AN INTERACTIVE BOUNDARY ELEMENT ANALYSIS SYSTEM

by

Fouad G. Tamer

B.E., Civil Engineering, American University of Beirut

(1983)

SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS OF THE

DEGREE OF

MASTER OF SCIENCE

IN CIVIL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1985

Signature of Author [illegible]
Department of Civil Engineering
May 16, 1985

Certified by [illegible]
Jerome J. Connor
Thesis Supervisor

Accepted by ______________________
Francois M.M. Morel
Chairman, Departmental Committee on Graduate Students

AN INTERACTIVE BOUNDARY ELEMENT ANALYSIS SYSTEM

by

FOUAD G. TAMER

Submitted to the Department of Civil Engineering
on May 16, 1985 in partial fulfillment of
the requirements for the Degree of Master of Science in
Civil Engineering

ABSTRACT

The framework for a general purpose boundary element analysis system is developed, and a capabilitiy for solving the potential problem is implemented. Primary emphasis is placed on creating a user-friendly environment for pre- and post-processing. Features such as totally menu driven, extensive graphics display, and dynamic memory allocation considerably enhance the program. The C language is used for the pre- and post-processing phases; FORTRAN is adopted for the numerical computation phase. A preliminary version of a rule-based accuracy advisor is also included.

Thesis Supervisor: Dr. Jerome J. Connor
Title: Professor of Civil Engineering

ACKNOWLEDGEMENTS

The Author would like to thank his parents for their continuous support, encouragement and understanding all through the years.

The author would also like to thank Professor Jerome J. Connor for his invaluable guidance and motivation throughout this thesis.

The author would also like to thank his friend Fadi Chehayeb for his help and suggestions.

The Author would also like to thank Ms. Pamela Howland for typing this thesis.

TO ALICE AND GEORGE

TABLE OF CONTENTS

PAGE

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 DISCRETE ELEMENT ANALYSIS

1.1.1 Discrete Systems

Problems occuring in the physical world can range from the most simple ones to the most complex and complicated ones. Unfortunately the human mind is limited in such a way that it cannot grasp the behavior of its complex surroundings and creations in one operation. Thus it resorts to other means of solving these problems. Computers and "artificial intelligence" contribute a big part in supporting the human intelligence. But before being able to use those powerful tools, humans subdivide all systems into their individual components or "elements", whose behavior is readily understood. And, then, the original system is rebuilt from these simple components to study its behavior.

In many situations an adequate model is obtained using a finite number of well-defined components. This is what is called a discrete system. In other systems you need an infinite number of elements to properly model the system, and the problem can only be defined using the mathematical fiction of an infinitesimal. Such systems are called continuous systems. But if computers are going to be used to help the human mind, it should be noted that the capacity of computers is finite. Discrete problems can be solved easily using computers even if the number of elements is very large. But exact solutions to continuous problems can only be obtained by mathematical manipulation.

To eliminate the unattractiveness of such manipulations, various methods of discretization were proposed for these continuum media. All involve an approximation which should get closer to the true solution as the number of discrete variables increases. This leads to a class of methods called discrete element methods. It includes finite difference approximations, finite element methods, and the more recently developed boundary element methods. The solution of continuum problems using those methods has been approached differently by mathematicians and engineers. The first have developed general techniques applicable directly to differential equations governing the problem. The engineer, on the other hand, approaches the problem more intuitively by creating an analogy between real discrete elements and finite portions of a continuum domain. Since the early 1960's much progress has been made, and the purely mathematical and 'analogy' approaches are fully reconciled.

On the other hand, in the analysis of problems of a discrete nature, a standard methodology has been developed over the years. The civil engineer, dealing with structures, first calculates his force-displacement relationships for each element of the structure and then proceeds to assemble the whole system according to a well-defined procedure of establishing local equilibrium at each 'node' or connecting point of the structure. From such equations the solution of the unknown displacements becomes possible. This sort of analysis follows a standard pattern which is universally adaptable to discrete systems.

Basically the continuum problem is solved by using similar techniques, by dividing it into a discrete number of elements with a finite number of parameters. All these different techniques can be termed approximate solution methods.

1.1.2 Approximate Solution Methods

Suppose there is a domain Ω which has a boundary Γ as shown in Fig. 1.1. Let the equation $\nabla^2 u = p$ be satisfied in Ω. This is equivalent to:

$$\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} = p \text{ in } \Omega$$

in a two-dimensional space. Let there be some boundary conditions on Γ. There are two different kinds of approximate solution methods: "Domain" methods and "Boundary" methods.

In domain methods an approximation $\hat{u}$ is introduced which satisfies exactly the boundary conditions, but not the differential equation in Ω. Among these methods are finite differences, finite elements, Galerkin, and Rayleigh Ritz methods. On the other hand in boundary methods the approximation $\hat{u}$ which is introduced, satisfies exactly the differential equation throughout Ω, but only satisfies in an approximate way the boundary conditions on Γ. One example of this class of methods is the boundary element method. In the next sections, discçussion will be focused on two major approximate solution methods: finite element and boundary element methods.

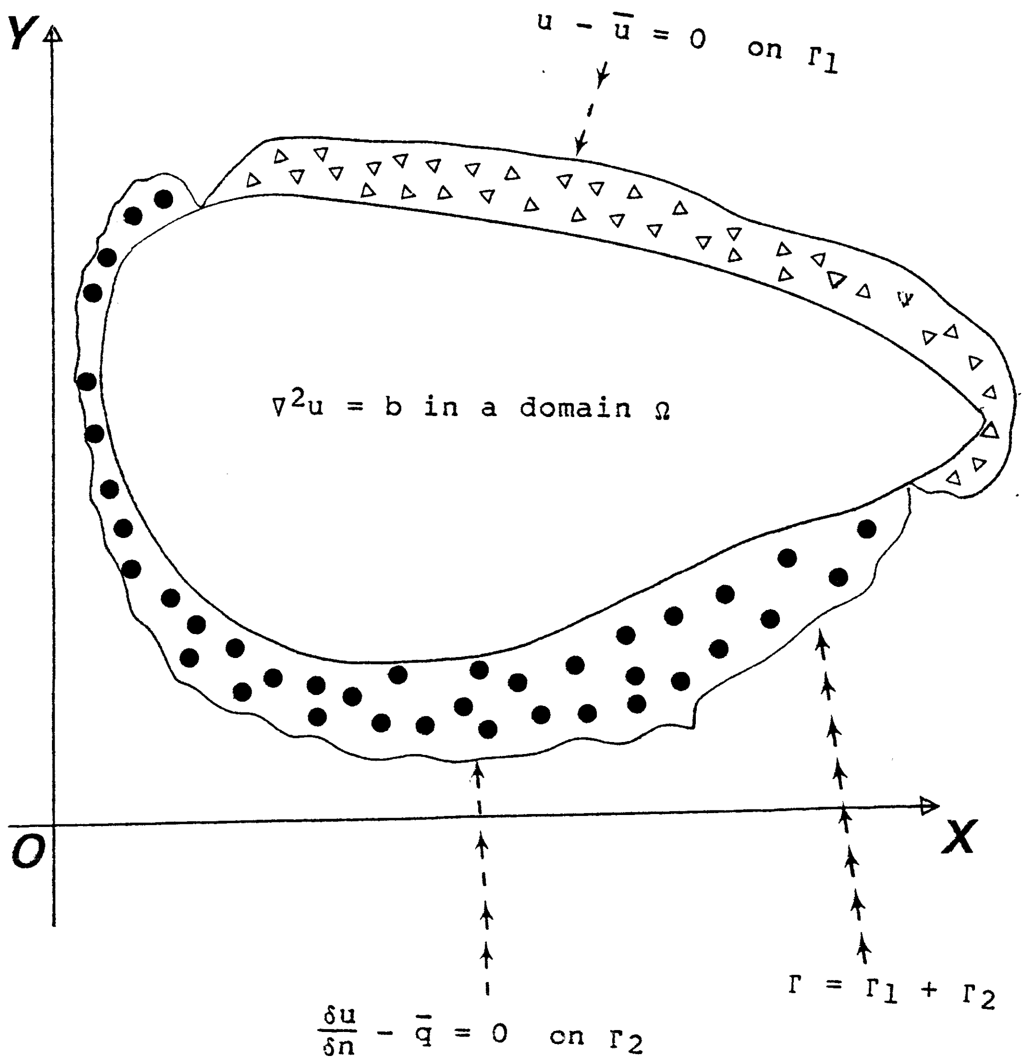

$u - \bar{u} = 0$ on Γ_1
$\nabla^2 u = b$ in a domain Ω
Y
X
O
$\Gamma = \Gamma_1 + \Gamma_2$
$\frac{\delta u}{\delta n} - \bar{q} = 0$ on Γ_2

FIGURE 1.1 Potential problem satisfying Poisson's equation.

1.1.3 The Finite Element Method

It is from the engineering "direct analogy" view that the finite element has been born. Table 1.1 shows how the mathematical point of view and the engineering ideas converged in the early 1960's to form the finite element method.[1] From the mathematical point of view finite differences lead to variational principles that introduced the present day finite element method.

From the engineering point of view one approach was to use variational methods and weighted residuals that lead in the late 1940's to piecewise continuous trial functions that could be applied to finite element methods. Another approach was the structural analogue substitution that originated direct continuum elements which found their application in finite element analysis. The basic concept of the finite element method is that any continuous quantity such as temperature, pressures, or displacements, can be approximated by a discrete model composed of a set of piecewise continuous functions defined over a finite number of subdomains [2]. The discrete model is constructed as follows:

(a) Choose a finite number of nodes in the domain.

(b) The value of the continuous quantity at each node is denoted as a variable to be determined.

(c) Divide the domain into subdomains called elements. These can be triangular, rectangular, or of other geometrical shape. They are connected at common nodes and approximate the shape of the domain.

