEST.LOADLIB,DISP=SHR
//FT05F001  DD    DUMMY
//FT06F001 DD SYSOUT=*
//FT04F001 DD UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(TRK,(25,5),RLSE),
//         DSN=&&SV,DCB=(LRECL=80,BLKSIZE=80,RECFM=F)
//FT09F001 DD DSN=F45L005.TWODEP.DATA(TDPASNGL),DISP=SHR
//FORT   EXEC PGM=IGIFORT,PARM=(NOSOURCE)
//SYSPRINT DD DUMMY
//SYSLIN   DD DSN=&OBJ,SPACE=(3040,(40,40),,,ROUND),UNIT=VIO,
//            DISP=(MOD,PASS),
//            DCB=(BLKSIZE=3040,LRECL=80,RECFM=FBS,BUFNO=1)
//SYSIN    DD DSN=&&SV,DISP=(OLD,DELETE)
//*===================================================================
//LKED   EXEC PGM=IEWL,COND=(5,LT,FORT),PARM='LIST,LET'
//SYSPRINT DD DUMMY
//SYSLMOD  DD SPACE=(CYL,(1,1,1)),DSN=&LOD(X),DISP=(,PASS),
//            UNIT=VIO,DCB=BUFNO=1
//SYSUT1   DD DSN=&SYSUT1,SPACE=(1024,(120,120),,,ROUND),UNIT=VIO,
//            DCB=BUFNO=1
//SYSLIN   DD DSN=&OBJ,DISP=(OLD,DELETE)
//         DD DDNAME=SYSIN
//SYSLIB   DD DSN=SYS1.FORTLIB,DISP=SHR
//*===================================================================
//GO      EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,LKED),(5,LT,FORT))
//FT06F001 DD SYSOUT=Y
//FT02F001 DD UNIT=SYSDA,DISP=(NEW,DELETE),DSN=&&FILE2,
//       SPACE=(TRK,(100,10),RLSE)
//FT08F001 DD UNIT=SYSDA,DISP=(NEW,DELETE),DSN=&&FILE8,
//      SPACE=(TRK,(5,5),RLSE),DCB=(LRECL=120,BLKSIZE=120,RECFM=F)
//*  NOTE THAT FILES 8 AND 9 ARE MODIFIED HERE FOR THE 'DISP' PARM
//*  TO BE  (NEW,DELETE)  INSTEAD OF (NEW,CATLG)
//FT09F001 DD UNIT=SYSDA,DISP=(NEW,DELETE),DSN=&&FILE9,
//      SPACE=(TRK,(5,5),RLSE),DCB=(LRECL=120,BLKSIZE=120,RECFM=F)
//FT12F001 DD UNIT=SYSDA,DISP=(NEW,DELETE),DSN=&&FILE12,
//       SPACE=(TRK,(100,10),RLSE)
//FT13F001 DD UNIT=SYSDA,DISP=(NEW,DELETE),DSN=&&FILE13,
//       SPACE=(TRK,(100,10),RLSE)
//FT14F001 DD UNIT=SYSDA,DISP=(NEW,DELETE),DSN=&&FILE14,
//       SPACE=(TRK,(100,10),RLSE)
//FT15F001 DD UNIT=SYSDA,DISP=(NEW,DELETE),DSN=&&FILE15,
//       SPACE=(TRK,(100,10),RLSE)
//
        PEND
```

6.8.2 Implementation on a Mini-Computer

It is possible to implement the package on a mini-computer. In fact, the package can run on relatively big mini's, or super mini's like the VAX 11/785 or the Prime computer family. On the relatively smaller mini-computers like the HP3000, the package was modified to run by defining all integer variables to be integer*4 and all other real variables are defined as double precision. The most severe problem on this machine is the limited address space for any executable process which is less than 32K words as explained in detail in Chapter 5. Although it is possible to use the out-of-core option (NDIM=2), the processing time is increased rapidly.

6.9 ENHANCEMENTS TO THE PACKAGE

Many enhancements have been done to the package since its first release. Some of them are implemented by the author, while others are by the original developers.

6.9.1 Performance Optimization

The original package was written in ANSI66 Fortran. When implementing it on the IBM mainframe computer we noticed that there are many Fortran compilers available. These are: the H, G1,WATFIV and VS Fortran compilers. The first two; the H and G1 compilers are based on the old Fortran i.e. ANSI66 while the last two are based on the ANSI78 (popularly known as Fortran 77). The WATFIV compiler is developed by the University of Waterloo in Canada and is mainly used for educational and teaching purposes. It is very useful during the testing and debugging stages but at execution and actual 'production' stage it is too slow. The VS-Fortran compiler is the IBM implementation of the 1978 ANSI Fortran. Its first release was in the year 1981. When implementing the package on the IBM machine, it was first run using the G1 compiler. Afterwards, it was locally modified to run under the relatively new VS-Fortran compiler. In this compiler an optimization feature exists [IBM, 1981]. It has 3 levels: 1,2 and 3. Although the compilation time is increased when using the optimization feature, it is usually profitable to use optimization since the savings in execution time are usually more than the increase in compilation time. The first level of compiler optimization is optimize (1). This is the lowest level of optimization and involves register and branch optimization. Optimize (2) specifies, in addition to that, partial code-movement

optimization, but it will not relocate any code when it has been determined that relocating the code under consideration would cause unplanned or unexpected interrupts. Optimize (3) specifies full code-movement optimization in addition to the optimize (1) functions. The main goal of optimization performed by the compiler is to produce a code that will be executed faster. Reducing the size of the object code is usually a secondary goal in this context. In principle, there are many rules that must be followed by the programmer in order to be able to make use of the compiler optimization. As a simple example consider the two codes A and B below:

Code A	Code B
COMMON/X/A	COMMON/W/A,B,C
COMMON/Y/B	C=A+B
COMMON/Z/C	
C=A+B	

In code A, 3 registers are required while in code B only one register is required. This is due to the fact that each reference to a variable in a COMMON block requires that the starting address of that common block to be placed in a register.

Since the source code of the package was not written in accordance with many of these rules and since it will be a very time-consuming task to re-write the whole source again, we used the optimization feature in the code 'as it is'. Some of the most important tasks done by the compiler to optimize the code can be stated as follows:

(i) Removal of unnecessary operations from the program. Here are some examples to illustrate this:

Example (1)

Normal Code	Optimized Code
Parameter (N=100)	Parameter (N=100)
K=N/2	K=50

In this example, the compiler will translate the statement K=N/2 as an assignment statement K=50 which is much faster than the first one at execution time.

Example (2)

Normal Code	Optimized Code
X=Y*Z+R	Temp=Y*Z
.....	X=Temp+R
.....	.....
A=X+B+C+Y*Z	A=X+B+C+Temp
B=Y*Z+Q	B=Temp+Q

Here, the compiler assigns the valye Y*Z to a temporary location Temp thus this multiplication is evaluated only once instead of 3 times.

Example (3)

	Normal Code		Optimized Code
	DO 10 I=1,N		Temp=10.0*Y
			DO 10 I=1,N
10	X=X+10*Y*A(I)	10	X=X+Temp*A(I)

In this case, the compiler moves the calculation of 10*Y outside the loop. Moreover, it does the code conversion from 10 (integer) to 10. (real) at compilation time. This code motion is, however, not always safe as shown in the following example.

Example (4)

	Normal Code		Optimized Code
	DO 10 I=1,N		DO 10 I=1,N
	DO 5 J=1,M		Temp=SQRT(X(I))
	IF(X(I).LT.0) GOTO 10		DO 5 J=1,M
5	Y=SQRT(X(I))		IF (X(I).LT.0) GOTO 10
10	CONTINUE	5	Y=Temp
		10	CONTINUE

In this case, the optimized code will work fine if X(I) is ≥0. However, execution error will result if X(I) is <0 since the SQRT function argument will be negative and the program will abort. To overcome such a situation, optimize (2) is used or the code is manually changed.

(ii) Replacing some operations with more efficient ones. This occurs most frequently in arithmetic expression evaluation where the compiler tries to use registers rather than ordinary memory locations. This results in faster execution time since operations using the hardware registers are much faster than those referencing memory locations. This is usually done by storing intermediate results in registers rather than temporary memory locations. Another situation for register usage is that of indexing arrays and storing the start address of a common block.

(iii) Optimizing execution of logical expressions. The compiler optimizing the execution of logical IF statements in some situations by evaluating only the necessary logical expressions. For example, a statement like: IF(E1.AND.E2)... will be executed by evaluating E1 first, if it was false, E2 will not be evaluated since the final result of the expression is obviously false. Another example is: IF(X(I,J,K).LT.Y(I.J.K) OR.A.EQ.B)... Here, the logical expression A.EQ.B is first evaluated and if it is true, the first logical expression will not be evaluated. This re-ordering is beneficial since a comparison of simple variables is faster than subscripted ones.

The generated Fortran program by the package was compiled using the optimize (2) option and the CPU time for two test problems was compared. Approximately around 10% improvement in speedup factor was

noticed. It is believed, however, that careful rewriting of some portions of the code may result in more savings.

6.9.2 Definition of User Variables

Recall Example 6.1, where the equation (6.25) is to be solved for three different values of the parameter ω. The package in its original design cannot handle this and similar situations efficiently. To solve this problem one has to solve the three cases as if each one is a stand-alone problem. It is obvious however, that considerable part of the processing is independent of the value of ω. For example, the discretization process and the node renumbering are both the same in all the three cases unless the parameter NTF is intentionally changed from one run to the other. To overcome this difficulty, it was proposed by the author to fetch the generated Fortran program and modify it manually at the correct places. In the considered example this can be done easily by specifying a greater outer loop of the form:

```
      DIMENSION OMEGA(3)
      DATA OMEGA/1.0,10.0,100.0/
      ..............
      Normal code up to the point of renumbering of nodes
      ..............
      DO 888 IOMEGA=1,3
      W=OMEGA(IOMEGA)
      ..............
      Rest of code
      ..............
  888 CONTINUE
```

This technique is useful in problems where the same domain and topology is repeatedly used with different parameters. However, later on, IMSL released the new software product PDE/PROTRAN in 1985 which is one further

step above TWODEPEP. It allows the mixing of Fortran statements with TWODEPEP keywords. To illustrate this consider the following example:

```
$  PDE2D
   UNKNOWNS=(U,V)
   NTRIANGLES=250
   FRONTAL
   A=(E1*UX+E2*VY,E3*(UY+VX))
   B=(E3*(UY+VX),E1*VY+E2*UX)
   DEFINE
   ====
   EM=10.6E6
   VNU=.3
   E1=EM/(1.-VNU**2)
   E2=E1*VNU
   E3=EM/(1.+VNU)/2.
   ====
   SYMMETRIC
   SAVEFILE=PLOT
   GB=(1,0,-1./2.08)(6,0,1./2.08)
   VERTICES=(7,1,8,1)(1,2,8,2)(2,3,8,3)
*    (3,9,8,0)(9,7,8,4)(9,3,10,0)(3,4,10,3)(4,5,10,5)
*    (5,6,10,6)(6,9,10,7)
$  PLOTSTRESSES
$  END
```

6.9.3 Supercomputer Implementation

In a very recent paper [Sewell, 1987] the package was released on the Cray-1 supercomputer. In fact, the advent of supercomputers and the highly concurrent systems in general has stimulated the analysts to review, redesign and even invent new algorithms that take advantage of the new capabilities provided by these machines. Since a significant portion of the solution time in the FEM is spent in solving the resulted

systems of linear algebraic equations, vectorization in the equation solver is most critical. A set of experiments have been done using the Fortran compiler vectorization switch alternately on, to activate the vectorization, and off to execute sequentially in scalar mode. In the stated reference a speedup factor of up to 2.49 was obtained. However, it is well known that the actual performance of supercomputers is very different from the peak performance theoretically calculated. In order to get realistic results, benchmark tests must be actually run on the supercomputer. If the package is to be run efficiently on a vector computer, some portions must be rewritten to be more amenable for vectorization.

CHAPTER 7

FINITE ELEMENT SOLUTION TO SOME GROUNDWATER FLOW PROBLEMS

TABLE OF CONTENTS

7.1 INTRODUCTION

Groundwater is an important source of water. In arid zones, it is usually the only available source of water. Studying the groundwater flow is, therefore, very important in order to optimize the exploitation of this vital resource. Before studying the groundwater flow problem it is convenient to give some preliminary definitions as follows [Bouwer, 1978]:

Aquifer: An aquifer is a groundwater-bearing formation which is sufficiently permeable to transmit and yield water in significant quantities. The most common aquifer materials are sands and gravels.

Aquiclude: This is a saturated but essentially impermeable formation that does not yield significant quantities of water. Typical material of which may be clay or solid limestone.

Aquitard: This is a formation which is sufficiently permeable to transmit water vertically to or from an adjacent aquifer, but not permeable enough to laterally transport water like an aquifer. Sandy clay is a typical material for aquitards.

Aquifuge: It is an impermeable formation that does not contain groundwater at all like solid granite. It does not have the ability to contain or transmit water.

7.1.1 Types of Aquifers

Aquifers can be classified into:

(i) Confined aquifers: where the aquifer is bounded from above and below by impervious formations. They are termed pressure aquifers since there is no watertable. If a well penetrates such an aquifer, the water level will be above the base of the upper impervious stratum.

The water levels in a number of such wells will define an imaginary surface called the piezometric surface. If the piezometric surface for a confined aquifer is above the ground surface, the aquifer is called an artesian aquifer.

(ii) Unconfined aquifers: where the watertable is the upper boundary of the aquifer. They are called watertable aquifers or phreatic aquifers.

Aquifers that can gain (or loose) water through the formations above and/or below (i.e. vertically) are called leaky aquifers. Accordingly, a confined aquifer which has at least one semi-pervious confining stratum is called a leaky confined aquifer, and a phreatic aquifer which rests on a semi-pervious layer is called a leaky phreatic aquifer.

Figures 7.1, 7.2 and 7.3 are schematics for different aquifer types.

7.1.2 Functions of Aquifers

The main function of an aquifer is, of course, as a source of water for various purposes. However, many other functions may be achieved by managing aquifers [Bear, 1979]. The main functions can be summarized as follows:

(i) Water supply: This is usually the major function of an aquifer. In general, an aquifer is replenished annually from precipitation. However, it is a fact that some aquifers are not renewable and contain water stored inside from the far past. Usually, under natural conditions, a quasi-equilibrium situation is maintained with inflow equal to outflow.

(ii) Storage reservoir: By using the technique of artifical recharge, large quantities of water can be stored in a phreatic aquifer. In this

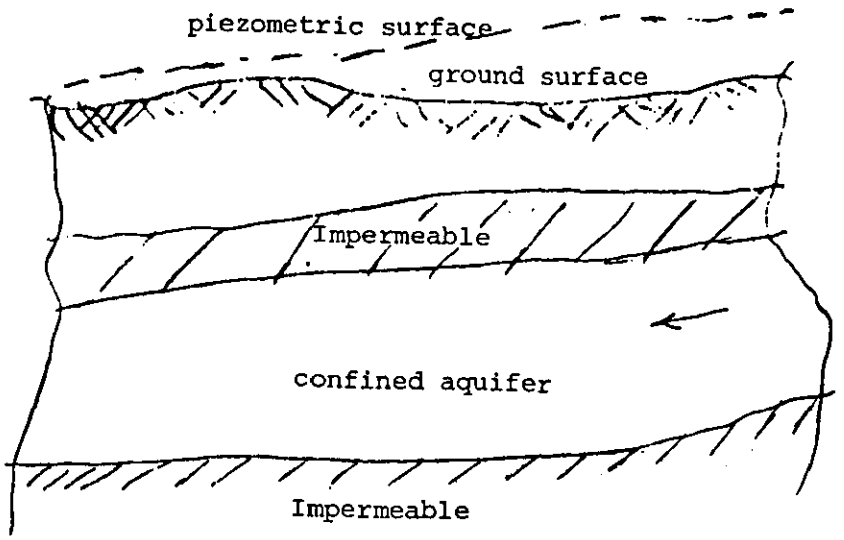


FIGURE 7.1: A schematic of a confined aquifer

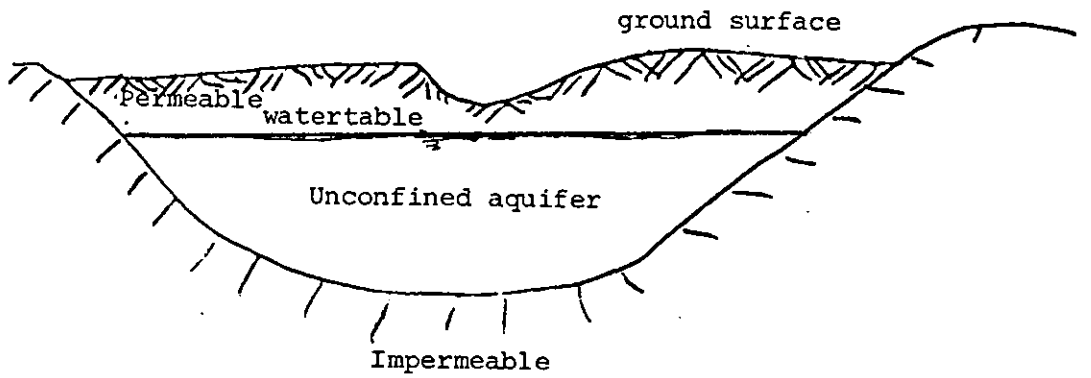


FIGURE 7.2: A schematic of an unconfined aquifer

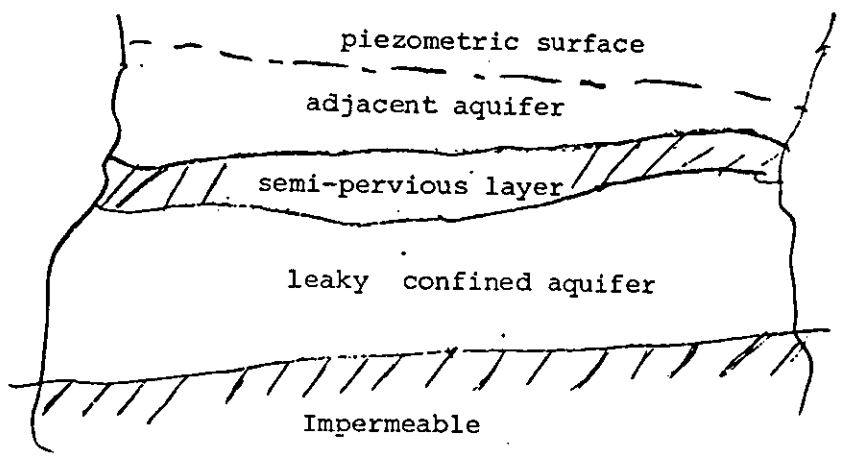


FIGURE 7.3: A schematic of a leaky confined aquifer

case, the aquifer is used as a storage reservoir. This can be helpful for either as time storage or for long term storage, e.g. from a wet sequence of years to a dry one.

(iii) Conduit-like: It is possible to introduce water into an aquifer by artificial recharge at one point and withdraw water by pumping at another point or set of points. The injected water will flow through the aquifer which is used as if it was a conduit. This technique may be used in place of a distribution system provided that certain hydraulic constraints are observed.

(iv) Filter plant: It is possible to use the techniques of artificial recharge to filter and purify water of inferior quality injected into an aquifer. In this sense, the aquifer acts as a filter plant. This may take several forms like: ion exchange phenomena on the solid surface of the porous material, chemical reactions, absorption,... etc. Generally, the ability of an aquifer to upgrade the quality of water depends on the chemical and physical properties of the aquifer material and on the type of mineral and organic impurities contained in the water.

7.1.3 Effect of Human Activities on Groundwater

Many human activities affect the groundwater level and/or quality. These can be summarized as follows [Bredehoeft, et al, 1982]:

(i) Activities that raise the water level:

The most important activity that raises the groundwater level is irrigation. Although there is usually a drainage system with every irrigation activity, but such drainage systems usually try to limit only the rise of the watertable. Dams and similar hydraulic structures which create reservoirs and lakes may also change patterns of groundwater

recharge and discharge. The change in surface-water level alters the boundary conditions for the groundwater system.

(ii) Activities that lower the water level:

Developments of groundwater lower the water level. Any structures that extract groundwater must create a gradient in the system which lowers the water levels. As an example, a production well will create a cone of depression. Continued declines in water levels are common in many areas of the world due to the excess of pumping from the aquifer or due to the mining of groundwater. Mining means extracting water as if it were a mine like gas or oil, i.e., in a non-renewable fashion.

(iii) Activities that change water quality:

Waste disposal is usually done to atmosphere, streams or other surface-water bodies on or into the earth. Waste disposal either as solids or liquids into the solid earth has associated hazards of contamination and transport by groundwater. Even disposal to the atmosphere or streams may indirectly affect the groundwater quality. Increased use of chemicals both in agriculture and other municipalities may also cause aquifer contamination.

7.1.4 Groundwater Problems

It is possible to classify the different types of problems concerned with groundwater into three major categories as follows:

(i) Prediction problems: these are the most commonly solved problems and indeed most of the research done so far in the groundwater field is concerned with prediction problems. In these problems, we are interested in studying physical phenomena associated primarily with the aquifer. Four types of problems may be identified as prediction

problems. These are: flow, land subsidence, mass transport and heat transport predictions. In the flow prediction problem, the data about the aquifer parameters, boundary conditions and sources or sinks in the domain are utilized to solve the PDE's, of flow in order to determine quantitative parameters of groundwater flow such as direction and rate of flow, changes in water level and the pressure at different points. In fact this class of problems is considered within this thesis. Land subsidence caused by withdrawals of groundwater is another type of prediction problem where the deformations and stresses are to be determined. A third type of prediction problem is that concerned with mass transport in groundwater. This is primarily associated with groundwater quality problems where the movement and concentration of various pollutants is predicted. Among the problems of this type is that of the sea water intrusion in coastal aquifers. In heat transport problems, the flow of heat is coupled with water or steam. An example of this type of problem is the analysis of hot springs, heat pumps and geothermal reservoirs. It should be noticed, however, that it is possible to have prediction problems of more than one type. Prediction problems can be solved either analytically, using analog techniques or numerically.

(ii) Resource management problems: these are concerned by the utilization of the groundwater as a resource which has to be optimized in a particular sense subjected to constraints.

So, the target is to determine the courses of action which will be consistent with predetermined management objectives and constraints. This system approach to groundwater management is relatively new compared to prediction problems. The management objectives may be, for

example, cost minimization or ensure a minimum of water supply; while the constraints may be legal, technological or economical.

(iii) Identification problems: these can be viewed as the inverse of the prediction problem. Here, we seek to determine the values of the parameters of the aquifers, e.g. transmissivity, based on actual data gathered from the field tests.

7.2 MODELLING OF GROUNDWATER FLOW

7.2.1 The Basic Equations

1. Darcy's Law

The generalized Darcy's law which relates the specific discharge to the hydraulic head can be written as [Wang and Anderson, 1982]:

$$q = -K\,\text{grad}\,h \tag{7.1}$$

where: q is the specific discharge or Darcy's velocity vector. It is the volume rate of flow per unit area, its unit is L/T i.e. length per time

$$Q = [q_x \quad q_y \quad q_z]^T$$

K is the hydraulic conductivity or the permeability tensor its unit is L/T and,

h is the piezometric head. Its unit is L

grad h or ∇h is the vector $[\frac{\partial h}{\partial x} \frac{\partial h}{\partial y} \frac{\partial h}{\partial z}]^T$.

Equation (7.1) can be written in its expanded form as:

$$\begin{Bmatrix} q_x \\ q_y \\ q_z \end{Bmatrix} = -\begin{bmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{yx} & K_{yy} & K_{yz} \\ K_{zx} & K_{zy} & K_{zz} \end{bmatrix} \begin{Bmatrix} \frac{\partial h}{\partial x} \\ \frac{\partial h}{\partial y} \\ \frac{\partial h}{\partial z} \end{Bmatrix} \tag{7.2}$$

Of the nine components of K there are only six different permeability coefficients since K is symmetric. This is due to the fact that for any soil, the flow in any direction is equal and opposite under an equal and opposite pressure gradient. If the cartesian coordinates are chosen to coincide with the principle axes of the hydraulic conductivity tensor K, then K will be a diagonal matrix, i.e.,

$$K = \begin{bmatrix} K_{xx} & 0 & 0 \\ 0 & K_{yy} & 0 \\ 0 & 0 & K_{zz} \end{bmatrix} \tag{7.3}$$

Although Darcy demonstrated his law experimentally in one dimension only, it has been proved theoretically from the general Navier-Stokes equations for viscous flow [DeWiest, 1965]. It is also important to notice that Darcy's law is valid only for laminar flow. For most cases of groundwater movement, the flow is laminar with low Reynolds number (R_e).

2. Aquifer transmissivity:

Assuming an aquifer of thickness B at any point, the quantity T is called the transmissivity and defined by,

$$KB = T \ . \tag{7.4}$$

The unit of T is L^2/T. The transmissivity of the aquifer is a characteristic of it which gives the rate of flow per unit width through the entire thickness of the aquifer per unit hydraulic gradient [Bear, 1979]. The concept is valid only in 2-D flow which, in fact, is usually encountered in aquifers.

3. Storage coefficient:

The storage coefficient of an aquifer S (also called storativity) is defined as the volume of water yielded per unit horizontal area and per unit drop of watertable (for unconfined aquifers) or piezometric surface (for confined aquifers). For example, if an unconfined aquifer released $4m^3$ water for a watertable drop of 2m over a horizontal area of $10m^2$, the storage coefficient is .2. For unconfined aquifers, the storage coefficient is called also the specific yield, which is the volume of water released from a unit volume of saturated aquifer material drained by a falling watertable [Bouwer, 1978]. Both S and T are determined by pumping tests in the field. S normally varies directly with aquifer thickness and as a thumb rule, it is given by:

$$S = 3\times10^{-6}B \quad , \tag{7.5}$$

where B is the saturated aquifer thickness in meters.

Note, however, that S is a dimensionless quantity involving a volume of water per unit volume of aquifer. It is customary to define the specific storage S_e as the ratio of S to the thickness B, i.e.,

$$S_e = S/B \quad . \tag{7.6}$$

4. The Dupuit Assumption:

Groundwater flow is assumed to be essentially horizontal, i.e., the vertical flow components are neglected. It is also assumed to be uniformly distributed with depth. This is called the Dupuit assumption. Its validity is noticed since the depth of aquifer is usually too small compared to its area and due to the very slow velocity of flow.

5. The fundamental equation of aquifer flow:

Assuming that the averaged flow field can be considered as a continuum and applying the principle of mass conservation, the following equation can be written [Liggett and Liu, 1983]:

$$-S_e \frac{\partial h}{\partial t} = \nabla.(q) \quad , \tag{7.7}$$

where S_e is the specific storage and $\nabla.(q)$ is the divergence of $q = \frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z}$.

Now, assuming the flow is essentially horizontal, i.e. the Dupuit assumption is valid, Equation (7.3) will be:

$$K = \begin{bmatrix} K_{xx} & 0 \\ 0 & K_{yy} \end{bmatrix} \tag{7.8}$$

substituting in (7.1) gives:

$$\begin{Bmatrix} q_x \\ q_y \end{Bmatrix} = - \begin{bmatrix} K_{xx} & 0 \\ 0 & K_{yy} \end{bmatrix} \begin{bmatrix} \frac{\partial h}{\partial x} \\ \frac{\partial h}{\partial y} \end{bmatrix} \tag{7.9}$$

Substituting in (7.7) for q from (7.9) yields:

$$S_e \frac{\partial h}{\partial t} = \frac{\partial}{\partial x} (K_{xx} \frac{\partial h}{\partial x}) + \frac{\partial}{\partial y} (K_{yy} \frac{\partial h}{\partial y})$$

Using equation (7.6) we get:

$$S \frac{\partial h}{\partial t} = \frac{\partial}{\partial x} (T_{xx} \frac{\partial h}{\partial x}) + \frac{\partial}{\partial y} (T_{yy} \frac{\partial h}{\partial y}) \tag{7.10}$$

This equation describes the flow in aquifers under the previously stated assumptions i.e. assuming that Darcy's law and the Dupuit assumption are both applicable and valid and considering the water to be an incompressible homogeneous fluid with constant density. If there are sources or sinks in the flow domain i.e. inputs and/or outputs, we add their effect in equation (7.10) to get:

$$S \frac{\partial h}{\partial t} = \frac{\partial}{\partial x} (T_{xx} \frac{\partial h}{\partial x}) + \frac{\partial}{\partial y} (T_{yy} \frac{\partial h}{\partial y}) - Q \tag{7.11}$$

where Q(x,y,t) is the net groundwater withdrawal or recharge including pumping with unit L/T.

In the case of steady conditions we get:

$$\frac{\partial}{\partial x} (T_{xx} \frac{\partial h}{\partial x}) + \frac{\partial}{\partial y} (T_{yy} \frac{\partial h}{\partial y}) - Q = 0 \tag{7.12}$$

If the aquifer is homogeneous isotropic i.e. T is independent on the position, then (7.12) will be further simplified to:

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = Q \ . \tag{7.13}$$

In the case of no inputs or outputs, i.e. Q=0, equation (7.13) will be:

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0 \ , \tag{7.14}$$

which is the Laplace equation.

7.2.2 Boundary Conditions in Aquifers

In managing and analyzing any aquifer, the boundary conditions must be carefully evaluated. The boundaries may be physical such as bedrock contacts, or they may be hydraulical, such as limited pressure areas. They may even be political boundaries such as boundaries between different countries if the aquifer passes through them [ASCE, 1972].

As usual, there are three types of boundary conditions:

(i) The prescription of the piezometric head h on part of the boundary, i.e.,

$$h = h_o(x,y,t) \tag{7.15}$$

This is called: 1st type (I) Dirichlet, geometric or essential boundary condition.

(ii) The prescription of the flux, q, normal to some part of the boundary, i.e.,

$$q = -(T_{xx} \frac{\partial h}{\partial x} \ell_x + T_{yy} \frac{\partial h}{\partial y} \ell_y) = q_o(x,y,t) \tag{7.16}$$

where ℓ_x and ℓ_y are the direction cosines between the normal to the boundary surface and the x and y directions, respectively. This is called the Neumann, natural, 2nd type (II) or forced boundary condition.

(iii) The prescription of a boundary flux which is proportional to the aquifer piezometric head on some part of the boundary. This is called the mixed or the 3rd type (III) boundary condition.

In addition to these boundary conditions, for unsteady problems, i.e. time dependent ones, the initial condition of the piezometric head should be specified over the entire aquifer domain at the initial time value $t=t_o$ (usually at time 0), i.e.,

$$h = h_o(x,y,t_o) \tag{7.17}$$

In reality, examples of these boundary conditions can be:

(i) First type boundary condition may be encountered when an aquifer is in direct hydraulic contact with a river or a lake in which the water level is known.

(ii) Second type boundary condition may be encountered in the case where an impervious strata is above or/and below the aquifer.

(iii) Third type boundary condition may be encountered in the case of induced infiltration from a lake or a stream into the aquifer.

These three types are shown schematically in Figure 7.4.

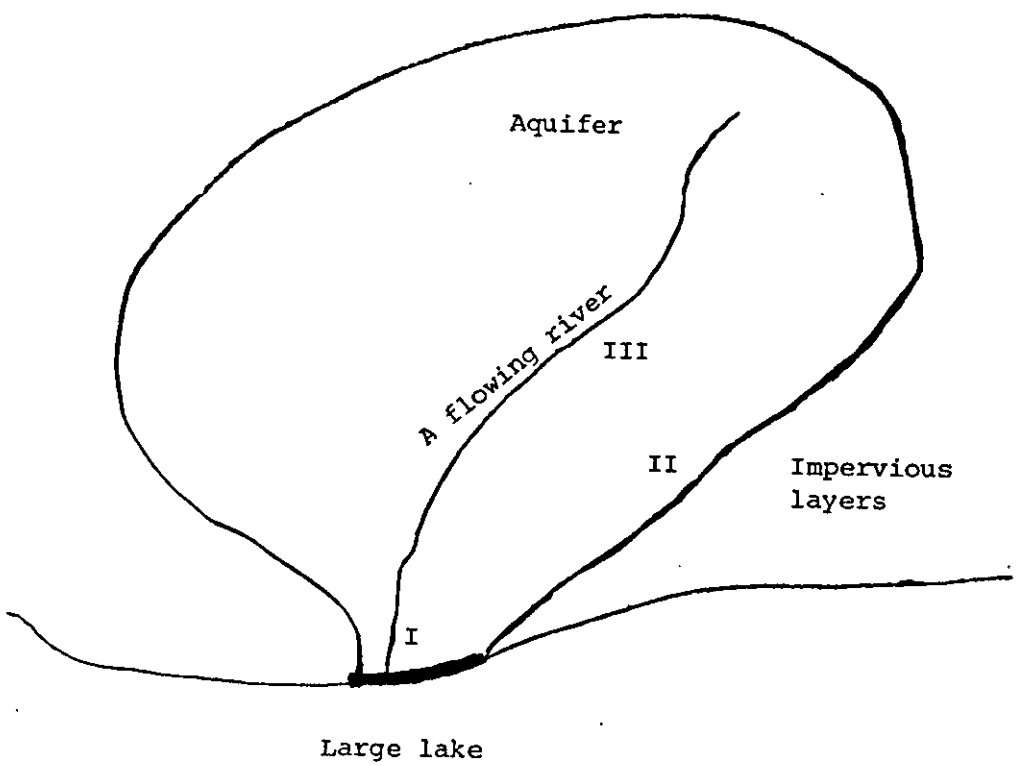


FIGURE 7.4: A schematic of boundary conditions in aquifers

7.2.3 Solution Methods

Solution methods for groundwater flow problems can be divided into three categories [Prickett, 1975]: sand tank models, analog models and mathematical models. Sand tank models are physical scale models used to simulate the actual flow systems. In these models a sand box is utilized and water or another liquid is allowed to flow through the sand. By measuring the actual flow in the model, the flow in the prototype can be calculated from [Bouwer, 1978]:

$$Q_p = Q_m \frac{K_p}{K_m} S_L^2 \tag{7.18}$$

and,

$$S_L = L_p / L_m \tag{7.19}$$

where, S_L is the scale factor

L_p and L_m are the prototype and model length, respectively,

K_p and K_m are the hydraulic conductivity of the aquifer and model materials, respectively,

and Q_p and Q_m are the prototype and model flow rates, respectively.

It is assumed that the model is a true scale model of the prototype, i.e. all dimensions, including the head h, are reduced by the same scale factor S_L. Sand tank models were among the first models built to study groundwater flow problems. However, one of the major difficulties in these models is that the height of the capilliary fringe may be out of proportion to the height of the flow system below the watertable if a free watertable is modelled.

In the electrical analog models, the similarity between Ohm's law and Darcy's law is utilized to build an electrical network model that simulates the groundwater flow system. Measuring electrical quantities: currents, volts,... etc., actual physical quantities in the aquifer can

be computed. Two types are usually used: the resistance-network analogs and the resistance-capacitance ones.

Mathematical models, which are in the form of sets of differential equations, are solved either analytically or numerically. Analytical solutions are available for many groundwater flow problems but only for simple geometries and idealized situations and hence, in otherwise cases, numerical techniques are the logical alternative. The three methods that can be used in this case are the finite difference methods (FDM), the finite element methods (FEM) and, more recently, the boundary integral equation methods (BIEM). Though the FDM were traditionally used to solve groundwater flow problems, the FEM is becoming more and more popular. Application of the BIEM for aquifer flow problems has recently started for some problems. In this work, the FEM is used to solve some of the aquifer flow problems. In fact, the software given in Chapter 6 will be used. Since some of the problems are time-dependent we use the FEM for the spatial analysis while the resulting set of ordinary differential equations are solved by the FDM.

7.2.4 Software for Groundwater Flow Problems

Several programs for the analysis of groundwater flow problems exist. Some of these have been developed by academics in universities and research centres while the others are developed by private firms and usually are licensed to users only. Since these firms make money from royalties paid by users, their programs are usually delivered in binary form only, i.e. no source code is given to the users. Scanning some of the most well known software programs for fluid flow it can be safely said that they are mostly based on FDM and FEM only. Here, we list some of

the software which may be used to solve some of the aquifer flow problems.

(i) Programs that require a mainframe or a super-minicomputer:

1. In the Australian Water Resources Council project 71/25, 8 programs for analysis of flow towards wells in an aquifer was developed [Huyakorn and Dudgeon, 1974]. These programs seem to be amongst the earliest FE programs used to solve well-flow problems of considerable complexity. They handle partial penetration, partial screening, gravel packing and the occurrence of non-Darcy flow. It can solve 1 and 2 dimensional flow towards a pumped well constructed in unconsolidated materials. The formation materials are assumed to be isotropic. Also, modifications of the programs to deal with anisotropy were outlined within the documentation of these programs. It is evident that these programs are useful only for a detailed study of well flow. On the regional level, these problems are of relatively less importance.
2. Four mainframe-based programs for general flow dynamics are quoted in [Johnson and Torok, 1985]. Two of these are: "Fidap" and "Flodyn" which are based on FEM for incompressible fluid flow. The other two are: "Phoenics" and "Fluent" which are based on FDM. These programs are usually used for general continuum problems described by the Navier-Stokes equations for fluid flow and are mathematically oriented based software. Their use for groundwater problems was not demonstrated but there is no reason to believe that they cannot be used for such problems.

(ii) Programs that require small size mini-computers or micro-computers:

1. The "IFEP" program developed by the author has already been presented

in Chapter 4 and is a FE based program for the analysis of steady flow in aquifers.

2. The "AQUIFER/1000" program was initially developed by Dr. E.O. Frind of the University of Waterloo in Canada in 1972. Later it was adopted by Compeng Computer Systems, Ltd. in Canada and implemented on the mini-computer HP1000. It can be used to compute the transient piezometric surfaces for two dimensional confined aquifers. It is now based on linear triangular finite elements.
3. Another aquifer simulation program for micro-computers is that developed by Briz-Kishore and Avadhanula [1981]. This program is based on the FDM and can be used for steady-state and dynamic conditions for both homogeneous and heterogeneous aquifer systems. Two versions of this program have been developed, one in Fortran and the other in Basic.
4. Two programs, one for aquifer flow [Wilson, et al, 1979] and the other for seawater intrusion [Sa Da Costa, 1980] have been developed at the MIT in the USA. They can be run on a small size computer and are based on the FEM.

7.3 THE FINITE ELEMENT FORMULATION

In this section the FE formulation of the general aquifer flow problem defined by Equation (7.11) is given. The formulation is based on the Galerkin's method. Since this problem is a time-dependent one, the second approach described in 3.10.1 is used, i.e. the nodal values are considered functions of time and the space variables are used in the FE analysis. This leads to a system of ordinary differential equations which can be solved by the finite difference method.

Since the method of weighted residuals (MWR) and the Galerkin's FEM formulation were discussed in detail in Chapter 3, unnecessary details will not be repeated again.

Assuming the trial solution for equation (7.11) to be $\hat{h}$, then:

$$\hat{h}(x,y,t) = N(x,y)H(t) \ , \tag{7.20}$$

where $N(x,y)$ are the interpolation functions and $H(t)$ are the nodal values of the piezometric head at time t. Since $\hat{h}$ is an approximate solution to equation (7.11), then this equation will not be satisfied exactly and hence a residual or error will be created. In other words, the expression,

$$S\frac{\partial \hat{h}}{\partial t} - \frac{\partial}{\partial x}\left(T_{xx}\frac{\partial \hat{h}}{\partial x}\right) - \frac{\partial}{\partial y}\left(T_{yy}\frac{\partial \hat{h}}{\partial y}\right) + Q$$

will not be zero.

In the method of weighted residual (MWR) the integral of the weighted residual is equated to zero, i.e.,

$$\int_A \left[S\frac{\partial \hat{h}}{\partial t} - \frac{\partial}{\partial x}\left(T_{xx}\frac{\partial \hat{h}}{\partial x}\right) - \frac{\partial}{\partial y}\left(T_{yy}\frac{\partial \hat{h}}{\partial y}\right) + Q\right] \omega_i \, dA = 0 \tag{7.21}$$

where ω_i are the weighting functions.

Rewriting equation (7.21) we have:

$$\int_A [S \frac{\partial \hat{h}}{\partial t} + Q]\omega_i dA - \int_A [\frac{\partial}{\partial x}(T_{xx} \frac{\partial \hat{h}}{\partial x}) + \frac{\partial}{\partial y}(T_{yy} \frac{\partial \hat{h}}{\partial y})]\omega_i dA = 0 \tag{7.22}$$

By using Green's theorem for the second term we get:

$$\int_A [S \frac{\partial \hat{h}}{\partial t} + Q]\omega_i dA - [\int_S (T_{xx} \frac{\partial \hat{h}}{\partial x} \ell_x + T_{yy} \frac{\partial \hat{h}}{\partial y} \ell_y)\omega_i dS$$

$$- \int_A (T_{xx} \frac{\partial \hat{h}}{\partial x} \frac{\partial \omega_i}{\partial x} + T_{yy} \frac{\partial \hat{h}}{\partial y} \frac{\partial \omega_i}{\partial y}) dA] = 0 \tag{7.23}$$

where S represents the boundary segment of the region. Assuming this boundary is divided into two parts S_1 and S_2 where S_1 represents a boundary of known head and S_2 a boundary of known flux then:

$$S = S_1 + S_2 \tag{7.24}$$

If the weighting functions are restricted so that they vanish along S_1, then the integration over S in equation (7.23) will be over S_2 only. If we recall equation (7.16) which defines the 2nd type boundary condition i.e., the prescribed flux boundary condition $q_o(x,y,t)$ then equation (7.23) will be:

$$\int_A (S \frac{\partial \hat{h}}{\partial t} + Q)\omega_i dA + \int_{S_2} q_o \omega_i dS + \int_A (T_{xx} \frac{\partial \hat{h}}{\partial x} \frac{\partial \omega_i}{\partial x} + T_{yy} \frac{\partial \hat{h}}{\partial y} \frac{\partial \omega_i}{\partial y}) dA] = 0 \tag{7.24}$$

Now, since, the weighting functions ω_i are chosen to be the same as the interpolation functions in the Galerkin's method i.e. $\omega_i = N_i$, then (7.24) will be:

$$\int_A [S \frac{\partial \hat{h}}{\partial t} + Q]N_i dA + \int_{S_2} q_o N_i dS + \int_A (T_{xx} \frac{\partial \hat{h}}{\partial x} \frac{\partial N_i}{\partial x} + T_{yy} \frac{\partial \hat{h}}{\partial y} \frac{\partial N_i}{\partial y}) dA = 0 \tag{7.25}$$

Since in the FEM the domain is discretized to elements, then equation (7.25) is valid in fact for each element in the domain.

Assuming the elements to be linear triangular, then, we, should obtain the element characteristics. Although these have been derived previously in Chapter 3 [see equations (3.190) to (3.210) for details], to complete the analysis here, they are summarized in the following:

$$\hat{h}^e = N_1^e H_1^e + N_2^e H_2^e + N_3^e H_3^e \tag{7.26}$$

which relates the piezometric head at any point in the element to the nodal values. The shape functions N_1, N_2 and N_3 can be expressed as:

$$N_1 = \frac{1}{2A}(a_1 x + b_1 y + c_1) \tag{7.27}$$

$$N_2 = \frac{1}{2A}(a_2 x + b_2 y + c_2) \tag{7.28}$$

$$N_3 = \frac{1}{2A}(a_3 x + b_3 y + c_3) \quad , \tag{7.29}$$

where (x_1,y_1), (x_2,y_2) and (x_3,y_3) are the coordinates of the vertices of the triangular element and the coefficients $a_1, a_2, \ldots$, etc. are given by:

$$a_1 = y_2 - y_3, \quad a_2 = y_3 - y_1, \quad a_3 = y_1 - y_2 \ , \tag{7.30}$$

$$b_1 = x_3 - x_2, \quad b_2 = x_1 - x_3, \quad b_3 = x_2 - x_1 \ , \tag{7.31}$$

$$c_1 = x_2 y_3 - x_3 y_2, \quad c_2 = x_3 y_1 - x_1 y_3 \text{ and } c_3 = x_1 y_2 - x_2 y_1 \tag{7.32}$$

and A is the area of the triangle given by:

$$A = \tfrac{1}{2} \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \tag{7.33}$$

It should be noted that in the natural coordinate system (area coordinates) the shape functions are given by,

$$N_1 = \frac{A_1}{A}, \quad N_2 = \frac{A_2}{A} \text{ and } N_3 = \frac{A_3}{A} \ , \tag{7.34}$$

where A_1, A_2 and A_3 are the areas of the subtriangles formed by any point inside the triangle [see Figure 3.11].

It is possible now to evaluate the derivatives in equation (7.25). For example,

$$\frac{\partial \hat{h}}{\partial x} = \frac{\partial N_1}{\partial x} H_1 + \frac{\partial N_2}{\partial x} H_2 + \frac{\partial N_3}{\partial x} H_3 = \frac{1}{2A}[a_1 \quad a_2 \quad a_3]H^e \tag{7.35}$$

Similarly,

$$\frac{\partial \hat{h}}{\partial y} = \frac{1}{2A}[b_1 \quad b_2 \quad b_3]H^e \tag{7.36}$$

and $$\frac{\partial \hat{h}}{\partial t} = [N_1 \quad N_2 \quad N_3] \frac{d}{dt}(H)^e \tag{7.37}$$

It is also known that [Davies, 1980] the area and arc segment integrals of the forms in equation (7.25) are:

$$\int_A N_1^m N_2^n N_3^p \, dA = \frac{2Am!n!p!}{(m+n+p+2)!} \tag{7.38}$$

and $$\int_S N_1^m N_2^n \, dS = S\frac{m!n!}{(m+n+1)!} \tag{7.39}$$

Substituting in equation (7.25), we get:

$$\int_{A^e} [S^e \sum_{j=1}^{3} N_j^e \frac{dH^e}{dt} + Q^e]N_i^e \, dA + \int_{S_2^e} q_o N_i^b dS + \int_A T_{xx}\left(\sum_{j=1}^{3} H_j^e \frac{\partial}{\partial x} N_j^e\right) \frac{\partial N_i^e}{\partial x} dA + \int_A T_{yy}\left(\sum_{j=1}^{3} H_j^e \frac{\partial}{\partial y} N_j^e\right) \frac{\partial N_i^e}{\partial y} dA = 0 \tag{7.40}$$

This equation can be written in a matrix form as:

$$K^e \frac{dH^e}{dt} + B^e H^e + F^e = 0 \, , \tag{7.41}$$

where, the matrix K^e has the elements k_{ij}^e defined by:

$$k_{ij}^e = S^e \int_{A^e} N_j^e N_i^e \, dA \tag{7.42}$$

$$b_{ij}^e = \int_{A^e} \left(T_{xx} \frac{\partial N_j^e}{\partial x} \frac{\partial N_i^e}{\partial x} + T_{yy} \frac{\partial N_j^e}{\partial y} \frac{\partial N_i^e}{\partial y} \right) dA \tag{7.43}$$

and $$F^e = F_Q^e + F_b^e \tag{7.44}$$

where F_Q stands for the contribution of source/sink flow and F_b stands for the boundary flow with elements

$$f_{iQ}^e = \int_{A^e} Q^e N_i^e \, dA \tag{7.45}$$

and $$f_{ib}^e = \int_{S_2^e} q_o N_i^b \, dS \tag{7.46}$$

By summing the element contributions, the overall equation for the whole domain will be,

$$\sum_{\forall \text{ elements}} \left(K^e \frac{dH^e}{dt} + B^e H^e + F_Q^e \right) + \sum_{\substack{\forall \text{ boundary} \\ \text{arcs}}} (F_b^e) = 0 \tag{7.47}$$

or globally as:

$$K \frac{dH}{dt} + BH = F \tag{7.48}$$

which is in fact, a set of ordinary differential equations to be solved for each time step. In the case of steady state flow, this will be:

$$BH = F \, , \tag{7.49}$$

which is indeed similar to the standard FE equation in structural analysis with B as the stiffness matrix, H is the unkknown vector of displacements and F the known vector of loads. The elements of these matrices may be obtained analytically in this particular case of linear triangular elements. To exemplify, equation (7.42) for the computation of k_{ij}^e will be given:

$$K^e = S^e \int_{A^e} \begin{bmatrix} N_1 \\ N_2 \\ N_3 \end{bmatrix} [N_1 \quad N_2 \quad N_3] \, dA \tag{7.50}$$

$$\therefore K^e = S^e \begin{bmatrix} \int N_1^2 & \int N_1 N_2 & \int N_1 N_3 \\ \int N_2 N_1 & \int N_2^2 & \int N_2 N_3 \\ \int N_3 N_1 & \int N_3 N_2 & \int N_3^2 \end{bmatrix} , \tag{7.51}$$

where all the integrals are over A^e. These integrals can be evaluated using equation (7.38). For example,

$$\int_{A^e} N_1^2 dA = \int_{A^e} N_1^2 N_2^0 N_3^0 \, dA = \frac{2A^e 2!}{4!} = \frac{A^e}{6} .$$

Similarly, other integrals can be evaluated and thus:

$$K^e = \frac{S^e A^e}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} . \tag{7.52}$$

7.4 STEADY FLOW IN AQUIFERS

In steady flow conditions, the governing equation will be,

$$\frac{\partial}{\partial x}\left(T_{xx}\frac{\partial h}{\partial x}\right) + \frac{\partial}{\partial y}\left(T_{yy}\frac{\partial h}{\partial y}\right) - Q = 0 \ , \tag{7.53}$$

In the special case where no sinks or sources exist, this is further simplified to:

$$\frac{\partial}{\partial x}\left(T_{xx}\frac{\partial h}{\partial x}\right) + \frac{\partial}{\partial y}\left(T_{yy}\frac{\partial h}{\partial y}\right) = 0 \ . \tag{7.54}$$

In the one-dimensional case, this will be:

$$\frac{d}{dx}\left(T_{xx}\frac{dh}{dx}\right) = 0 \ . \tag{7.55}$$

In this and the next section, some problems of flow in aquifers are solved. The results are compared with known solutions, if any. The modelling of the aquifers is done in several ways, e.g. increasing the number of triangles (h-refinement) or increasing the order of the elements used (p-refinement) in order to study the effect of these parameters on the accuracy of the obtained solutions and the computational cost.

7.4.1 Steady Flow in Confined Aquifers

We start with a very simple problem of 1-D flow in an homogeneous aquifer. In this case, T_{xx} is independent of h or x, and thus equation (7.55) can be integrated directly twice. Consider the case of prescribed head boundary of values: $h=h_o$ at $x=0$ and $h=h_L$ at $x=L$, the solution is:

$$h = h_o + \frac{(h_L - h_o)}{L}\, x \ . \tag{7.56}$$

Although the head is independent on the transmissivity, T_{xx}; the flux per unit width of the aquifer, Q_x is given by,

$$q_x = -T_{xx}\frac{dh}{dx} = T_{xx}(h_o - h_L)/L \ , \tag{7.57}$$

and thus is directly proportional on T_{xx}.

For 2-D problems of flow in homogeneous aquifers where T_{xx} and T_{yy} are both independent of h, x or y, the governing equation will be:

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0 \ , \qquad (7.58)$$

which is Laplace's equation.

Problem 7.1

In this problem, a rectangular confined aquifer under steady state conditions with constant transmissivity is considered as shown in Figure (7.5). The model data is:

Aquifer length = 10,000 m

" width = 1,000 m

" thickness = 20 m

" transmissivity $T_{xx} = T_{yy} = T = 20{,}000\ m^2/day$.

No flow conditions are assumed on the long sides while on the other sides the head is assumed to be 100m on one of them and 50m on the other. This problem has been solved with an increasing number of triangles (NTF) of 8, 16, 32, 64, 128 and 256. Other factors tested are: the out-of-core versus the in-core computation, the shape of the triangles and the degree

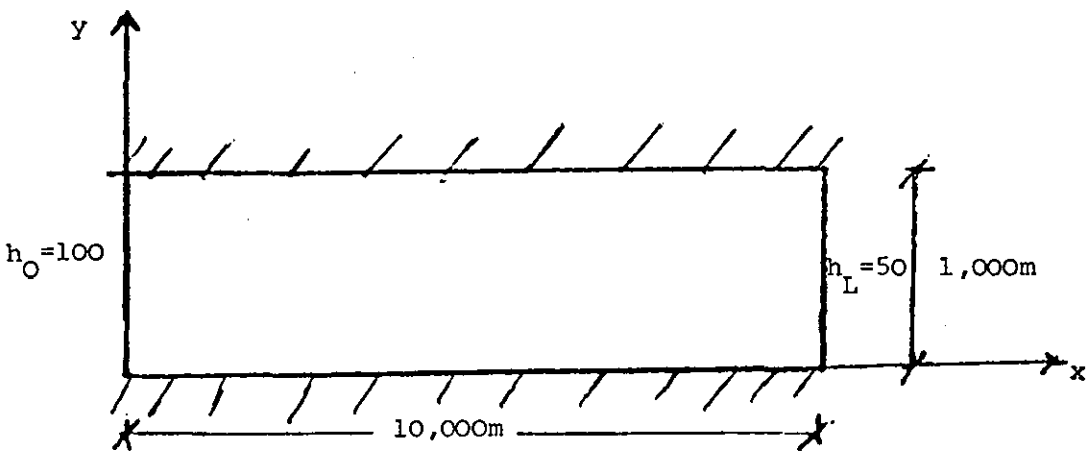


FIGURE 7.5: Aquifer of Problem 7.1

of interpolating polynomial. The results can be summarised as follows:

(i) Since the aquifer geometry has a high ratio of length to width (=10:1) with no flow boundary conditions across the long sides, the flow can be approximated by a 1-D flow and hence equation (7.56) can be used for comparison purposes. The results obtained show an accuracy of 5 significant digits using double precision arithmetic with the number of quadratic triangles $\geq$32. Use of higher order elements results in an increase in the processing time without a corresponding increase in the accuracy.

(ii) Since the number of triangles is relatively small, the required memory is very small, in fact the NDIM parameter is only 1380 words. Thus, the in-core solution is always preferred to out-of core due to the limited size of the memory required.

Figures (7.6) and (7.7) show the initial and final triangulation of this problem. Note, however, that they are reduced to A4 size which results in dimensions distortion. Moreover, since these are printer plottings, more distortion occurs. The results are shown in Table (7.1).

7.4.2 Steady Flow in Unconfined Aquifers

We follow the same line as that in 7.4.1 where confined aquifers were considered. In the unconfined (or phreatic) aquifers, the transmissivity is dependent on the head h which is now the water surface elevation. Thus, the governing equation for the flow will be:

$$\frac{\partial}{\partial x}(K_{xx} h \frac{\partial h}{\partial x}) + \frac{\partial}{\partial y}(K_{yy} h \frac{\partial h}{\partial y}) = 0 , \tag{7.59a}$$

or

$$K_{xx} \frac{\partial^2 h^2}{\partial x^2} + K_{yy} \frac{\partial^2 h^2}{\partial y^2} = 0 . \tag{7.59b}$$

In the case of homogeneous aquifer $K_{xx}=K_{yy}=K$ and (7.59) will be:

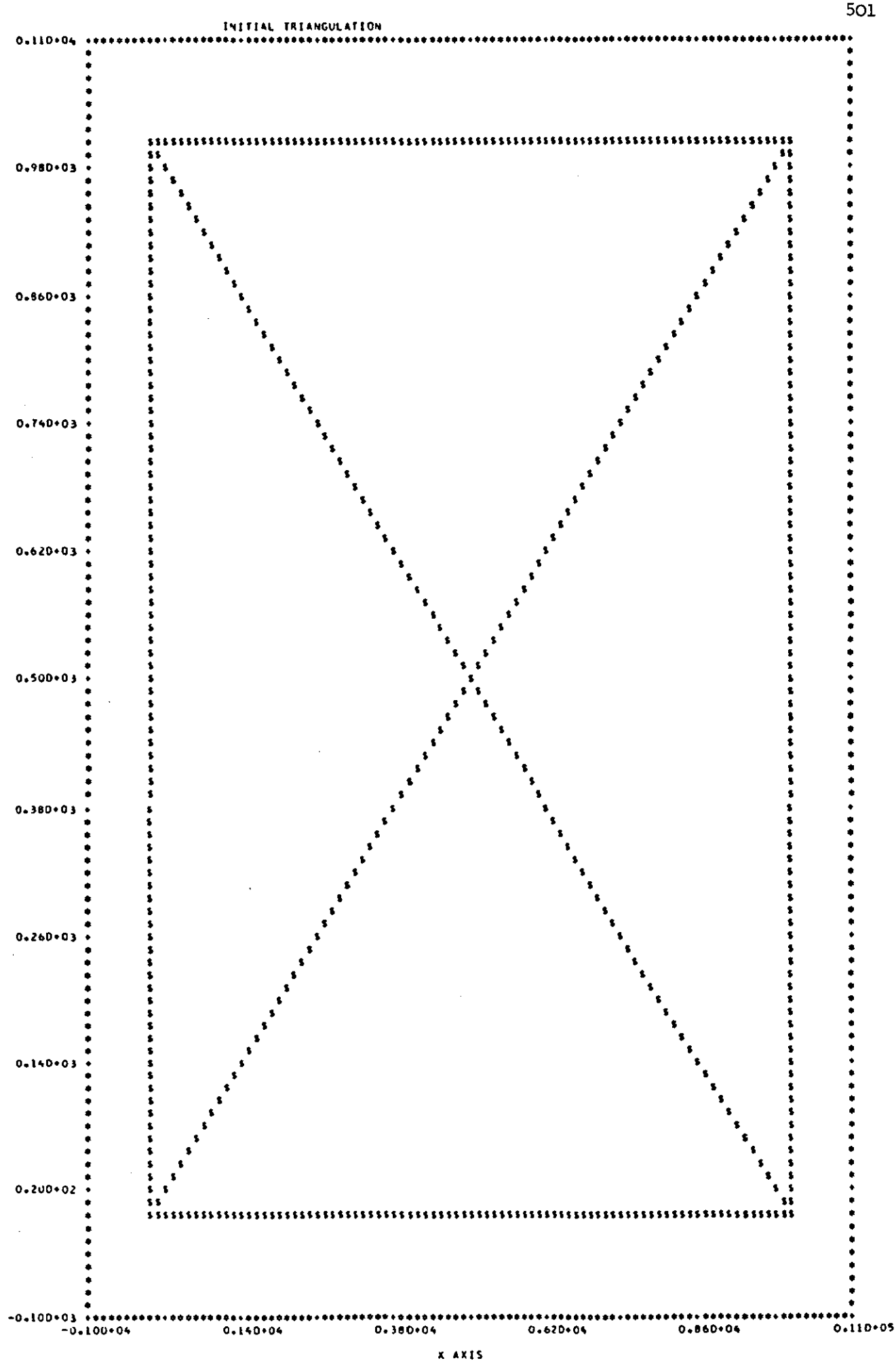


FIGURE 7.6: Initial triangulation for Problem 7.1

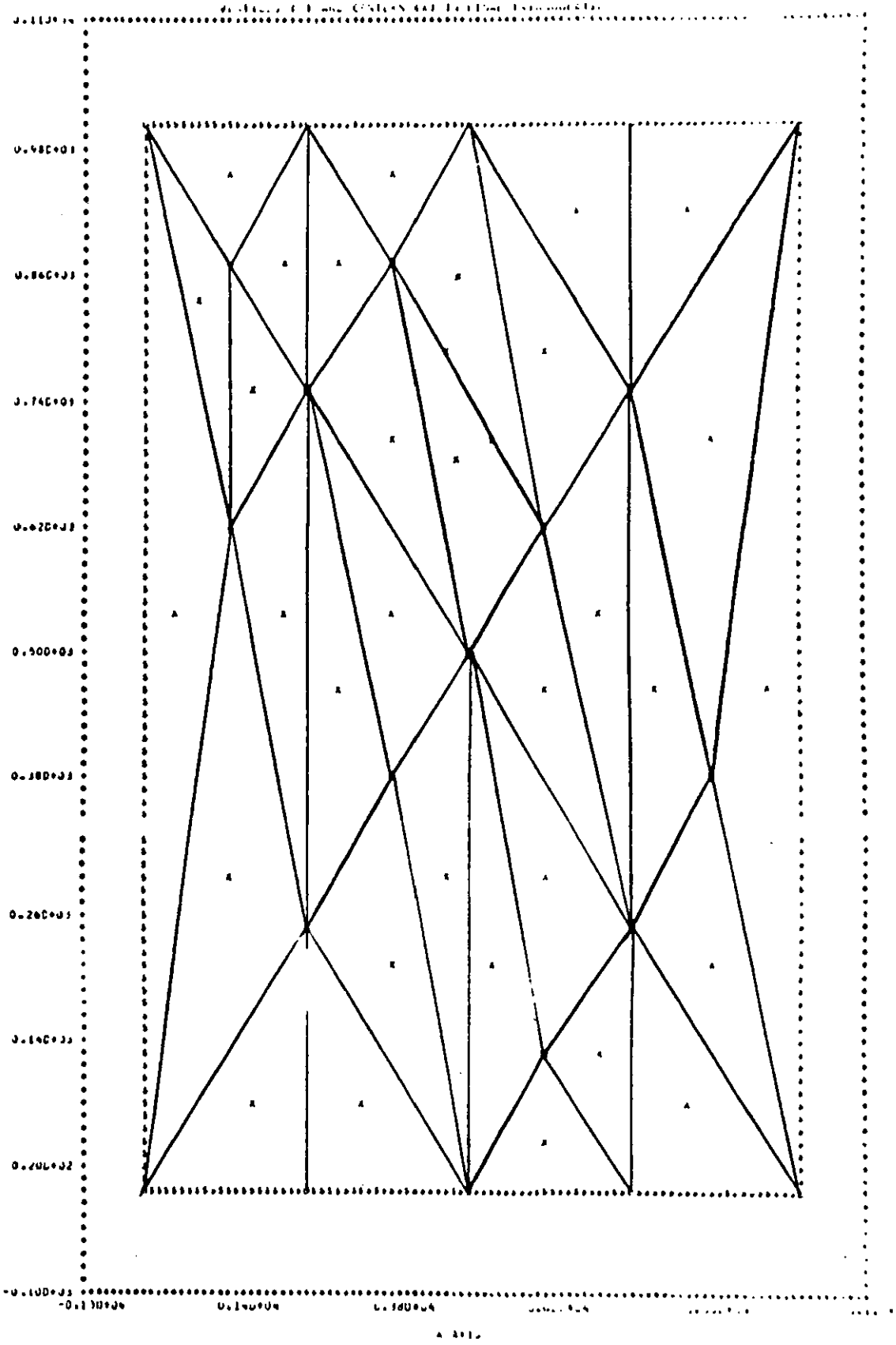


FIGURE 7.7: Final triangulation for Problem 7.1

X	Y	h
0.0	0.0	0.10000D+03
0.50000D+03	0.0	0.97500D+02
0.10000D+04	0.0	0.95000D+02
0.15000D+04	0.0	0.92500D+02
0.20000D+04	0.0	0.90000D+02
0.25000D+04	0.0	0.87500D+02
0.30000D+04	0.0	0.85000D+02
0.35000D+04	0.0	0.82500D+02
0.40000D+04	0.0	0.80000D+02
0.45000D+04	0.0	0.77500D+02
0.50000D+04	0.0	0.75000D+02
0.55000D+04	0.0	0.72500D+02
0.60000D+04	0.0	0.70000D+02
0.65000D+04	0.0	0.67500D+02
0.70000D+04	0.0	0.65000D+02
0.75000D+04	0.0	0.62500D+02
0.80000D+04	0.0	0.60000D+02
0.85000D+04	0.0	0.57500D+02
0.90000D+04	0.0	0.55000D+02
0.95000D+04	0.0	0.52500D+02
0.10000D+05	0.0	0.50000D+02
0.0	0.50000D+03	0.10000D+03
0.50000D+03	0.50000D+03	0.97500D+02
0.10000D+04	0.50000D+03	0.95000D+02
0.15000D+04	0.50000D+03	0.92500D+02
0.20000D+04	0.50000D+03	0.90000D+02
0.25000D+04	0.50000D+03	0.87500D+02
0.30000D+04	0.50000D+03	0.85000D+02
0.35000D+04	0.50000D+03	0.82500D+02
0.40000D+04	0.50000D+03	0.80000D+02
0.45000D+04	0.50000D+03	0.77500D+02
0.50000D+04	0.50000D+03	0.75000D+02
0.55000D+04	0.50000D+03	0.72500D+02
0.60000D+04	0.50000D+03	0.70000D+02
0.65000D+04	0.50000D+03	0.67500D+02
0.70000D+04	0.50000D+03	0.65000D+02
0.75000D+04	0.50000D+03	0.62500D+02
0.80000D+04	0.50000D+03	0.60000D+02
0.85000D+04	0.50000D+03	0.57500D+02
0.90000D+04	0.50000D+03	0.55000D+02
0.95000D+04	0.50000D+03	0.52500D+02
0.10000D+05	0.50000D+03	0.50000D+02
0.0	0.10000D+04	0.10000D+03
0.50000D+03	0.10000D+04	0.97500D+02
0.10000D+04	0.10000D+04	0.95000D+02
0.15000D+04	0.10000D+04	0.92500D+02
0.20000D+04	0.10000D+04	0.90000D+02
0.25000D+04	0.10000D+04	0.87500D+02
0.30000D+04	0.10000D+04	0.85000D+02
0.35000D+04	0.10000D+04	0.82500D+02
0.40000D+04	0.10000D+04	0.80000D+02
0.45000D+04	0.10000D+04	0.77500D+02
0.50000D+04	0.10000D+04	0.75000D+02
0.55000D+04	0.10000D+04	0.72500D+02
0.60000D+04	0.10000D+04	0.70000D+02
0.65000D+04	0.10000D+04	0.67500D+02
0.70000D+04	0.10000D+04	0.65000D+02
0.75000D+04	0.10000D+04	0.62500D+02
0.80000D+04	0.10000D+04	0.60000D+02
0.85000D+04	0.10000D+04	0.57500D+02
0.90000D+04	0.10000D+04	0.55000D+02
0.95000D+04	0.10000D+04	0.52500D+02
0.10000D+05	0.10000D+04	0.50000D+02

TABLE 7.1: Results of Problem 7.1

$$\frac{\partial}{\partial x}(h \frac{\partial h}{\partial x}) + \frac{\partial}{\partial y}(h \frac{\partial h}{\partial y}) = 0 \ . \tag{7.60}$$

In the case of 1-D flow with water surface elevation of h_o at x=0 and h_L at x=L, the equation will be:

$$\frac{d}{dx}(h \frac{dh}{dx}) = 0 \ . \tag{7.61}$$

By integrating twice and imposing the boundary conditions will give the solution as follows:

$$h^2 = h_o^2 + \frac{(h_L^2 - h_o^2)}{L} x \ . \tag{7.62}$$

The flux q will be:

$$q_x = -K_{xx} h \frac{\partial h}{\partial x} = -K h \frac{dh}{dx} = k \frac{(h_o^2 - h_L^2)}{2L} \ . \tag{7.63}$$

Problem 7.2

Consider an unconfined aquifer with dimensions of 10,000m x 1,000m and thickness of 120m. The permeability is assumed to be constant and = 1000m/day. The boundary conditions are identical to those in Problem 7.1, i.e. no flow across the long sides and the head (=water elevation) is 100m at one of the short sides and 50m at the other. The problem is solved using a different number of triangles as before. In fact, all the considered problems have been solved using this strategy. This is particularly useful not only to study the effect on computational costs but also in the case of difficult problems where no known analytical solution exists. In this case, solving the same problem with doubling the number of triangles and observing the obtained solution may be an indicator to decide whether the obtained solution is satisfactory or not. "Stability" of the results can be used as a pointer to the convergence to the correct solution. It should be noted that the region

X	Y	h
0.0	0.0	0.100000+03
0.500000+03	0.0	0.981070+02
0.100000+04	0.0	0.96177D+02
0.150000+04	0.0	0.942070+02
0.200000+04	0.0	0.921950+02
0.250000+04	0.0	0.901390+02
0.300000+04	0.0	0.880340+02
0.350000+04	0.0	0.858780+02
0.400000+04	0.0	0.83666D+02
0.450000+04	0.0	0.813940+02
0.500000+04	0.0	0.790570+02
0.550000+04	0.0	0.76649D+02
0.600000+04	0.0	0.74162D+02
0.650000+04	0.0	0.715880+02
0.700000+04	0.0	0.689200+02
0.750000+04	0.0	0.661440+02
0.800000+04	0.0	0.632460+02
0.850000+04	0.0	0.60208D+02
0.900000+04	0.0	0.57009D+02
0.950000+04	0.0	0.53619D+02
0.100000+05	0.0	0.50000D+02
0.0	0.500000+03	0.100000+03
0.500000+03	0.500000+03	0.981070+02
0.100000+04	0.500000+03	0.96177D+02
0.150000+04	0.500000+03	0.942070+02
0.200000+04	0.500000+03	0.921950+02
0.250000+04	0.500000+03	0.90139D+02
0.300000+04	0.500000+03	0.880340+02
0.350000+04	0.500000+03	0.858780+02
0.400000+04	0.500000+03	0.836660+02
0.450000+04	0.500000+03	0.813940+02
0.500000+04	0.500000+03	0.790570+02
0.550000+04	0.500000+03	0.766490+02
0.600000+04	0.500000+03	0.741620+02
0.650000+04	0.500000+03	0.715890+02
0.700000+04	0.500000+03	0.689200+02
0.750000+04	0.500000+03	0.66144D+02
0.800000+04	0.500000+03	0.632460+02
0.850000+04	0.500000+03	0.602080+02
0.900000+04	0.500000+03	0.570090+02
0.950000+04	0.500000+03	0.536190+02
0.100000+05	0.500000+03	0.500000+02
0.0	0.100000+04	0.100000+03
0.500000+03	0.100000+04	0.981070+02
0.100000+04	0.100000+04	0.961770+02
0.150000+04	0.100000+04	0.942070+02
0.200000+04	0.100000+04	0.921950+02
0.250000+04	0.100000+04	0.901390+02
0.300000+04	0.100000+04	0.880340+02
0.350000+04	0.100000+04	0.858780+02
0.400000+04	0.100000+04	0.836660+02
0.450000+04	0.100000+04	0.813940+02
0.500000+04	0.100000+04	0.790570+02
0.550000+04	0.100000+04	0.766490+02
0.600000+04	0.100000+04	0.741620+02
0.650000+04	0.100000+04	0.715890+02
0.700000+04	0.100000+04	0.689200+02
0.750000+04	0.100000+04	0.661440+02
0.800000+04	0.100000+04	0.632460+02
0.850000+04	0.100000+04	0.602080+02
0.900000+04	0.100000+04	0.570090+02
0.950000+04	0.100000+04	0.536190+02
0.100000+05	0.100000+05	0.500000+02

TABLE 7.2: Results of Problem 7.2.

is identical to that of Problem 7.1, so the initial and final triangulations will not be repeated again. The problem is basically non-linear but using the form of equation (7.59b) it can be considered as linear in h^2. The results of this problem are shown in Table (7.2) which compares very well with the approximate values obtained by equation (7.62).

7.4.3 Steady Flow in a Confined Aquifer with Leakage From an Adjacent One

The governing differential equation for steady flow in a confined aquifer with leakage from an adjacent one is [Bear, 1979]:

$$\frac{\partial}{\partial x}\left(T_{xx}\frac{\partial h}{\partial x}\right) + \frac{\partial}{\partial y}\left(T_{yy}\frac{\partial h}{\partial y}\right) - \frac{K'}{b'}(h-h_\ell) = 0 \ , \tag{7.64}$$

where:

T_{xx} is the aquifer transmissivity in the x-direction

T_{yy} is the aquifer transmissivity in the y-direction

h the piezometric head

$K'(x,y)$ is the vertical permeability of the semi-pervious leaky layer

$b'(x,y)$ is the thickness of the semi-pervious leaky layer.

h_ℓ is the piezometric head in an adjacent aquifer separated from the main aquifer by a semi-pervious, leaky layer.

In the case of an homogeneous aquifer, $T_{xx}=T_{yy}=T$ and equation (7.64) will be:

$$T\frac{\partial^2 h}{\partial x^2} + T\frac{\partial^2 h}{\partial y^2} - \frac{K'}{b'}(h-h_\ell) = 0 \ , \tag{7.65}$$

since T=KB in confined aquifers, then:

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} - \frac{K'}{b'KB}(h-h_\ell) = 0 \ . \tag{7.66}$$

In the case of 1-D flow, this is further simplified to:

$$\frac{d^2 h}{dx^2} = \frac{K'}{b'KB}(h-h_\ell) \ . \tag{7.67}$$

Assuming further that the boundary conditions for the aquifer are:

$$\left.\begin{aligned} & h = h_o \text{ at } x=0 \\ \text{and} \quad & h = h_L \text{ at } x=L \end{aligned}\right\} \tag{7.68}$$

then the analytical solution of (7.67) is given by:

$$h = h_\ell + (h_o - h_\ell)\frac{\sinh[c_1(L-x)]}{\sinh(c_1 L)} + (h_L - h_\ell)\frac{\sinh(c_1 x)}{\sinh(c_1 L)} \tag{7.69}$$

where,

$$c_1 = \sqrt{\frac{K'}{b'KB}} \quad . \tag{7.70}$$

The flux $q_x = -T_{xx}\frac{\partial h}{\partial x}$ will be,

$$q_x = c_1 T[(h_o - h_\ell)\frac{\cosh[c_1(L-x)]}{\sinh(c_1 L)} - (h_L - h_\ell)\frac{\cosh(c_1 x)}{\sinh(c_1 L)} \quad . \tag{7.71}$$

Problem 7.3

A confined aquifer of 10,000m length, 1,000m width and 20m thickness is considered. The value of T is assumed to be 20,000 m^2/day. The leaky aquifer permeability is assumed to be .0015 m/day. The thickness of the semi-pervious leaky layer b' is assumed to be 10m. The head in the adjacent aquifer h_ℓ is 95m. The prescribed boundary head values are h_o=100m and h_L=90m. The solution of this problem is obtained in a similar way to that of Problem 7.1. The results can be summarized as follows:

(i) The function F_1 is set to be: .00015*(95-U) where the factor K'/b'=.00015.

(ii) The results obtained are of high accuracy giving 5 decimal places using 32 triangles only as shown in Table 7.3.

X	Y	h
0.0	0.0	0.100000+03
0.500000+03	0.0	0.99475D+02
0.100000+04	0.0	0.98956D+02
0.150000+04	0.0	0.98445D+02
0.200000+04	0.0	0.97940D+02
0.250000+04	0.0	0.97443D+02
0.300000+04	0.0	0.96949D+02
0.350000+04	0.0	0.96459D+02
0.400000+04	0.0	0.95970D+02
0.450000+04	0.0	0.95484D+02
0.500000+04	0.0	0.95000D+02
0.550000+04	0.0	0.94516D+02
0.600000+04	0.0	0.940300+02
0.650000+04	0.0	0.935410+02
0.700000+04	0.0	0.930510+02
0.750000+04	0.0	0.925580+02
0.800000+04	0.0	0.920600+02
0.850000+04	0.0	0.915550+02
0.900000+04	0.0	0.910440+02
0.950000+04	0.0	0.905250+02
0.100000+05	0.0	0.900000+02
0.0	0.500000+03	0.100000+03
0.500000+03	0.500000+03	0.994740+02
0.100000+04	0.500000+03	0.989560+02
0.150000+04	0.500000+03	0.984450+02
0.200000+04	0.500000+03	0.979400+02
0.250000+04	0.500000+03	0.97443D+02
0.300000+04	0.500000+03	0.96949D+02
0.350000+04	0.500000+03	0.96459D+02
0.400000+04	0.500000+03	0.959700+02
0.450000+04	0.500000+03	0.95484D+02
0.500000+04	0.500000+03	0.950000+02
0.550000+04	0.500000+03	0.945160+02
0.600000+04	0.500000+03	0.940300+02
0.650000+04	0.500000+03	0.935410+02
0.700000+04	0.500000+03	0.930510+02
0.750000+04	0.500000+03	0.92557D+02
0.800000+04	0.500000+03	0.920600+02
0.850000+04	0.500000+03	0.915550+02
0.900000+04	0.500000+03	0.910440+02
0.950000+04	0.500000+03	0.905260+02
0.100000+05	0.500000+03	0.900000+02
0.0	0.100000+04	0.100000+03
0.500000+03	0.100000+04	0.994750+02
0.100000+04	0.100000+04	0.989560+02
0.150000+04	0.100000+04	0.984450+02
0.200000+04	0.100000+04	0.979400+02
0.250000+04	0.100000+04	0.974430+02
0.300000+04	0.100000+04	0.969500+02
0.350000+04	0.100000+04	0.964590+02
0.400000+04	0.100000+04	0.959700+02
0.450000+04	0.100000+04	0.954840+02
0.500000+04	0.100000+04	0.950000+02
0.550000+04	0.100000+04	0.945160+02
0.600000+04	0.100000+04	0.940300+02
0.650000+04	0.100000+04	0.935410+02
0.700000+04	0.100000+04	0.930510+02
0.750000+04	0.100000+04	0.925570+02
0.800000+04	0.100000+04	0.920600+02
0.850000+04	0.100000+04	0.915550+02
0.900000+04	0.100000+04	0.910440+02
0.950000+04	0.100000+04	0.905250+02
0.100000+05	0.100000+04	0.900000+02

TABLE 7.3: Solution of Problem 7.3

7.4.4 Modelling of Sources/Sinks in Aquifers

Often pumping and recharge wells are represented by sources and sinks in the flow domain. This is acceptable if the details of the flow patterns near the well and at its wall are not required. In the early groundwater models, the FDM was used to solve the flow problems. A well is modelled at the nearest mesh intersection point. In the FEM, and also in some FDM techniques, the well is represented by a small size element $\Delta x * \Delta y$. The well flow is assumed to be uniformly distributed over the element area. After several experiments to determine the best size of Δx and Δy, it seems that for large aquifers a value of 100m for Δx and Δy may be suitable. This is best illustrated by the following problem for a confined aquifer with pumping well.

Problem 7.4

A confined infinite aquifer whose transmissivity is 400 m^2/day has a static piezometric head of 20m and a thickness of 50m. A well is pumped at the rate of 2,000 m^3/day. We assume that a steady state situation is reached after a sufficiently long time of continuous pumping. In fact, theoretically, true equilibrium will never be reached, but practically it is considered to be reached after some days of pumping. The drawdown of the piezometric surface extends to a limited area around the well. We assume that this area is a circle with its centre at the well and of radius 4,000m. Note that the PDE in this case is (7.53). Assuming an homogeneous isotropic aquifer this equation will be

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = \frac{Q}{T} \quad . \tag{7.72}$$

This is to be solved in the circle centred at the origin and of radius 4,000m. The function Q has a value of 0 elsewhere except in the

vicinity of the well, i.e. near (0,0). The considered "small" area is taken to be 100×100; thus the flux will be $\frac{-2000}{100\times100}$ = -0.2 m/day and $\frac{Q}{T}$ will be -5×10^{-4}. The head is assumed to be constant at the whole boundary and equal to 20.

The modelling of this problem is shown in Figures (7.8) and (7.9) for the initial and final triangulations. The obtained solution shown in Table 7.4, compares well with that obtained by the Thiem equation [Bear, 1979] which relates the piezometric head h at any radius r to that initially in the aquifer. The Thiem equation is:

$$h(r) = h(r_e) + \frac{Q}{2\pi T} \log_e \frac{r}{r_e} \quad . \tag{7.73}$$

In the considered problem Q=2,000; r_e = radius of the cone of depression after which the effect of pumping in the piezometric head is negligable (also called the radius of influence of the well) = 4,000 and $h(r_e)$=20. Therefore we have,

$$h(r) = 20 + \frac{2000}{2\pi\times400} \log \frac{r}{4000} \quad . \tag{7.74}$$

An important point that should be considered when solving problems with point sources/sinks (well, springs,...) is to have a finer mesh near these singular points. This can be specified by the D3EST parameter in TWODEPEP. Experimentally, it is found that the following equation gives a gradual transition of the element density from the singular point to the outer boundary, i.e. most dense triangulation at the origin (at the well position) and a less dense triangulation as we go far from the well. The equation is:

```
D3EST = 1.0/(1.0+X**2+Y**2) .                    (7.75)
```

It is worth mentioning that this equation is valid for steady and unsteady

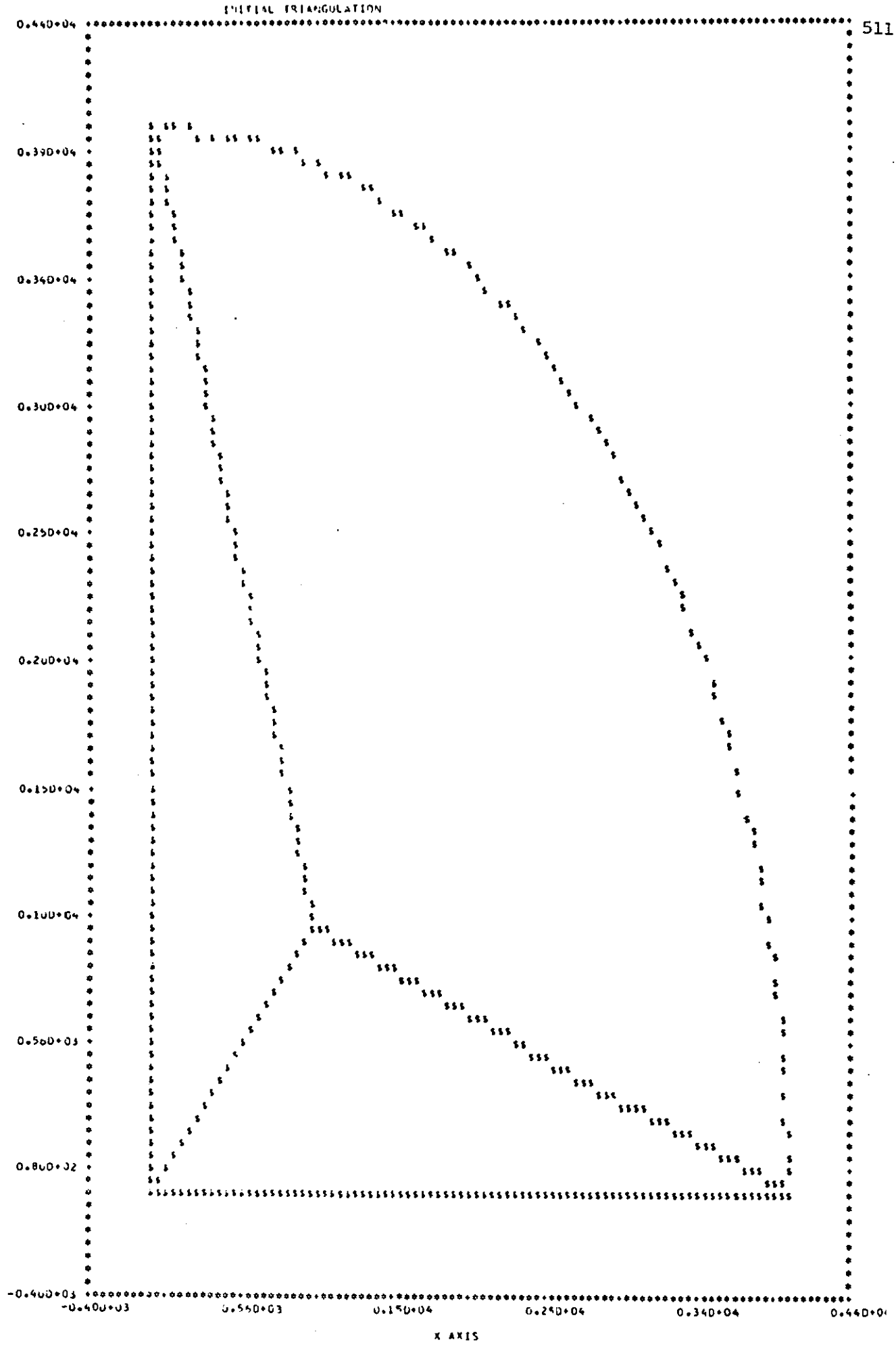


FIGURE 7.8: Initial triangulation for Problem 7.4

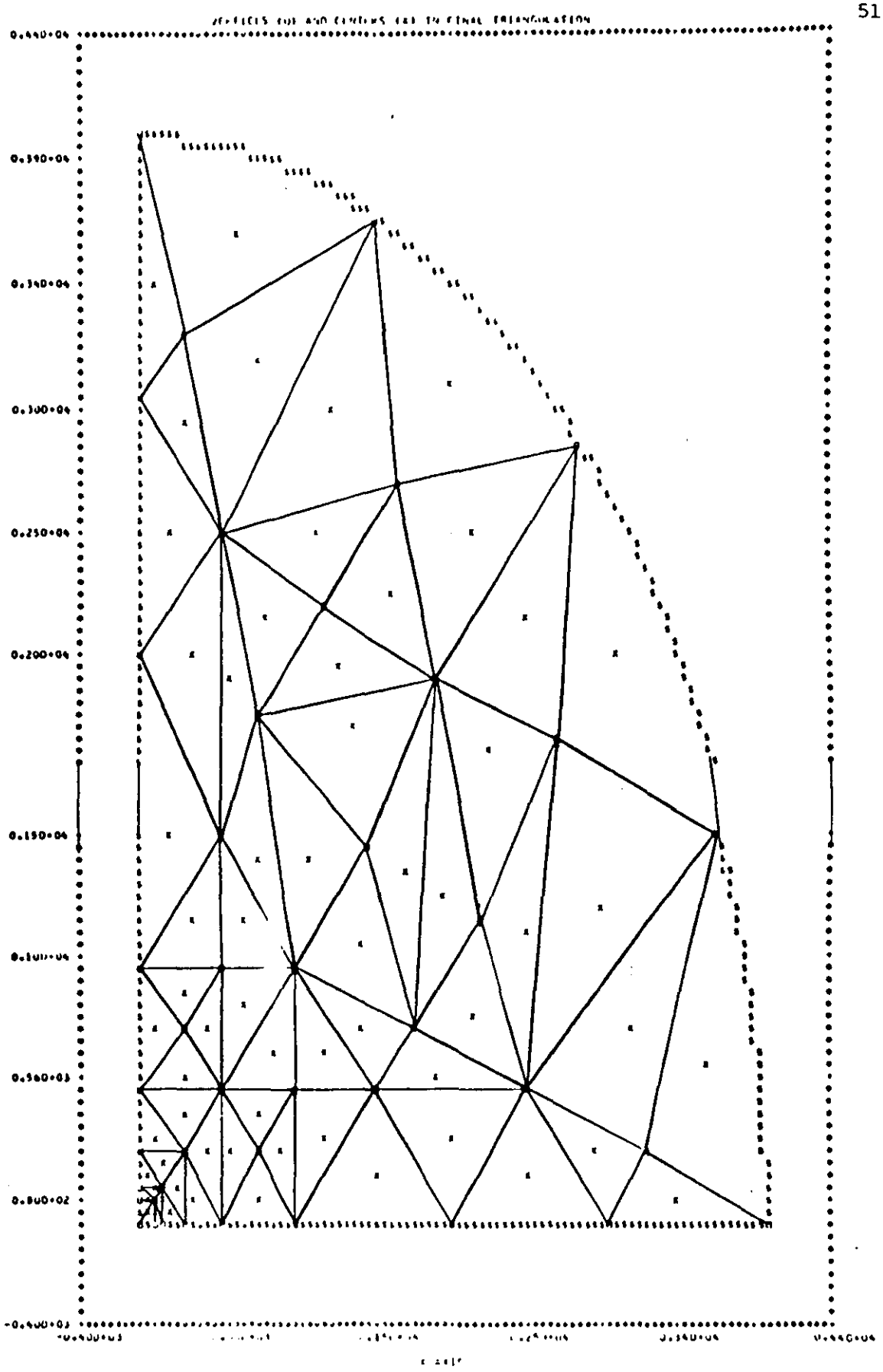


FIGURE 7.9: Final triangulation for Problem 7.4

X	Y	h
0.100000D+03	0.0	0.173570+02
0.200000D+03	0.0	0.178340+02
0.300000D+03	0.0	0.181160+02
0.400000D+03	0.0	0.183300+02
0.500000D+03	0.0	0.184910+02
0.600000D+03	0.0	0.186200+02
0.700000D+03	0.0	0.187350+02
0.800000D+03	0.0	0.188360+02
0.900000D+03	0.0	0.189230+02
0.100000D+04	0.0	0.189950+02
0.0	0.100000D+03	0.173570+02
0.100000D+03	0.100000D+03	0.175520+02
0.200000D+03	0.100000D+03	0.179130+02
0.300000D+03	0.100000D+03	0.181630+02
0.400000D+03	0.100000D+03	0.183500+02
0.500000D+03	0.100000D+03	0.185120+02
0.600000D+03	0.100000D+03	0.186350+02
0.700000D+03	0.100000D+03	0.187450+02
0.800000D+03	0.100000D+03	0.188410+02
0.900000D+03	0.100000D+03	0.189230+02
0.100000D+04	0.100000D+03	0.190000+02
0.0	0.200000D+03	0.178340+02
0.100000D+03	0.200000D+03	0.179130+02
0.200000D+03	0.200000D+03	0.180780+02
0.300000D+03	0.200000D+03	0.182620+02
0.400000D+03	0.200000D+03	0.184120+02
0.500000D+03	0.200000D+03	0.185490+02
0.600000D+03	0.200000D+03	0.186610+02
0.700000D+03	0.200000D+03	0.187640+02
0.800000D+03	0.200000D+03	0.188560+02
0.900000D+03	0.200000D+03	0.189370+02
0.100000D+04	0.200000D+03	0.190100+02
0.0	0.300000D+03	0.181160+02
0.100000D+03	0.300000D+03	0.181630+02
0.200000D+03	0.300000D+03	0.182620+02
0.300000D+03	0.300000D+03	0.183660+02
0.400000D+03	0.300000D+03	0.184900+02
0.500000D+03	0.300000D+03	0.186000+02
0.600000D+03	0.300000D+03	0.187020+02
0.700000D+03	0.300000D+03	0.187950+02
0.800000D+03	0.300000D+03	0.188800+02
0.900000D+03	0.300000D+03	0.189570+02
0.100000D+04	0.300000D+03	0.190260+02
0.0	0.400000D+03	0.183300+02
0.100000D+03	0.400000D+03	0.183500+02
0.200000D+03	0.400000D+03	0.184120+02
0.300000D+03	0.400000D+03	0.184900+02
0.400000D+03	0.400000D+03	0.185820+02
0.500000D+03	0.400000D+03	0.186660+02
0.600000D+03	0.400000D+03	0.187580+02
0.700000D+03	0.400000D+03	0.188360+02
0.800000D+03	0.400000D+03	0.189110+02
0.900000D+03	0.400000D+03	0.189830+02
0.100000D+04	0.400000D+03	0.190480+02
0.0	0.500000D+03	0.184910+02

TABLE 7.4: Solution of Problem 7.4

problems since the topology in the FE modelling is essentially the same.

It should be noticed also that when modelling this problem, no flow conditions are specified along arcs 1 and 3. This is justified by symmetry since the flow is radial towards the well.

7.5 UNSTEADY FLOW IN AQUIFERS

The unsteady flow in aquifers is governed by the basic equation (7.11) which for homogeneous isotropic aquifers is simplified to:

$$\frac{S}{T}\frac{\partial h}{\partial t} = \frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} - \frac{Q}{T} \quad . \tag{7.76}$$

In the case of no sources or sinks this will be:

$$\frac{S}{T}\frac{\partial h}{\partial t} = \frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} \quad . \tag{7.77}$$

As stated before, this type of equation can be solved by FEM in two ways. In one of them, the FEM is used for the spatial and temporal analysis, while in the other, FEM is used for the spatial analysis while FDM is used for the temporal one. This second approach is used in our analysis. However, as expected, care must be taken for the proper choice of the time step.

7.5.1 The Time Interval

In linear problems it is advantageous to keep fixed time steps. The main reason is that the master matrix will be constant for all time steps. However, since it is not always possible to keep the time step fixed, it seems that variable time steps with a reduced number of master matrix reformulation is the logical alternative choice. If the master matrix is the same, it will be factored only once. In non-linear time-dependent problems (unsteady or transient state), the master matrix must be formed and factored several times in each time step. This will make the computational cost very high. One of the possible solutions to this situation is to update the master matrix every n time steps. Usually n is 2 or 3. Indeed this approach is the one used for the

solution of unsteady problems. The maximum initial time step that is used is from the following emperical equation by Wilson et al [1979]:

$$\Delta t \leq \left(\frac{L^2 S}{T}\right)_{max} \quad , \tag{7.78}$$

where L^2, T and S are the characteristic values of the element area, transmissivity and storativity, respectively. Of course, after a reasonable time, this time step can be increased considerably. The parameters NUPDT, TO, TF, DT can, therefore, be specified in a manner that keeps the time step in the correct range without excessive computations. In linear unsteady, symmetric problems, NUPDT is set to O, in other problems, values of O,2,3 were used and tested. It seems that a value of 2 or 3 may be sufficient. The value of DT used varies from .0005 days to 10 depending upon the value of t itself. A function is defined which gives the correct time step at each time. This is the FUNT (T) function. The details of which are shown in the programs of the following problems which deal with unsteady flow in aquifers.

7.5.2 Unsteady Flow in Confined Aquifers

We start with the case of 1-D flow in an homogeneous aquifer. In this case, T_{xx} is constant and equation (7.77) will be:

$$S\frac{\partial h}{\partial t} = T\frac{\partial^2 h}{\partial x^2} \quad . \tag{7.79}$$

Boundary conditions are:

$$h = h_o \text{ at } x = O,\ t>O \ , \tag{7.80}$$

$$h = h_o/2 \text{ at } x = L;\ t>O \ , \tag{7.81}$$

and

$$h = h_o \ \forall x \text{ at } t = O \ . \tag{7.82}$$

The analytical solution of this problem is [Wilson et al, 1979] given by:

$$h = h_o - \tfrac{1}{2}h_o \frac{x}{L} + \frac{h_o}{\pi} \sum_{i=1}^{\infty} \frac{(-1)^{i-1}}{i} \exp\left(-\frac{T\, i^2 \pi^2 t}{SL^2}\right) \sin\left(\frac{i\pi x}{L}\right). \qquad (7.83)$$

Problem 7.5

Consider a confined aquifer of dimensions 10,000 ,100m. The thickness of the aquifer is 20m. The transmissivity T=20,000 m^2/day and storage coefficient S=.001. Initially the head is assumed to be 95 m at all points. Then at x=10,000, the head is lowered to 90m.

Thus, the problem is mathematically:

$$\text{P.D.E.:} \quad \frac{S}{T}\frac{\partial h}{\partial t} = \frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} \qquad (7.84)$$

$$h = 95 \;\forall\; x \text{ at } t = 0\ , \qquad (7.85)$$

$$h = 95 \text{ at } x = 0,\ t > 0\ , \qquad (7.86)$$

$$h = 90 \text{ at } x = 10{,}000,\ t > 0\ , \qquad (7.87)$$

$$\text{and} \quad \frac{\partial h}{\partial y} = 0 \text{ at } y = 0 \text{ and } y = 100\ . \qquad (7.88)$$

The final and initial triangulations are similar to those in Problem 7.1. The function of time interval is designed so that the initial time steps are small enough and, then, increased gradually. The following function gives good results for the considered problem:

```
      DOUBLE PRECISION FUNCTION FUNT(T)
      Double Precision T
      IF(T.EQ.0.) goto 10
      IF(T.LT.1) goto 20
      IF(T.GE.1.AND.T.LT.2) goto 30
      IF(T.GE.2.AND.T.LT.10) goto 40
      IF(T.GE.10.) goto 50
10       FUNT=0.0005
         return
20       FUNT=.005
         return
30       FUNT=.05
         return
40       FUNT=.5
         return
50       FUNT=10
         return
         end
```

If a Fortran 77 compiler is available to a user, this function can be rewritten in an improved form using IF-THEN-ELSE constructs.

The time step according to equation (7.78) is dependent on the number of elements in the final triangulation. It is noticed that the final triangulation does not contain any vertices of intermediate triangles along the side in the y direction (100m length) if NTF is ≤ 64 triangles. Thus, in these runs equation (7.78) gives a time step of $\leq .05$ days. It is clear that the value returned by the function FUNT is on the safe side for a wide range of practical problems.

The obtained results are in good agreement with those evaluated by (7.83). The steady state situation is reached (approximately) after around 1 day. Note that in order to have a reasonable size of printed outputs a large spacing of Hx and Hy and small values of Nx and Ny are used. Some of these values are shown in Table 7.5.

X	Y	h
0.0	0.0	0.95000D+02
0.25000D+04	0.0	0.94910D+02
0.50000D+04	0.0	0.94151D+02
0.75000D+04	0.0	0.92421D+02
0.10000D+05	0.0	0.90000D+02
0.0	0.50000D+03	0.95000D+02
0.25000D+04	0.50000D+03	0.94909D+02
0.5000CD+04	0.50000D+03	0.94155D+02
0.75000D+04	0.50000D+03	0.92720D+02
0.10000D+05	0.50000D+03	0.90000D+02
0.0	0.10000D+04	0.95000D+02
0.25000D+04	0.10000D+04	0.94908D+02
0.50000D+04	0.10000D+04	0.94170D+02
0.75000D+04	0.10000D+04	0.92420D+02
0.10000D+05	0.10000D+04	0.90000D+02

TABLE 7.5: Solution of Problem 7.5 at t=1

7.5.3 Unsteady Flow in an Unconfined Aquifer

The main problem when dealing with unconfined aquifers is the location of the phreatic surface at each time step. This free surface problem is extremely complex and, in fact, even for the simplest cases, no exact analytical solution is known. The available solutions are mostly numerical or empirical formula based on greater approximations. It should be noted that T_{xx} and T_{yy} are dependent on the head h, this makes the PDE non-linear. Thus, in solving unsteady or transient flow in phreatic aquifers attention must be paid not only to the time interval, but also, for the iterative procedure required at each time step to get the solution to converge at this particular time.

Problem 7.6

Consider a phreatic aquifer of 10,000×100m and thickness of 100m. The permeability $K=1\times10^{4}$ m/day while S=.1. The boundary conditions are: initial head = 100m at time = 0 along the short sides. No flow is assumed across the long sides. The head is maintained at 100m at one end while at the other end it was lowered to 50m. Approximate solution to this free surface problem is usually done by iterative procedures as explained later. As a rough approximation, the free surface is assumed to be a plane. The results obtained are, in fact, approximate and can be used in a qualitative sense rather than rigorous quantitative values. Some of the results are shown in Table 7.6.

X	Y	h
0.0	0.0	0.10000D+03
0.25000D+04	0.0	0.91785D+02
0.50000D+04	0.0	0.81814D+02
0.75000D+04	0.0	0.68611D+02
0.10000D+05	0.0	0.50000D+02
0.0	0.50000D+02	0.10000D+03
0.25000D+04	0.50000D+02	0.91785D+02
0.50000D+04	0.50000D+02	0.81814D+02
0.75000D+04	0.50000D+02	0.68611D+02
0.10000D+05	0.50000D+02	0.50000D+02
0.0	0.10000D+03	0.10000D+03
0.25000D+04	0.10000D+03	0.91785D+02
0.50000D+04	0.10000D+03	0.81814D+02
0.75000D+04	0.10000D+03	0.68611D+02
0.10000D+05	0.10000D+03	0.50000D+02

TABLE 7.6: Solution of Problem 7.6 at time t=3

7.6 FREE SURFACE PROBLEMS IN AQUIFER FLOW

In free surface problems, some part of the boundary of the considered region is not known a priori and the determination of which is part of the solution. This occurs, for example, in phreatic aquifers where the upper surface of the aquifer is a free surface. Its position is not known precisely beforehand, all that is known is the boundary conditions that must be satisfied on the free surface. There are very few analytical solutions for the time-varying location of the phreatic surface in a phreatic aquifer. In fact, the available solutions are either exact solutions of gross physical approximations, or approximate numerical solutions of more realistic physical approximations. Standard FEM cannot be applied directly to solve free surface flow problems since these methods deal with known domains of fixed boundaries. Therefore, iterative procedures are used where a tentative position of the free surface is assumed, the problem is solved and the boundary conditions on the free surface are checked. This cycle is repeated until the boundary conditions are satisfied within a predetermined tolerance. This method is called the trial free boundary method. It can be implemented in different ways as will be explained later. Another approach is based on variational inequalities. In this section, the trial free boundary method is employed for the solution of a seepage through an earth dam problem. However, the same ideas apply to other similar problems like sluice gate flows and flows over weirs. The following example [Connor and Brebbia, 1976] gives a typical free surface groundwater problem. The considered problem is shown schematically in Figure (7.10) which represents seepage through a porous media.

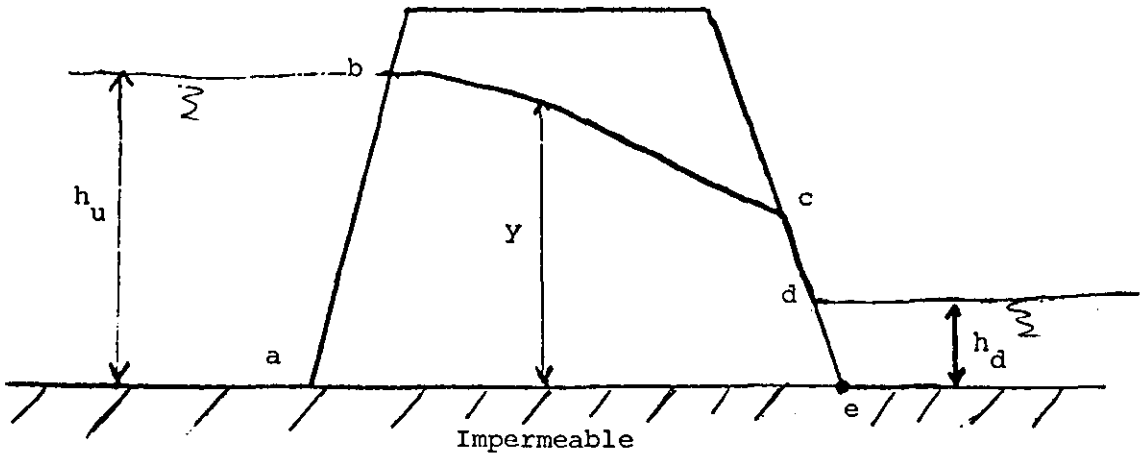


FIGURE 7.10: The dam problem

In this problem, the boundary conditions are as follows:

(i) on the line ab which is a water boundary, the potential head is h_u, while on the down stream side, the potential head is h_d on the water boundary de, i.e. $h=h_u$ or $=h_d$ on ab and de.

(ii) on the free surface, bc, the potential head is the elevation y itself since there is no pressure on this phreatic surface (i.e. atmospheric pressure only). Also no water seeps out from the free surface i.e. $\frac{\partial h}{\partial n} = 0$.

(iii) the seepage face, cd is a boundary where the water seeps out of the soil into air, i.e. $h = y$.

(iv) the impervious boundary, ae is a boundary where $\frac{\partial h}{\partial n} = 0$ as usual.

The governing equation in the steady case for the dam problems is Laplace. The main difficulty is that the location of the line bc is not known in advance and its position must be determined as part of the solution. However, in the FEM the domain must be known beforehand. To overcome this difficulty using the trial free boundary method several

strategies have been proposed. It is possible to classify them in the following categories:

(i) Variable domain method:

In this method a tentative location is assumed for the free surface. The problem is then solved. The computed heads on the free surface are compared to their elevation. Generally, there will be some difference. In the next step, the free surface is moved, according to a specific strategy. The problem is resolved again and the whole process is repeated until the difference between the computed heads on the free surface and the elevations are small enough whence the problem is solved. Several strategies for moving the free surface have been proposed. For example, the free surface is moved according to the computed heads. The main drawback of the variable domain method of locating the free surface is that the domain is always changing from one iteration to another, and thus, the computation of many of the element matrices are repeated in every cycle. The global matrix, of course, is reformulated everytime.

(ii) Variable element properties:

In order to avoid the changing of the FE mesh every cycle of iteration as in the variable domain method, the elements are assumed fixed and the "critical" elements properties are changed in order to satisfy the free surface conditions.

(iii) Variable boundary conditions:

In this approach the FE mesh is fixed and the solution is performed on the assumed position of the free surface as before. However, the boundary conditions on the assumed free surface are changed according to the actual computed values. In other words, since the position of

the free surface is unknown within the elements on the free boundary, it is possible to express the boundary conditions on the assumed free surface in terms of the actual free surface [France, 1975].

In order to solve a free surface problem using the TWODEPEP software, we must notice that this package is not designed to handle free surface problems. Moreover, it seems that there is no general software available for general free surface problems. However, it is possible to set a strategy that will, hopefully, minimize the inconveniences in solving free surface problems by software which is primarily designed for fixed domain applications. The steps can be summarized as follows:

(i) Perform the initial triangulation as usual but locate some (3 to 5) initial nodes on the assumed free surface.

(ii) Write the problem description as usual.

(iii) Solve the problem, obtain the outputs for the head at the free surface.

(iv) Modify the problem description by considering the computed values of the free surface. Thus, only parts of the VXY array will be changed. All other parameters will be unchanged.

(v) The procedure is repeated until the relative difference between the computed head and elevation is $\leq .1\%$ (say).

Problem 7.7

This problem represents a rectangular dam as shown in Figure (7.11). The soil is assumed to be homogeneous. This problem has been solved by several researchers by different methods. Thatcher and Askew [1982] used a complementary energy approach based on a stream function formulation, while Aitchison [1977] used a variational inequality solution.

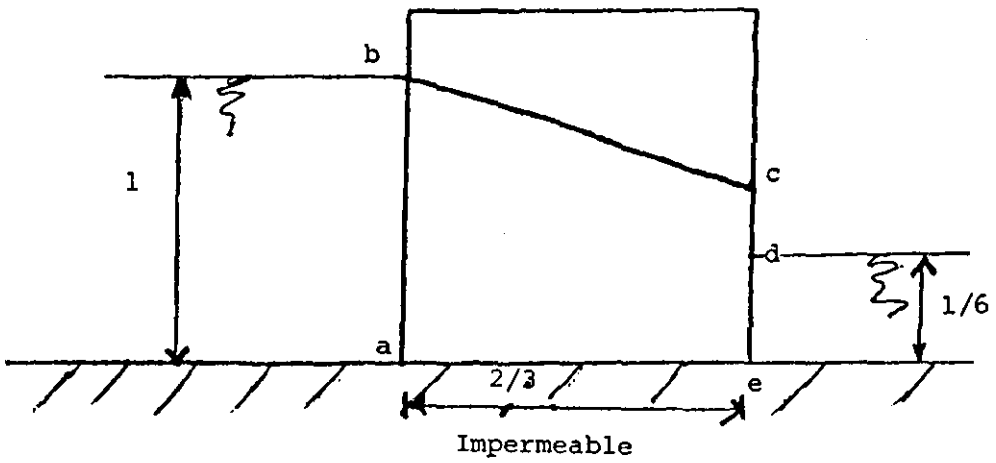


FIGURE 7.11: Schematic of Problem 7.7.

The modelling of this problem is shown in Figure (7.12).

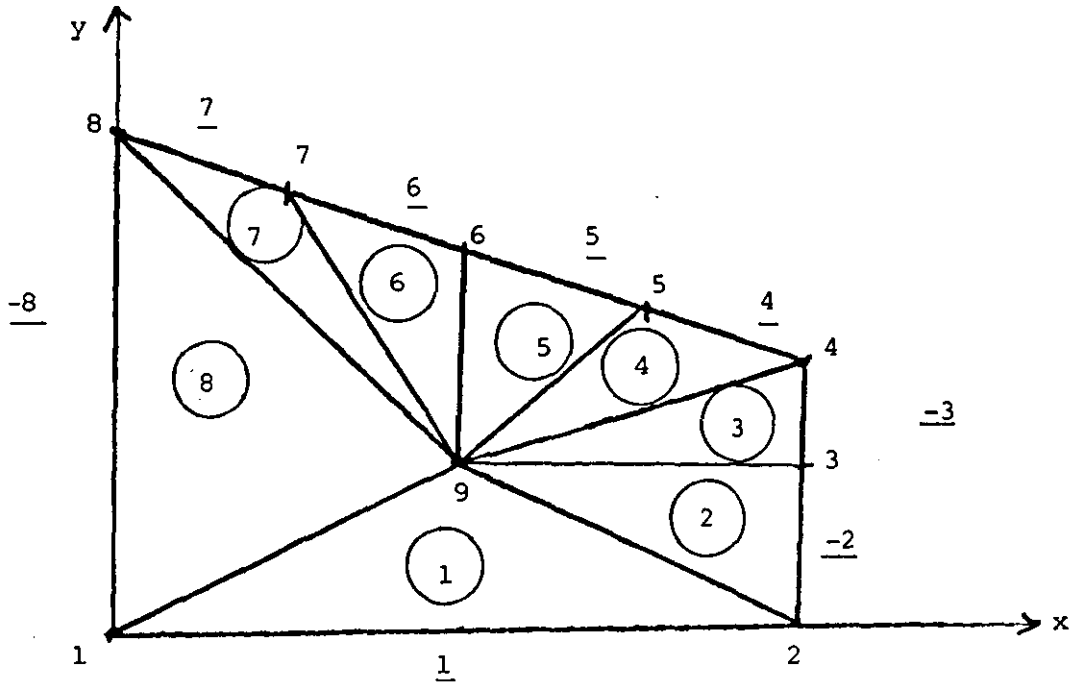


FIGURE 7.12: Nodes and elements for Problem 7.7

Note that initially, the free surface is assumed to be linear, node number 4 is assumed to be (2/3,.5). The condition of no seepage across the free surface is satisfied in this triangulation while the other condition of h=y is to be checked. The program of this problem is as follows:

```
        1       8       9      60       1
****    Classical dam problem
****    A free surface problem
****    Solved iteratively by the trial free boundary method
OXX         UX
OXY         UY
PLOT        1
XA          0.0
XH          .166667
NX          4
YA          0.0
HY          .125
NY          8
D3EST       1.0/(1+(X-.6667)**2+(y-.5)**2)
VXY     0.0,0.0      .66667,0.0    .666667,.166667      .666667,.5
VXY     ,.5,.625     .333334,.75   .166667,.875     0,1     .3333,.5
IABC    1  2  9     2  3  9       3  4  9        4  5  9    5  6  9
I          1           -2            -3              4          5
IABC    ,6  7  9     7  8  9       8  1  9
I          ,6           7            -8
ARC=-2
FB1     .166667
ARC=-3
FB1      Y
ARC=-8
FB1      1
END.
```

Figures (7.13) and (7.14) give the initial and final triangulations.

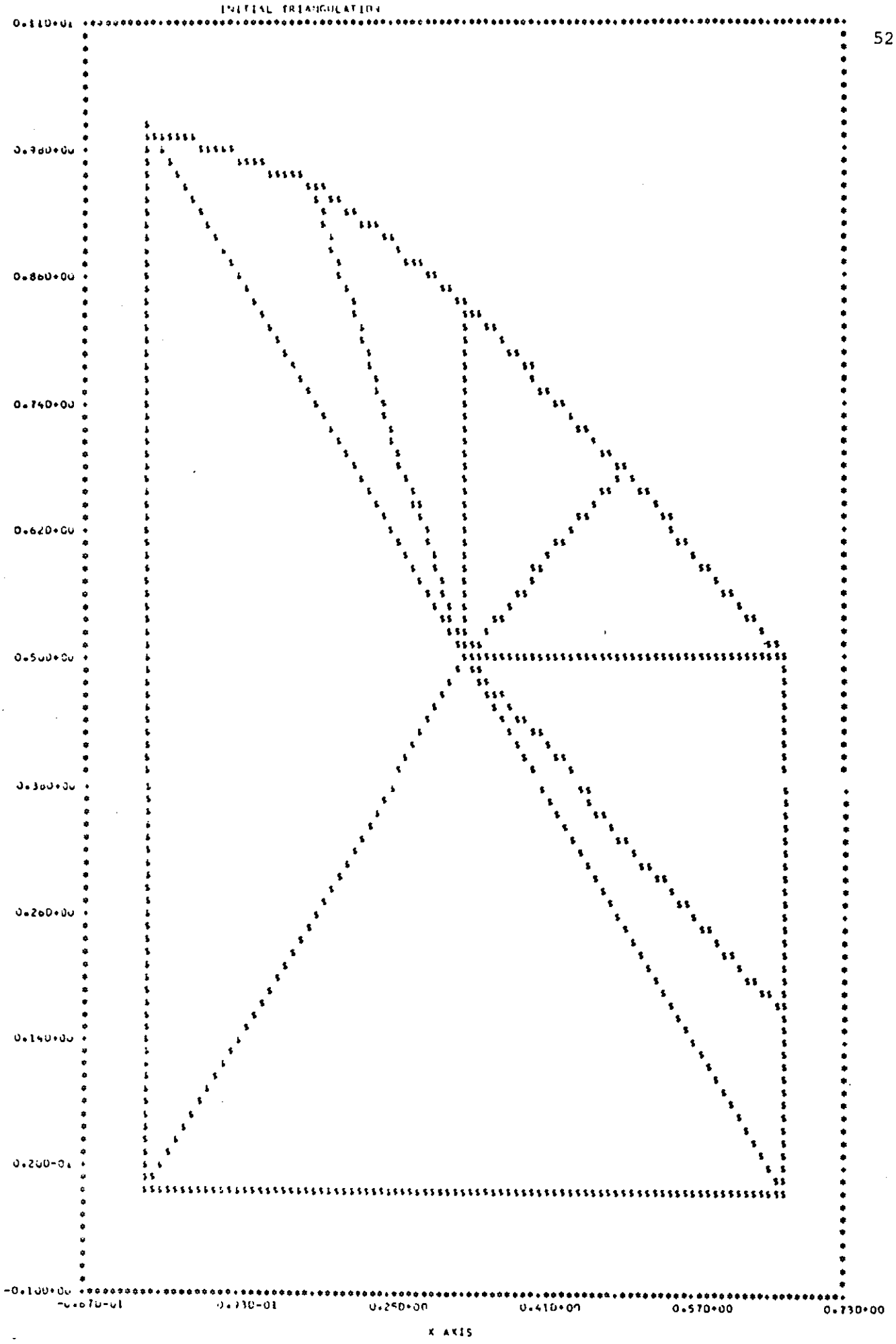


FIGURE 7.13: Initial triangulation for Problem 7.7

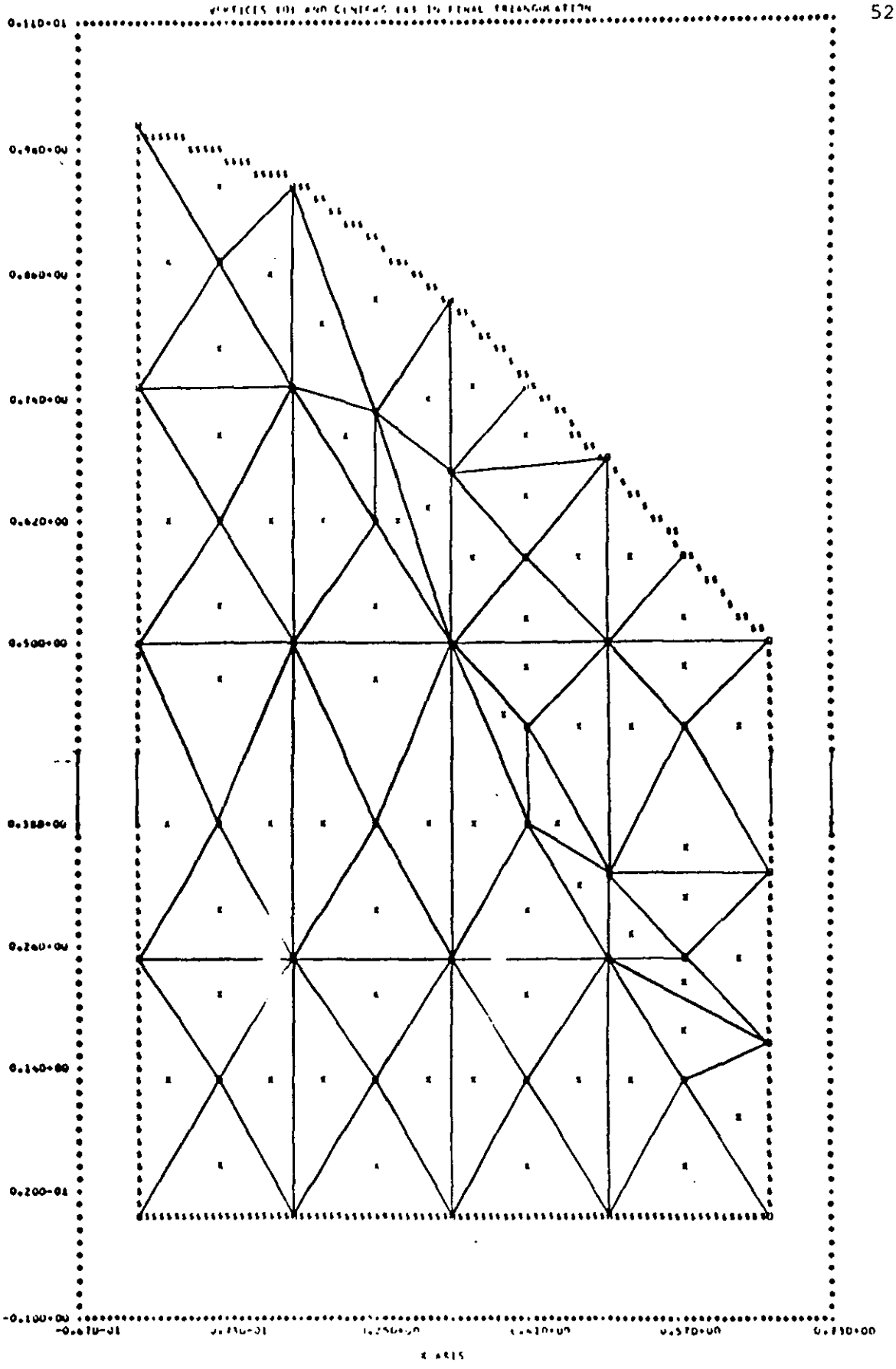


FIGURE 7.14: Final triangulation for Problem 7.7

'The results after the first iteration at these particular points: 4,5,6 and 7 are shown in Table 7.7.

Point	Assumed head	Computed head
4	.50	.5
5	.625	.67571
6	.75	.83666
7	.875	.94950

TABLE 7.7: Computed heads after 1 iteration

Now the problem is resolved using the new coordinates for these points in the VXY array. Note that in the 2nd and later iterations, the head at these points is computed by interpolation since the outputs specified by the XA, HX, NX, YA, HY and NY parameters will not coincide exactly on these points. Table 7.8 shows the computed head after the 2nd iteration. The steady state solution is arrived at after 5 iterations

Point	Assumed head	Computed head
4	.50	.52
5	.67571	.69305
6	.83666	.86279
7	.94950	.94347

TABLE 7.8: Computed heads after 2 iterations

where the final results are shown in Table 7.9. A tolerance of .1% is allowed.

Comparing these results with those by Thatcher and Askew [1982] and Aitchison [1977] shows good agreement. It should be noticed that the final triangulation in this problem is done most dense in the vicinity

Point	Head	Thatcher solution	Aitchison solution
4	.53	.5382	.5356
5	.72	.7312	.7290
6	.86	.854	.8515
7	.95	.944	.9412

TABLE 7.9: Solution of Problem 7.7

of the seepage face. This is realised by the function D3EST defined by:

$$D3EST = 1/[1+(x-\frac{2}{3})^2 + (y-.5)^2] \qquad (7.89)$$

Another important point is that the determination of the seepage point, i.e. point c or node 4. Since at all times, the condition of free surface will be automatically realised at this point. The most known satisfactory algorithm is to move this node according to the moving of other related nodes on the free surface.

Problem 7.8

Consider a radial flow to a well in an unconfined aquifer. The well is completely penetrating the aquifer. The water level in the well is maintained, by pumping, at 20m above the impermeable bed. The permeability of the aquifer is k=1,000 m/day. The assumed radius of influence of the well is 100m above the impermeable bed. The problem is schematically drawn in Figure (7.15). Such problems can be approximated utilizing the Dupuit assumptions and an analytical solution can be found. However, in this case, the seepage face bc is neglected and the free surface is assumed to be db rather than dc. The obtained solutions can be accepted far from the well, but near the well, the approximation is poor. This problem is solved using the same strategy used in the previous problem.

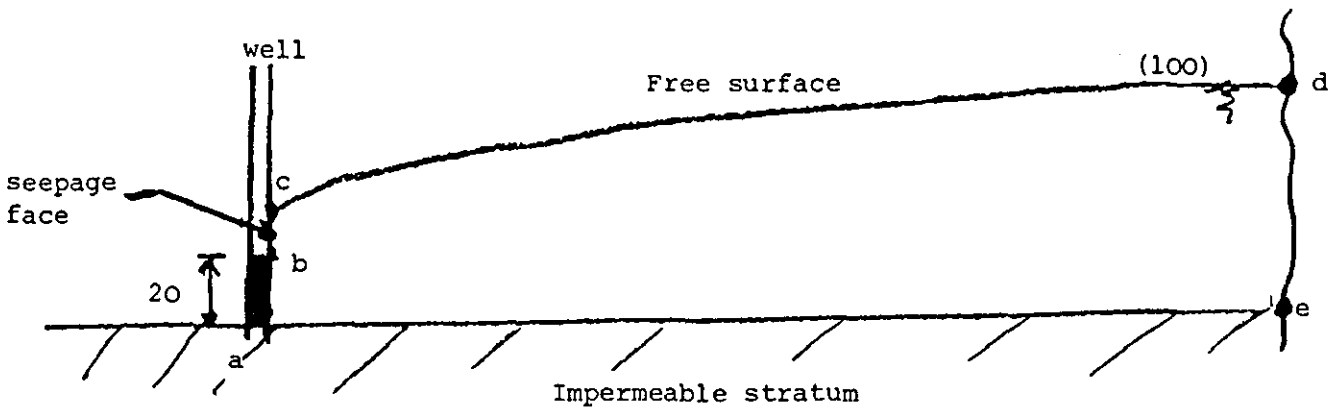


FIGURE 7.15: Schematic of Problem 7.8

The FE modelling is shown in Figure (7.16) where 15 nodes and 14 elements are done. The function D3EST is defined in this example so as to produce

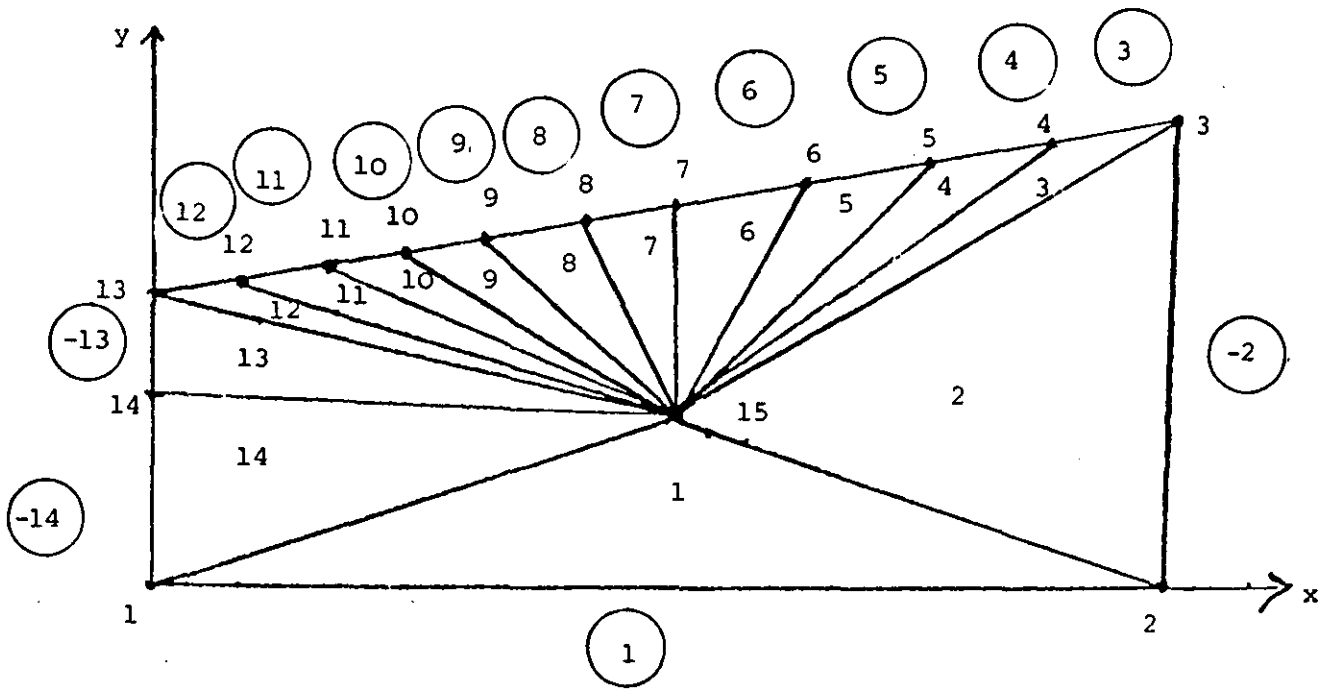


FIGURE 7.16: FE modelling of Problem 7.8

the most dense triangulation in the vicinity of the well. The function used is

$$D3EST = 1/[x^2 + (y-20)^2 + 1] \tag{7.90}$$

The program for this problem is as follows:

```
        1     14     15     250     1
****    flow towards a well in phreatic aquifer
****    a free surface problem
****    solved iteratively by the trial free boundary method
OXX       UX
OXY       UY
PLOT      1
XA        0.0
HX        100
NX        10
YA        0.0
HY        10
NY        10
D3EST     1.0/(x*x+(y-20)**2+1)
ARC=-2
FB1       100
ARC=-13
FB1       Y
ARC=-14
FB1       20
VXY       0.,0.      1000.,0.0      1000.,100.     900,95.    800.,90.
VXY     ,700.,85.    600.,80.       500.,75.       400.,70.   300.,65.
VXY     ,200.,60.    100.,55.       0.0,50.        0.0,20.0   500.,20.
IABC    1  2  15     2  3  15       3  4  15       4  5  15   5  6  15
I         1            -2             3              4          5
IABC    ,6  7  15     7  8  15       8  9  15    9  10  15  10  11. 15  11 12 1
I         ,6           7              8              9          10         11
IABC    ,12  13  15   13  14  15     14  1  15
I         ,12          -13            -14
END.
```

The initial and final triangulations for this problem are shown in Figures (7.17) and (7.18). After the first iteration, the solution obtained is as shown in Table 7.10.

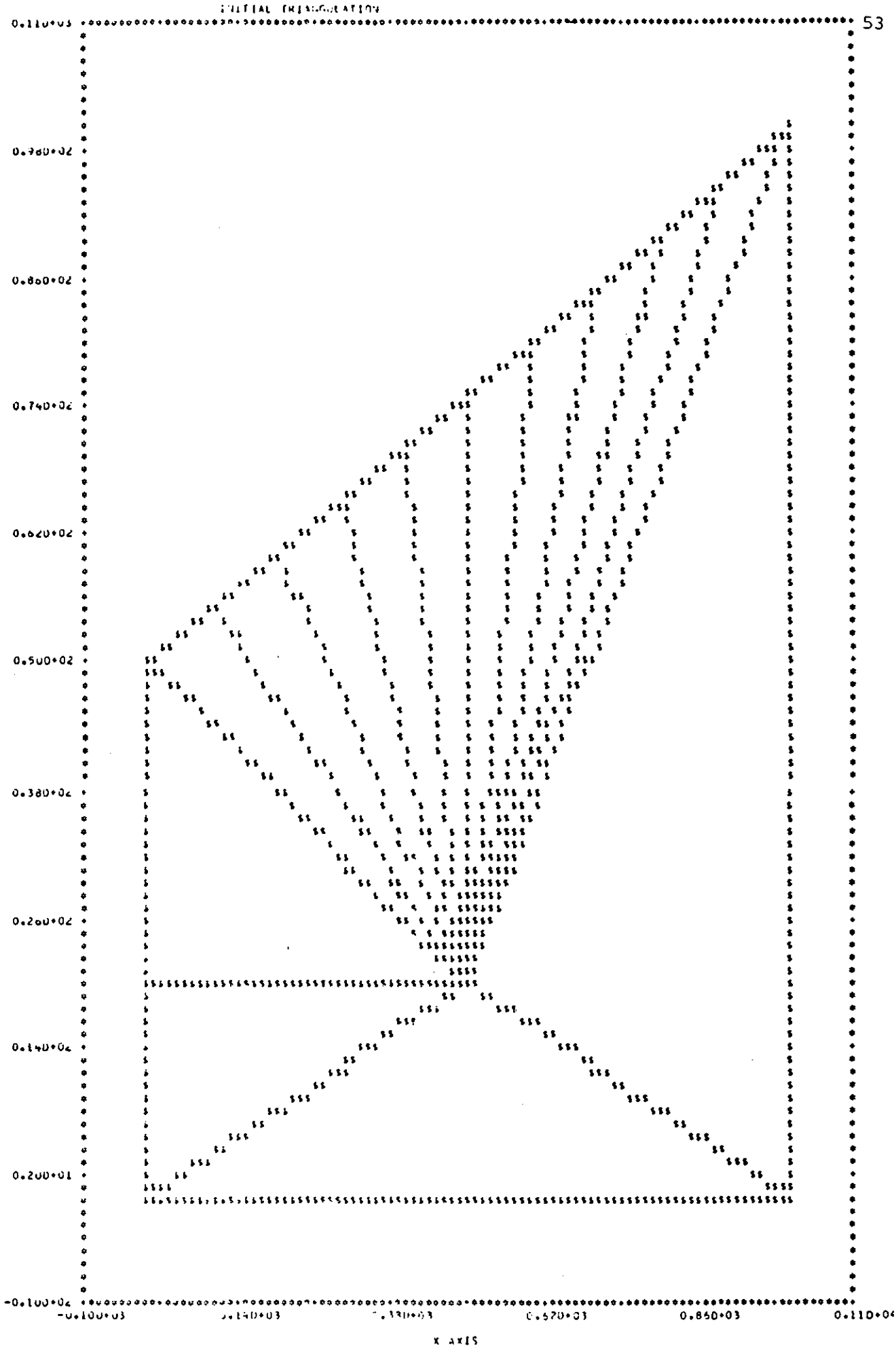


FIGURE 7.17: Initial triangulation for Problem 7.8

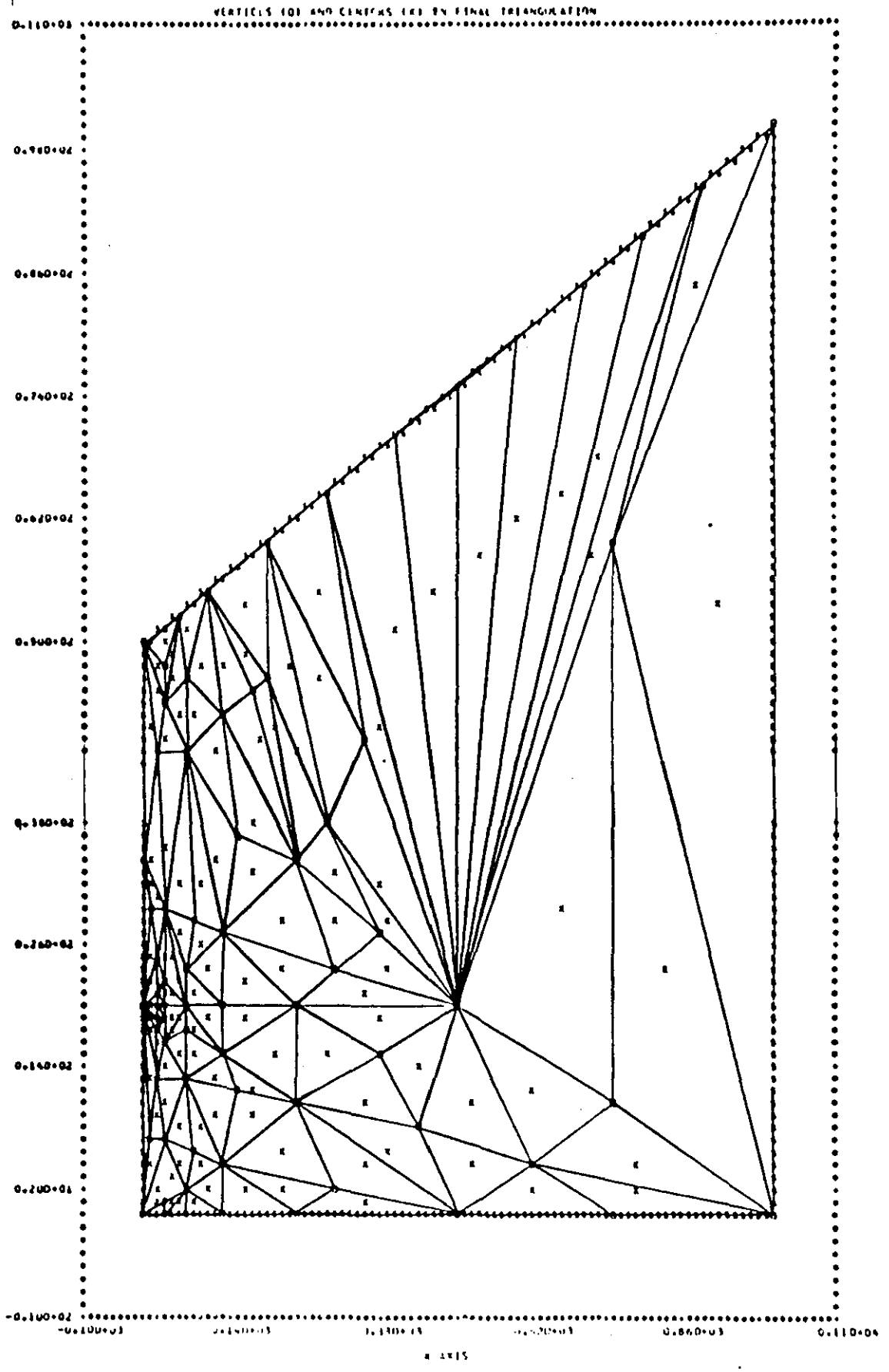


FIGURE 7.18: Final triangulation for Problem 7.8

X	Y	h
0.0	0.0	0.200000D+02
0.100000D+03	0.0	0.389010D+02
0.200000D+03	0.0	0.478160D+02
0.300000D+03	0.0	0.559970D+02
0.400000D+03	0.0	0.635820D+02
0.500000D+03	0.0	0.705920D+02
0.600000D+03	0.0	0.771550D+02
0.700000D+03	0.0	0.833620D+02
0.800000D+03	0.0	0.892260D+02
0.900000D+03	0.0	0.947740D+02
0.100000D+04	0.0	0.100000D+03
0.0	0.100000D+02	0.200000D+02
0.100000D+03	0.100000D+02	0.389050D+02
0.200000D+03	0.100000D+02	0.478210D+02
0.300000D+03	0.100000D+02	0.560010D+02
0.400000D+03	0.100000D+02	0.635850D+02
0.500000D+03	0.100000D+02	0.705920D+02
0.600000D+03	0.100000D+02	0.771600D+02
0.700000D+03	0.100000D+02	0.833630D+02
0.800000D+03	0.100000D+02	0.892270D+02
0.900000D+03	0.100000D+02	0.947700D+02
0.100000D+04	0.100000D+02	0.100000D+03
0.0	0.200000D+02	0.200000D+02
0.100000D+03	0.200000D+02	0.389550D+02
0.200000D+03	0.200000D+02	0.478320D+02
0.300000D+03	0.200000D+02	0.560100D+02
0.400000D+03	0.200000D+02	0.635900D+02
0.500000D+03	0.200000D+02	0.706010D+02
0.600000D+03	0.200000D+02	0.771660D+02
0.700000D+03	0.200000D+02	0.833720D+02
0.800000D+03	0.200000D+02	0.892310D+02
0.900000D+03	0.200000D+02	0.947690D+02
0.100000D+04	0.200000D+02	0.100000D+03
0.0	0.300000D+02	0.300000D+02
0.100000D+03	0.300000D+02	0.390110D+02
0.200000D+03	0.300000D+02	0.478500D+02
0.300000D+03	0.300000D+02	0.560270D+02
0.400000D+03	0.300000D+02	0.636020D+02
0.500000D+03	0.300000D+02	0.706130D+02
0.600000D+03	0.300000D+02	0.771770D+02
0.700000D+03	0.300000D+02	0.833330D+02
0.800000D+03	0.300000D+02	0.892390D+02
0.900000D+03	0.300000D+02	0.947760D+02
0.100000D+04	0.300000D+02	0.100000D+03
0.0	0.400000D+02	0.400000D+02
0.100000D+03	0.400000D+02	0.390700D+02
0.200000D+03	0.400000D+02	0.478750D+02
0.300000D+03	0.400000D+02	0.560490D+02
0.400000D+03	0.400000D+02	0.636190D+02
0.500000D+03	0.400000D+02	0.706290D+02
0.600000D+03	0.400000D+02	0.771920D+02
0.700000D+03	0.400000D+02	0.833970D+02
0.800000D+03	0.400000D+02	0.892500D+02
0.900000D+03	0.400000D+02	0.947860D+02

TABLE 7.10: Solution to Problem 7.8, first iteration

7.7 MISCELLANEOUS PROBLEMS IN GROUNDWATER FLOW

Here some problems in groundwater flow are solved utilizing the FEM. In Problem 7.9, a small watershed is considered. Problems 7.10 and 7.11 are concerned with radial flow towards a well in a confined aquifer, without leakage and with leakage, respectively. Finally, in Problem 7.12 a case study of the Yun-Lin aquifer on the island of Taiwan is considered.

7.7.1 Problem 7.9: Small watershed

A small watershed is bounded on three sides by a no flow condition, while the upper bound is approximated by a horizontal line with the head = .02x+100. The problem is modelled in Figure (7.19).

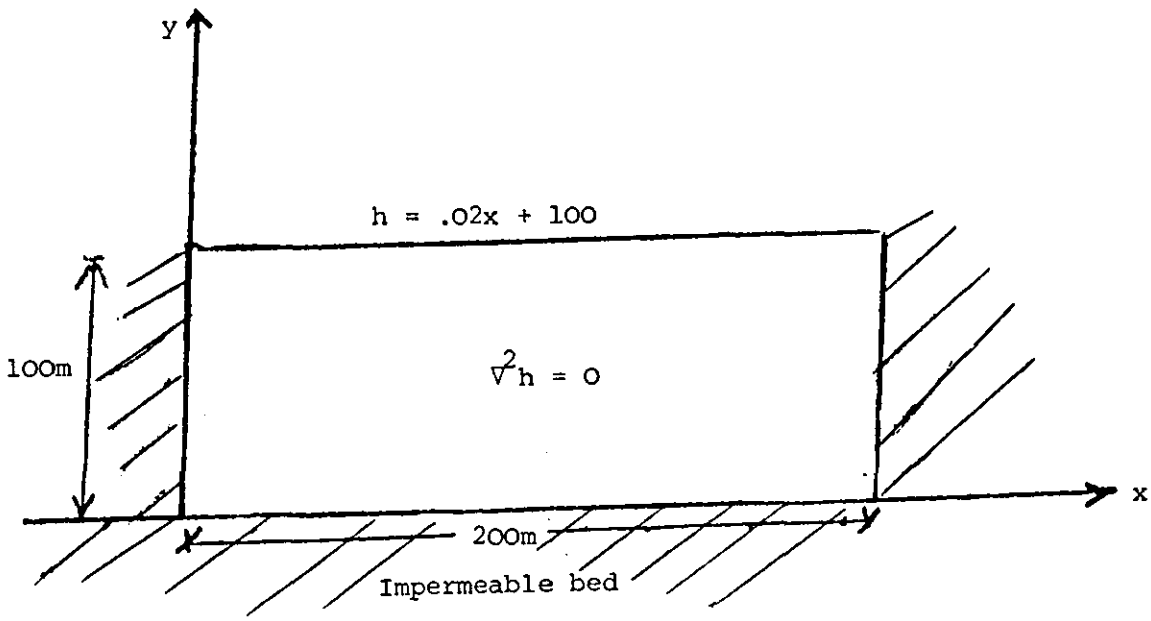


FIGURE 7.19: FE modelling of Problem 7.9

In fact, this problem has been introduced by Wang and Anderson [1982] where it was solved by the FDM. This problem is solved here by the FEM and the results are compared with their solution. The initial and final triangulations for this problem are shown in Figures (7.20) and (7.21), respectively. The results are given in Table 7.11.

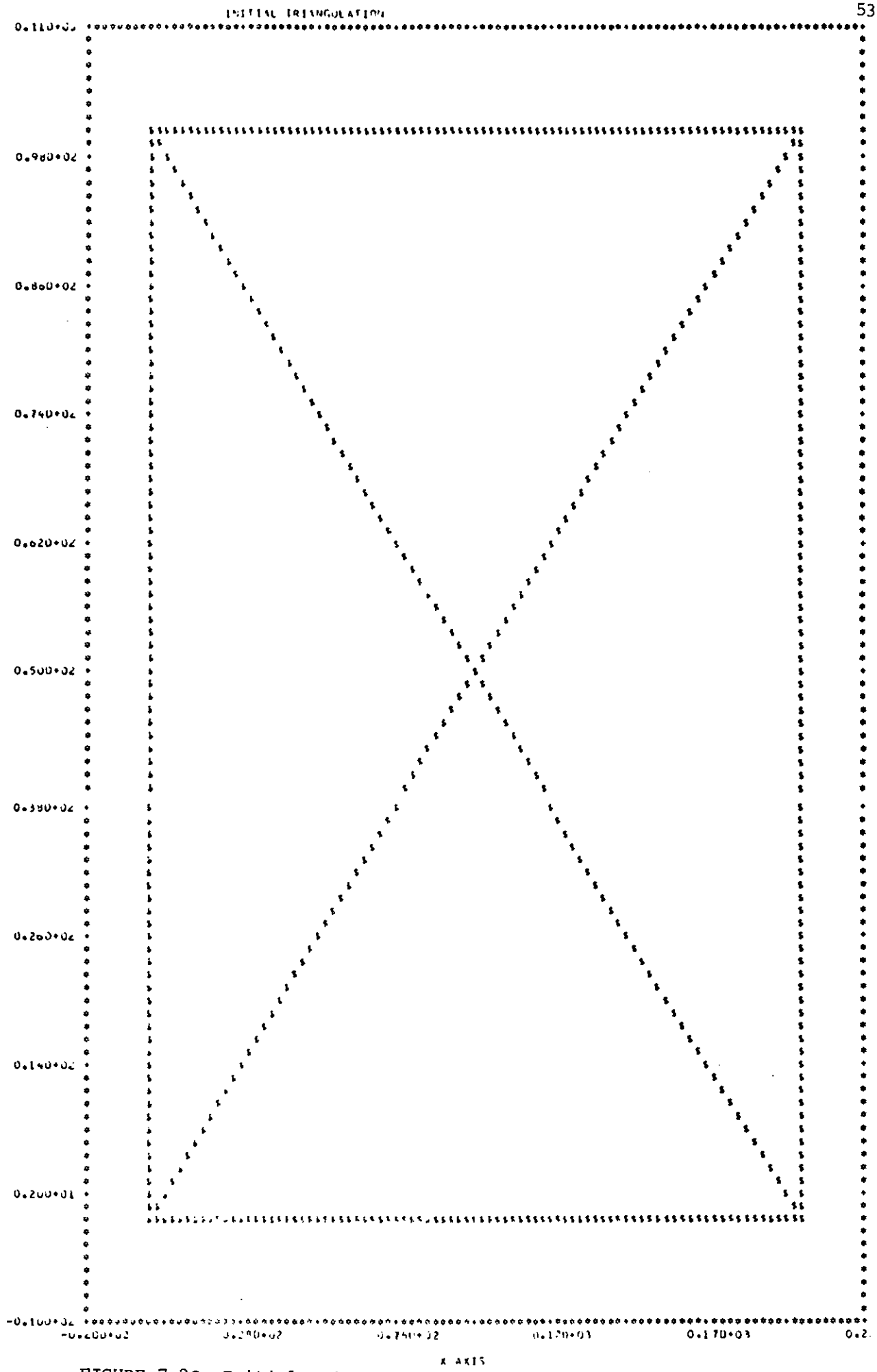


FIGURE 7.20: Initial triangulation for Problem 7.9

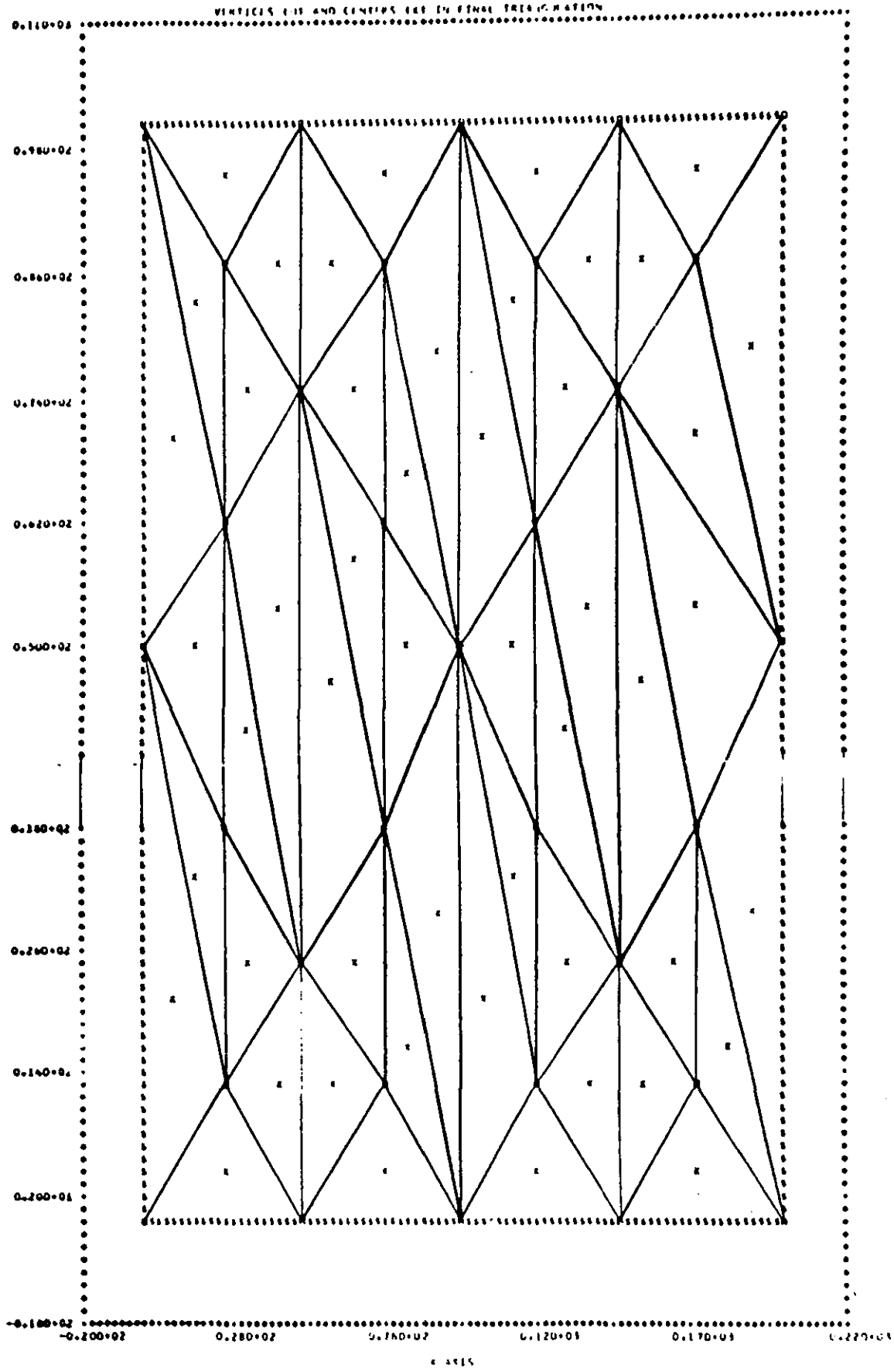


FIGURE 7.21: Final triangulation for Problem 7.9

X	Y	h
0.0	0.0	0.101350+03
0.200000+02	0.0	0.101380+03
0.400000+02	0.0	0.101480+03
0.600000+02	0.0	0.101630+03
0.800000+02	0.0	0.101800+03
0.100000+03	0.0	0.102000+03
0.120000+03	0.0	0.102200+03
0.140000+03	0.0	0.102370+03
0.160000+03	0.0	0.102520+03
0.180000+03	0.0	0.102620+03
0.200000+03	0.0	0.102650+03
0.0	0.200000+02	0.101320+03
0.200000+02	0.200000+02	0.101350+03
0.400000+02	0.200000+02	0.101450+03
0.600000+02	0.200000+02	0.101610+03
0.800000+02	0.200000+02	0.101790+03
0.100000+03	0.200000+02	0.102000+03
0.120000+03	0.200000+02	0.102210+03
0.140000+03	0.200000+02	0.102390+03
0.160000+03	0.200000+02	0.102550+03
0.180000+03	0.200000+02	0.102650+03
0.200000+03	0.200000+02	0.102680+03
0.0	0.400000+02	0.101210+03
0.200000+02	0.400000+02	0.101250+03
0.400000+02	0.400000+02	0.101380+03
0.600000+02	0.400000+02	0.101560+03
0.800000+02	0.400000+02	0.101770+03
0.100000+03	0.400000+02	0.102000+03
0.120000+03	0.400000+02	0.102230+03
0.140000+03	0.400000+02	0.102450+03
0.160000+03	0.400000+02	0.102630+03
0.180000+03	0.400000+02	0.102750+03
0.200000+03	0.400000+02	0.102790+03
0.0	0.600000+02	0.101040+03
0.200000+02	0.600000+02	0.101080+03
0.400000+02	0.600000+02	0.101240+03
0.600000+02	0.600000+02	0.101460+03
0.800000+02	0.600000+02	0.101730+03
0.10000D+03	0.600000+02	0.102000+03
0.120000+03	0.600000+02	0.102270+03
0.140000+03	0.600000+02	0.102530+03
0.160000+03	0.600000+02	0.102760+03
0.180000+03	0.600000+02	0.102920+03
0.200000+03	0.600000+02	0.102960+03
0.0	0.800000+02	0.100660+03
0.200000+02	0.800000+02	0.100800+03
0.400000+02	0.800000+02	0.101040+03
0.600000+02	0.800000+02	0.101340+03
0.800000+02	0.800000+02	0.101670+03
0.100000+03	0.800000+02	0.102000+03
0.120000+03	0.800000+02	0.102330+03
0.140000+03	0.800000+02	0.102660+03
0.160000+03	0.800000+02	0.102960+03
0.180000+03	0.800000+02	0.103190+03
0.200000+03	0.800000+02	0.103340+03
0.0	0.100000+03	0.100000+03
0.200000+02	0.100000+03	0.100400+03
0.400000+02	0.100000+03	0.100800+03
0.600000+02	0.100000+03	0.101200+03
0.800000+02	0.100000+03	0.101600+03
0.100000+03	0.100000+03	0.102000+03
0.120000+03	0.100000+03	0.102400+03
0.140000+03	0.100000+03	0.102800+03
0.160000+03	0.100000+03	0.103200+03
0.180000+03	0.100000+03	0.103600+03
0.200000+03	0.100000+03	0.104000+03

TABLE 7.11: Solution of Problem 7.9

It is obvious that these results are approximate in nature, since the problem is, in fact, a free surface one. Apart from a few points the results are generally in good agreement with those computed by the FDM given in the cited reference.

7.7.2 Problem 7.10: Transient well flow

In this problem, radial flow towards a well in a homogeneous confined aquifer is considered. The aquifer is assumed to extent to ∞. This problem is described by the following P.D.E.:

$$S \frac{\partial h}{\partial t} = T\left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2}\right) + Q \quad \forall\, x,y \tag{7.91}$$

Since this problem possesses radial symmetry, it is usually described by the following P.D.E. [Wilson et al, 1979]

$$S \frac{\partial h}{\partial t} = \frac{1}{r} \frac{\partial}{\partial r} \left(rT \frac{\partial h}{\partial r}\right) \tag{7.92}$$

or

$$\frac{S}{T} \frac{\partial h}{\partial t} = \frac{1}{r} \frac{\partial h}{\partial r} + \frac{\partial^2 h}{\partial r^2} \tag{7.93}$$

where r>0.

The initial condition is

At time t = 0, $h = h_o$ $\forall$ x,y , (7.94)

and the boundary conditions are

$$h = h_o \text{ as } r \to \infty,\ t > 0 , \tag{7.95}$$

and

$$\lim_{r \to 0} r \frac{\partial h}{\partial r} = \frac{Q}{2\pi T} \quad , t > 0 , \tag{7.96}$$

where Q is the well discharge.

The analytical solution to this problem is given by Theis [1935] as:

$$h = h_o - \frac{Q}{4\pi T} * W(u) \tag{7.97}$$

where:

$$u = \frac{r^2 S}{4Tt} \tag{7.98}$$

and the Theis well function W(u) is given by:

$$W(u) = \int_u^{\infty} \frac{e^{-v}}{v}\, dv \qquad (7.99)$$

The values of the Theis well function W(u) are tabulated in many textbooks on groundwater flow e.g. Bear [1979]. This equation relates the head at any radius r at any time t. The numerical values considered in our problem are as follows:

Aquifer transmissivity	$T = 1\times10^{5}\ m^2/day$
Aquifer storativity	$S = .001$
Aquifer thickness	$B = 100m$
Initial head	$h_o = 100m$
Well pumping rate	$Q = 1\times10^{6}\ m^3/day$
Radius of influence	$R = 10{,}000m$

Since this problem possesses radial symmetry, only one quadrant of the region needs to be considered in the FE modelling. The well is handled as stated earlier. The finite elements distribution is also based on the D3EST functions given before and finally the time interval is computed subjected to the constraints given in (7.5.1). The obtained results compare very well with those obtained by the Theis equations. The initial and final triangulations are shown in Figures (7.22) and (7.23).

7.7.3 Problem 7.11: Transient well flow with leakage

This problem is similar to Problem 7.10 but there is a leakage from an adjacent aquifer through a leaky layer. The P.D.E. is given by

$$S\frac{\partial h}{\partial t} = T\left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2}\right) + \frac{K'}{b'}(h-h_a) + Q\ , \qquad (7.100)$$

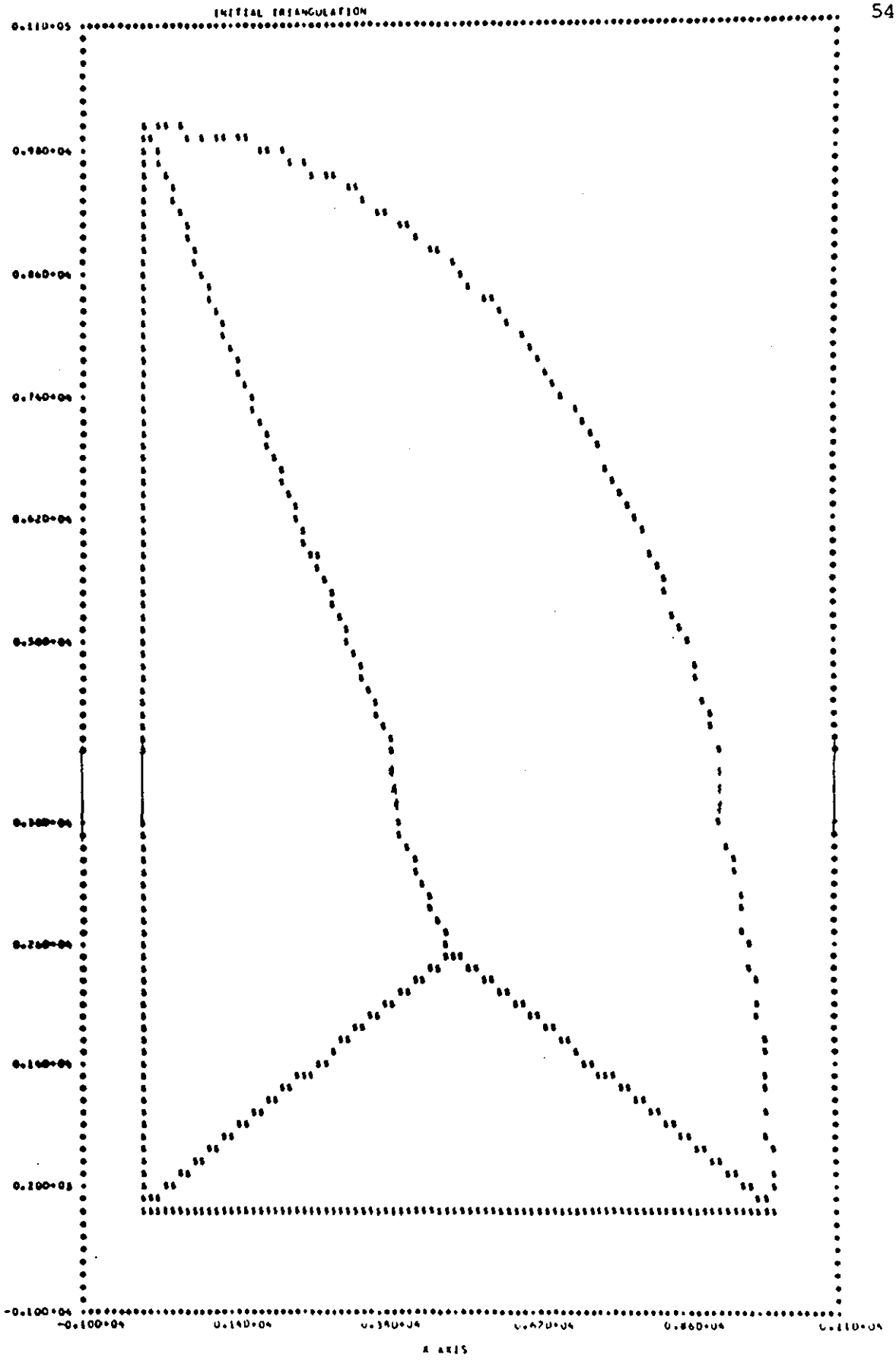


FIGURE 7.22: Initial triangulation for Problem 7.10

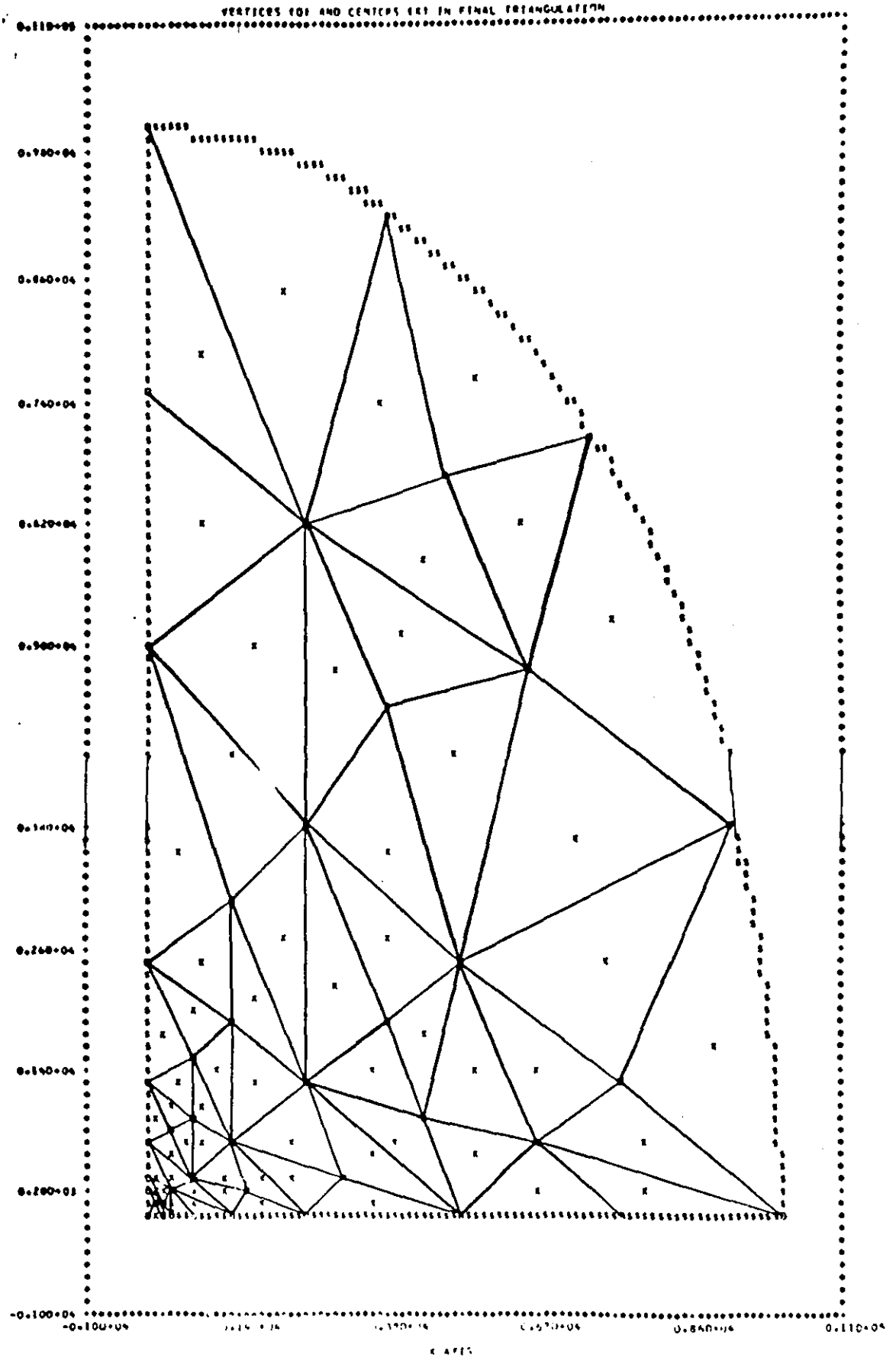


FIGURE 7.23: Final triangulation for Problem 7.10

where K' and b' are the permeability and the thickness of the semi-pervious layer. h_a is the piezometric head in an adjacent aquifer separated from the main aquifer by the semi-pervious layer. The boundary conditions are as in Problem 7.10. In the special case where $h_a=h_o$, Hantush and Jacob [1955] solved the problem analytically:

$$h = h_o - \frac{Q}{4\pi T} W(u,r/\beta) \tag{7.101}$$

where:

$$u = \frac{r^2 S}{4Tt} \tag{7.102}$$

and,

$$\beta = \sqrt{\frac{T}{K'/b'}} \tag{7.103}$$

The leaky well function $W(u,r/\beta)$ is given by

$$W(u,r/\beta) = \int_u^\infty \frac{1}{v} \exp(-v- \frac{r^2}{4\beta^2 v}) \, dv \tag{7.104}$$

The values of this function are also tabulated in several text books on groundwater.

The problem which is solved numerically is identical to that of Problem 7.10, but now the adjacent head h_a=95 and h_o =100m and the leakey layer parameter K'/b' = .1 day^{-1}. The obtained results by the FEM are in good agreement with those obtained by the Hantush equations.

7.7.4 Problem 7.12: Anisotropic aquifer flow

This problem is concerned with the analysis of the Yun-Lin aquifer on the island of Taiwan. This problem has been solved by Liggett and Liu [1983] using the boundary integral equation method. It is assumed that the aquifer is non-leakey and anisotropic, however, the global axes are selected to coincide with the principal axes of the transmissivity tensor. The aquifer is shown schematically in Figure (7.24)

which is essentially as given in Liggett and Liu [1983]. The properties of the different zones are given in Table 7.12.

Zone number	T_x	T_y
1	542.88	5428.8
2	112.3	1123
3	185.76	1857.6
4	764.64	7646.4
5	290.88	2908.8
6	69.12	691.2
7	120.96	1209.6

TABLE 7.12: Zones properties of Problem 7.12

Table 7.13 gives the coordinates of the different nodes in the aquifer measured in Km. A well is located at node 44 with discharge of 100 m^3/h, i.e. 2,400 m^3/day. The initial FE modelling is shown in Figure (7.25).

The boundary conditions are as follows:

(i) No flow condition along the sides:
1-2, 2-3, 3-4, 4-5, 5-6, 6-7, 7-8, 8-9,
14-15, 15-16, 16-17 and 17-18

(ii) Specified head on the remaining boundary. These specified heads are given in Table 7.14.

In order to express these boundary conditions in the TWODEPEP notation, they should be expressed as a function of (x,y). This is done assuming a linear variation of the head. The boundary conditions are thus expressed in the following equations where x and y are in Km.

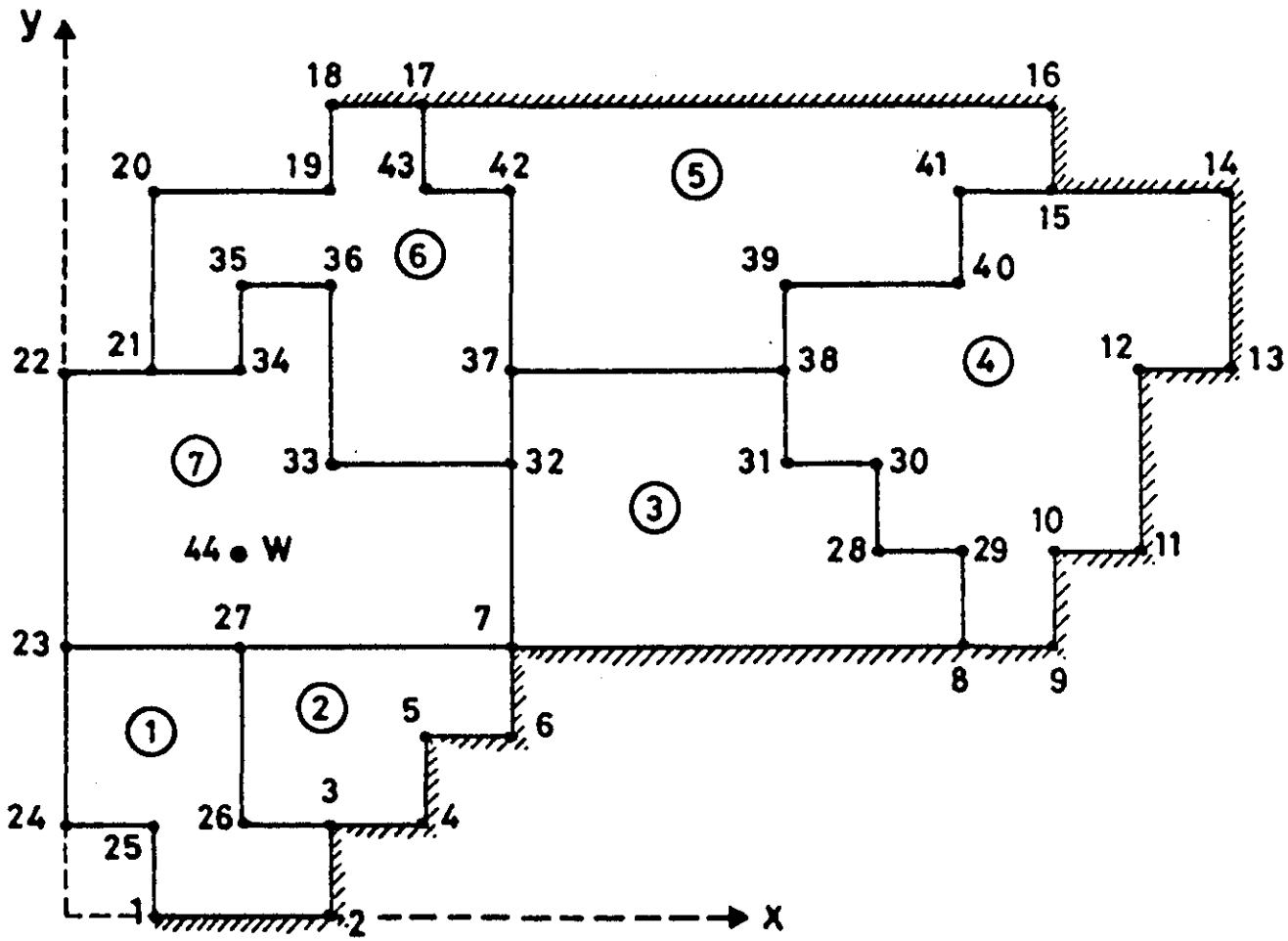


FIGURE 7.24: Schematic of Problem 7.12

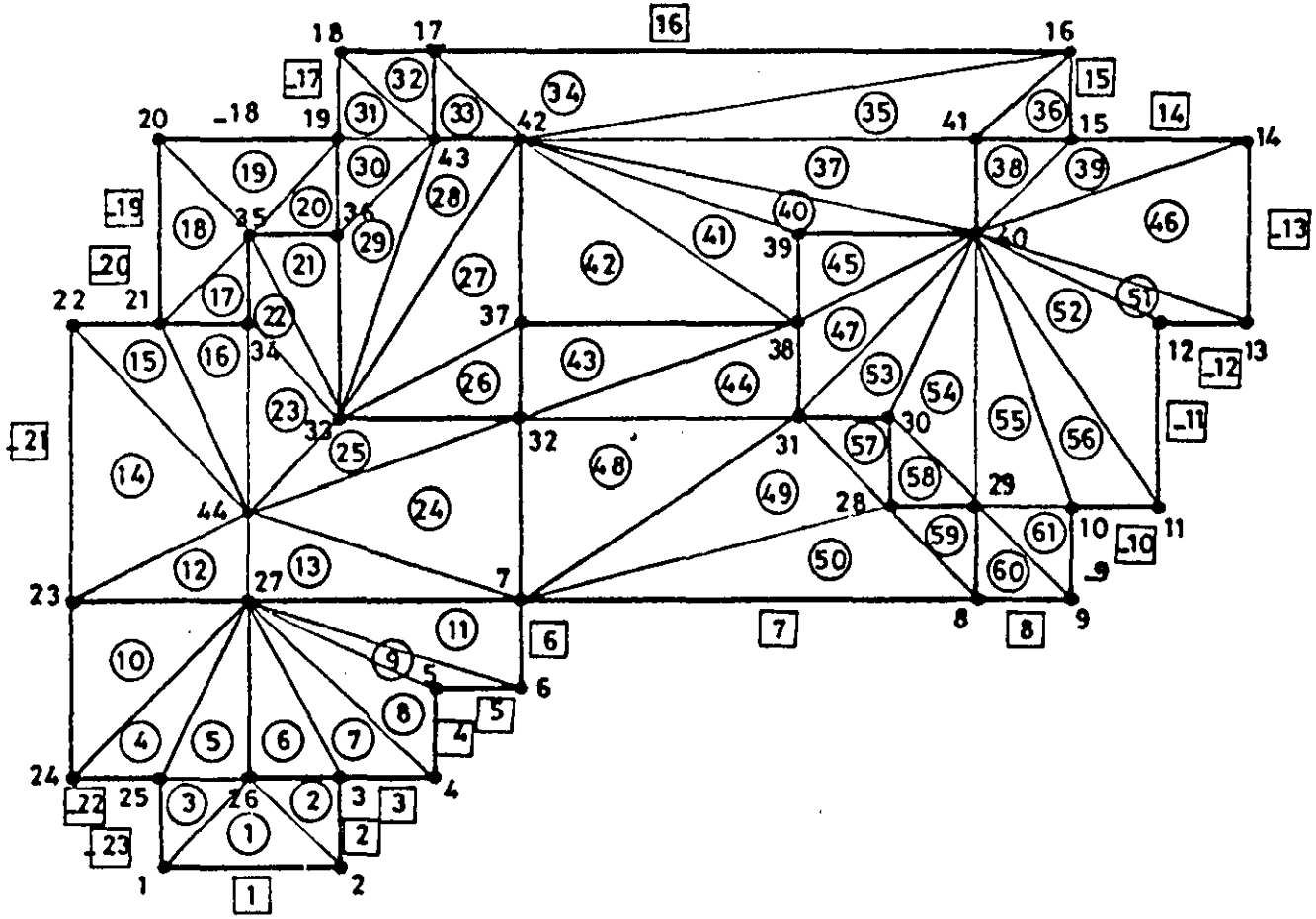


FIGURE 7.25: FE modelling of Problem 7.12

Node	X	Y	Node	X	Y
1	4	0	23	0	12
2	12	0	24	0	4
3	12	4	25	4	4
4	16	4	26	8	4
5	16	8	27	8	12
6	20	8	28	36	16
7	20	12	29	40	16
8	40	12	30	36	20
9	44	12	31	32	20
10	44	16	32	20	20
11	48	16	33	12	20
12	48	24	34	8	24
13	52	24	35	8	28
14	52	32	36	12	28
15	44	32	37	20	24
16	44	36	38	32	24
17	16	36	39	32	28
18	12	36	40	40	28
19	12	32	41	40	32
20	4	32	42	20	32
21	4	24	43	16	32
22	0	24	44	8	16

TABLE 7.13: Coordinates of the nodes of Problem 7.12.

Node No.	Head	Node No.	Head
1	1.5	18	3
9	60.28	19	2.8
10	60.0	20	0
11	70	21	0
12	57	22	0
13	59	23	0
14	55	24	0
		25	2

TABLE 7.14: Specified heads

(1) along the line 9-10 (i.e. arc=-9)

$$h = 60.28 - .07(y-12) \tag{7.105}$$

(2) along the line 10-11

$$h = 60 + 2.5(x-44) \tag{7.106}$$

(3) along the line 11-12

$$h = 70 - \frac{13}{8}(y-16) \tag{7.107}$$

(4) along the line 12-13

$$h = 57 + .5(X-48) \tag{7.108}$$

(5) along the line 13-14

$$h = 59 - .5(y-24) \tag{7.109}$$

(6) along the line 18-19

$$h = 3 + .05(y-36) \tag{7.110}$$

(7) along the line 19-20

$$h = \frac{2.8}{8}(x-4) \tag{7.111}$$

(8) along the lines 20-21, 21-22, 22-23 and 23-24

$$h = 0 \tag{7.112}$$

(9) along the line 24-25

$$h = .5x \tag{7.113}$$

(10) along the line 25-1

$$h = 1.5 + y/8 \tag{7.114}$$

Some of the important notes to be mentioned when modelling this problem are as follows:

(i) The coordinates must be entered in metres to have consistent units.

(ii) The triangulation near the well is more dense than other places by the definition of the D3EST function as explained earlier. The initial and final triangulations are given in Figures (7.26) and (7.27).

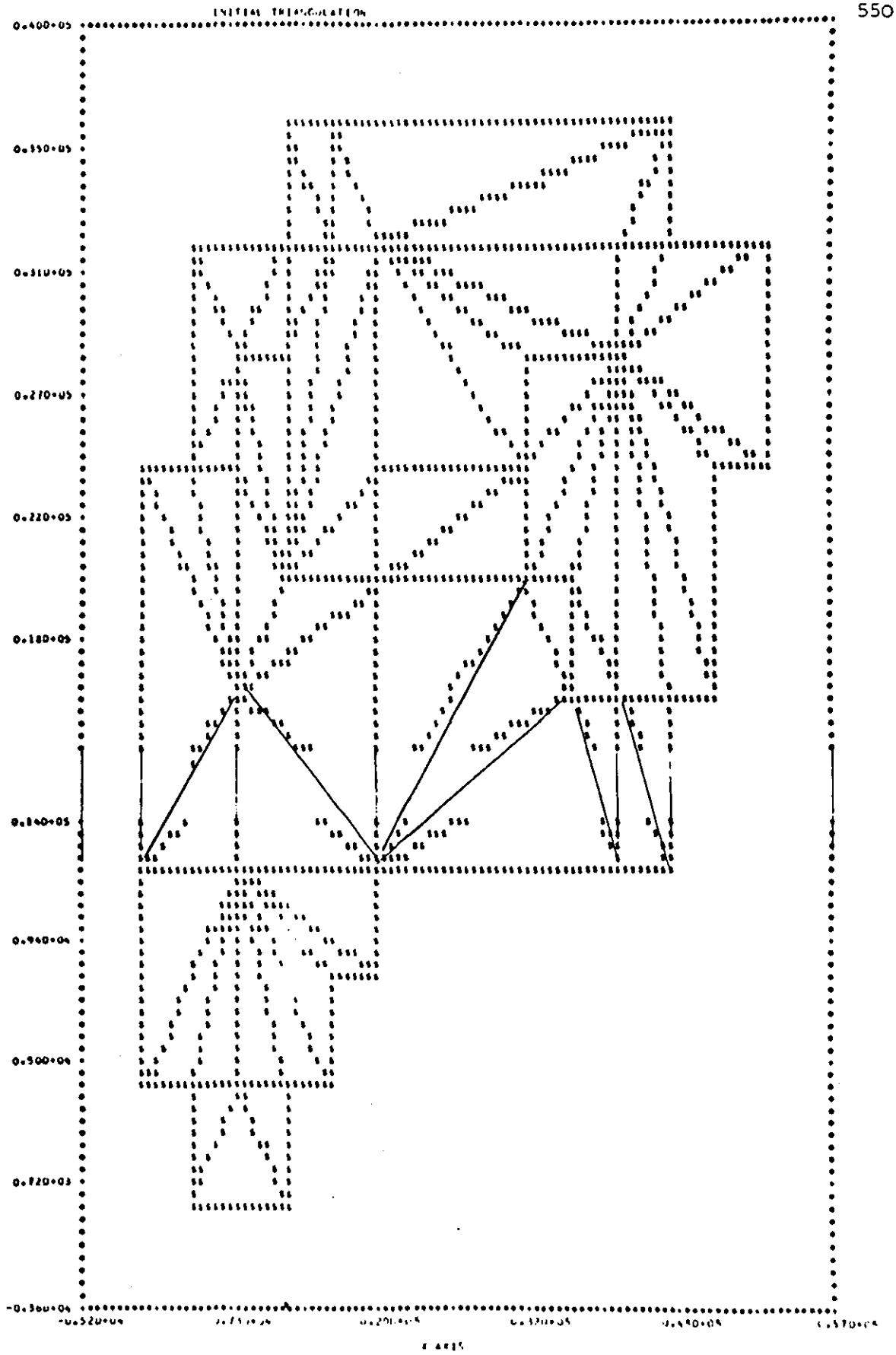


FIGURE 7.26: Initial triangulation for Problem 7.12

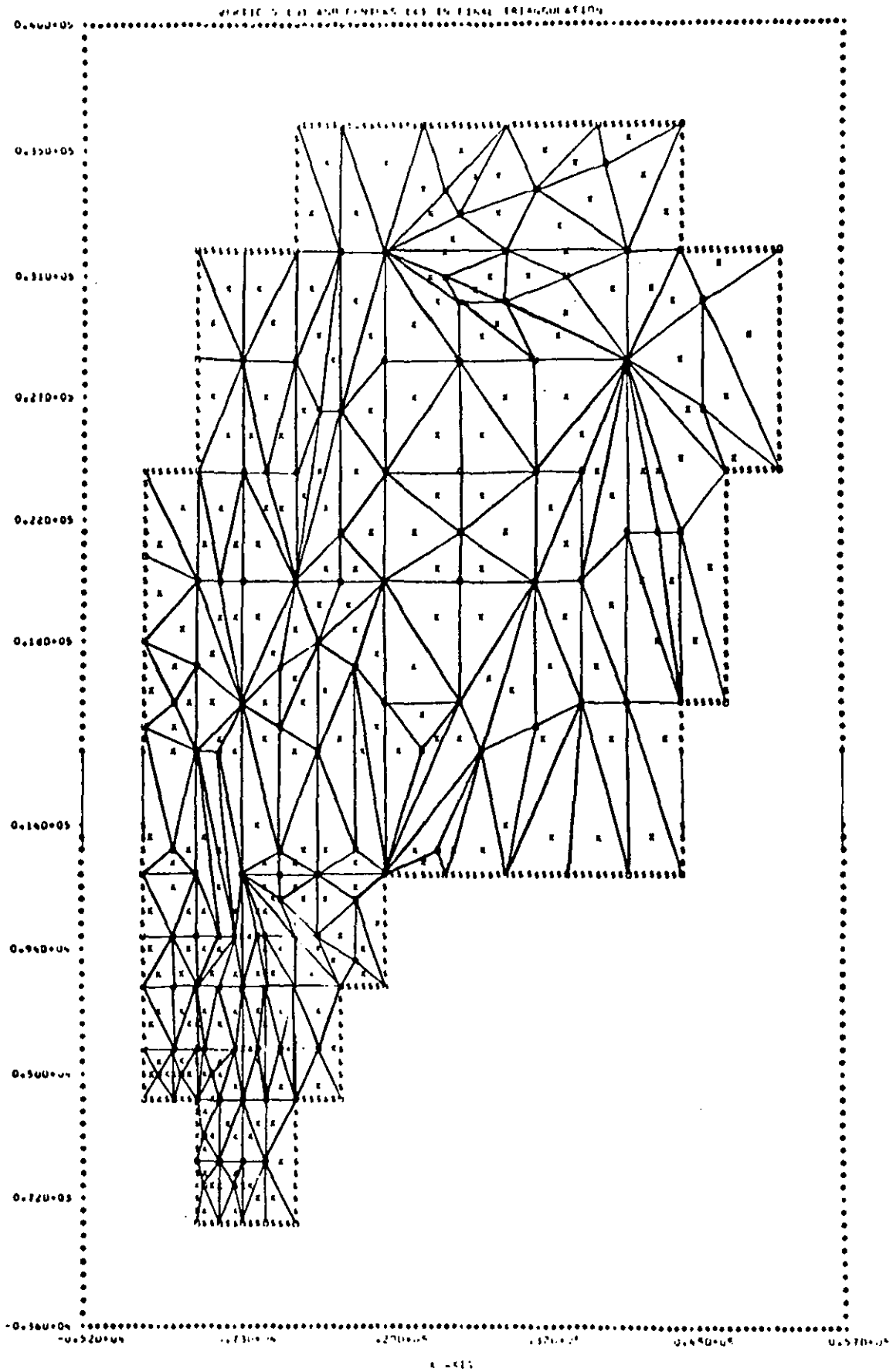


FIGURE 7.27: Final triangulation for problem 7.12

(iii) Since the transmissivity is not constant, a function is defined TRANSX(x,y) which returns the value of T_x at any point. Similarly the function TRANSY(x,y) is defined for T_y. The zone number of any point is defined by the integer function IZONE(x,y).

The following boolean expressions determine the zone number according to the (x,y) coordinates (in Km) of any point:

(1) Zone=1 if:

$x \geq 4$ and $x \leq 12$ and $y \leq 4$

or $x \leq 8$ and $y \geq 4$ and $y \leq 12$

(2) Zone=2 if:

$x \geq 8$ and $x \leq 16$ and $y \geq 4$ and $y \leq 12$

or $x \geq 16$ and $x \leq 20$ and $y \geq 8$ and $y \leq 12$

(3) Zone=3 if:

$x \geq 20$ and $x \leq 32$ and $y \geq 12$ and $y \leq 24$

or $x \geq 32$ and $x \leq 36$ and $y \geq 12$ and $y \leq 20$

or $x \geq 36$ and $x \leq 40$ and $y \geq 12$ and $y \leq 16$

(4) Zone=4 if:

$x \geq 40$ and $x \leq 44$ and $y \geq 12$ and $y \leq 16$

or $x \geq 36$ and $x \leq 48$ and $y \geq 16$ and $y \leq 20$

or $x \geq 32$ and $x \leq 48$ and $y \geq 20$ and $y \leq 24$

or $x \geq 32$ and $x \leq 52$ and $y \geq 24$ and $y \leq 28$

or $x \geq 40$ and $x \leq 52$ and $y \geq 28$ and $y \leq 32$

(5) Zone=5 if:

$x \geq 20$ and $x \leq 32$ and $y \geq 24$ and $y \leq 28$

or $x \geq 20$ and $x \leq 40$ and $y \geq 28$ and $y \leq 32$

or $x \geq 16$ and $x \leq 44$ and $y \geq 32$ and $y \leq 36$

(6) Zone=6 if:

$x \geq 4$ and $x \leq 8$ and $y \geq 24$ and $y \leq 32$

or $x \geq 8$ and $x \leq 12$ and $y \geq 28$ and $y \leq 32$

or $x \geq 12$ and $x \leq 20$ and $y \geq 20$ and $y \leq 32$

or $x \geq 12$ and $x \leq 16$ and $y \geq 32$ and $y \leq 36$

(7) Zone=7:

otherwise.

However, the boolean expressions for the 7th zone are:

$x \leq 20$ and $y \geq 12$ and $y \leq 20$

or $x \leq 12$ and $y \geq 20$ and $y \leq 24$

or $x \geq 8$ and $x \leq 12$ and $y \geq 24$ and $y \leq 28$

(iv) The results are computed for the points of intersection on a 4×4 grid.

(v) The well is modelled as explained earlier.

The computed heads at selected nodes are shown in Table 7.15, where the solution by the BIEM is also shown.

It can be seen that generally, good agreement exists.

Node Number	FEM	BIEM
1	1.5	1.5
2	6.419	6.8
3	7.099	7.4
4	15.759	18.2
5	17.866	19.0
6	27.497	30.6
7	29.931	31.9
8	58.009	59.7
9	60.28	60.3
14	55	55
15	59.03	60.3
16	58.892	60.4
17	24.67	26.3
18	3	3
26	4.2317	3.9
27	4.0516	3.7
32	32.132	32.4
33	9.4852	9.7
35	2.8733	3.2
36	7.0657	7.0

TABLE 7.15: Computed head for Problem 7.12

7.7.5 Conclusions

Several problems in groundwater flow have been presented in this chapter. All of these problems have been solved with a varying number of triangles (elements) and of different degrees, namely, quadratic, cubic and quartic elements. The total memory required for the whole job and the CPU time spent in each run are observed for different problems. Although other parameters like shape were tested, if seems that their effect on these two factors (i.e. memory and CPU time) is not significant. So only the number triangles and the degree of the

interpolating polynomials are considered. The following results are typically found:

(i) The relationship between the number of triangles (NTF) and CPU time and between the memory required and NTF are shown in Figures (7.28) and (7.29). It is obvious that the CPU time increases rapidly with the NTF parameter. The relationship may be approximated by a quadratic equation. The same seems true for the total memory required.

(ii) By performing a statistical analysis using the SAS [1982] system, the following empirical equations can be written for quadratic triangles:

(1) $$M = an^2 + bn + c \ , \qquad (7.115)$$

where: M is the memory (Kbytes)

a = .00347

b = .8277

c = 132.56

and n is the number of final triangles (NTF).

(2) $$T = an^2 + bn + c \ , \qquad (7.116)$$

where: T is the CPU time (seconds)

a = .0000337

b = .0139

c = 16.116

and n is the number of final triangles (NTF).

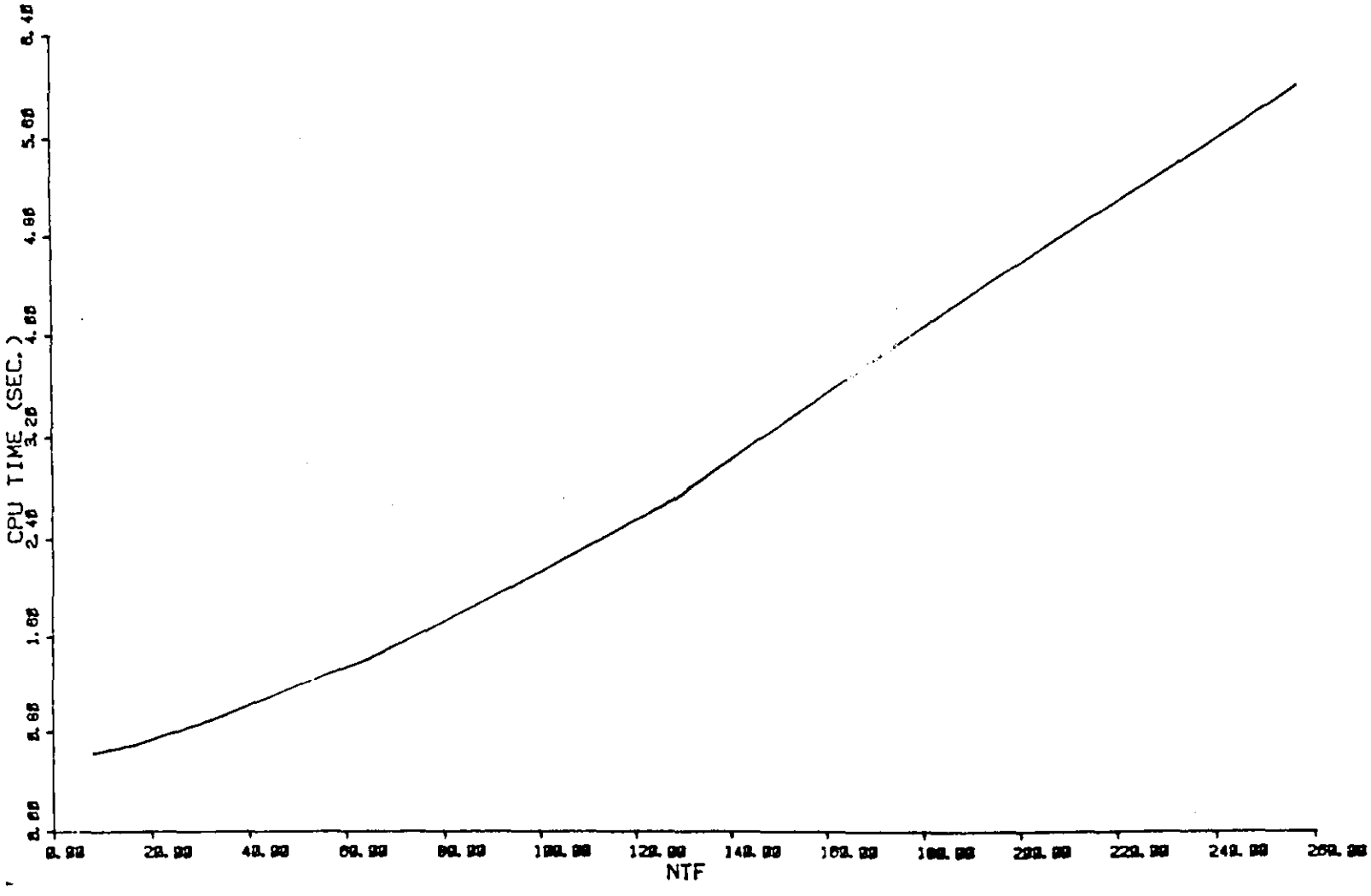


FIGURE 7.28: Relationship between the CPU time and number of elements

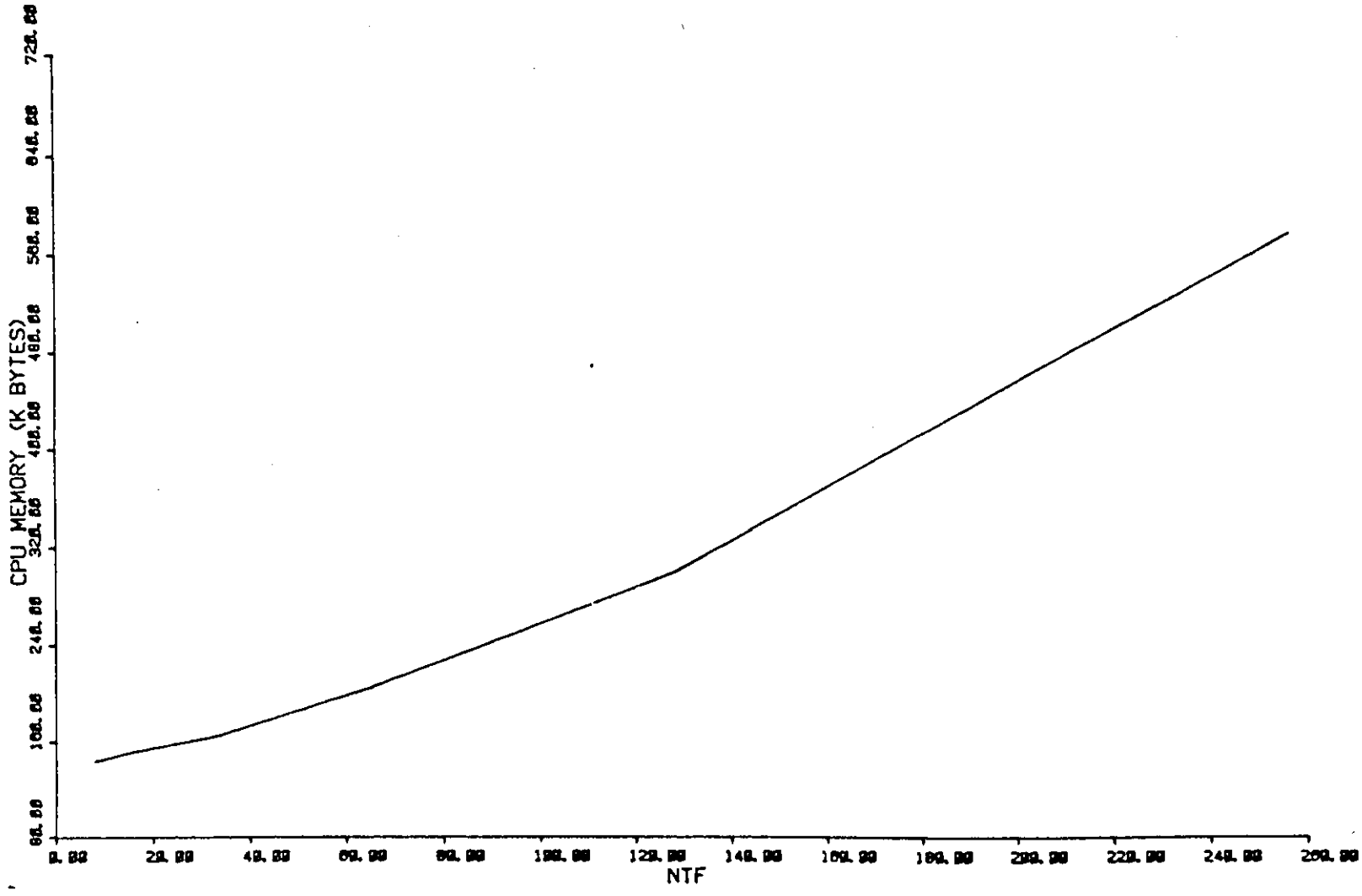


FIGURE 7.29: Relationship between the memory required and number of elements

CHAPTER 8

CONCLUSIONS

TABLE OF CONTENTS

8.1 CONCLUSIONS

The objective of this research was to study in detail and develop software for the FEM and to try to solve some of the computational problems encountered in the computer implementation of this method. In addition to that, several applications of the presented software have been solved. The research presented in this thesis is essentially of a software nature. However, it has two "wings"; one is mathematical while the other is engineering. This can be observed in the structure of the thesis where the FEM has been presented from both engineering (in Chapter 2) and mathematical (in Chapter 3) approaches and where engineering software (in Chapter 4) and mathematical software (in Chapter 6) has been given.

In addition to that, the solution to the problem of limited stack, which exist in many mini- and micro-computers, is given (in Chapter 5), and several groundwater problems using the FEM software were solved (in Chapter 7). Several programs for the FEM have been included: MSAP, STRAP, ELASTIC, IFIP and TWODEPEP. They, in total, represent the software of the FEM on computers of varying capabilities that range from the main-frame large computers to the small-size microcomputers.

As a result of this research, the following conclusions can be drawn:

(i) Theory and fundamentals of the FEM

- In many problems, the FEM formulation based on a variational principle will lead to an identical set of equations to that produced by a Galerkin MWR approach.
- The h- and p- versions of the FEM were tested in several problems. Although it is generally accepted that the p-version

will give better results, there are some situations where the use of the p-version will result in unnecessary additional costs. In fact, the accuracy of the FEM depends on how close the chosen trial functions simulates the actual behaviour of the field variable in the considered problem.

- Although it is not possible to compute the round-off errors in the FEM computations, there are some guidelines that, when followed, will minimize the effect of round-off errors. The two most important points are the use of computers of greater word length (like the CDC computers) and using double or higher precision arithmetic. Avoidance of a too refined FE mesh is also recommended since the relative round-off error may be increased.

(ii) Programming of the FEM

- The requirements of a programming language for the large-scale computational software, like FE, are established. Since Fortran is the most widely used language for FE programming, it was examined in view of these requirements. It was concluded that Fortran lacks some of these requirements and extensions to Fortran were proposed to overcome this situation. This, undoubtly, will make FE programming easier.
- The solution of equations in the FE analysis consumes a major part of the total solution time. However, in higher dimensional problems, it was observed that the stiffness matrices computation and the assembly process (if it has to be done) consumes a considerable amount of time. This raises the need for more investigations in this area.
- The FE equations are usually solved either by a banded or

frontal algorithm. If the available computer's fast memory is large enough, then the use of the banded algorithm will normally result in faster turn-around time. This is due to the relative slowness in file handling in the frontal algorithm. Generally, a trade-off should be done between the computer fast memory and the execution time.

- There are two versions of routines for the solution of equations: space economizer and high accuracy. For medium size of equations n ($n \cong 400$) both will consume the same time approximately. However, for larger values of n, the high accuracy routines will spend much more time.
- Many of the mainframe-based FEM programs lack adequite user interface. This may be due to the fact that most of the known large-scale FE programs, like SAP IV, ASKA, NASTRAN, etc., have their roots in the early 70's. This is not the case with mini- and micro-based systems. One of the recommended strategies to solve this problem is to use one of these smaller systems for the preparation of the FE model and to do the actual processing on a larger computer.
- There are three major problems that face a sophisticated FE user on a micro-computer: the limited memory size, the slow processing speed and the lack of supporting libraries. The problem of limited address space can be removed by the use of the virtual stack facility (VSF) presented in Chapter 5. While it is recommended to install mathematical co-processor to speed-up the floating point arithmetic.
- The definitions and functions of FE pre- and post-processors

have been established and demonstrated. Their importance have also been emphasised. In fact, more work is needed in this direction.

- The direct subdivision methods for mesh generation proved to be the most efficient among the presented methods of mesh generation. They are usually easier to program compared to other competitive methods. The only exception to this was the method of isoparametric mapping which is easily programmed. The reversed Cuthill-McKee (RCM) algorithm is also the most popular node numbering algorithm.
- The selection of the 'best fit' FE software depends upon the purpose for which this software is going to be used. For large-scale FE programs, it is recommended to use one (or more) of the three quantitative approaches presented in Chapter 4.
- The proposed solution of the limited stack size, namely, the virtual stack facility, proved to be an effective means to overcome some of the problems associated with mini- and micro-computers.
- The proposed K-R replacement algorithm also proved to be more efficient than some of the well-known replacement algorithms.
- The use of optimized compilers for FE programs can speed up the execution of the compiled program by a factor of up to 10%. So it is highly recommended to modify existing FE programs to make use of this feature.

(iii) Applications of the FEM

- The applicability of the FEM to solve a very wide-range of problems has been proved. In fact, the fields of applications are ever increasing with greater success in most of them.
- FE modelling of the considered groundwater problems proved the

versatility of the method. The results obtained when compared with analytical solutions, when they exist, showed very good agreement. In other problems, comparisons with other numerical methods support the evidence that the FEM mocelling is generally better compared to other numerical methods.

- The mesh refinement in regions where a singularity occurs is necessary. This has been done automatically by specifying the mesh refinement functions presented within the thesis. There is no formal way to formulate this function, but rather it can be formulated by experience according to the topology of the FE model. The given functions within the thesis proved to be efficient and produce well graded meshes.
- The strategy given to handle free surface problems is of a general nature and can be used in other fields. It can be automated by modifying the existing software.

8.2 SCOPE OF FURTHER RESEARCH

There is a good deal of work to be done in the FEM. However, here, we list some of the points that directly relate to our work and which seem to be of immediate need:

1. The use of multiple microprocessor-based systems for the FE analysis. This will open a new trend in FE analysis where complicated problems can be handled by distributing processor systems of cheap processors.
2. The efficient use of the parallel computing systems for the matrix computations and element assembly in FE analysis. It is clear that greater attention has been paid to the solution of equations while very little attention has been paid to the stiffness matrix computations and element assembly. In fact, even for single processor systems, there is a need for more "elegant" methods for the formulation and computation of element stiffnesses.
3. The development of new data structures, programming languages (new or extensions to an existing one) is a major area of research, the basis of which has been given within this research. Designing an efficient compiler for the extended Fortran given in Chapter 4 will no doubt be an asset in FE programming.
4. Implementing of the VSF with Fortran compiler modifications and implementing the K-R algorithm within existing operating systems environment.
5. Design of a FE software of general nature than can handle free surface problems automatically.

REFERENCES

Aitchinson, J.M. [1977], *The Numerical Solution of a Minimisation Problem Associated with a Free Surface Flow*,
J.Inst.Maths.Applics., 20, pp.33-44.

Akhras, G. and G. Dhatt, [1976], *An Automatic Node Relabelling Scheme for Minimizing a Matrix or Network Bandwidth*,
International Journal for Numerical Methods in Engineering, Vol.10, pp.787-797.

Akin, J.E. [1982], *Application and Implementation of Finite Element Methods*,
Academic Press.

Alway, G.G. and D.W. Martin, [1965], *An Algorithm for Reducing the Bandwidth of a Matrix of Symmetrical Configuration*,
Computer Journal, 8, pp.264-272.

ANSI [1978], *American National Standard Programming Languages*,
ANSI Publication X3.9-1978, American National Standards Institute, New York, U.S.A.

Argyris, J.H. [1964], *Recent Advances in Matrix Methods of Structural Analysis*,
Progress in Aeronautical Science, Vol. 4.

ASCE [1972], *Manuals and Reports on Engineering Practice*,
No. 40, Ground Water Management.

Axelsson, O. and U. Navert [1977], *On a Graphical Package for Nonlinear Partial Differential Equation Problems*,
Information Processing 1977, Ed. B. Gilchrist, pp. 103-108.

Babuska, I. and B. Szabo [1982], *On the Rates of Convergence of the Finite Element Method*,
International J. for Numerical Methods in Engineering, Vol. 18, pp.323-341.

Babuska, I., B. Szabo and I. Katz [1981], *The p-Version of the Finite Element Method*,
SIAM, J.Numer.Anal., 18, pp.515-545.

Babuska, I. and M. Dorr [1981], *Error Estimates for the Combined h and p Versions of the Finite Element Method*,
Num.Math., 37, pp.257-277.

Babuska, I., R.B. Kellogg and J. Pitkaranta [1979], *Direct and Inverse Error Estimates for Finite Elements with Mesh Refinements*,
Num.Math. 33, pp.447-471.

Baer, J-L. [1980], *Computer Systems Architecture*,
Pitman Publishing Limited.

Bathe, K-J, [1978], *ADINA - A Finite Element Program for Automatic Dynamic Incremental Non-Linear Analysis*,
Report 82448-1, MIT Press.

Bathe, K.J., E.L. Wilson and F.E. Peterson [1974], *SAP IV - A Structural Analysis Program for Static and Dynamic Response of Linear Systems*,
Publication No. EERC 73-11, College of Engineering, University of California, Berkeley, U.S.A.

Bear, J. [1979], *Hydraulics of Groundwater*,
McGraw-Hill Publ.

Becker, E.B., G.F. Carey and J.T. Oden [1981], *Finite Elements - An Introduction*,
Vol. I, Prentice-Hall.

Beckmann, P. [1971], *The History of* π
St. Martin's Press, New York.

Belady, L.A. [1966], *A Study of Replacement Algorithms for a Virtual-Storage Computer*,
IBM Systems Journal, Vol.5, No.2.

Bettess, A.J. [1977], *A Data Structure for Finite Element Analysis*, International Journal for Numerical Methods in Engineering, Vol. 11, pp.1779-1799.