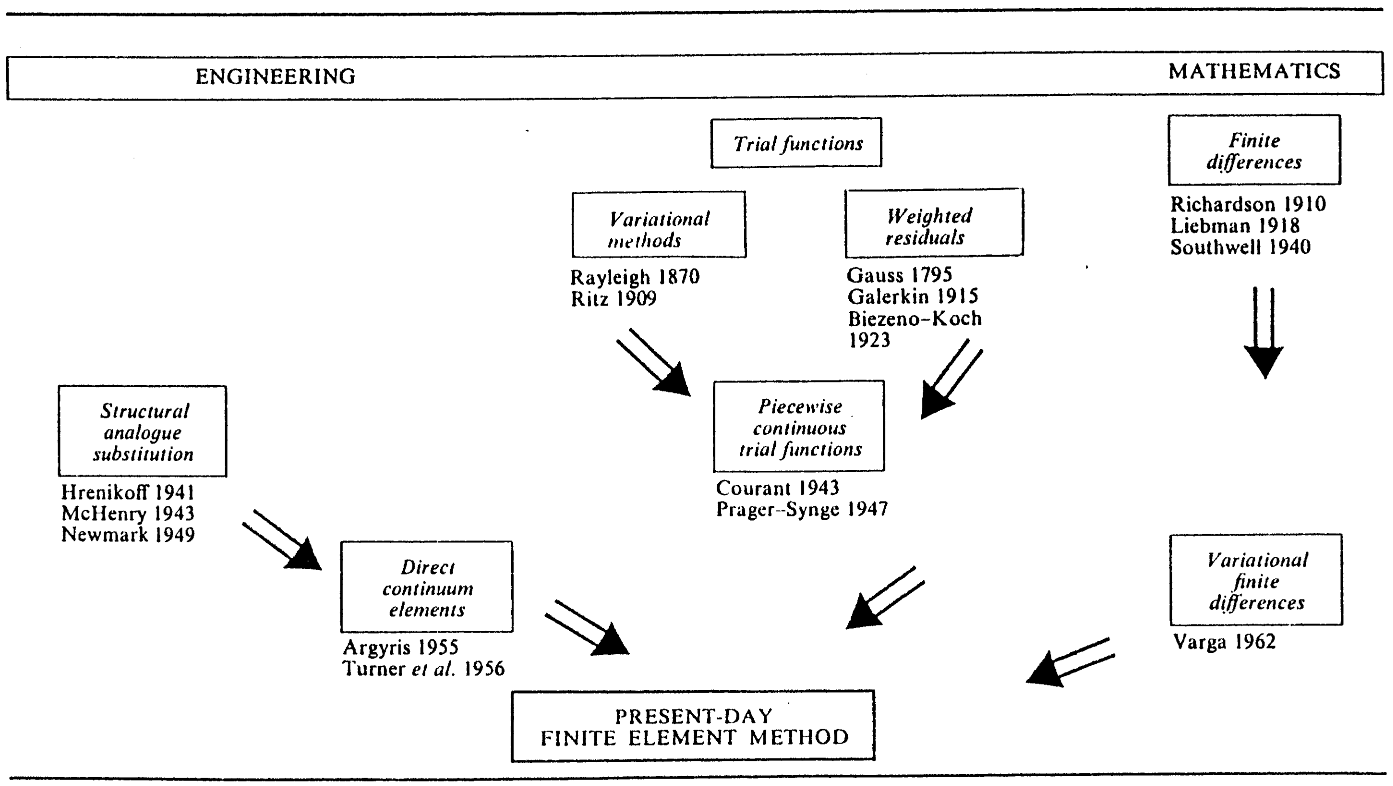


TABLE 1.1 Origin of the finite element method.

(d) The continuous quantity is approximated over each element by a polynomial defined using the values at the nodes of the continuous quantity.

Three main requirements should be satisfied:

(i) Equilibrium equations;

(ii) Compatibility along the element boundaries; and

(iii) Constitutive relations should be verified.

The conventional finite element formulation is simply a Rayleigh-Ritz method using piecewise assumed shape functions with nodal displacements q as generalized coordinates. The final equation is of the form:

$$Kq = Q$$

where K is the assembled stiffness matrix of the structure and Q the assembled vector of equivalent nodal forces. This is, of course, a matrix displacement method.

The finite element method can also be looked at as a special weak formulation obtained by integrating by parts n times the original statement, as shown in Fig. 1.2 [9], where the orignal statement contains a derivative of order 2n. The finite element method starts from the balanced form of the fundamental equation. Integrating by parts n more times, the inverted statement is obtained. This is the start of the boundary element method, which is going to be introduced in the next section.

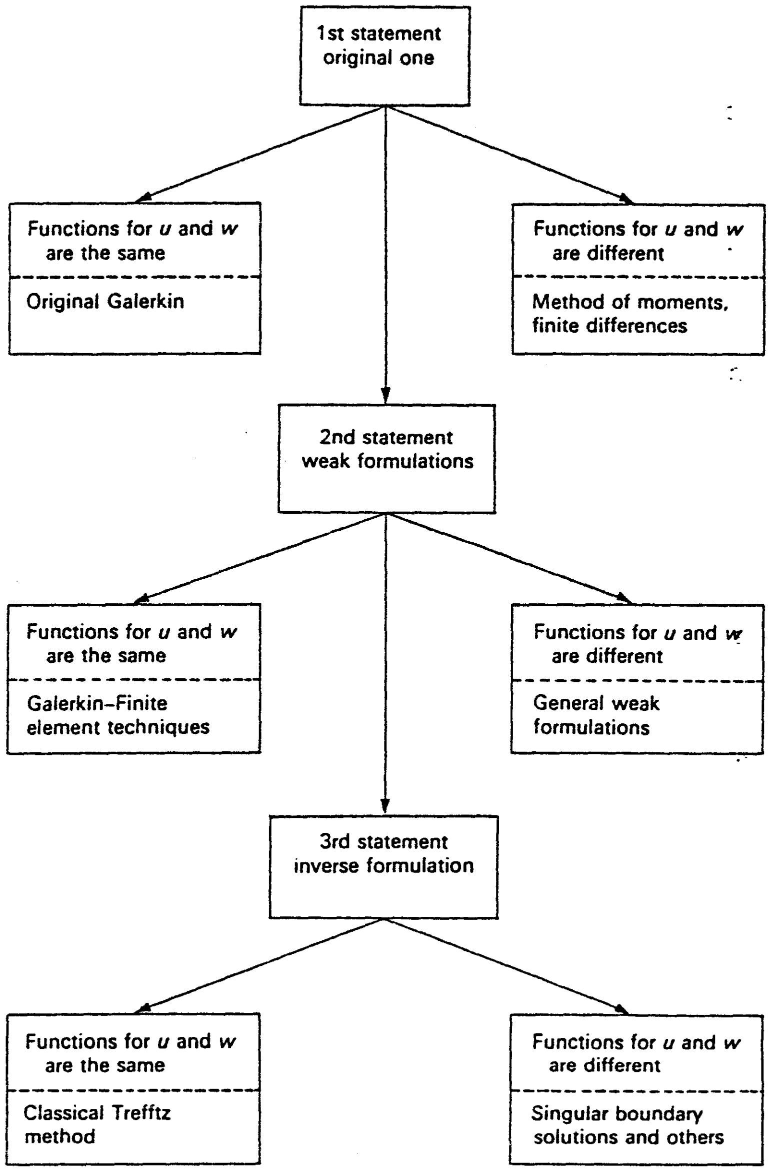

1st statement
original one
Functions for *u* and *w* are the same
Original Galerkin
Functions for *u* and *w* are different
Method of moments, finite differences
2nd statement
weak formulations
Functions for *u* and *w* are the same
Galerkin–Finite element techniques
Functions for *u* and *w* are different
General weak formulations
3rd statement
inverse formulation
Functions for *u* and *w* are the same
Classical Trefftz method
Functions for *u* and *w* are different
Singular boundary solutions and others

FIGURE 1.2 Classification of different approximate techniques.

1.1.4 The Boundary Element Method

Integral equations were until recently considered to be a different type of analytical method, somewhat unrelated to approximate methods. They became popular in Europe through the work of a series of Russian authors such as Muskhelish and Smirnov but were not very popular with engineers.

Considering the Laplace's type problem:

$\Delta^2 u = \emptyset$ in the domain Ω with boundary conditions:

$u = \overline{u}$: essential boundary condition on Γ_1

$\frac{\delta u}{\delta n} = \overline{q}$: natural boundary condition on Γ_2

where $\Gamma = \Gamma_1 + \Gamma_2$ is the total surface, as shown in Fig. 1.1. When Γ_1 is equivalent to Γ, the problem is called the Divichlet problem, with all boundary conditions being essential boundary conditions. When Γ_2 is equivalent to Γ, the problem reduces to the Neumann problem with all boundary conditions being natural ones; and when $\overline{q} = \alpha_o + (\alpha_1 u)*u$ the problem is referred to as the Robin problem.

In 19Ø3 Fredholm tried to define appropriate Green's functions to solve these problems. In 1963, with the introduction of approximation to integral equations by Hildebrand,and later on Jaswon [4] and Symm [5], indirect methods for solving Divichlet and Neumann problems were developed. In 1967 Hess and Smith from aeronautical engineering attacked the Neumann problem. And, later on in 1969, Harrington from electrical engineering proposed solutions to the Robin problem.

A need for a less mathematical approach was proposed by engineers from Carnegie Mellon University in 1971. The work of Cruse and Rizzo [6] in elastostatics was the beginning of direct methods of solving the problem taking both u and $q = \delta u/\delta n$ as variables. Significant developments were made in the early 1970's at Southampton University [7]. Pioneering work was done there by Butterfield and Tottenham. Extensive work has been done by Brebbia [8,9,10,11,12] as a follow-up. Last, but not least, innovative work has been done at the Massachusetts Institute of Technology by J. Connor, in the development and implementation of boundary element methods.

There are two kinds of boundary element methods: direct and indirect. Both will be discussed in detail in Chapter Two.

The basic difference with the finite element method is that the boundary element method is based on the discretization of the exterior boundary only. An expansion for the variables (which are the "essential" and "natural" boundary quantities) is introduced in terms of parameters. For N parameters, N equations are needed, and, thus, N "source" points. These "source" points are applied on the exterior boundary in the boundary element method. Therefore, the exterior boundary is discretized into straight or curved (isoparametric) elements, with different number of "sources" (points) applied depending on the expansion of the elements.

Clearly, the key points for obtaining accurate results in both finite and boundary element methods are a good discretization, and a correct interpretation of the results.

1.2 JUSTIFICATION OF THE USE OF BOUNDARY ELEMENT METHODS

1.2.1 Advantages and Disadvantages of the Discrete Element Methods

Some of the advantages of the finite and boundary element methods include:

(1) Irregularly shaped boundaries can be approximated using elements with straight sides or matched exactly using elements with curved boundaries. The methods are not limited to "nice" shapes with easily defined boundaries.

(2) The size of the elements can be varied. This properly allows the mesh to be expanded or refined as the need exists.

(3) Boundary conditions, such as discontinuous surface loadings, present no difficulties for these methods. Mixed boundary conditions can be easily handled.

(4) The above properties can be incorporporated into one general computer program for a particular subject area.

Finite element methods are very useful in problems involving non-homogeneous medias, plasticity, or time dependent problems while boundary elements are very much needed in modeling problems in an infinite domain, taking care of singularities at crack tips, or dealing with incompressible materials. Those advantages and disadvantages will be discussed in the next section.

The user should be careful in interpreting the results of the huge computer codes based on the discrete element analysis. A trend exists to accept those results without questioning them thoroughly.

As pointed out in Reference 13: "in the field of linear elastic stress analysis, the finite element method is now used almost indiscriminately, and the results produced are often accepted without question". The belief in calculated results should be tempered with caution.

There are three main areas where errors can arise in discrete element analysis. They are:

(i) the creation of the discrete element model;

(ii) the failure to allow for the limitations of the programs used;

(iii) the assumption of high accuracy for the results.

The creation of the discrete element model involves assumptions about the applied loads, the boundary conditions and the behavior of the material. So an overall understanding of the physical behavior of the component and the limitations of the model is crucial. Furthermore, a set of assumptions which seem reasonable may produce a set of results consistent with the assumptions made but which are nonetheless incorrect. Figure 1.3(a) shows a cantilever beam under concentrated loading. The finite element method's answers are compared to Timoshenko and Bernouilli beam theories in Fig. 1.3(c). The exact solution gives an answer consistent with the finite element model assumed in Fig. 1.3(b). And as the number of elements N increases, the

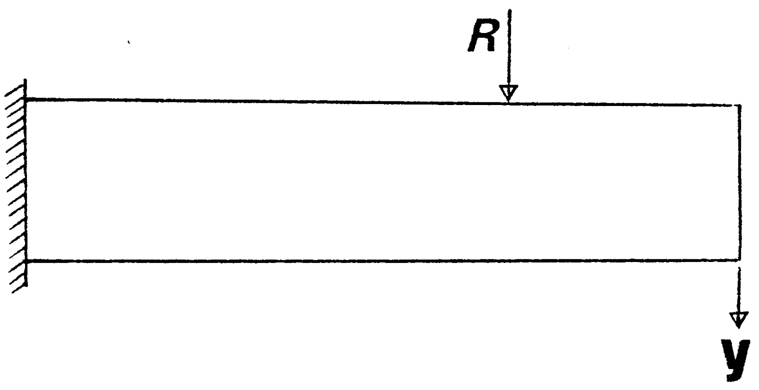


FIGURE 1.3 (a) Actual Problem

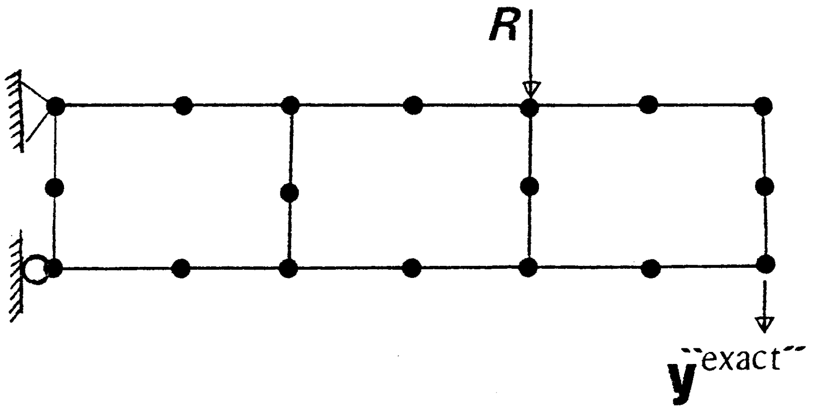


FIGURE 1.3 (b) Finite elements' model

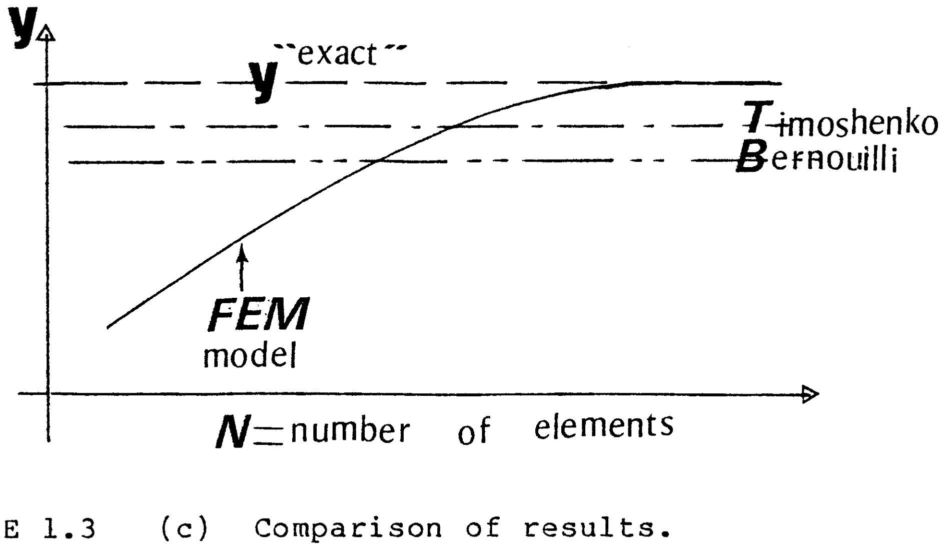


FIGURE 1.3 (c) Comparison of results.

FIGURE 1.3 Accuracy measures for approximate methods.

answer converges to δ^{exact}. But this solution is the correct one for the "model" assumed and it is bigger than both Bernouilli and Timoshenko beam theory predictions, as shown in Fig. 1.3(c). Therefore, to have a good answer, one should have a consistent model. And to be able to achieve this, a thorough preknowledge of the physical behavior of the component under analysis is needed. Therefore analysis should include some assessment of the validity of the assumptions made. Errors from bad assumptions combine with limitations of the different discrete element programs used. Round off errors in computations can give rise to very bad answers, especially if single precision is used. Double precision is recommended for any large size problem using discrete element analysis. Errors can arise in solving large numbers of simultaneous equations, as in the solution for the vector of displacements $\underline{U}$, in $\underline{KU} = \underline{B}$. Errors can also result from time integration, or numerical integration. Appropriate integration rules should be used for every kind of problem. Finally, results from the computer codes should be checked carefully before using them. High level of accuracy can only be achieved by very experienced analysts or if there is considerable relaxation of time and cost constraints.

1.2.2 Comparison of Finite Element and Boundary Element Methods

The boundary element method generally offers a series of advantages over 'domain' type techniques such as finite difference, or finite elements. The first one is that a boundary element model is much easier to develop than a finite element model. Only discretization of the boundary is needed in boundary element methods, compared to discretization of the whole domain in finite element methods. This results in smaller systems of equations to be solved, which results in the reduction of round off errors. But the boundary element methods have a more complicated analytical formulation, which results in the use of more numerical and time integration rules. Some error might arise from the use of a non-appropriate rule. The finite element method would rather be used in a certain class of problems, like plasticity, time-dependent problems, and problems with non-homogeneous materials, where the boundary element method is not as powerful. The boundary element method would be better when modelling a physical system of infinite extent. In this case an appropriate approximation to reality by means of a finite number of "truly" finite elements is clearly very difficult. Another situation where the finite element method encounters difficulties is when stress singularities are present, such as occur at crack tips. The boundary element method is also preferred when solving for materials with Poisson's ratio equal to 0.5 (i.e. incompressible materials).

On the other hand, the finite element method is better known than the boundary element method, and more computer codes are available using finite elements. However, people are sometimes too confident about the results of finite element methods, and this might result in a careless use of the answers. While boundary element methods are relatively more recent and one is generally more conservative with respect to the results.

It may be very convenient in many applications to combine boundary elements with finite elements. It would be very useful to have a knowledge-based system, which would advise the user to use one method or the other or a combination of both depending on the kind of problem faced.

1.2.3 Problems to be Examined using Boundary Element Techniques

Many kinds of problems can be solved using boundary element methods. I am going to concentrate on two-dimensional problems. The extension to three-dimensional problems can be made easily once the procedure is established for two dimensions. Changes in computer codes are easier if appropriate languages have been used (the change to three dimensions is easier in the "C" language than in FORTRAN).

The main emphasis of programming is going to be on potential problems. The applications of Poisson's equation are numerous, and most of them are potential problems. Heat transfer, fluid problems, electrostatics, torsion of constant cross-section members and magnetostatics are just a few examples. A more complete listing is shown in Table 1.2 [14]. Elasticity problems and time-dependent problems are also applications to be considered. Recent advances in time dependant problems using boundary elements, make these problems more attractive to solve using boundary elements [11,12,15].

Natural boundary condition: $k\frac{\partial u}{\partial n} + h(u - u_{\infty}) = q$; essential boundary condition: $u = \hat{u}$

Field of application	Primary variable u	Material constant k	Source variable Q	Secondary variables $q, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}$
1. Heat transfer	Temperature T	Conductivity k	Heat source Q	Heat flow q [comes from conduction $k(\partial T/\partial n)$ and convection $h(T - T_{\infty})$]
2. Irrotational flow of an ideal fluid	Stream function ψ	Density ρ	Mass production σ (normally zero)	velocities: $\frac{\partial \psi}{\partial x} = -v$ $\frac{\partial \psi}{\partial y} = u$
	Velocity potential ϕ	Density ρ	Mass production σ (normally zero)	$\frac{\partial \phi}{\partial x} = u, \frac{\partial \phi}{\partial y} = v$
3. Ground-water flow	Piezometric head ϕ	Permeability K	Recharge Q (or pumping, $-Q$)	Seepage q $q = K\frac{\partial \phi}{\partial n}$ velocities: $u = -K\frac{\partial \phi}{\partial x}$ $v = -K\frac{\partial \phi}{\partial y}$
4. Torsion of constant cross-section members	Stress function ϕ	$k = \frac{1}{G}$, G = shear modulus	$Q = 2\theta$, θ = angle of twist per unit length	$\frac{\partial \phi}{\partial x} = -\tau_{zy}$ $\frac{\partial \phi}{\partial y} = \tau_{zx}$ τ_{zx}, τ_{zy} are shear stresses
5. Electrostatics	Scalar potential ϕ	Dielectric constant ε	Charge density ρ	Displacement flux density D_n
6. Magnetostatics	Magnetic potential ϕ	Permeability μ	Charge density ρ	Magnetic flux density B_n
7. Transverse deflection of elastic membranes	Transverse deflection u	$k = T$, T = tension in membrane	Transversely distributed load	Normal force q

TABLE 1.2 Examples of the Poisson equation.

1.3 COMPUTER APPLICATIONS

1.3.1 Evolution of Computer-Based Analysis

The practical use of discrete element analysis is based on matrix algebra and the use of the computer; because it is only in matrix form that the complete solution process can be expressed in a compact and elegant manner, and because it is only with the use of the computer that solutions to large problems can be handled practically, accurately, and quickly. Indeed, the rapid progress in the development of discrete element methods is owed to high-speed automatic digital computers which have also made the most rapid advancements during the last quarter of the century. Due to these advancements, the direct numerical analysis approach to engineering design is now adopted on a routine basis. Humans are getting very used to the idea of their computer friend doing very large and tedious computations for them. Huge systems of equations are solved in record time. Thus, computers are a necessity in each engineering design or research problem. But with human minds relying more and more on the computer, larger problems are handled every day by these artificial minds. Thus, this leads to huge data files that are not easy to prepare. Making a mistake in one of these files is very easy, and very difficult to detect. Furthermore, the computer outputs are huge lists of numbers that are not practical to look at. This is why the need for pre- and post-processing capabilities arised. The pre-processor's task is to help the user in creating his problem, and especially the discretization, while the post-processor will present the results in a more user-friendly way. But the inexperienced user can have problems in choosing the correct discretization, and in interpreting the results of the analysis. The development of a "knowledge-based" accuracy advisor is underway to perform these tasks. It will give advice about the mesh used, and help in interpreting the results. It can be done independently of the pre-and post-processors, or be included within them.

1.3.2 Pre- and Post-Processing Capabilities

The main function of the pre- and post-processor is the intelligent handling of data. Most big computer codes, at present, have very powerful capabilities but they require an extensive numerical input, and print out huge quanties of numerical results. The user is very susceptible to making mistakes in these long input files and spends too much time in creating these big input files. Furthermore, it might be difficult to use the enormous lists of results that the code provides. This is where the pre- and post-processors come into play.

The pre-processor creates the geometry and discretization of the problem. It also allows the user to impose boundary conditions, edit his problem, and obtain a plot or an output of the discretization. The post-processor displays the results by either printing them or plotting them. More options are also available and will be discussed in Chapter 3.