Bettess, P. [1977], *Infinite Elements*, International Journal for Numerical Methods in Engineering, Vol. 11, pp.53-64.

Blackburn, C.L., O.O. Storsli and R.E. Fulton [1982], *The Role and Application of Data Base Management in Integrated Computer-Aided Design*, Presented at the AIAA/ASME/ASCE/AHS 23rd. Structures, Structural Dynamics and Material Conference, New Orleans, L.A.

Blakely, K., R. Lahey and D. Mclean [1985], *MSC/PAL: An FE Companion for the PC*, Computers in Mechanical Engineering, Vol. 3, No.4, pp.32-41.

Bornat, R. [1979], *Understanding and Writing Compilers*, The Macmillan Press Ltd.

Borsetto, M., G. Carradori and R. Ribacchs [1981], *Coupled Seepage, Heat Transfer and Stress Analysis with Application to Geothermal Problems*, In Numerical Methods in Heat Transfer, pp. 233-259, Editors: R.W. Lewis, K. Morgan and O.C. Zienkiewicz, John Wiley.

Bouwer, H. [1978], *Groundwater Hydrology*, McGraw-Hill.

Brebbia, C. and J. Dominguez [1978], *Boundary Element Method Versus Finite Elements*, In Applied Numerical Modelling, Ed. C.A. Brebbia, Pentech Press, pp.571-586.

Bredehoeft, J.D. and P. Betzinski [1982], *Ground-water Models, Vol.I: Concepts, Problems and Methods of Analysis*,
The Unesco Press.

Briz-Kishore, B.H. and R.V.S.S. Avadhanulu [1981], *Aquifer Simulation Program for Micro-Based Processors*,
Ground Water, Vol. 19, No. 4, pp.400-406.

Brown, C.B. and J.T. Yao [1983], *Fuzzy Sets and Structural Engineering*,
J. of Structural Engineering, Vol. 109, pp.1211-1225.

Brown, P.J. [1981], *Writing Interactive Compilers and Interpreters*,
John Wiley.

Browne, J.C. [1976], *Data Definition, Structures, and Management in Scientific Computing*,
In Computer Science and Scientific Computing, Ed. J.M. Ortega, pp. 25-26, Academic Press.

Broyden, C.G. [1965], *A Class of Methods for Solving Non-linear Simultaneous Equations*,
Mathematics of Computation 19, pp.577-593.

Bykat, A. [1977], *A Note on an Element Ordering Scheme*,
International J. for Numerical Methods in Engineering, Vol. 11, pp.194-198.

Cavendish, J.C. [1974], *Automatic Triangulation of Arbitrary Planar Domains for the Finite Element Method*,
International J. for Numerical Methods in Engineering, Vol. 8, pp.679-696.

Cheung, Y.K. and M.F. Yeo [1979], *A Practical Introduction to Finite Element Analysis*,
Pitman International Text.

Christ, W. [1982], *ICES on VAX-Computers, An Introduction to the ICES Basic System for VAX*,
ICES J., Vol. 14, pp.45-57.

Christiansen, H.N. [1976], *Computer Simulation of Distorted Structural Frameworks*, Computers & Structures, Vol. 6, pp.497-501.

Christiansen, H.N. and M.B. Stephenson [1979], *MOVIE.BYE - A Computer Graphics Software System*,
Journal of the Technical Councils of ASCE, Vol. 5, No. TCI, pp.3-12.

Clough, R.W. [1960], *The Finite Element in Plane Stress Analysis*,
Proceedings 2nd. ASCE Conference on Electronic Computation,
Pittsburgh, pp. 345-378.

Coel, C.M. [1984], *The Big Squeeze: Moving Mainframe Fortran to a Micro Computer*,
Computers in Mechanical Engineering, Vol. 3, No.2, pp.57-60.

Coffman, E.G. and P.J. Denning, [1973], *Operating Systems Theory*,
Prentice-Hall Inc.

Collins, R.J. [1980], *A Programming Language for the Finite Element Method*,
A Ph.D. Thesis, The City University, London.

Computer Aided Design Centre [1976], *GINO-F User Manual*,
Issue 2, Computer Aided Design Centre, Cambridge, England.

Connor, J.J. and C.A. Brebbia, [1976], *Finite Element Techniques for Fluid Flow*,
Newnes-Butterworths.

Conte, S.D. and C. De Boor, [1980], *Elementary Numerical Analysis, An Algorithmic Approach*,
3rd. Ed., McGraw-Hill, New York.

Cook, R.D. [1982], *Loubignac's Iterative Method in Finite Element Elastostatics*,
International J. for Numerical Methods in Engineering, Vol. 18, pp.67-75.

Cornell, A. [1980], *The Decision-Makers Handbook*,
Prentice-Hall.

Cuthill, E. and J.M. McKee [1969], *Reducing the Bandwidth of Sparse Symmetric Matrices*,
Proc. 24th National Conference, Association for Computing Machinary,
ACM Pub. P69, New York, pp. 157-172.

Date, C.J. [1982], *An Introduction to Data Base Systems*,
4th Edition, Addison-Wesley, Reading, Massachusetts.

Davis, A.J. [1980], *The Finite Element Method: A First Approach*,
Clarendon Press, Oxford.

DeWiest, R.J.M. [1965, *Geohydrology*,
John Wiley and Sons.

Dongarra, J.J., C.B. Moler, J.R. Bunch and G.W. Stewart [1979],
Linpack User's Guide,
SIAM, Philadelphia.

Duff, I.S. and J.K. Reid [1983], *The Multifrontal Solution of Indefinite Sparse Symmetric Linear Equations*,
ACM Transactions on Mathematical Software, Vol. 9, No.3, pp.302-325.

Dunavont, D. and B. Szabo [1983], *A Posteriori Error Indicators for the p-Version of the Finite Element Method*,
International J. for Numerical Methods in Engineering, Vol. 19,
pp.1851-1870.

Durocher, L.L. and A. Gaspar [1979], *A Versatile Two-Dimensional Mesh Generator with Automatic Bandwidth Reduction*, Computers & Structures, Vol. 10, pp.561-575.

Eisenstat, S.C., M.C. Gursky, M.H. Schultz and A.H. Sherman [1976], *Yale Sparse Matrix Package: I. The Symmetric Codes*, Research Report 112, Yale Computer Science Department, Yale University, New Haven, Conn., U.S.A.

Ergatoudis, J., B.M. Irons and O.C. Zienkiewicz [1968], *Curved Iso-Parametric Quadrilateral Elements for Finite Element Analysis*, Int.J. Solids Struct., Vol. 4, pp.31-42.

Evans, D.J., [1973], *The Analysis and Application of Sparse Matrix Algorithms in the Finite Element Method*, In the Mathematics of Finite Elements and Applications, Ed. J.R. Whiteman, Academic Press, pp.427-447.

Evans, D.J. [1982], *Parallel Numerical Algorithms for Linear Systems*, In Parallel Processing Systems, Ed. D.J. Evans, Cambridge University Press, pp. 357-383.

Everstine, G.C. and E.H. Cuthill [1983], *The Optimal Ordering of Tree Networks*, Computers & Structures, Vol. 17, No. 4, pp.621-622.

Faust, C.R. and J.W. Mercer [1980], *Ground Water Modelling: Numerical Models*, Ground Water, Vol. 18, No. 4, pp.395-409.

Felippa, C.A. [1972], *An Alphanumeric Finite Element Mesh Plotter*, International J. for Numerical Methods in Engineering, Vol. 5, pp.217-236.

Felippa, C.A. [1979], *Data base Management in Scientific Computing - I. General Description*, Computers & Structures, Vol. 10, pp.53-61.

Finlayson, B.A. and L.E. Scriven [1966], *The Method of Weighted Residuals - A Review*,
Appl.Mech.Rev., Vol. 19, No. 9, pp.735-748.

Firkins, N.L. and J.Q. Hossack [1977], *A Command Structured Approach to Structural Object Plotting and Automated Line Dimensioning*,
Computers & Structures, Vol. 7, pp. 587-598.

Fong, H.H. [1984], *Standards for Finite Element Codes: the Long Road Ahead*,
Computers in Mechanical Engineering, Vol. 3, No. 3, pp.10-14.

Fossen, D.B.V. [1978], *FESAP - Design Program for Static and Dynamic Structural Analysis*,
Computers & Structures, Vol. 9, pp. 371-376.

Fox, L. [1966], *An Introduction to Numerical Linear Algebra*,
Oxford University Press, New York.

France, P.W. [1975], *An Improved Finite Element Technique for the Analysis of Free Surface Flow Problems*,
Computers & Fluids, Vol. 3, pp.149-153.

Gallagher, R.H., B.R. Simon, P.C. Johnson and J.F. Gross [1982], *Finite Elements in Bio Mechanics*,
Editors,
John Wiley.

Gattas, M. and J.F. Abel [1983], *Three-dimensional Linear Dynamic Analysis of Buildings with 32-Bit Virtual-Memory Mini-Computers*,
Computers & Structures, Vol. 17, No. 1, pp. 97-104.

George A. and J. W-H Liu [1981], *Computer Solution of Large Positive Definite Systems*,
Prentice-Hall.

George, J.A. [1971], *Computer Implementation of the Finite Element Method*,
Ph.D. Dissertation, Dept. Comput.Sci., Stanford Univ., Cal., USA.

Gerald, C.F. [1978], *Applied Numerical Analysis*,
2nd edition, Addison-Wesley, Pub.Co.

Gibbs, N.E., W.G. Poole and P.K. Stockmeyer, [1976], *An Algorithm for Reducing Bandwidth and Profile of a Sparse Matrix*,
SIAM J. Numerical Analysis Vol. 13, pp.236-250.

Goetschel, D. [1984], *A Personal Look at Virtual Memory Micro-computers*,
Computers in Mechanical Engineering, Vol. 2. No. 4, pp.48-52.

Green, G. [1828] *An Essay on the Application of Mathematical Analysis to the Theory of Electricity and Magnetism*,
Nottingham.

Griffin, O.H. and C.R. Wilson [1983], *Finite Element Analysis on A Microprocessor-Based Personal Workstation*,
Computers & Structures Vol. 17, No. 4, pp.617-619.

Groover, M.P. and E.W. Zimmers [1983], *CAD/CAM: Computer-Aided Design and Manufacturing*,
Prentice-Hall.

Haber, R. and J.F. Abel [1982], *Discrete Transfinite Mappings for the Description and Meshing of Three-Dimensional Surfaces Using Interactive Computer Graphics*,
International J. for Numerical Methods in Engineering, Vol. 18, pp.41-66.

Haber, R., M.S. Shephard, J.F. Abel, R.H. Gallagher, and D.P. Greenberg [1981], *A General Two-Dimensional Graphical Finite Element Preprocessor*

Utilizing Discrete Transfinite Mappings,
International J. for Numerical Methods in Engineering, Vol. 17, pp. 1015-1044.

Hantush, M.S. and C.E. Jacob [1955], *Non-Steady Radial Flow in an Infinite Leaky Aquifer*,
Am. Geophys. Un. Trans. 36, pp.95-100.

Haugeneder, E., W. Prochazka and P. Tavolato, [1981], *A Pre Processor for the Finite Element Program SAP IV*,
International Journal for Numerical Methods in Engineering, Vol. 17, pp.1779-1789.

Hellen, T.K. [1969], *A Front Solution for Finite Element Techniques*,
Central Electricity Board R & D Dept., RD/B/N 1459.

Hildebrand, F.B. [1965], *Methods of Applied Mathematics*,
Prentice-Hall, Englewood Cliffs, N.J.

Hinton, E., A. Razzaque, O.C. Zienkiewicz and J.D. Davies, [1975], *A Simple Finite Element Solution for Plates of Homogeneous, Sandwich and Cellular Construction*.
Proc.Inst. Civil Eng. 59, Part 2, pp.43-65.

Hinton, E. and D.R.J. Owen [1979], *Finite Element Programming*,
Academic Press.

Hinton, E., P. Bettess and R.W. Lewis [1981], *Numerical Methods for Coupled Problems*,
Pineridge Press.

Hitchings, D. [1975], *FINEL - A Finite Element Language for Teaching, Research and Development*,
Proceedings 3rd. Post Conference on Computational Aspects of the Finite Element Method, pp.59-67.

Hogben, L. [1967], *Mathematics for the Millions*,
Pan Books, London.

Hoit, M. and E.L. Wilson [1983], *An Equation Numbering Algorithm Based on a Minimum Front Criteria*,
Computers & Structures, Vol. 16, No. 1-4, pp.225-239.

Hood, P. [1976], *Frontal Solution Program for Unsymmetric Matrices*,
International J. for Numerical Methods in Engineering, Vol. 10, pp.379-400.

Hopgood, F.R.A. [1974], *Compiling Techniques*,
MacDonald Pub.

Houstis, E.N., R.E. Lynch, T. Papatheodorou and J.R. Rice [1975], *Development, Evaluation and Selection of Methods for Elliptic Partial Differential Equations*,
In Advances in Computer Methods for Partial Differential Equations, Ed. R. Vichnevetsky, AICA Publ., pp. 1-6.

HP 3000 [1979], *HP 3000 General Information Manual*,
Hewlett-Packard, California, USA.

Hrenikoff, A. [1941], *Solution of Problems in Elasticity by the Framework Method*,
Journal of Applied Mechanics, Vol.8.

Huebner, K.H. and E.A. Thornton [1982], *The Finite Element Method for Engineers*,
2nd. Edition, Wiley.

Hurst, T.N. and B.A. Ross [1984], *Fortran That Travels: Programming for Portability*,
Computers in Mechanical Engineering, Vol. 3, No. 3, pp.25-27.

Hurwitz, A., J.P. Citron and J.B. Yeaton [1967], *GRAF * Graphical Extensions to Fortran*,
Proc. SJCC 1967, Thompson Book Co., Washington, D.C., pp.553-557.

Huyakorn, P.S. and C.R. Dudgeon [1974], *Finite Element Programs for Analysing Flow Towards Wells*,
Australian Water Resources Council Project 71/25.

IBM [1981], *VS Fortran Application Programming Guide*,
IBM,

IMSL [1983], *TWODEPEP User's Manual*,
Edition 5, IMSL Inc., Texas, U.S.A.

IMSL [1984], *International Mathematical and Statistical Library*,
Edition 9.2, IMSL Inc., Texas, U.S.A.

Intel [1981], *IAPX 286 Preliminary User Manual*,
Santa Clara, California, Intel Corp.

Irons, B.M. [1966], *Engineering Application of Numerical Integration in Stiffness Method*,
AIAA J., Vol. 14, pp.2035-2037.

Irons, B.M. [1970], *A Frontal Solution Program for Finite Element Analysis*,
International J. for Numerical Methods in Engineering, Vol. 2, No. 1, pp.5-32.

Irons, B.M. and A. Razzaque [1973], *Introduction of Shear Deformations into a Thin Plate Displacement Formulation*,
AIAA J., Vol. 11, No.10, pp.1438-1439.

Irons, B.M. and N. Shrive [1983], *Finite Element Primer*,
Ellis Horwood Ltd.

Irons, B.M. and S. Ahmad [1980], *Techniques of Finite Elements*,
Ellis Horwood Ltd.

Jacobsen, K.P. [1983], *Fully Integrated Superelements: A Data Base Approach to Finite Element Analysis*,
Computers & Structures, Vol. 16, No. 1-4, pp.307-315.

Janssen, T.L. [1983], *A Simple Efficient Hidden Line Algorithm*,
Computers & Structures, Vol. 17, No. 4, pp.563-571.

Jennings, A. [1966], *A Compact Storage Scheme for the Solution of Symmetric Linear Simultaneous Equations*,
Computer Journal, Vol. 9, pp. 281-285.

Jennings, A. [1977], *Matrix Computation for Engineers and Scientists*,
John Wiley & Sons.

Johnson, A.E. and D. Torok [1985], *Software for Fluid Flow and Heat Transfer Analyses of Electronic Packaging*,
Computers in Mechanical Engineering, Vol. 4, No. 1, pp. 41-46.

Jordan, H.F. and P.L. Sawyer, [1979], *A Multi-Microprocessor System for Finite Element Structural Analysis*,
Computers & Structures, Vol. 10, pp. 21-29.

Kaldjian, M.J. [1976], *Interactive Data Preprocessor Program for Michigan SAP (MSAP)*,
Computers & Structures, Vol. 6, pp. 405-412.

Kaldjian, M.J. [1977], *Three Dimensional Interactive Graphic Display Program for Michigan SAP (MSAP)*,
Computers & Structures, Vol. 7, pp. 183-187.

Kaldjian, M.J., M.S. El-Nashie, O. Yuzugullu, G. Siddiqui and A. Sharaf Eldin [1982], *Computer Aided Structural Design*,
A Short Course offered by the College of Engineering,
King Saud University from 29 May - 9 June 1982. Published by King Saud University Press.

Kalkani, E.C. [1976], *Computer Plotting of Stress Contours in Excavated Slopes*,
International Journal for Numerical Methods in Engineering, Vol. 10, pp.1261-1280.

Kamel, H.A., A.V. Mobley, R. Nagulpally and D. Kumar [1985], *A GIFT for FE Analysis on a Microcomputer*,
Computers in Mechanical Engineering, Vol. 3, No. 4, pp.12-20.

Kamel H.A. and M.W. McCabe [1976], *Applications of GIFTS III to Structural Engineering Problems*,
Computers & Structures, Vol. 7, pp. 399-415.

Kamel, H.A. and Z. Navabi [1980], *Digitizing for Computer-Aided Finite Element Model Generation - Part 2 Use of Digitizing in Mesh Generation*,
Transactions of the ASME, Vol. 102, pp.560-565.

Karplus, W.J. and D. Cohen [1981], *Architectural and Software Issues in the Design and Application of Peripheral Array Processors*,
Computer, Vol. 14, No. 9, pp.11-17.

Keeney, R.L. and H. Raiffa [1976], *Decision with Multiple Objectives: Preferences and Value Tradeoffs*,
John Wiley & Sons, Inc.

Kelly, D., J. Gago, O.C. Zienkiewicz and I. Babuska [1983], *A Posteriori Error Analysis and Adaptive Processes in the Finite Element Method: Part I - Error Analysis*,
International J. for Numerical Methods in Engineering, Vol. 19, pp.1593-1619.

Kirk, D.B. [1986], *Curved Surfaces in Solid Modelling: New Hardware Improves the View*,
Computers in Mechanical Engineering, Vol. 4, No. 6, pp.10-14.

Knuth, D.E. [1975], *The Art of Computer Programming*,
Addison-Wesley.

Kulsrud, H.E. [1968], *A General Purpose Graphic Language*,
Comm. ACM 11, pp.247.

Liggett, J.A. and P. L-F Liu [1983], *The Boundary Integral Equation Method for Porous Media Flow*,
George Allen & Unwin.

Livesley, R.K. [1964], *Matrix Methods of Structural Analysis*,
Pergamon Press.

Liu, W.H. and A.H. Sherman [1976], *Comparative Analysis of the Cuthill-McKee and the Reverse Cuthill-McKee Ordering Algorithm for Sparse Matrices*
SIAM J. Numerical Analysis, Vol. 13(2), pp.198-213.

Loubignac, G., G. Cantin and G. Touzot [1977], *Continuous Stress Fields in Finite Element Analysis*,
AIAA J., Vol. 15, No. 11, pp.1645-1647.

MacNeal, R.H. [1970], *The NASTRAN Theoretical Manual*,
NASA Report No. NASA SP-221.

Majid, K.I. [1980], *Introduction to Matrix and Numerical Methods for Civil Engineers*,
Wood Stock Publishing.

Marc Analysis Research Corp. [1980], *MENTAT, Interactive Graphics Finite Element Mesh Editor, Preprocessor and Postprocessor*,
California, U.S.A.

Marcal, P.V. [1976], *General Purpose Program for Finite Element Analysis: Some Computational Considerations*,
In Computer Science and Scientific Computing, Ed. J.M. Ortega,
pp. 155-162, Academic Press.

Martin, J. [1977], *Data-base Organization*,
2nd Edition, Prentice-Hall Inc.

Melosh, R. [1973], *Inherited Error in Finite Element Analysis of Structures*,
Computers & Structures, Vol. 3, pp. 1205-1217.

Melosh, R. and R.M. Banford [1969], *Efficient Solution of Load Deflection Equations*,
J. Struct. Division Proc., ASCE, 95, ST 4, pp.661-676.

Melosh, R., S. Utku, M. Islam and M. Salama [1984], *An Emulator for Minimizing Computer Resources for Finite Element Analysis*,
Computers & Structures, Vol. 18, No. 4, pp. 567-574.

Mindlin, R.D. [1951], *Influence of Rotatory Inertia and Shear on Flexural Motions of Isotropic Elastic Plates*,
J.Appl.Mech. 18, pp.31-38.

Mitchell, A.R., G. Phillips and E. Wachspress [1971], *Forbidden Shapes in the Finite Element Method*,
J.Inst.Maths. Applics., 8, pp.260-269.

More, J., B. Garbow and K. Hillstrom [1980], *User Guide for MINPACK-1*,
Argonne National Laboratory Report ANL-80-74, Argonne, Illinois, U.S.A.

Motorola [1981], *MC 68451*,
Advance Information, Austin, Texas, Morotola Inc.

Newman, W.M. and R.F. Sproull [1982], *Principles of Interactive Computer Graphics*,
Second Edition, McGraw-Hill Inc.

Noor, A.K. [1981], *Survey of Computer Programs for Solution of Nonlinear Structural and Solid Mechanics Problems*,
Computers & Structures, Vol. 13, pp.425-465.

Oden, J.T. [1972], *Finite Element of Non-Linear Continua*,
McGraw-Hill.

Oden, J.T. and N. Kikuchi [1980], *Theory of Variational Inequalities with Applications to Problems of Flow Through Porous Media*,
Int.J. Energy Sci., 18, pp.1173-1284.

Ogden, D. [1979], *Extended Memory Systems for Microprocessor*,
International Micro and Mini Computer Conference, Houston, Texas, USA.

Oliver, M.A. and N.E. Wiseman [1983], *Operations on Quadtree Encoded Images*,
The Computer Journal, Vol. 26, No. 1, pp. 83-92.

Ortega, J.M. and W.C. Rheinboldt [1970], *Iterative Solution of Nonlinear Equations in Several Variables*,
Academic Press, New York.

Papadimitriou, C.H. [1974], *The NP-Completeness of the Bandwidth Minimization Problem*,
Report No. 173, Computer Science Laboratory, Department of Electrical Engineering, Princeton University, USA.

Pavlidis, T. [1977], *Structural Pattern Recognition*,
Springer-Verlag.

Pesquera, C.I., U. McGuire and J.F. Abel [1983], *Interactive Graphical Preprocessing of Three Dimensional Framed Structures*,
Computers & Structures, Vol. 17, No. 1, pp.1-12.

Pina, H.L. [1981], *An Algorithm for Frontwidth Reduction*,
International Journal for Numerical Methods in Engineering, Vol.17, pp.1539-1546.

Plunkett, B. [1985], *Portable Pictures: A Case for Graphics Standards*,
Computers in Mechanical Engineering, Vol. 3, No.5, pp.37-40.

Prickett, T.A. [1975], *Modelling Techniques for Groundwater Evaluation*, In Advances in Hydroscience, Vol. 10, Academic Press, New York, pp.1-43.

Przemieniecki, J.S. [1968], *Theory of Matrix Structural Analysis*, McGraw-Hill.

Quoc, L. Vu and J.R. Oleary [1984], *Automatic Node Resequencing with Constraints*, Computers & Structures, Vol. 18, No. 1, pp.55-69.

Raiffa, H. [1968], *Decision Analysis*, Addison-Wesley Publ.

Rao, S.S. [1982], *The Finite Element Method in Engineering*, Pergamon Press.

Razzaque, A. [1980], *Automatic Reduction of Frontwidth for Finite Element Analysis*, International J. for Numerical Methods in Engineering, Vol.15, pp.1315-1324.

Rice, J.R. [1975], *A Metalgorithm for Adaptive Quadrature*, J. ACM, 22, pp.61-82.

Rivara, M.C. [1984], *Algorithms for Refining Triangular Grids Suitable for Adaptive and Multi Grid Techniques*, International J. for Numerical Methods in Engineering, Vol.20, pp.745-756.

Romo, J.B. and D.M. Burns [1986], *Parallelism in Engineering Analysis*, Computers in Mechanical Engineering, Vol.4, No.5, pp.12-20.

Rose, D.J., G.G. Whitten, A.H. Sherman and R.E. Tarjan [1980], *Algorithms and Software for In-Core Factorization of Sparse Symmetric Positive Definite Matrices*, Computers & Structures, Vol. 11, pp.597-608.

Ross, B.A., J.J. Cox, T.N. Hurst and S.E. Benzley [1985], *Rosetta Stone for FE Modelling*,
Computers in Mechanical Engineering, Vol. 3, No. 4, pp.43-51.

Sa Da Costa, A.A.G. [1980], *A Numerical Model of Seawater Intrusion in Aquifers*,
Ph.D. Dissertation, Dept. of Civil Engineering, MIT, Mass., USA.

Samet, H. [1984], *The Quadtree and Related Hierarchical Data Structure*,
Computing Surveys, Vol. 16, No. 2, pp.187-247.

SAS [1982], *SAS Users' Guide 1982 Edition*,
SAS Institute, USA.

Schmitt, S. [1983], *Virtual Memory for Micro Computers, Four New Memory-Management Chips Pave the Way*,
Byte Magazine.

Schreiber, R. [1982], *A New Implementation of Sparse Gaussian Elimination*,
ACM Transactions on Mathematical Software, Vol. 8, No. 3, pp.256-276.

Schrem, E. [1979], *Trends and Aspects of the Development of Large Finite Element Software Systems*,
Computers & Structures, Vol. 10, pp.419-425.

Sewell, G. [1972], *Automatic Generation of Triangulations for Piecewise Polynomial Approximation*,
Ph.D. Thesis, Purdue University, USA.

Sewell, G. [1987], *Vectorization and PDE/PROTRAN*,
Directions, Vol. 4, No. 1, pp.2-3.

Sharaf Eldin, A. [1983a], *Numerical Simulation of Ground Water Resources Analysis Using Finite Elements*,
Proc. Symposium on Water Resources in the Kingdom of Saudi Arabia, April, 1983, pp.298-313.

Sharaf Eldin, A. [1983b], *Preprocessors for Finite Element Groundwater Analysis Programs*,

Proc. Symposium on Water Resources in the Kingdom of Saudi Arabia, April, 1983, pp.155-168.

Sharaf Eldin, A. [1983c], *Postprocessors for Finite Element Groundwater Analysis Programs*,

Proc. Symposium on Water Resources in the Kingdom of Saudi Arabia, April, 1983, pp.144-154.

Sharaf Eldin, A. [1983d], *Introduction to Digitizer*,

Lecture Notes, King Saud University.

Sharaf Eldin, A. [1984], *Introducing the IMSL Software*,

Lecture Notes, King Saud University,

Sharaf Eldin, A. [1985a], *A Micro-Computer Based Interactive Program for Aquifer Simulation Using Finite Elements*,

Proc. 16th Annual Pittsburgh Conference on Modelling and Simulation, University of Pittsburgh, Pittsburgh, USA, April 25-26, 1985, pp.907-910.

Sharaf Eldin, A. [1985b], *PDS/3000, A Man-Machine Interface*,

Proc. IUG Conference, Amsterdam, The Netherlands, March 31-5 April, 1985, pp.139-147.

Sharaf Eldin, A., A.R. Abdul-Fattah and W.H. Abulfaraj [1986], *A Multi-Attribute Utility Package for Decision Making*,

Proc. 17th Annual Pittsburgh Conference on Modelling and Simulation, University of Pittsburgh, Pittsburgh, USA, April 24-25, 1986, 1839-1844.

Sharaf Eldin, A. and A.S. Nouh [1984], *Virtual Stack Facility for Mini- and Micro-computers in Time-Sharing Environment*,

Proc. 7th NCC, pp.416-426.

Sharaf Eldin, A. and D.J. Evans, [1986], *Finite Elements Software Selection for Computational Mechanics: Three Quantitative Approaches*, Proc. International Conference on Computational Mechanics, ICCM86-Tokyo, Japan. 25-29 May, 1986, pp.XII-23 - XII-29.

Sharaf Eldin, A. and D.J. Evans, [1987], *ELASTIC - An Interactive Finite Elements Program for the Analysis of Two Dimensional Elasticity Problems*, Symp. on Concrete and Concrete Structures in the Middle East. Riyadh, Saudi Arabia 25-29 April, 1987.

Shephard, M.S. and M.A. Yerry, [1983], *Approaching the Automatic Generation of Finite Element Meshes*, Computers in Mechanical Engineering, Vol. 1, No. 4, pp.49-56.

Smith, G.D., [1969], *Numerical Solution of Partial Differential Equations*, Oxford University Press.

Stabrowski, M.M., [1981], *An Algorithm for the Solution of Very Large Banded Unsymmetric Linear Equation Systems*, International J. for Numerical Methods in Engineering, Vol. 17, pp. 1103-1117.

Stafford, J.R., [1983], *A Desk-Top Personal Computer for Finite Element Post-Processing*, Computers & Structures, Vol. 17, No. 5-6, pp.689-695.

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

Strohkorb, G.A. and A.K. Noor, [1984], *Potential of Minicomputer-Array Processor System for Nonlinear Finite Element Analysis*, Computers & Structures, Vol. 18, No. 4, pp.703-718.

Sutherland, I.E., [1965], *SKETCHPAD: A Man-Machine Graphical Communication System*, MIT Lincoln Lab., Tech. Rep. 296.

Sutherland, I.E., R.F. Sproull and R.A. Schumacker, [1974], *A Character-*

ization of Ten Hidden-Surface Algorithms,
Computer Survey, 6, No. 1, pp.1.

Swanson, J.A., [1977], *Use of Mini-computers for Large Scale Structural Analysis Programs*,
Computers & Structures, Vol. 7, pp.291-294.

Taweel, A., A. Sharaf Eldin, A. Nouh and G. Khader, [1978], *Numerical Simulation of Groundwater Resources in Saudi Arabia*,
Proceedings 4th NCC, March, 1978.

Thacker, W.C., [1980], *A Brief Review of Techniques for Generating Irregular Computational Grids*,
International J. for Numerical Methods in Engineering, Vol.15, pp.1335-1341.

Thatcher, R.W. and S.L. Askew, [1982], *A Complementary Solution to the Dam Problem*,
IMA J. of Numerical Analysis, 2, pp.229-239.

Theis, C.V., [1935], *The Relation Between the Lowering of the Piezometric Surface and the Rate and Duration of Discharge of a Well Using Groundwater Storage*,
Trans. Am. Geoph, Un., 16, pp.519-524.

Timoshenko, S. and J.N. Goodier, [1951], *Theory of Elasticity*,
McGraw-Hill, New York.

Turaby, A. and A. Sharaf Eldin, [1978], *Structural Analysis Package (STRAP)*
Proc. 4th NCC, March, 1978.

TWODEPEP, [1980], *TWODEPEP User's Manual, Version 1*,
IMSL Inc., Texas, USA.

Ungless, R.F., [1973], *An Infinite Finite Element*,
M.Sc. Thesis, Dept. of Civil Eng. University of British Columbia, Canada.

Utku, S. and R.J. Melosh, [1984], *Solution Errors in Finite Element Analysis*,
Computers & Structures, Vol. 18, pp.379-393.

Van Fossen, D.B., [1978], *FESAP - Design Program for Static and Dynamic Structural Analysis*,
Computers & Structures, Vol. 9, pp.371-376.

Vemuri, V. and W.J. Karplus, [1981], *Digital Computer Treatment of Partial Differential Equations*,
Prentice-Hall, Inc.

Walker, B.S., J.R. Gurd, and E.A. Drawnak, [1973], *Interactive Computer Graphics*,
Edward Arnold Pub.

Wang, H.F. and M.P. Anderson, [1982], *Introduction to Groundwater Modelling: Finite Difference and Finite Element Methods*,
W.H. Freeman and Company, San Fransisco.

Warnock, J.E., [1969], *A Hidden-Surface Algorithm for Computer Generated Half-tone Pictures*,
Univ. Utah, Computer Science Dept., TR 4-15.

Watkins, G.S., [1970], *A Real-Time Visible Surface Algorithm*,
Univ. Utah, Computer Science Dept., UTEC-CSC-70-101.

Wilkinson, J.H., [1965], *Rounding Errors in Algebraic Processes*,
N.P.L. Notes on Applied Science, 32, London.

Wilson, E.L., [1963], *Finite Element Analysis of Two Dimensional Structures*,
SESM 63-2, University of California at Berkely, USA.

Wilson, E.L., [1970], *SAP - A General Structural Analysis Program*,
SESM Report 70-20, Dept. of Civil Engineering, Uni. of California, Berkeley, USA.

Wilson, E.L. [1974], *The Static Condensation Algorithm*, International J. for Numerical Methods in Engineering, Vol. 8, No. 1, pp.198-203.

Wilson, E.L.,[1985], *Tailor Made: Structural FE Analysis to Suit the User and the Computer*, Computers in Mechanical Engineering, Vol. 3, No. 4, pp.22-28.

Wilson, E.L., K.J. Bathe and W.P. Doherty, [1974], *Direct Solution of Large Systems of Linear Equations*, Computers & Structures, Vol. 4, pp.363-372.

Wilson, J.L., R.L. Townley and A.S.D. Costa, [1979], *Mathematical Development and Verification of a Finite Element Aquifer Flow Model - Aquifem-1*, Report. No.248, Ralph M. Parsons Laboratory for Water Resources and Hydrodynamics, MIT, USA.

Wu, E-R, [1982], *Techniques to Avoid Duplicate Nodes and Relax Restrictions on the Super Element Numbering in a Mesh Generator*, Computers & Structures, Vol. 15, No. 4, pp.419-422.

Yamada, Y. and H. Okumara, [1980], *Micro-computer Systems for Medium-Sized and Experimental Finite Element Analysis*, In NASA Conference Publication 2147, NASA, pp.277-289.

Yamada, Y., H. Okumara and T. Sakurai, [1980], *Microcomputer Systems for Medium Sized and Experimental Finite Element Analysis*, NASA Conference Publication 2147, Research in Nonlinear Structural and Solid Mechanics, Ed. H.G. McComb and A.K. Noor, pp.277-289.

Yettram, A.L. and H.M. Husain, [1966], *Plane Framework Models for Plates in Extension*, Journal of the Engineering Mechanics Division, Proc. of ASCE, Vol.92, No. EM1, pp.157-168.

Zadeh, L.A., [1965], *Fuzzy Sets*,
Information & Control, Vol. 8, pp.338-353.

Zadeh, L.A., [1976], *A Fuzzy Algorithmic Approach to the Definition of Complex or Imprecise Concepts*,
Int.J. Man-Machine Studies, 8, pp.249.

Zave, P. and E.C. George Jr., [1983], *A Quantitative Evaluation of the Feasibility of, and Suitable Hardware Architectures for, an Adaptive, Parallel Finite Element System*, ACM Trans. on Mathematical Software, Vol. 9, No. 3, pp.271-292.

Zienkiewicz, O.C., [1977], *The Finite Element Method*,
McGraw-Hill.

Zienkiewicz, O.C. and D.V. Phillips, [1971], *An Automatic Mesh Generation Scheme for Plane and Curved Surfaces by 'Isoparametric' Co-ordinates*,
International J. for Numerical Methods in Engineering, Vol. 3, pp.519-528.

Zienkiewicz, O.C. and D.W. Kelly, [1982], *Finite Elements A Unified Problem-Solving and Information Transfer Method*,
In Finite Elements in Biomechanics, ed. by R.H. Gallagher et al., Wiley.

Zienkiewicz, O.C. and Y.K. Cheung, [1965], *Finite Elements in the Solution of Field Problems*,
The Engineer, Vol. 220, pp.507-510.

APPENDICES

TABLE OF CONTENTS

A1. Sample Programs that Demonstrate the Existing Problem of Limited Stack Size