The strong point of the pre- and post-processors is their user and programmer friendly environment. To achieve this, the main innovations with respect to other systems are:

- menu-driven processors;
- graphics at each step of the problem discretization and of the results analysis; and
- a powerful editor that allows the user to correct mistakes in the discretization, or to create a new discretization without many efforts.

The menu-driven option is done in such a way that the user does not need to refer to any manual to be able to operate the system. Both pre- and post-processors are written in the "C" language. They will be discussed in detail in Chapter 3, along with some examples to show their capabilities.

Some intelligence is needed in the pre-processor. A "knowledge-based" pre-processor will give the user good advice about the discretization of the problem, the assumption of material properties, and the boundary conditions used.

At the post-processor stage, some previous knowledge about similar problems, or some intelligence about the method used, will allow the post-processor to look at the consistency of the answers, and to advise the user if he should refine the mesh or not.

At a more advanced stage of programming, the system will learn from the results of each new run. This will be discussed in Chapter 4.

2. GENERAL OVERVIEW OF BOUNDARY ELEMENT METHODS

2.1 BASIC CONCEPTS

2.1.1 Adjoint Operators

Assume an equation can be represented by an operator L such that:

$$L(u) = b \text{ in } \Omega$$

where L is an operator which, when applied to the function u, produces another function b. Ω is the spatial domain represented by coordinates x_i (i=1,2,3). The concept of an adjoint operator can be introduced using the Lagrange's identity:

$$vL(u) - uL^*(v) = \frac{d}{dx} P(u,v)$$

where L^* is defined as the adjoint of L; P is referred to as the "bilinear concomitant". If L(u) contains derivatives of u of order n, then P is linear in the derivatives of u,v of order up to n-1. u and v are two artibrary functions. Given L one can determine L^* and P. To find them one multiplies L(u) by v, and integrates them by parts until all the derivatives of u are transferred over to derivatives of v. This is equivalent to transforming the integral:

$\int_{x_1}^{x_2} vL(u)\, dx$ to the following expression:

$$\int_{x_1}^{x_2} uL^*(v)\, dx + |P(u,v)|_{x_1}^{x_2}$$

Lagrange's identity follows from there, and the relation between L^*, P and L is easily seen. The inner product of two vectors a and b can be written as:

$$\langle a,b \rangle = \quad ab\ d$$

If $L^* = L$, then L is said to be self-adjoint. A self-adjoint positive definite operator is defined as:

$$(\langle L(u),\ u \rangle) \quad 0 \text{ for } u \neq 0$$

$$(\langle L(u),\ u \rangle) = 0 \text{ for } u = 0$$

where $\langle L(u),u \rangle$ is the inner product of L(u) with u.

2.1.2 Green's Formula

Considering the homogeneous form of the original problem:

$L(u) = 0$ in ˆ ,

it can be written as:

$$\langle L(u),v \rangle = \langle u,L^*(v) \rangle$$
$$+ \int [\underline{G}^*(v)\underline{S}(u) - \underline{G}(u)\underline{S}^*(v)]\ d$$

This is Green's formula; is the exterior surface, and S and G are differential operators due to the integration by parts. The set G(u) represents essential boundary conditions, and S(u) represents the natural boundary conditions. Green's formula is equivalent to requiring:

$$P(u,v) = [\underline{G}^*(v)\underline{S}(u) - \underline{G}(u)\underline{S}^*(v)]$$

in Lagrange's identity.

In order for the solution to be unique the essential boundary conditions should be satisfied. If the operator L is self-adjoint, then $G = G^*$ and $S = S^*$.

For self adjoint operators we can define the boundary conditions as follows:

$\underline{G}(u) = \underline{g}$ on Γ_1 would be the essential b.c. and $\underline{S}(u) = \underline{f}$ on Γ_2 would be the natural b.c. where $\Gamma_1 + \Gamma_2 = \Gamma$. In potential problems Γ_1 would correspond to prescribed potentials, and Γ_2 to prescribed fluxes. In elasticity problems, Γ_1 would correspond to prescribed displacements and Γ_2 to prescribed forces.

2.1.3 Variational Formulation

Rewriting Green's formula as:

$$\langle L(u), v\rangle = \langle u, L^*(v)\rangle + \int_\Gamma [\underline{F}^*(v)\underline{G}(u) - \underline{F}(u)\underline{G}^*(v)]\, d\Gamma$$

let's consider the problem:

$$L(u) + b = 0 \text{ in } \Gamma$$

with boundary conditions:

$\underline{F}(u) = \underline{f}$ on Γ_1: Essential b.c.

$\underline{G}(u) = \underline{g}$ on Γ_2: Natural b.c.

If the operator L is self adjoint, the inner product $\langle L(u),v\rangle$ can be written after the first integration phase (i.e., when order of derivatives of u are equal to order of derivatives of v) as:

$$\langle \underline{L}(u),v\rangle = \int \underline{F}(V)\ \underline{G}(u)\, d\Gamma + \int \underline{D}(v)\underline{D}(u)\, d\Omega$$

To interpret the equation as a stationary requirement for a functional, we define v as the variation of u, which gives

$$v = \delta u$$

Thus the formal equation becomes:

$$\langle L(u), \delta u\rangle = \int \underline{F}(\delta u)\underline{G}(u)\, d\Gamma + \langle \underline{D}(u), \underline{D}(\delta u)\rangle$$

Next a functional H is defined such that:

$$\delta H(u) = \underline{D}(v)\ \underline{D}(u), \text{ i.e.,}$$

$$\langle H(u)\ \rangle = \frac{1}{2} \langle \underline{D}(u) \cdot \underline{D}(u)\rangle$$

Again, this is only possible because the operator L was assumed to be self-adjoint.

Rewriting the equation gives:

$$\langle L(u) + b,\ \delta u\rangle - \int \underline{F}(\delta u)\ \underline{G}(u)\ d\Gamma = \delta \langle H(u) + bu\rangle$$

Requiring $\underline{F}(\delta u) = 0$ on Γ_1, and the natural boundary condition: $\underline{G}(u) - \underline{g} = 0$ on Γ_2 to be satisfied, as well as the original problem: $L(u) + b = 0$ in Ω, is equivalent to requiring the functional I_1 to be stationary. Where I_1 is given by:

$$\langle L(u) + b,\ \delta u\rangle - \int_{\Gamma_2} \underline{F}(\delta u)\{\underline{G}(u)-\underline{g}\}\ d\Gamma$$

$$= \delta\{\int[H(u) + bu]\ d\Omega + \int_{\Gamma_2} \underline{F}(u)\underline{g}\ d\Gamma\}$$

$$= \delta\{\langle H(u) + bu\rangle + \langle \underline{F}(u),\ \underline{g}\rangle\ \Gamma_2\}$$

$$= \delta I_1 .$$

Requiring I_1 to be stationary is equivalent to $\delta I_1 = 0$. This approach is equivalent to solving the following problem:

$$L(u) + b = 0 \text{ in } \Omega$$

$$\underline{G}(u) = \underline{g} \text{ on } \Gamma_2$$

$$\underline{F}(\delta u) = 0 \text{ on } \Gamma_1$$

A more general approach which is followed in the boundary element method and mixed variational formulations is to require the essential boundary conditions on Γ_1 to be satisfied, as well as the natural boundary conditions on Γ_2. This approach does not require $F(\delta u)$ to be constrained, but only $F(u)$ and $G(u)$ to be constrained on Γ_1 and Γ_2 respectively. This gives:

$$\langle L(u) + b, \delta u\rangle + \int_{\Gamma_1} [\underline{F}(u) - \underline{f}]\, \underline{G}(\delta u)\, d\Gamma$$

$$- \int_{\Gamma_2} \underline{F}(\delta u)[\underline{G}(u) - \underline{g}]\, d\Gamma$$

$$= \delta\{\int_{\Omega} [H(u) + bu]\, d\Omega + \int_{\Gamma_2} \underline{F}(u)\underline{g}\, d\Gamma + \int_{\Gamma_1} [\underline{F}(u) - \underline{f}]\, \underline{G}(u)\, d\Gamma\}$$

$$= \delta I_2$$

where I_2 is the quantity between brackets. In this case, requiring $\delta I_2 = 0$ for arbitrary δu (i.e., I_2 is stationary) is equivalent to the complete set of conditions:

$$L(u) + b = 0 \text{ in } \Omega$$

$$\underline{F}(u) = \underline{f} \text{ on } \Gamma_1$$

$$\underline{G}(u) = \underline{g} \text{ on } \Gamma_2$$

This approach satisfies both essential and natural boundary conditions, and leads to the establishment of the general weighted residual form. The difference is that while the above approach is only valid for self-adjoint operators, the weighted residual technique allows the solution of any boundary element type statement.

2.2 GENERAL WEIGHTED RESIDUAL SCHEMES

2.2.1 Approximate Solutions

Most engineerng problems which are expressed in a differential form can only be solved in an approximate manner due to their complexity. The exact solution is usually impossible to find, but it can be approximated by a set of functions $\phi_k(x)$, such that:

$$u = \sum_{k=1}^{n} \alpha_k \phi_k$$

where α_k are undetermined parameters and ϕ_k are linearly independent functions taken from a complete sequence of functions such as: $\phi_1(x)$, $\phi_2(x)$, ... $\phi n(x)$; where x represents the spatial coordinates in the domain under consideration. These functions satisfy boundary and continuity conditions.

Considering the problem:

$$L(\hat{u}) + b = 0 \text{ in } \Omega$$

$$\underline{F}(\hat{u}) = \underline{f} \text{ on } \Gamma_1$$

$$\underline{G}(\hat{u}) = \underline{g} \text{ on } \Gamma_2$$

The introduction of the approximate solution is going to give rise to some errors, which can be represented by error functions called residuals. The function error is:

$$\varepsilon_I = L(u) + b$$

and the boundary conditions residual functions are:

$$\varepsilon_1 = F(u) - f$$

$$\varepsilon_2 = G(u) - g$$

These errors are zero for the exact solution but not for the approximate one.

2.2.2 Method of Weighted Residuals

In this method the error ε_i or residual is forced to zero in the average sense, by setting weighted integrals of the residual equal to zero, i.e.:

$$\int \varepsilon\, \omega_i\, dx = 0 \qquad i = 1,2,\ldots n$$

where ω_i is a set of weighting functions, which are part of a linearly independent, complete set. The solution will converge towards the exact solution as the number of terms used increases. The weighted residual is:

$$W.R. = \int_{\Omega} \varepsilon_I\, w d\Gamma + \int_{\Gamma_1} \underline{\varepsilon}_1\, \underline{w}_1\, d\Gamma + \int_{\Gamma_2} \underline{\varepsilon}_2\, \underline{w}_2\, d\Gamma$$

We want to achieve W.R. = 0.

If the order of the approximation function u is n, then one should apply the above equations n times in order to determine the n parameters of the approximating function.