```
00001000  C       EXAMPLE  1
00002000  C       A SAMPLE PROGRAM TO DEMONSTRATE THE LIMITIMG STACK PROBLEM
00003000  C       THIS PROGRAM WILL NOT BE COMPILED.
00004000  C       THE DIMENSION OF THE NEXT STATEMENT IS TOO BIG
00005000          DIMENSION  A(20000)
00006000          STOP
00007000          END
*** ERROR  128 ***  ARRAY EXCEEDS 32767 WORDS


  SYMBOL MAP

NAME              TYPE       STRUCTURE   ADDRESS                    NAME

A                 REAL       ARRAY       Q+%1   ,I


****   1 ERROR,   NO WARNINGS  ****
PROGRAM UNIT MAIN' FLUSHED
```

```
PAGE 0001   HP32102B.01.05  FORTRAN/3000  (C) HEWLETT-PACKARD CO. 1980  MON, J

00001000  C       EXAMPLE 2
00002000  C       A SAMPLE PROGRAM TO DEMONSTRATE THE LIMITIMG STACK PROBLEM
00003000  C       THIS PROGRAM WILL NOT BE COMPILED
00004000  C       THE DIMENSION OF THE NEXT STATEMENT IS TOO BIG
00005000          DIMENSION  A(33000)
                             ^
*** ERROR   77 ***  INTEGER EXCEEDS CONTEXTUAL LIMITS
00006000          STOP
00007000          END


  SYMBOL MAP

NAME              TYPE        STRUCTURE   ADDRESS                    NAME

A                 REAL        ARRAY


****   1 ERROR,   NO WARNINGS  ****
PROGRAM UNIT MAIN' FLUSHED
```

```
00001000  C       EXAMPLE 3
00002000  C       A SAMPLE PROGRAM TO DEMONSTRATE THE LIMITIMG STACK PROBLEM
00003000  C       THIS PROGRAM WILL NOT BE COMPILED
00004000  C       THE DIMENSIONS IN THE NEXT STATEMENT ARE TOO BIG
00005000          DIMENSION  A(10000),B(6000),K(2000)
00006000          STOP
00007000          END
*** ERROR  212 ***  DATA SPACE OVERFLOW


  SYMBOL MAP

NAME              TYPE       STRUCTURE   ADDRESS                    NAME

A                 REAL       ARRAY       Q+%2   ,I                  B
K                 INTEGER    ARRAY       Q+%1   ,I


****   1 ERROR,   NO WARNINGS  ****
PROGRAM UNIT MAIN' FLUSHED
```

A2. Samples of the Outputs Produced by the VSF Compiler

```
          VIRTUAL STACK FACILITY              MON, JUN 29, 1987,  1:09 PM
                                V  S  F
(C) Ahmed Sharaf Eldin Ahmed.  APRIL 84
PHASE I:  COMPILING OF THE VSF BLOCK.
$ARRAY
$LIST
$COMMENT A TEST EXAMPLE WITH AN ERROR IN THE COMMAND
$COMMENT '$MAP' WHICH IS MISSPELLED AS '$MOP'
$COMMENT
$COMMENT
$MOP
**ERROR** CANNOT RECOGNISE VSF COMMAND.
    AVAILABLE VSF COMMANDS ARE: $ARRAY,$NOLIST,$NOMAP,$INITIALIZE,
    $CROSSREF,$COMMENT,$MAP,$LIST AND $END.
$CROSSREF
INTEGER A(10000,200),KLM(20000),AGF(20,1000,2000)
REAL    STIFFNESS(2000)
**ERROR** ILLEGAL VSF ARRAY NAME.
    A LEGAL VSF NAME MUST BE 1 TO 6 CHARACTERS AND START
    WITH ALPHABET,CONTAINS NO SPECIAL SYMBOLS AND NOT A FORTRAN
    KEY WORD.
**ERROR** ILLEGAL VIRTUAL ARRAY SPECIFICATIONS
DOUBLE DEFLEC(2000)
COMPLEX CSOS(200,100,10)
**ERROR** ILLEGAL VSF STATEMENT.
    AVAILABLE VSF TYPE DECLARATIONS ARE: INTEGER,REAL,DOUBLE
     AND SIZE.
    A SPACE MUST BE LEFT AFTER THE KEYWORD.
$END
END OF PHASE I.
CPU TIME FOR VSF COMPILATION :  1.07100      SECONDS
     4 ERRORS ARE FOUND IN THE VSF BLOCK.
PLEASE CORRECT THE VSF BLOCK AND RE-COMPILE AGAIN.
```

```
          VIRTUAL STACK FACILITY                MON, JUN 29, 1987,  1:09 PM
                                V  S  F
(C) Ahmed Sharaf Eldin Ahmed.  APRIL 84
PHASE I:  COMPILING OF THE VSF BLOCK.
$ARRAY
$LIST
$MAP
$CROSSREF
REAL    A(640,640),B(640,640),C(640,640)
$END
END OF PHASE I.
CPU TIME FOR VSF COMPILATION :   .537000      SECONDS
NO ERRORS ARE FOUND.
```

VIRTUAL ARRAY	TYPE	DIMENSION	BOUNDS			TOTAL SIZE IN WORDS
A	REAL	2	640	640	0	819200
B	REAL	2	640	640	0	819200
C	REAL	2	640	640	0	819200

VIRTUAL ARRAY	START SECTOR	END SECTOR
A	1	6400
B	6401	12800
C	12801	19200

3 VIRTUAL ARRAYS ARE DECLARED.

REQUESTED REAL STACK SIZE = 0 WORDS

TOTAL SIZE OF VIRTUAL ARRAYS = 2457600 WORDS

```
                              V  S  F
(C) Ahmed Sharaf Eldin Ahmed.  APRIL 84
PHASE I:  COMPILING OF THE VSF BLOCK.
$ARRAY
$LIST
$COMMENT A TEST EXAMPLE
$COMMENT
$COMMENT
$MAP
$CROSSREF
INTEGER A(10000,200),KLM(20000),AGF(20,1000,2000)
REAL    STIFNS(2000)
DOUBLE DEFLEC(2000)
SIZE 12000
$END
END OF PHASE I.
CPU TIME FOR VSF COMPILATION :   .831000      SECONDS
NO ERRORS ARE FOUND.
```

VSF MAP

VIRTUAL ARRAY	TYPE	DIMENSION	BOUNDS			TOTAL SIZE IN WORDS
A	INTEGER	2	10000	200	0	2000000
KLM	INTEGER	1	20000	0	0	20000
AGF	INTEGER	3	20	1000	2000	40000000
STIFNS	REAL	1	2000	0	0	4000
DEFLEC	DOUBLE	1	2000	0	0	8000

VIRTUAL ARRAY	START SECTOR	END SECTOR
A	1	15625
KLM	15626	15782
AGF	15783	328282
STIFNS	328283	328314
DEFLEC	328315	328377

5 VIRTUAL ARRAYS ARE DECLARED.

REQUESTED REAL STACK SIZE = 12000 WORDS

TOTAL SIZE OF VIRTUAL ARRAYS = 42032000 WORDS

```
          VIRTUAL STACK FACILITY                MON, JUN 29, 1987,  1:13 PM
                                V  S  F
(C) Ahmed Sharaf Eldin Ahmed.  APRIL 84
PHASE I:  COMPILING OF THE VSF BLOCK.
$ARRAY
$LIST
$COMMENT A TEST EXAMPLE
$COMMENT
$COMMENT
$MAP
$CROSSREF
INTEGER A(10000,200),KLM(20000),AGF(20,1000,2000)
REAL    STIFNS(2000)
DOUBLE DEFLEC(2000)
SIZE 12000
$END
END OF PHASE I.
CPU TIME FOR VSF COMPILATION :   .806000      SECONDS
NO ERRORS ARE FOUND.
```

VSF MAP

VIRTUAL ARRAY	TYPE	DIMENSION	BOUNDS			TOTAL SIZE IN WORDS
A	INTEGER	2	10000	200	0	2000000
KLM	INTEGER	1	20000	0	0	20000
AGF	INTEGER	3	20	1000	2000	40000000
STIFNS	REAL	1	2000	0	0	4000
DEFLEC	DOUBLE	1	2000	0	0	8000

VIRTUAL ARRAY	START SECTOR	END SECTOR
A	1	15625
KLM	15626	15782
AGF	15783	328282
STIFNS	328283	328314
DEFLEC	328315	328377

5 VIRTUAL ARRAYS ARE DECLARED.

REQUESTED REAL STACK SIZE = 12000 WORDS

TOTAL SIZE OF VIRTUAL ARRAYS = 42032000 WORDS

A3. Error Messages Produced by the VSF Compiler

ERROR MESSAGES FOR THE VSF COMPILER

```
!!FATAL ERROR!! FIRST RECORD IS NOT A '$ARRAY' STATEMENT.
**ERROR** CANNOT RECOGNISE VSF COMMAND.
    AVAILABLE VSF COMMANDS ARE: $ARRAY,$NOLIST,$NOMAP,$INITIALIZE,
    $CROSSREF,$COMMENT,$MAP,$LIST AND $END.
**ERROR** ILLEGAL VSF STATEMENT.
    AVAILABLE VSF TYPE DECLARATIONS ARE: INTEGER,REAL,DOUBLE
     AND SIZE.
    A SPACE MUST BE LEFT AFTER THE KEYWORD.
**ERROR** ILLEGAL VSF ARRAY NAME.
    A LEGAL VSF NAME MUST BE 1 TO 6 CHARACTERS AND START
    WITH ALPHABET,CONTAINS NO SPECIAL SYMBOLS AND NOT A FORTRAN
    KEY WORD.
**ERROR** ILLEGAL NUMBER OF SUBSCRIPTS IN A VSF ARRAY DECLARATION.
    NUMBER OF SUBSCRIPTS MUST BE 1 TO 3.
**ERROR** VSF ARRAY BOUNDS MUST BE POSITIVE INTEGERS.
**ERROR** DUPLICATE DECLARATION OF A VSF ARRAY.
**ERROR** NO DIMENSION IS GIVEN FOR A VSF ARRAY.
**ERROR** INITIALIZED VSF ARRAY IS NOT DECLARED.
**ERROR** MORE THAN 100 VIRTUAL ARRAYS ARE DECLARED.
**ERROR** YOUR MAIN FORTRAN PROGRAM CONTAINS ERRORS AND/OR WARNINGS
    VSF COMPILER REQUIRES A SYNTAX ERROR FREE MAIN PROGRAM.
!!FATAL ERROR!! NO $END IS FOUND.
    LAST STATEMENT IN THE VIRTUAL STACK BLOCK MUST BE $END.
**ERROR** TOO MANY CONTINUATION LINES.
    VSF LIMITS THE NUMBER OF CONTINUATION LINES TO 10 IN THE MAIN
    FORTRAN PROGRAM.
**ERROR** ILLEGAL VSF STATEMENT
**ERROR** ILLEGAL VIRTUAL ARRAY SPECIFICATIONS
**ERROR** ILLEGAL REAL STACK SIZE
**ERROR** REAL STACK SIZE IS TOO BIG ()32000 WORDS).
**ERROR** AN INTEGER IS EXPECTED AS ARRAY DIMENSION.
*WARN* REAL STACK SIZE IS TOO SMALL TO KEEP REQUESTED SEGMENT
```

B. Programs for the Problems Solved in Chapter 7

```
 1  4   5 32  1
****    PROBLEM 7-1.
****    AQUIFER ANALYSIS
****    CASE NO. :     1
****    CONFINED AQUIFER, STEADY STATE, CONSTANT TRANSMISSIVITY
****    B.C.  PRESCRIBED HEAD ON BOUNDARY
****    MODEL DATA :
****       AQUIFER LENGTH                 =  10000  M.
****               WIDTH                  =  1000   M.
****               THICKNESS              =  20     M.
****               TRANSMISSIVITY  TXX    =  20000  M2/DAY
****               TRANSMISSIVITY  TYY    =  20000  M2/DAY
****       BOUNDARY CONDITIONS :
****           HEAD ON SIDE 2  IS  50 M.
****           HEAD ON SIDE 4  IS  100 M.
****           NO FLOW ACROSS THE LONG SIDES.
****    TO TEST THE EFFECT OF NO. OF TRIANGLES ON   CPU TIME(H-REFINMENT)
****    SEVERAL RUNS ARE DONE WITH MULTIPLE MAX. NO. OF TRIANGLES
****    NTF TAKES THE VALUES OF 8,16,32,64,128,256
****    OTHER FACTORS TESTED ARE :
****       - THE OUT-OF-CORE  COMPUTATION VERSUS IN-CORE IN TERMS OF
****         TIME,CORE REQUIREMENTS.
****       - THE SHAPE OF TRIANGLES EFFECT.
****       - THE INCREASED POLYNOMIAL (P-REFINEMENT) EFFECT ON CPU TIME
****         AND PRECISION.
OXX      20000.0*UX
OXY      20000.0*UY
SYMMETRIC
SHAPE    1
PLOT     1
NX       20
NY       2
XA       0
YA       0
HX       500
HY       500
ARC=     1
GBI      0
ARC=     -2
FBI      50.0
ARC=     3
GBI      0
ARC=     -4
FBI      100
VXY      0 0     10000.0 0      10000.0 1000.    0. 1000.    5000. 500.
IABC     1 2 5    2 3 5    3 4 5   4 1 5
I          1        -2       3       -4
END.
```

```
 1  4  5  32  1
****   PROBLEM 7-2.
****   AQUIFER ANALYSIS
****   UNCONFINED AQUIFER, STEADY STATE
****   NONLINEAR AQUIFER, THE VARIABLE IS H**2 RATHER THAN H.
****   B.C.  PRESCRIBED HEAD ON TWO SIDES
****   B.C.  NO FLUX ON THE OTHER TWO SIDES.
****   MODEL DATA :
****      AQUIFER LENGTH                 =  10000  M.
****              WIDTH                  =  1000   M.
****              THICKNESS              =  120    M.
****              PERMEABILITY    KXX    =  1000   M/DAY
****              PERMEABILITY    KYY    =  1000 M/DAY
****      BOUNDARY CONDITIONS :
****          HEAD ON SIDE 2  IS  50 M.
****          HEAD ON SIDE 4  IS  100 M.
****   TO TEST THE EFFECT OF NO. OF TRIANGLES ON   CPU TIME,
****   SEVERAL RUNS ARE DONE WITH MULTIPLE MAX. NO. OF TRIANGLES
****   NTF TAKES THE VALUES OF 8,16,32,64,128,256.
OXX     1000.*UX
OXY     1000.*UY
NX      20
NY      2
XA      0
YA      0
HX      500
HY      500
UPRINT  DSQRT(U)
ARC=    -2
FBI     2500.
ARC=    -4
FBI     10000.
VXY     0 0    10000.0 0     10000.0 1000.  0. 1000.  5000. 500.
IABC    1 2 5   2 3 5   3 4 5  4 1 5
I         1       -2      3      -4
END.
```

```
 1  4   5  32 1
****   PROBLEM 7-3.
****   AQUIFER ANALYSIS
****   CASE NO. :     3
****   CONFINED AQUIFER,NO WELLS,STEADY STATE,CONSTANT TRANSMISSIVITY
****   WITH LEAKY INFLOWS FROM AN ADJACENT AQUIFER
****   B.C.  PRESCRIBED HEAD ON BOUNDARY
****   MODEL DATA :
****      AQUIFER LENGTH                 =  10000  M.
****              WIDTH                  =  1000   M.
****              THICKNESS              =  20     M.
****              TRANSMISSIVITY  TXX    =  20000  M2/DAY
****              TRANSMISSIVITY  TYY    =  20000  M2/DAY
****              HEAD IN ADJACENT AQUIFER  PHI A IS :
****                                     =  95 M.
****              K       :   PERMEABILITY OF LEAKY AQUIFER   = .0015
****              B DASH  :   THICKNESS OF THE LEAKEY LAYER = 10 M.
****              K DASH/B DASH I.E. THE LEAKY AQUIFER PARAMETER IS :
****                                     =  .00015 DAY(-1)
****      BOUNDARY CONDITIONS :
****          HEAD ON SIDE 2  IS  90 M.
****          HEAD ON SIDE 4  IS  100 M.
****   TO TEST THE EFFECT OF NO. OF TRIANGLES ON   CPU TIME,
****   SEVERAL RUNS ARE DONE WITH MULTIPLE MAX. NO. OF TRIANGLES
****   NTF TAKES THE VALUES OF 8,16,32,64,128,256.
OXX     20000.0*UX
OXX/UX  20000.0
OXX/UY  0
OXY     20000.0*UY
OXY/UX  0
OXY/UY  20000.
FI      .00015*(95-U)
SYMMETRIC
SHAPE   1
PLOT    1
NX      20
NY      2
XA      0
YA      0
HX      500
HY      500
ARC=    1
GBI     0
ARC=    -2
FBI     90.0
ARC=    3
GBI     0
ARC=    -4
FBI     100.0
VXY     0 0    10000.0 0      10000.0 1000.  0. 1000.  5000. 500.
IABC    1 2 5   2 3 5   3 4 5   4 1 5
I         1       -2       3       -4
END.
```

```
 1  3   4  64    1
****    PROBLEM 7-4.
****    AQUIFER ANALYSIS
****    CASE NO. :     5
****    CONFINED AQUIFER,RADIAL FLOW TO A WELL WITH NO LEAKAGE
****    B.C.  PHI = PHIO AS R --> INFENITY IS 20.
****    MODEL DATA :
****       AQUIFER RADUIS                =  4000    M.
****               THICKNESS             =  50      M.
****               TRANSMISSIVITY  TXX   =  400 M2/DAY
****               TRANSMISSIVITY  TYY   =  400 M2/DAY
****               STORATIVITY     S     =  .001
****       WELL DISCHARGE IS 2000 M3/DAY
****       BOUNDARY CONDITIONS :
****           HEAD AT TIME T IS = 0 IS 20 M.
****       ONLY ONE QUARTER IS CONSIDERED WITH WELL DISCHARGE = 500
****       Q/T = 500/400 = 1.25
****       ONLY ONE QUARTER IS CONSIDERED
OXX      UX
OXY      UY
F1       FUN(X,Y)
SHAPE    1
PLOT     1
D3EST    1.0/(1.0+X*X+Y*Y)
NX       20
NY       20
XA       0
YA       0
HX       200
HY       200
ARC=     1
GB1      0
ARC=     -2
X        4000.*DCOS(1.570945*S)
Y        4000.*DSIN(1.570945*S)
FB1      20.0
ARC=     3
GB1      0
VXY      0 0     4000.0 0    0  4000.0     1000. 1000.
IABC     1 2 4     2 3 4     3 1 4
I          1         -2         3
ADD.
          DOUBLE PRECISION FUNCTION FUN(X,Y)
          DOUBLE PRECISION X,Y
          FUN=0.
          IF(X.LE.100. .AND. Y.LE. 100)FUN=-.0001*1.25
          RETURN
          END
END.
```

```
 1  4   5 64 1
****    PROBLEM  7-5.
****    AQUIFER ANALYSIS
****    CONFINED AQUIFER, TRANSIENT STATE, CONST. TRANSMISSIVITY
****    B.C.  PRESCRIBED HEAD ON BOUNDARY
****    MODEL DATA :
****       AQUIFER LENGTH                =  10000  M.
****               WIDTH                 =  100    M.
****               THICKNESS             =  20     M.
****               TRANSMISSIVITY  TXX   =  20000  M2/DAY
****               TRANSMISSIVITY  TYY   =  20000  M2/DAY
****               STORATIVITY     S     =  .001
****       BOUNDARY CONDITIONS :
****           HEAD ON ALL SIDES AT TIME T = 0 IS 95  M.
****           I.E. HEAD H = 95  FOR ALL X,Y AT T = 0.
****           HEAD = 95  AT X = 0,  T> 0.
****           HEAD = 90 AT X = 10000,  T> 0.
****           DH-DY = 0 AT Y = 0 AND Y = 100.
****    TO TEST THE EFFECT OF NO. OF TRIANGLES WITH CPU TIME,
****    SEVERAL RUNS ARE DONE WITH MULTIPLE MAX. NO. OF TRIANGLES
****    NTF TAKES THE VALUES OF 8,16,32,64,128,256.
OXX      20000.0*UX
OXX/UX   20000.0
OXX/UY   0
OXY      20000.0*UY
OXY/UX   0
OXY/UY   20000.
C1       .001
UO       95.
NUPDT    0
SHAPE    1
PLOT     1
TF       1
DT       FUNT(T)
DTINV    1.0/DSQRT(T)
NX       4
NY       2
XA       0
YA       0
HX       2500
HY       50
ARC=     1
GB1      0
ARC=     -2
FB1      90.0
ARC=     3
GB1      0
ARC=     -4
FB1      95.0
VXY      0 0    10000.0 0     10000.0 100.   0. 100.   5000. 50.
IABC     1 2 5   2 3 5   3 4 5  4 1 5
I          1       -2       3       -4
ADD.
         DOUBLE PRECISION FUNCTION FUNT (T)
         DOUBLE PRECISION T
         IF (T.EQ.0)GOTO10
         IF (T.LT.1)GOTO20
```

```
      IF(T.GE.1 .AND. T.LT. 2)GOTO30
      IF(T.GE.2 .AND. T.LT.10)GOTO40
      IF (T.GE.10)GOTO50
10    FUNT=.0005
      RETURN
20    FUNT=.005
      RETURN
30    FUNT=.05
      RETURN
40    FUNT=.5
      RETURN
50    FUNT=10.
      RETURN
      END
END.
```

```
 1  4   5 32 1
****    PROBLEM  7-6.
****    AQUIFER ANALYSIS
****    PHREATIC AQUIFER, TRANSIENT STATE, CONST. TRANSMISSIVITY
****    B.C.  PRESCRIBED HEAD ON BOUNDARY
****    MODEL DATA :
****       AQUIFER LENGTH                 =  10000  M.
****               WIDTH                  =  100    M.
****               THICKNESS              =  100    M.
****               PERMEABILITY K         =  10000  M/DAY
****               STORATIVITY      S     =  .1
****       BOUNDARY CONDITIONS :
****           HEAD ON ALL SIDES AT TIME T = 0 IS 100 M.
****           I.E. HEAD H = 100 FOR ALL X,Y AT T = 0.
****           HEAD = 100 AT X = 0,  T> 0.
****           HEAD = 50 AT X = 10000,  T> 0.
****           DH/DY = 0 AT Y = 0 AND Y = 1000.
****    TO TEST THE EFFECT OF NO. OF TRIANGLES WITH CPU TIME,
****    SEVERAL RUNS ARE DONE WITH MULTIPLE MAX. NO. OF TRIANGLES
****    NTF TAKES THE VALUES OF 8,16,32,64,128,256.
OXX      10000.0*U*UX
OXX/UX   10000.0*U
OXX/UY   0
OXY      10000.0*U*UY
OXY/UX   0
OXY/UY   10000.*U
C1       .1
UO       100.
NUPDT    0
SHAPE    1
PLOT     1
TF       3
DT       .1
DTINV    1.0/DSQRT(T)
NX       4
NY       2
XA       0
YA       0
HX       2500
HY       50
ARC=     1
GB1      0
ARC=     -2
FB1      50.0
ARC=     3
GB1      0
ARC=     -4
FB1      100.0
VXY      0 0     10000.0 0      10000.0 100.    0. 100.    5000. 50.
IABC     1 2 5    2 3 5    3 4 5   4 1 5
I          1        -2       3       -4
ADD.
         DOUBLE PRECISION FUNCTION FUNT (T)
         DOUBLE PRECISION T
         IF (T.EQ.0)GOTO10
         IF (T.LT.1)GOTO20
         IF(T.GE.1 .AND. T.LT. 2)GOTO30
```

```
      IF(T.GE.2 .AND. T.LT.10)GOTO40
      IF (T.GE.10)GOTO50
10    FUNT=.0005
      RETURN
20    FUNT=.005
      RETURN
30    FUNT=.05
      RETURN
40    FUNT=.5
      RETURN
50    FUNT=10.
      RETURN
      END
END.
```

```
 1  8  9 64  1
****   PROBLEM 7-7. (FIRST ITERATION)
****   A FREE SURFACE PROBLEM
****   THE CLASSICAL DAM PROBLEM
****   SOLVED ITERATIVELY BY THE TRIAL FREE SURFACE METHOD
OXX     UX
OXY     UY
SHAPE   1
PLOT    1
NX      4
NY      8
XA      0
YA      0
HX      .1666667
HY      .125
D3EST   1.0/(1.+(X-.6667)**2+(Y-.5)**2)
ARC=    -2
FB1     .166667
ARC=    -3
FB1     Y
ARC=    -8
FB1     1
VXY     0,0  .66667,0  .66667,.16667 .66667,.5 .5,.625 .33334,.75
VXY     ,.166667,.875  0,1   .33333,.5
IABC    1 2 9   2 3 9   3 4 9  4 5 9  5 6 9  6 7 9  7 8 9  8 1 9
I         1       -2     -3      4       5       6       7     -8
END.
```

```
 1  8  9 64  1
****   PROBLEM 7-7. (SECOND ITERATION)
****   A FREE SURFACE PROBLEM
****   THE CLASSICAL DAM PROBLEM
****   SOLVED ITERATIVELY BY THE TRIAL FREE SURFACE METHOD
OXX     UX
OXY     UY
SHAPE   1
PLOT    1
NX      10
NY      10
XA      0
YA      0
HX      .06666667
HY      .1
D3EST   1.0/(1.+(X-.6667)**2+(Y-.5)**2)
ARC=    -2
FB1     .166667
ARC=    -3
FB1     Y
ARC=    -8
FB1     1
VXY     0,0  .66667,0  .66667,.16667 .66667,.5 .5,.67571 .33334,.83666
VXY     ,.166667,.94950  0,1   .33333,.5
IABC    1 2 9   2 3 9   3 4 9  4 5 9  5 6 9  6 7 9  7 8 9  8 1 9
I         1       -2      -3      4       5       6       7      -8
END.
```

```
 1  14  15  120  1
****   PROBLEM 7-8. (FIRST ITERATION)
****   A FREE SURFACE PROBLEM
****   FLOW TOWARDS A WELL IN PHREATIC AQUIFER
****   SOLVED ITERATIVELY BY THE TRIAL FREE SURFACE METHOD
OXX     UX
OXY     UY
SHAPE   1
PLOT    1
NX      10
NY      10
XA      0
YA      0
HX      100.0
HY      10.0
D3EST   1.0/(1.+X*X+(Y-20.0)**2)
ARC=    -2
FB1     100
ARC=    -13
FB1     Y
ARC=    -14
FB1     20
VXY     0,0  1000.,0.  1000.,100.  900.,95.  800.,90.  700.,85.
VXY     ,600.,80.  500.,75.  400.,70.  300.,65.  200.,60.  100.,55.
VXY     ,0.0,50.  0.0,20.  500.,20.
IABC    1 2 15  2 3 15  3 4 15  4 5 15  5 6 15  6 7 15  7 8 15  8 9 15
IABC    ,9 10 15  10 11 15  11 12 15  12 13 15  13 14 15  14 1 15
I         1       -2      3        4       5       6       7       8
I        ,9         10          11        12      -13      -14
END.
```

```
 1  4   5   50   1
****   PROBLEM  7-9.
****   A SMALL WATERSHED BOUNDED ON THREE SIDES BY NO FLOW CONDITION
****   THE UPPER BOUND IS APPROXIMATED BY A HORIZONTAL LINE
****   WITH THE HEAD = ACTUAL HEIGHT OF WATER TABLE
****   THE REGION ITSELF IS A RECTANGLE OF S*YO SIDES
****   S = 200  M.,  Y = 100  M.  AND    C=.02
****      BOUNDARY CONDITIONS :
****           DH/DX =  0 ON THE TWO VERTICAL SIDES
****           DH/DY =  0 AT THE GROUND (IMPERMEABLE BASEMENT).
****           H = C*X +YO  ON THE UPPER BOUND.
OXX      UX
OXX/UX   1
OXY      UY
OXY/UY   1
PLOT     1
NX       10
NY       5
XA       0
YA       0
HX       20
HY       20
ARC=     -3
FB1      0.02*X+100.0
VXY      0 0    200,0    200,100   0,100   100,50
IABC     1 2 5    2 3 5    3 4 5   4 1 5
I          1        2       -3       4
END.
```

```
 1  3  4  64   1
****   PROBLEM 7-10.
****   AQUIFER ANALYSIS
****   CONFINED AQUIFER,TRANSIENT RADIAL FLOW TO A WELL WITH NO LEAKAGE
****   B.C.  PHI = PHIO AS R --> INFENITY IS 100.
****   MODEL DATA :
****      AQUIFER RADUIS                =  10000   M.
****              THICKNESS             =  100     M.
****              TRANSMISSIVITY  TXX   =  1E5 M2/DAY
****              TRANSMISSIVITY  TYY   =  1E5 M2/DAY
****              STORATIVITY     S     =  .001
****      WELL DISCHARGE IS 1E6  M3/DAY
****      BOUNDARY CONDITIONS :
****          HEAD AT TIME T IS = 0 IS 100 M.
****      ONLY ONE QUARTER IS CONSIDERED WITH WELL DISCHARGE = 250000
OXX     100000.0*UX
OXY     100000.0*UY
F1      FUN(X,Y)
CI      .001
U0      100.
SHAPE   1
PLOT    1
D3EST   1.0/(1.0+X*X+Y*Y)
NX      10
NY      10
XA      0
YA      0
HX      400
HY      400
TF      1
DT      .1
DTINV   1.0/DSQRT(T)
ARC=    1
GBI     0
ARC=    -2
X       10000.*DCOS(1.570945*S)
Y       10000.*DSIN(1.570945*S)
FBI     100.
ARC=    3
GBI     0
VXY     0 0    10000.0 0   0  10000.0    5000. 2500.
IABC    1 2 4    2 3 4    3 1 4
I         1       -2       3
ADD.
         DOUBLE PRECISION FUNCTION FUN(X,Y)
         DOUBLE PRECISION X,Y
         FUN=0.
         IF(X.LE.100. .AND. Y.LE. 100)FUN=-25.
         RETURN
         END
END.
```

```
 1  3   4  64    1
****    PROBLEM 7-11.
****    AQUIFER ANALYSIS
****    CONFINED AQUIFER,TRANSIENT RADIAL FLOW TO A WELL WITH  LEAKAGE
****    B.C.  PHI = PHIO AS R --> INFENITY IS 100.
****    MODEL DATA :
****       AQUIFER RADUIS                =  10000    M.
****               THICKNESS             =  100      M.
****               TRANSMISSIVITY  TXX   =  1E5 M2/DAY
****               TRANSMISSIVITY  TYY   =  1E5 M2/DAY
****               STORATIVITY     S     =  .001
****               LEAKY LAYER PARAMETER K DASH/B DASH = .1 DAY-1.
****               ADJACENT HEAD = 95  M.
****       WELL DISCHARGE IS 1E6  M3/DAY
****       BOUNDARY CONDITIONS :
****           HEAD AT TIME T IS = 0 IS 100 M.
****       ONLY ONE QUARTER IS CONSIDERED WITH WELL DISCHARGE = 250000
OXX      100000.0*UX
OXY      100000.0*UY
F1       FUN(X,Y)
C1       .001
U0       100.
SHAPE    1
PLOT     1
O3EST    1.0/(1.0+X*X+Y*Y)
NX       10
NY       10
XA       0
YA       0
HX       500
HY       500
TF       1
DT       .1
DTINV    1.0/DSQRT(T)
ARC=     1
GB1      0
ARC=     -2
X        10000.*DCOS(1.570945*S)
Y        10000.*DSIN(1.570945*S)
FB1      100.
ARC=     3
GB1      0
VXY      0 0     10000.0 0    0  10000.0     5000. 2500.
IABC     1 2 4     2 3 4     3 1 4
I          1         -2         3
ADD.
          DOUBLE PRECISION FUNCTION FUN(X,Y,U)
          DOUBLE PRECISION X,Y,U
C         THE LEAKY LAYER PARAMETER IS .1, THE ADJACENT AQUIFER HEAD =100
          FUN=.1*(U-95. )
          IF(X.LE.100. .AND. Y.LE. 100)FUN=FUN-100.
          RETURN
          END
END.
```

```
 1  61 44 250 2
****    PROBLEM 7-12.
****    ANISOTROPIC AQUIFER ANALYSIS
****    CONFINED AQUIFER, STEADY STATE
****    B.C.  PRESCRIBED HEAD ON PART OF BOUNDARY
****          AND NO FLOW ACROSS THE REST OF BOUNDARY
****    MODEL DATA :
****       AQUIFER TOPOLOGY AS IN THE TEXT AND PROGRAM .
****               THICKNESS              =  50      M.
****               TRANSMISSIVITY  IS ACCORDING TO ZONE (X,Y).
****       BOUNDARY CONDITIONS AS IN THE TEXT AND PROGRAM.
****       A WELL IS LOCATED AT NODE 44 WITH Q = 2400 M3/DAY.
OXX      TRANSX(X,Y)*UX
OXY      TRANSY(X,Y)*UY
FI       WELL(X,Y)
PLOT     1
NX       13
NY       9
XA       0
YA       0
HX       4000
HY       4000
D3EST    1.0/(1.0+X*X+Y*Y)
ARC=     1
GBI      0
ARC=     2
GBI      0
ARC=     3
GBI      0
ARC=     4
GBI      0
ARC=     5
GBI      0
ARC=     6
GBI      0
ARC=     7
GBI      0
ARC=     8
GBI      0
ARC=     14
GBI      0
ARC=     15
GBI      0
ARC=     16
GBI      0
ARC=     -9
FBI      60.28-.07*(Y-12000.0)*.001
ARC=     -10
FBI      60+2.5*(X-44000)*.001
ARC=     -11
FBI      70-13.-8.*(Y-16000)*.001
ARC=     -12
FBI      57+.5*(X-48000)*.001
ARC=     -13
FBI      59-.5*(Y-24000)*.001
ARC=     -17
FBI      3+.05*(Y-36000)*.001
```

```
ARC=    -18
FBL     2.8-8*(X-40001*.001
ARC=    -19
FBL     0
ARC=    -20
FBL     0
ARC=    -21
FBL     0
ARC=    -22
FBL     .5*X*.001
ARC=    -23
FBL     1.5+Y*.001
VXY       4000,0      12000,0      12000,4000   16000,4000    16000,8000
VXY     ,20000,8000  20000,12000  40000,12000  44000,12000
VXY     ,44000,16000  48000,16000  48000,24000  52000 24000
VXY     ,52000,32000  44000,32000  44000,36000  16000,36000 12000,36000
VXY     ,12000,32000  4000,32000  4000,24000  0000,24000  0000,12000
VXY     ,0000,4000  4000,4000  8000,4000 8000,12000  36000,16000
VXY     ,40000,16000  36000,20000  32000,20000 20000,20000
VXY     ,12000,20000  8000,24000  8000,28000  12000,28000
VXY     ,20000,24000  32000,24000  32000,28000  40000,28000
VXY     ,40000,32000  20000,32000  16000,32000  8000,16000
IABC     01 02 26     02 03 26     25 01 26     24 25 27     25 26 27
IABC    ,26 03 27     03 04 27     04 05 27     05 06 27     23 24 27
IABC    ,06 07 27     23 27 44     27 07 44     22 23 44     21 22 44
IABC    ,34 21 44     21 34 35     20 21 35     19 20 35     35 36 19
IABC    ,36 35 33     35 34 33     34 44 33     44 07 32     33 44 32
IABC    ,33 32 37     42 33 37     42 43 33     43 36 33     43 19 36
IABC    ,18 19 43     17 18 43     17 43 42     16 17 42     16 42 41
IABC    ,15 16 41     40 41 42     15 41 40     14 15 40     39 40 42
IABC    ,38 39 42     37 38 42     37 32 38     32 31 38     39 38 40
IABC    ,13 14 40     38 31 40     31 32 07     31 07 28     07 08 28
IABC    ,12 13 40     11 12 40     30 40 31     40 30 29     40 29 10
IABC    ,10 11 40     28 30 31     28 29 30     08 29 28     08 09 29
IABC    ,09 10 29
I           +01           +02           -23           -22           +00
I         ,   0             3             4             5           -21
I         ,   6             0             0           -21           -20
I         ,   0             0           -19           -18             0
I         ,   0             0             0             0             0
I         ,   0             0             0             0             0
I         , -17            16             0            16             0
I         ,  15             0             0            14             0
I         ,   0             0             0             0             0
I         , -13             0             0             0             7
I         , -12           -11             0             0             0
I         , -10             0             0             0             8
I         , -09
ADD.
          DOUBLE PRECISION FUNCTION WELL(X,Y)
          DOUBLE PRECISION X,Y
          WELL=0.DO
          IF(X.GE.7900 .AND. X.LE.8100  .AND.
     1       Y.GE.15900 .AND. Y.LE.16100 )WELL=-.24
          RETURN
          END
          DOUBLE PRECISION FUNCTION TRANSX(X,Y)
          DOUBLE PRECISION X,Y
          REAL TT(7)
          INTEGER KK
          DATA TT/542.68,112.3,135.75,754.64,290.93,69.12,120.96/
```

```
      KK=IZONE(X,Y)
      TRANSX=TT(KK)
      RETURN
      END
      DOUBLE PRECISION FUNCTION TRANSY(X,Y)
      DOUBLE PRECISION X,Y
      REAL TT(7)
      INTEGER KK
      DATA TT/5428.8,1123.,1857.6,7646.4,2908.8,691.2,1209.6/
      KK=IZONE(X,Y)
      TRANSY=TT(KK)
      RETURN
      END
      INTEGER FUNCTION IZONE(U,V)
      DOUBLE PRECISION U,V
      REAL X,Y
C     X AND Y ARE THE COORDINATES IN KM.
      X=.001*U
      Y=.001*V
      IF((X.GE.4 .AND. X.LE.12 .AND. Y.GE.0 .AND. Y.LE.4)
     *                .OR.
     *   (X.GE.0 .AND. X.LE.8  .AND. Y.GE.4 .AND. Y.LE.12))IZONE=1
C
      IF((X.GE.8 .AND. X.LE.16 .AND. Y.GE.4 .AND. Y.LE.12)
     *                .OR.
     *   (X.GE.16 .AND. X.LE.20  .AND. Y.GE.8 .AND. Y.LE.12))IZONE=2
C
      IF((X.GE. 20.AND. X.LE.32 .AND. Y.GE.12 .AND. Y.LE.24)
     *                .OR.
     *   (X.GE.32 .AND. X.LE.36  .AND. Y.GE.12 .AND. Y.LE.20)
     *                .OR.
     *   (X.GE.36 .AND. X.LE.40  .AND. Y.GE.12 .AND. Y.LE.16))IZONE=3
C
      IF((X.GE. 40.AND. X.LE.44 .AND. Y.GE.12 .AND. Y.LE.16)
     *                .OR.
     *   (X.GE.36 .AND. X.LE.48  .AND. Y.GE.16 .AND. Y.LE.20)
     *                .OR.
     *   (X.GE.32 .AND. X.LE.48  .AND. Y.GE.20 .AND. Y.LE.24)
     *                .OR.
     *   (X.GE.32 .AND. X.LE.52  .AND. Y.GE.24 .AND. Y.LE.28)
     *                .OR.
     *   (X.GE.40 .AND. X.LE.52  .AND. Y.GE.28 .AND. Y.LE.32))IZONE=4
C
      IF((X.GE. 20.AND. X.LE.32 .AND. Y.GE.24 .AND. Y.LE.28)
     *                .OR.
     *   (X.GE.20 .AND. X.LE.40  .AND. Y.GE.26 .AND. Y.LE.32)
     *                .OR.
     *   (X.GE.16 .AND. X.LE.44  .AND. Y.GE.32 .AND. Y.LE.36))IZONE=5
C
      IF((X.GE. 04.AND. X.LE.08 .AND. Y.GE.24 .AND. Y.LE.32)
     *                .OR.
     *   (X.GE.08 .AND. X.LE.12  .AND. Y.GE.28 .AND. Y.LE.32)
     *                .OR.
     *   (X.GE.12 .AND. X.LE.20  .AND. Y.GE.20 .AND. Y.LE.32)
     *                .OR.
     *   (X.GE.12 .AND. X.LE.16  .AND. Y.GE.32 .AND. Y.LE.36))IZONE=6
C
      IF((X.GE. 00.AND. X.LE.20 .AND. Y.GE.12 .AND. Y.LE.20)
     *                .OR.
     *   (X.GE.00 .AND. X.LE.12  .AND. Y.GE.20 .AND. Y.LE.24)
     *                .OR.

     *   (X.GE.08 .AND. X.LE.12  .AND. Y.GE.24 .AND. Y.LE.28))IZONE=7
      RETURN
      END
END.
```