The key issue is the choice of the weighting functions. Let us first review several well-known approximate techniques:

(a) The collocation method: Instead of trying to satisfy the differential equation in an average form, we try to satisfy it at only a series of chosen points. Considering again the problem: $\int \varepsilon w_i \, dx = 0$ $i = 1,2...n$; the collocation method assumes $w_i = \Delta_i$ which gives $\int \varepsilon \Delta_i \, dx = 0$ $i = 1,2...n$; where $\Delta_i \equiv \Delta(xi)$ which is a Dirac function such that $\Delta(x_i) = 0$ if $x \neq x_i$ and

$$\int_{x_i-c}^{x_i+c} \Delta(x_i) \, dx = 1, \; c \to 0$$

(b) The method of sub-regions: It is similar to the collocation method, but instead of requiring the residual to be zero at certain chosen points, we require it to be zero over different regions.
Thus we obtain:

$$\int_{\Omega_i} \varepsilon \, d\Omega = 0 \text{ for different regions } \Omega_i \; i = 1, 2, ... n.$$

(c) The method of moments: It consists of setting $w_i = x^i$ $i = 0,1,2,...$. i.e., the weighting functions are taken from the series $1, x, x^2, x^3, ...$ for a one-dimensional problem. In this way successive higher moments of the residual are forced to be zero.

At this point let us recall that we are trying to find the different coefficients α such that:

$$u = \sum_{k=1}^{n} \alpha_k \phi_k$$

The methods of weighted residuals requires solving the following system of simultaneous equations:

$$\underline{a}\ \underline{\alpha} = \underline{b}$$

to find the vector of coefficients $\underline{\alpha}$. But all three weighted residual methods mentioned until now give an unsymmetric matrix $\underline{a}$, i.e. $a_{ij} \neq a_{ji}$. Let us examine the method of least squares that lead to a symmetric matrix $\underline{a}$.

(d) <u>Method of least squares</u>: It consists of weighting the residual function with its derivative with respect to each of the n different α's parameters. Thus, let:

$$\int_\Omega \varepsilon\ \varepsilon\ d\Omega = I$$

Then, minimize I with respect to each of the n different parameters:

$$\frac{\delta I}{\delta \alpha_j} = 0 \qquad j = 1,2,\ldots n$$

which gives:

$$\int_\Omega \varepsilon \frac{\delta \varepsilon}{\delta \alpha_j}\ d\Omega = 0$$

This means that we are setting:

$$w_j = \frac{\delta \varepsilon}{\delta \alpha_j}$$

But for linear problems, we can write:

$$\frac{\delta \varepsilon}{\delta \alpha_j} = L(\phi_j)$$

thus $a_{ij} = \int L(\phi_i)\, L(\phi_j)\, d\Omega = a_{ji}$ and the $\underline{a}$ matrix will be symmetric.

(e) <u>Method of Galerkin</u>: This is the most popular method used. The weighting functions are the same as the trial functions, i.e. $w_j = \phi_j$ for $j = 1, 2, \ldots n$. This gives:

$$a_{ij} = \phi_i\, L(\phi_j) \neq a_{ji},$$

But, through manipulations, the $\underline{a}$ matrix can be turned into a symmetric matrix, using variational formulation.

2.2.3 The Inverse Statement

Let us look at the expression for W.R. (developed in Section 2.2.2), and to the functional I_2 (developed in Section 2.1.3) whose stationary requirement is equivalent to the same problem for which the W.R. is written. The key issue is to choose the weighting functions, using the result obtained with the variational approach. Comparing the expressions for W.R. and I_2, one notices that:

$$w \equiv \delta u$$
$$\underline{w}_1 \equiv \underline{G}(w)$$
$$\underline{w}_2 \equiv -\underline{F}(w)$$

With this choice, the weighted residual is identical to the stationary requirement. Replacing $\underline{w}_1$ and $\underline{w}_2$ by $\underline{G}(w)$ and $-\underline{F}(w)$ the fundamental form of the problem is obtained:

$$\int_{\Omega} [L(u)+b]\ w d\Omega + \int_{\Gamma_1} [\underline{F}(u)-\underline{f}] \cdot \underline{G}(w)\ d\Gamma$$

$$+ \int_{\Gamma_2} (\underline{G}(u)\ -\underline{g})[-\underline{F}(w)]\ d\Gamma = 0$$

To obtain the variational principle, one uses $w = \delta u$. The above equation satisfies the differential equation and both essential and natural boundary conditions. One can generate alternate forms by integrating $wL(u)$ by parts. In the conventional finite element method, one integrates until the order of differention for u and w is balanced, i.e., equalized. By doing this we obtain the balanced form, which is:

$$\int_{\Omega} (\underline{D}(u) \cdot \underline{D}(w) + bw)\ d\Gamma$$

$$+ \int_{\Gamma_1} \{[\underline{F}(u) - \underline{f}]\ \underline{G}(w) + \underline{F}(w)\ \underline{G}(u)\}\ \delta\Gamma$$

$$+ \int_{\Gamma_2} g\ \underline{F}(w)\ d\Gamma = 0$$

This is the form used in finite element analysis. Again, for the variational formulation, w is replaced by δu.

In the boundary element approach, the fundamental form is transformed to a form where differentiation of u in the interior, Ω, is eliminated. This is the inverted form. It can be obtained directly from the fundamental form, or it can be obtained by integrating by parts the balanced form until there is no more derivatives of u. The inverted form is found to be:

$$\int_{\Omega} [uL(w) + bw] \, d\Omega +$$

$$\int_{\Gamma_1} [\underline{F}(w) \cdot \underline{G}(u) - \underline{f} \; \underline{G}(w)] \, d\Gamma$$

$$\int_{\Gamma} [\underline{F}(w) \cdot \underline{g} - \underline{F}(u) \; \underline{G}(w)] \, d\Gamma = 0$$

The advantage of the balanced form (F.E.M.) is that the order of the derivatives in u and w is lower than in the inverted form. But, the superiority of the inverted form lies in the choice of w. Suppose one uses, as weighting functions, solutions of the homogeneous equations:

$$L(w) = 0$$

Then, the interior integral involving u vanishes and one has only to work with expansion for u on the boundary. Interior node points are not required. This shows very clearly the advantage of the boundary element method. It also shows that in some instances it would be simpler to use the finite element method. It would be very useful to define a class of problems to be used by each method. The inverted form shown in this section is the basis of the boundary element method.

2.3 POTENTIAL PROBLEMS

2.3.1 General Overview

Many engineering problems such as seepage, heat conduction, electrical problems, etc. are governed by a Laplace or Poisson equation of the type: $\nabla^2 u = b$.

In most two-dimensional potential problems, two quantities are of interest: the potential, u, and the potential derivative, q. In heat flow problems, u and q can be taken as heat flow per unit volume and flux respectively, or u could represent temperature while q would be the temperature gradient. In fluid flows u could be specified as either velocity potential: ϕ, or stream function: ψ; in which cases q would correspond to normal flow and stream function derivative respectively.

To introduce potential problems a simple two-dimensional heat flow is considered. A domain , Ω, with a surface, Γ, is subjected to an input f which could be a positive heat flow. Let u be the potential on Γ, and q_x and q_y the fluxes in the x and y direction respectively as shown in Fig. 2.1. The fluxes: q_x and q_y are related to the potential u by the Fickian law:

$$q_x = - k_{xx} \frac{\delta u}{\delta x} - k_{xy} \frac{\delta u}{\delta y}$$

$$q_y = - k_{xy} \frac{\delta u}{\delta x} - k_{yy} \frac{\delta u}{\delta y}$$

These two equations are equivalent to stress-strain relationships for solids. The problem shown in Fig. 2.1 is governed by the following equation:

$$\frac{\delta u}{\delta t} = f - \frac{\delta q_x}{\delta x} - \frac{\delta q_y}{\delta y}$$

This equation is similar to the force equilibrium relations for two-dimensional elasticity.

In steady state problems, $\delta u/\delta t = 0$. The input f can be modelled as a combination of a constant term and a function of u.

Assume $f = Q - a_1 u$, which could be a representation of a heat quantity. Laplace's equation is obtained by setting:

(i) $k_{xx} = k_{yy} = 1$

(ii) $k_{yx} = k_{xy} = 0$

(iii) $a_1 = 0$

(iv) $Q = 0$

Introducing the operator $\nabla^2 = \frac{\delta^2}{\delta x^2} + \frac{\delta^2}{\delta y^2}$, and considering the steady state problem, Laplace's equation reduces to:

$$\nabla^2 u = 0 .$$

Poisson's equation satisfies conditions (i), (ii) and (iii) above, but $Q \neq 0$. Thus, Poisson's equation can be written as:

$$\nabla^2 u = Q.$$

Finally, Holme's equation only satisfies conditions (i) and (ii) with $Q \neq 0$ and $a_1 \neq 0$, thus reducing to:

$$\nabla^2 u = Q - a_1 u.$$

Laplace's, Poisson's and Holmes' equations represent the main type of potential problems.

In general for orthotropic materials: $k_{xy} = k_{yx} = 0$ and $k_{yy} \neq k_{xx}$; while $k_{xx} = k_{yy}$ for isotropic media.

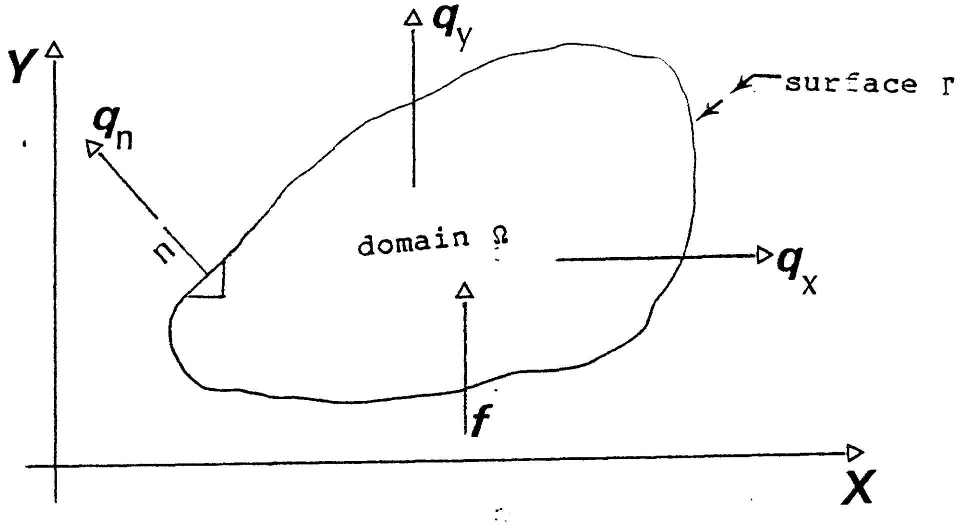


FIGURE 2.1 Two-dimensional heat flow problem.

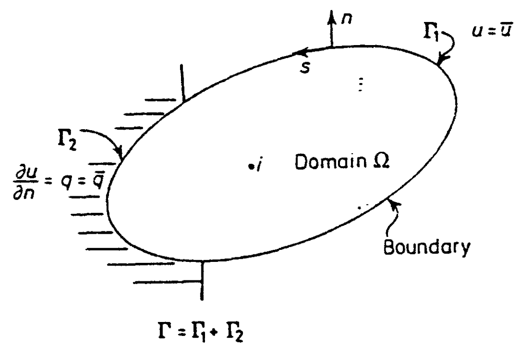

n
Γ_1
$u = \bar{u}$
s
Γ_2
$\frac{\partial u}{\partial n} = q = \bar{q}$
•i
Domain Ω
Boundary
$\Gamma = \Gamma_1 + \Gamma_2$

FIGURE 2.2 Potential problem definition.

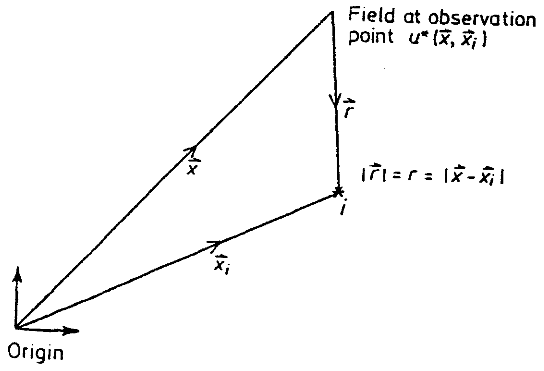

Field at observation point $u^*(\vec{x}, \vec{x}_i)$
$\vec{r}$
$|\vec{r}| = r = |\vec{x} - \vec{x}_i|$
$\vec{x}$
i
$\vec{x}_i$
Origin

FIGURE 2.3 Fundamental solution definition.

The normal flux, q_n, is usually the quantity of interest rather than q_x and q_y. They are related through a simple geometric relation:

$$q_n = \alpha_{nx}\, q_x + \alpha_{ny}\, q_y \ ,$$

where $\alpha_{nx} = \cos(n,x)$ and $\alpha_{ny} = \cos(n,y)$, n being the outward normal to the surface Γ. For simplicity, q_n is generally denoted by q (dropping the subscript n), and this convention will be adopted for this section. Thus, $q = \delta u/\delta n$. Emphasis in this thesis was given to the solution of Poisson's equation:

$$\nabla^2 u = b \text{ in a domain } \Omega \ ,$$

with boundary conditions:

$$u - \bar{u} = 0 \quad \text{on } \Gamma_1 \ ,$$

$$\frac{\delta u}{\delta n} - \bar{q} = 0 \quad \text{on } \Gamma_2 \ .$$

This is shown in Fig. 2.2. As discussed in Section 2.2 the different error's residuals can be written as:

$$\varepsilon_1 = \nabla^2 u - b \quad \text{in } \Omega$$

$$F(u) - \bar{u} = \varepsilon_1 = u - \bar{u} \quad \text{on } \Gamma_1$$

$$G(u) - \bar{q} = \varepsilon_2 = \frac{\delta u}{\delta n} - \bar{q} \text{ on } \Gamma_2$$

Using weighted residual methods, the fundamental form of the problem can be written:

$$\int_{\Omega}(\nabla^2 u-b)\ w d\Omega + \int_{\Gamma_1} (u-\bar{u})\ (\frac{\delta w}{\delta n})\ d\Gamma$$

$$+ \int_{\Gamma_2} (\frac{\delta u}{\delta n} -\bar{q})\ (-w)\ d\Gamma = 0$$

Integration by parts twice leads to the inverse statement, as shown below:

$$\int_{\Omega} (u\nabla^2 w-bw)\ d\Gamma + \int_{\Gamma_1} (w\ \frac{\delta u}{\delta n} - \bar{u}\ \frac{\delta w}{\delta n})\ d\Gamma$$

$$+ \int_{\Gamma_2} (w\bar{q}-u\ \frac{\delta w}{\delta n})\ d\Gamma = 0$$

This is the starting point of the boundary element method for potential problems.

2.3.2 <u>The direct boundary element method</u>

The equations for the direct boundary element formulation are generated by using Green's formula, or the method of weighted residuals, or an equivalent procedure such as Betti's law for solids. The variables are the "essential" and "natural" boundary quantities such as the potential and normal flux in potential problems, or the displacements associated with surface tractions, rotations along with moments, and displacements related to transverse shear in elasticity problems.

The starting idea is to find a weighting function w, that satisfies $\nabla^2 w = 0$. This would leave only one domain integral in terms of b in the inverse statement developed in Section 2.3.1. The weighting function satisfying Laplace's equation within Ω is called the fundamental solution, and is denoted by u^*. It is required to have continuous first derivatives within Ω. Assuming a concentrated change is acting at point 'i', the Laplace's equation reduces to:

$$\nabla^2 u^* + \delta_i = 0$$

where δ_i is a Dirac delta function function.

The point of application of the concentrated charge is called the "source" and involves a singularity in the Dirac delta function. The fundamental solution is a function only of the distance between the source point and the observation point under consideration. This distance is denoted by r, and defined as:

$r = |\vec{r}| = |\vec{x}-\vec{x}_i|$, as shown in Fig. 2.3.

For an isotropic medium the fundamental solution of the Laplace's equation is:

$u^* = \frac{1}{2\pi} \ln(\frac{1}{r})$ in a two-dimensional space, and

$u^* = \frac{1}{4\pi r}$ in a three-dimensional space.

It can be shown that the fundamental solution satisfies the Laplace's equation for all points within the domain Ω.

Using the fact that $\nabla^2 u^* + \delta_i = 0$, we can write:

$$\int_\Omega u(\nabla^2 u^* + \delta_i)\, d\Omega = 0 .$$

But $\int_\Omega u(\nabla^2 u^* + \delta_i)\, d\Omega = \int_\Omega u \nabla^2 u^*\, d\Omega + u^i$.

Thus, combining both equations, we obtain:

$$\int_\Omega u(\nabla^2 u^*)\, d\Omega = -u^i$$

Replacing for this in the inverse statement obtained in Section 2.3.1 and making use of the fact that:

$$\delta u/\delta n = q$$
$$w = u^*$$
$$\delta w/\delta n = q^*,$$

the following equation is obtained:

$$u^i + \int_\Omega b u^*\, d\Omega - \int_{\Gamma_1} (u^* q - \bar{u} q^*)\, d\Gamma$$
$$- \int_{\Gamma_2} (u^* \bar{q} - u q^*)\, d\Gamma = 0$$

The source points can be placed anywhere, inside or outside the domain Ω, or on the surface Γ. It is more convenient to have it placed on the boundary itself directly. Bringing the problem on the boundary, one first considers the source as a sphere in the three-dimensional space, as shown in Fig. 2.4. Then the behavior of the different components of the equation above, involving the fundamental solution, are studied as the sphere's radius: ε approaches zero. This will be shown for dimensional problems, using an arc of circle of radius ε, around the source point 'i', as shown in Fig. 2.5.

For two-dimensional problems,

$$u^* = -\frac{1}{2\pi} \ln r$$

and,

$$\frac{\delta u^*}{\delta r} = \frac{-1}{2\pi r}$$

Thus,

$$\int_{\Gamma_\varepsilon} u\, q^* = \lim_{\varepsilon \to 0} \left[u\left(\frac{-1}{2\pi\varepsilon}\right)\right] [(\pi+\alpha)\ \varepsilon]$$

$$= -\frac{1}{2}\left(1+\frac{\alpha}{\pi}\right)u_i \text{ for } -\pi < \alpha < \pi$$

and,

$$\int_{\Gamma_\varepsilon} u^*\, q\, d\Gamma = 0.$$

Thus the new coefficient of u_i in the modified inverse statement changes from:

$$1 \text{ to: } 1 - \frac{1}{2}\left(1 + \frac{\alpha}{\pi}\right) = \frac{1}{2}\left(1 - \frac{\alpha}{\pi}\right).$$

For a smooth boundary, $\alpha = 0$, and the coefficient of u_i reduces to 1/2.

Assuming that $u = \bar{u}$ on Γ_1 and that $q = \bar{q}$ on Γ_2, and making use of the fact that: $\Gamma = \Gamma_1 + \Gamma_2$, then the general equation for the direct boundary element method can be written as:

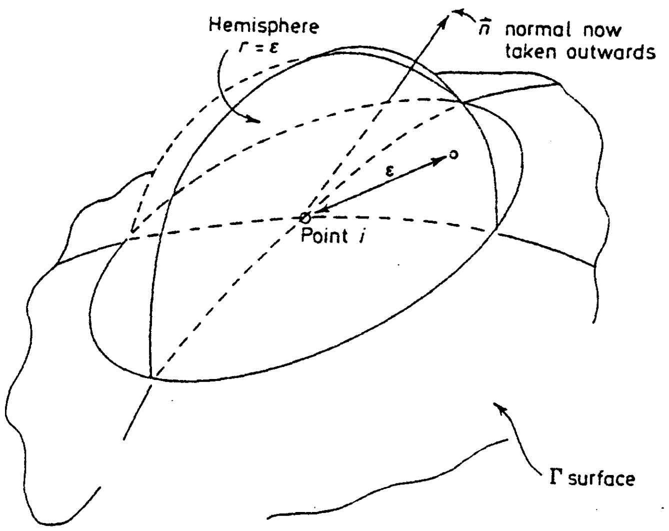


FIGURE 2.4 Hemisphere surrounding boundary point in 3-D space.

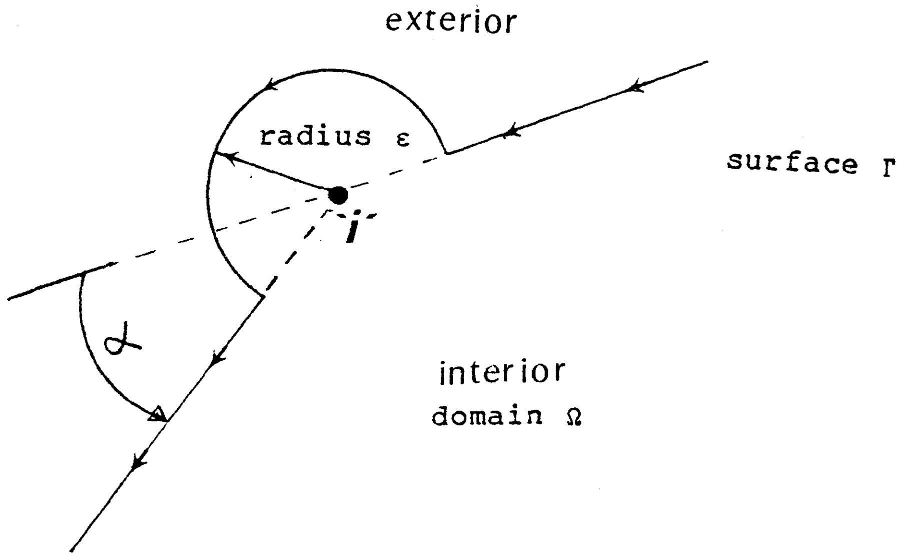


FIGURE 2.5 Arc of circle surrounding boundary point in 2-D space.

$$c_i u_i + {}_{\Gamma}\int uq^* \, d\Gamma - {}_{\Gamma}\int qu^* \, d\Gamma + {}_{\Omega}\int bu^* \, d\Omega = 0$$

where $c_i = 1$ for a point inside Ω,

$c_i = 0$ for a point outside Ω,

$c_i = \frac{1}{2}\,(1-\frac{\alpha}{\pi})$ for a point on a discontinuous boundary, as shown in Fig. 2.5, and

$c_i = \frac{1}{2}$ for a smooth boundary point.

The next step is to introduce an expansion for u and q on the boundary in terms of parameters. For N parameters a need for N equations arises, and, thus, N source points should be used. The boundray is discretized into straight or curved elements, which can have different expansions. This introduces a great flexibility, since there is no inter-element constraints, or expansion constraints on u and q. Constant, linear, quadratic or higher order expansion can be used. The position of the nodal points on the elements is arbitrary. Both continuous and discontinuous elements are permitted, which would have nodes at the edge or at the interior respectively. Less inter-element constraints exist for boundary element than for finite element methods.

The direct formulation leads to a very well structured matrix representation of the problem, to use in computer codes. This will be discussed in the next section.

2.3.3 Matrix formulation of the direct boundary element method

Considering the general equation for a problem of the Laplace's type (b=0), and specializing it for points on the boundary, the following is obtained:

$$\frac{1}{2}\left(1-\frac{\alpha}{\pi}\right) u_i + \int_\Gamma uq^* \, d\Gamma - \int_\Gamma qu^* \, d\Gamma = 0$$

Assume that the boundary is discretized into N elements, and that it is two-dimensional. The points where the unknown values are calculated are called the nodes. Depending on the expansion used for q and u, there can be a different number of nodes per element. As an example, one, two, or three nodes are required with constant, linear, or quadratic expansions respectively. The elements can be continuous or discontinuous. More details on the different type of elements and expansions, will be given in Section 3.2, since all of them have been implemented in the pre-processor facility. In this section, the matrix formulation for linear elements only is considered. Similar considerations can be easily applied to constant or quadratic expansions. In section 3.4.2 a general matrix formulation scheme is indicated that would facilitate the use of different expansions for the same problem.

Figures 2.6(a) and 2.6(b) show discontinuous and continuous elements respectively, with linear expansions.

The "discontinuous" elements have 2 nodes each that are generally placed at the quarter points from each end. In the "continuous" case, the nodes are placed at the intersection between elements.

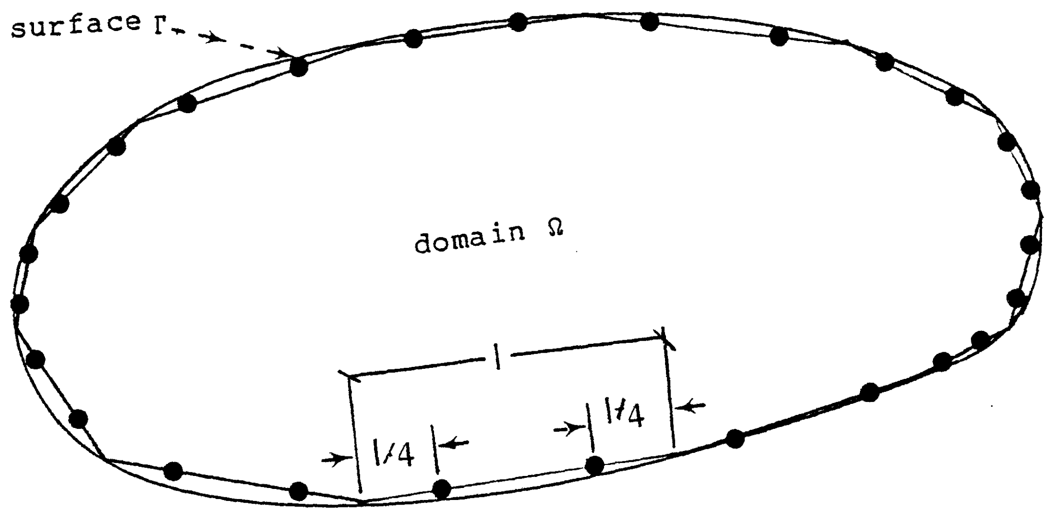


FIGURE 2.6 (a) Linear "discontinuous" elements.

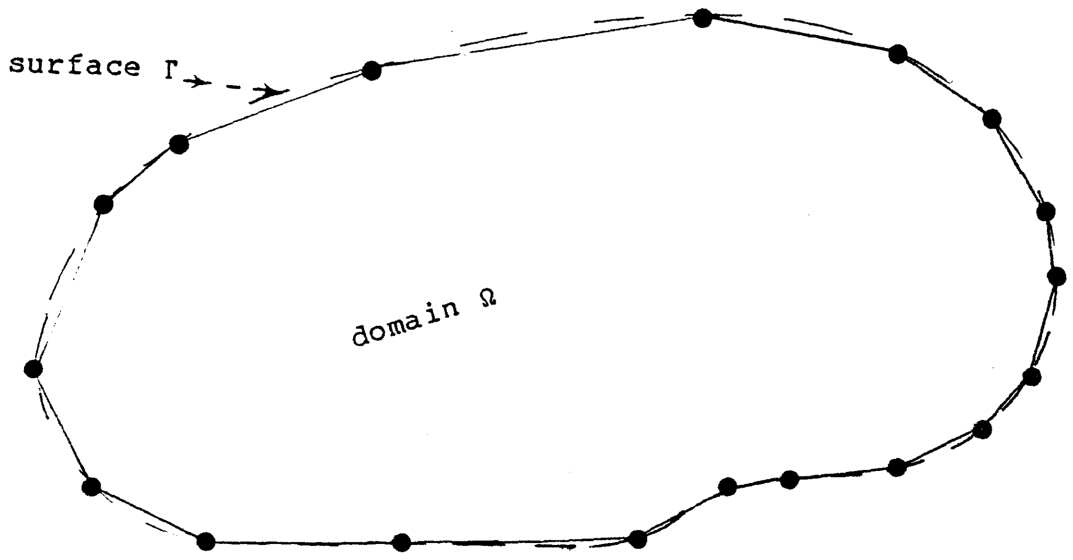


FIGURE 2.6 (b) Linear "continuous" elements.

FIGURE 2.6 Comparison of linear expansion, "dincontinuous" and "continuous" elements.

Let u, u_2 and q_1, q_2 be the values of the potential and the flux at the nodes. The formulation that is going to follow is for continuous elements. This can be easily extended to discontinuous ones. Let x be the distance along the rod, and η an isoparametric parameter such as:

$\eta = \frac{x}{2\ell}$. The linear interpolation functions can be written in terms of η and are given by:

$$\phi_1 = \frac{1}{2}(1-\eta)$$

$$\phi_2 = \frac{1}{2}(1+\eta)$$

The values of u and q can now be defined in terms of the nodal values and the interpolations functions:

$$u = \phi_1 u_1 + \phi_2 u_2$$

$$q = \phi_1 q_1 + \phi_2 q_2$$

The general direct boundary element formulation can be written for the N elements as:

$$\frac{1}{2}(1-\frac{\alpha}{\pi})u_i + \sum_{j=i}^{N} \int_{\Gamma_j} uq^* \, d\Gamma$$

$$- \sum_{j=1}^{N} \int_{\Gamma_j} qu^* \, d\Gamma = 0,$$

where 'i' is the point of application of the fictitious concentrated charge, and 'j' is the segment under consideration. Replacing for the expansion of u and q, one gets:

$$\int_{\Gamma_j} u\, q^* \, d\Gamma = \left[\int_{\Gamma_j} \phi_1 \, q^* \, d\Gamma \right] u_1 + \left[\int_{\Gamma_j} \phi_2 \, q^* \, d\Gamma \right] U_2$$

$$= h^1_{ij} \, u_1 + h^2_{ij} \, u_2$$

Similarly,

$$\int_{\Gamma_j} q u^* \, d\Gamma = g^1_{ij} \, q_1 + g^2_{ij} \, q_2$$

where

$$g^k_{ij} = \int_{\Gamma_j} \phi_k \, u^* \, d\Gamma$$

The coefficients h^k_{ij} and g^k_{ij} refer to the point 'i' considered, and node 'k' on an element 'j'.

Assembling properly all the different contributions from adjoining elements, the equation for node 'i' can be written as:

$$\frac{1}{2} \left(1 - \frac{\alpha}{\pi}\right) u_i + \sum_{j=1}^{N} \hat{H}_{ij} u_j = \sum_{j=1}^{N} G_{ij} \, q_j ,$$

where the $\hat{H}_{ij}$ and G_{ij} terms are equal to the h^2 term of element 'j-1' plus the h^1 term of element 'j'. This can be written as:

$$\sum_{j=1}^{N} H_{ij} \, u_j = \sum_{j=1}^{N} G_{ij} \, q_j$$

where

$$H_{ij} = \hat{H}_{ij} \qquad \text{for } i \neq j,$$

$$H_{ij} = \hat{H}_{ij} + \frac{1}{2}\left(1-\frac{\alpha}{\pi}\right) \quad \text{for } i = j.$$

In matrix form, this is equivalent to:

$$\underline{H}\ \underline{U} = \underline{G}\ \underline{Q},$$

which represents a NxN system of equations, in N unknowns.

Part of the u and q values are prescribed and the other part are unknown. Taking all the known values of u to the right hand side, and all the unknown values of q to the left hand side, one ends up with a modified system of equations:

$$\underline{A}\ \underline{X} = \underline{B},$$

where $\underline{X}$ is a column vector containing all the unknown values of u and q, $\underline{A}$ is a NxN matrix, and $\underline{B}$ is a column vector resulting from the multiplication of a NxN matrix of coefficients with a column vector of known values of u and and q.

Next, the values of u and q on the whole boundary can be determined, from the solution of this system of equations. Similar considerations can be applied to other expansions such as constant or quadratic.

2.3.4 Special comments

(a) Internal points

So far, interest was focused on the behavior of the boundary quantities. If an interest exists in the behavior of u and q inside the domain, then, internal points can be defined in Ω. As discussed before, the values of u and q at those internal points do not enter in the solution process. But, once the values of u and q are known on the boundary, the values at the internal points can be easily calculated.

For a point inside Ω, $c_i = 1$ in the general equation for the direct boundary element method, reducing it to:

$$u_i + {}_{\Gamma}\int uq^* \, d\Gamma - {}_{\Gamma}\int qu^* \, d\Gamma = 0$$

Using the $\underline{\hat{G}}$ and $\underline{\hat{H}}$ matrices developed in Section 2.2.3, one can calculate the value of u at any interior point 'i' using:

$$u_i = \sum_{j=1}^{N} q_j G_{ij} - \sum_{j=1}^{N} u_j \hat{H}_{ij} \; .$$

Differentiating the next to last equation, one can calculate the fluxes, at the interior point 'i', as:

$$(q_x)i = {}_{\Gamma}\int q \frac{\delta u^*}{\delta x} \, d\Gamma - {}_{\Gamma}\int u \frac{\delta q^*}{\delta x} \, d\Gamma$$

$$(q_y)i = {}_{\Gamma}\int q \frac{\delta u^*}{\delta y} \, d\Gamma - {}_{\Gamma}\int u \frac{\delta q^*}{\delta y} \, d\Gamma$$

(b) Different surfaces

The boundary element method can handle different surfaces. An exterior surface is generally defined counterclockwise, and an interior surface defined clockwise. There is no basic difference in the formulation between a problem with one surface or different surfaces.

(c) Material properties

If a body is non-homogeneous it is advisable to divide it into a series of regions each having the same properties. Then the different regions can be added together, as long as equilibrium and continuity conditions are satisfied.

All the cases that have been discussed involved isotropic materials, but orthotropic properties can also be treated with minor changes.

3. THE FIDES SYSTEM

3.1 A COMPARISON WITH OTHER SYSTEMS

3.1.1 System Structure

Pre- and post-processing capabilities for discrete element methods, along with a solver for boundary element potential problems, form the backbone of the FIDES system. The name FIDES stands for Friendly Interactive Discrete Element Solver.

The field of software development is moving so quickly that certain tools that seemed to be new ideas at the time of writing the system might seem standard usage for people reading this thesis. But it is interesting to expose them, since they are the key in developing friendly interactive menu-driven software. FIDES development began on the VMS operationg system, and continued on the UNIX operating system. Both versions were running on VAX 11-780 minicomputers. But with minor changes (that will be discussed later) those versions can be adapted to microcomputers or personal computers. In fact, one version of FIDES was installed on the APOLLO machine with the AEGIS operating system. The version of FIDES discussed in this thesis consists roughly of a 9000 lines code mainly written in the "C" language. The system can be divided into a pre-processor, a solver, and a post-processor. Since parts of the solver are written in FORTRAN, one translator is used to convert data from the data structure form of "C" language to the array form of FORTRAN, and another one is used to convert back the data from arrays to data structures. Figure 3.1 shows the complete configuration of the system.

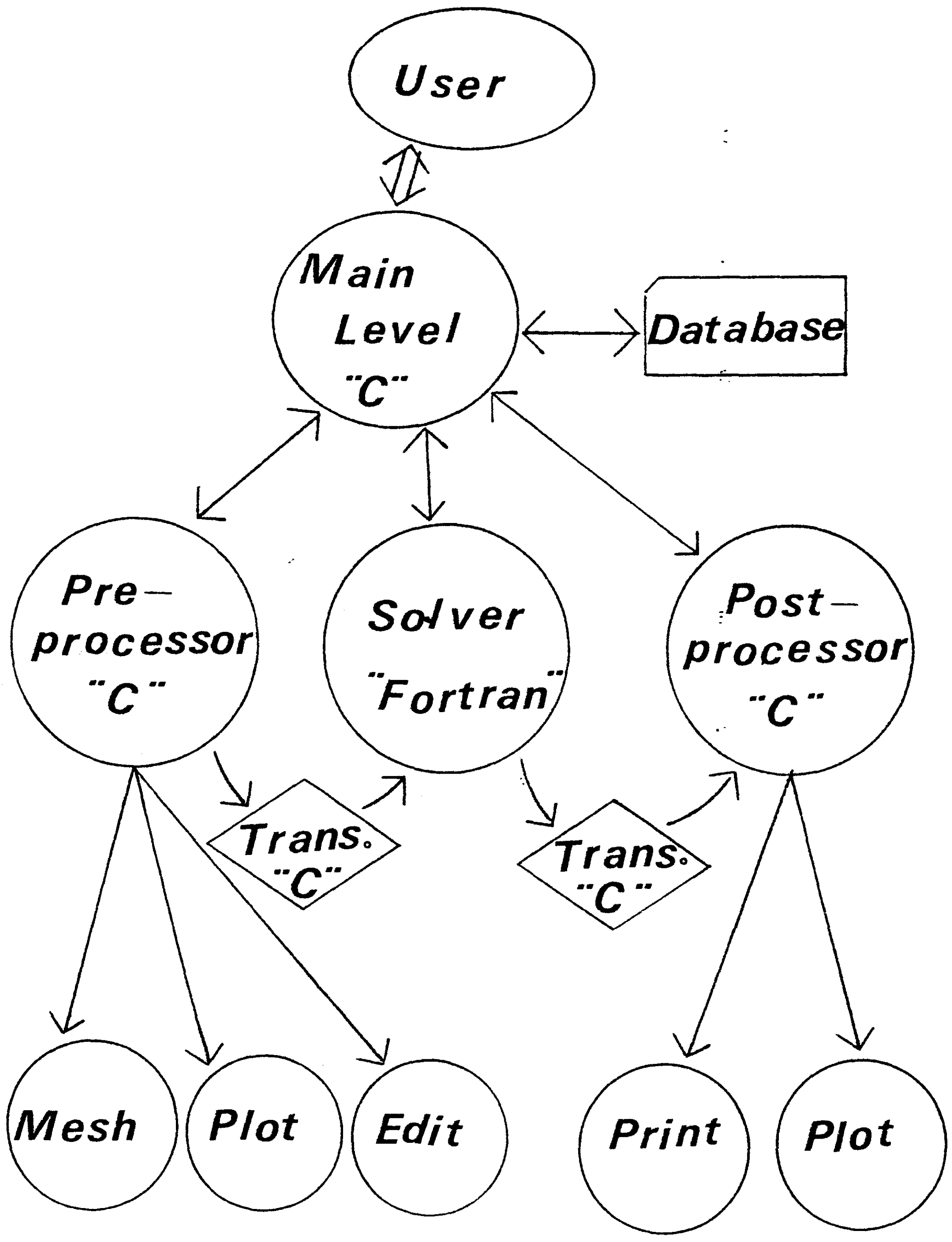


FIGURE 3.1 FIDES' structure.

Additional work on converting the solver to the "C" language and incorporating more element types is planned for the future. FIDES has been used for teaching of a discrete element course at MIT in the spring of the academic year 1984-85, and its improved capabilities will, hopefully, make it even more useful. The strong point of the system at present time is its user and programmer friendly environment. This is discussed in Sections 3.1.3 and 3.1.4.

Addition of a knowledge-based advisor to assist the user in the discretization of the problem, and in the interpretation of the results, is planned for the future. This will be discussed in Chapter 4.

3.1.2 Development Tools

The main development tools for the program are:

the operating system commands;

the graphics facilities;

the different programming languages used; and

the data base management concepts.

The different programming languages with emphasis on the "C" language will be discussed first, since this really is the key development tool. Database management concepts are generated by making intelligent use of the "C" language. And, finally, since FIDES was developed on three different operating systems, it is interesting to expose the advantages and disadvantages of each.

3.1.2.1 <u>The "C" programming language</u>

Both the pre-and post-processors are written in the "C" language, which is a "low-level" general purpose programming language. It offers many advantages over traditional programming languages such as FORTRAN. Many programmers have opted for the use of the "C" language, but care should be taken in using it. Some programmers have succeeded to make the "C" language look like FORTRAN, by always storing data in arrays, by not using data structures or pointers, by choosing incomprehensible variables' name, and by making the code very difficult to read and not programmer friendly. The advantages and disadvantages of the "C" language are discussed next. It is compared to FORTRAN since most commercially available engineering codes are written in FORTRAN.

(a) <u>Capabilities offered by "C" as well as FORTRAN</u>

1. Different data types and sizes are available (such as: char, int, short, long, float, ...).
2. The available arithmetic, relational and logical operators make representation of formulas very easy (such as: +, -, *, /, >, >=, <, <=).
3. Conditional expressions (such as: if, if... else, ...) often leads to succinct code.
4. Loops in "C" are similar to FORTRAN loops but more varied. (The different kind of loops are the "while", "for", and "do-while" loops). The "while" and "for" loops share the desirable attribute of testing the termination condition at the top, rather than at the

bottom. The "do-while" loop, tests at the bottom after making each pass through the loop body. The "for" loop is clearly superior when there is a simple initialization and re-initialization.

5. The GOTO statement exists as well in "C", but it is seldom used in programming. It was never used in writing the pre- and post-processors.
6. Both "external" and "internal" variables are present in "C". "External" variables (in contrast to "internal" variables) are defined outside any function, and are thus potentially available to many functions.
7. Subroutines do not exist in "C"; however, functions can be called as subroutines. Functions have been used extensively to make the code "transparent". As an example, the main program is shown in Fig. 3.2. Its only task is to call the appropriate subroutines.
8. Arrays exist in "C". Their use is not always recommended. They are not used in the pre- and post-processors of FIDES, and the data is stored in data-structures.
9. "C" has the same speed of execution as FORTRAN.
10. "C" and FORTRAN are both compiled as opposed to other languages, such as LISP, that are interpreted.

(b) <u>Advantages of "C" over FORTRAN</u>

1. The main advantage of "C" over FORTRAN is the existence of data-structures and pointers.

```
/************************************************************************
*                         START OF MAIN PROGRAM                         *
************************************************************************/


#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include "ndatabase.h"

#define begin           {
#define end             ;}
#define toupper(c)      (islower(c)?((c)-('a'-'A')):(c))


main()
begin                                       /*BEGIN MAIN PROGRAM*/


        char command[10];
        extern FILE *master, *prompt;

        master=stdin;
        prompt=stdout;


        n_dots=0;
        n_segs=0;
        n_nodes=0;
        n_elems=0;
        n_intpts=0;
        n_surfs=0;

        fprintf(prompt,"\n\nYOU ENTERED THE MAIN PROGRAM");


        fprintf(prompt,"\n\nCOMMANDS ARE : MESH, SOLVE, POST, EXIT");
        fprintf(prompt,"\n\nENTER YOUR COMMAND :  ");
        fscanf(master,"%s",command);

        while(strncmp(command,"exit",2)&&strncmp(command,"EXIT",2))
        begin                                       /*BEGIN WHILE LOOP*/

          switch(toupper(command[0]))
          begin                                     /*BEGIN SWITCH*/


            case 'M':                               /* MESH */

                  mesh_boundary();

             break;


            case 'S':                               /* SOLVE */

                  solve_problem();

             break;
```

FIGURE 3.2   Listing of main program.

```
          case 'P':                                          /* POST */

                post_processor();

           break;


          default:
           fprintf(prompt,"\n\n\nERROR , NO SUCH COMMAND !");
           break;

        end                                          /*END OF SWITCH*/


        fprintf(prompt,"\n\nCOMMANDS ARE : MESH, SOLVE, POST, EXIT");
        fprintf(prompt,"\n\nENTER YOUR COMMAND :");
        fscanf(master,"%s",command);

      end                                            /*END OF WHILE LOOP*/

end                                                  /*END OF MAIN PROGRAM*/


/************************************************************************
*                    END OF MAIN PROGRAM                                *
************************************************************************/
```

FIGURE 3.2   Listing of main program.

(Continued)

The intelligent use of those leads to excellent database management tools, such as linked lists, to be discussed in Section 3.1.2.2. A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling [16]. Structures help to organize complicated data, particularly in large programs, because they permit a group of related variables to be treated as a unit instead of as separate entities.

A pointer is a variable that contains the address of another variable. Pointers are very much used in "C". If they are used carelessly, they are a marvelous way to create impossible-to-understand programs [16]. However, with discipline, they can be used to achieve clarity and simplicity. As an example, the data structure for a segment representation is shown in Fig. 3.3. A segment's structure contains short integers for the segment number and the number of dots in the segment, double precision real number for the segment's length, and a single character byte to distinguish between straight, circular, or polynomial segments. It also contains pointers to the next segments, and to a list of dots. Thus, this structure contains very diverse variables that are treated as an entity, and that are all necessary to define a segment. The list of dots is part of the linked lists concept to be discussed in Section 3.1.2.2.

Since only the first 8 letters in a name are considered, segment has been given the type definition "seg". Suppose we are interested in a certain segment: seg _a. Its number is

```
                                                /*-------------------------------*/
struct segment_struct                           /*    GEOMETRICAL SEGMENT        */
        {                                       /*-------------------------------*/
        short          number,                  /* SEGMENT NUMBER                */
                       dots;                    /* NUMBER OF DOTS IN SEGMENT     */
        double         length;                  /* ORIGINAL LENGTH OF SEGMENT    */
        char           type;                    /* TYPE OF SEGMENT (S,C,P)       */
       struct dot_list_struct *dot_list;/* POINTER TO LIST OF DOTS      */
        struct segment_struct *seg_next;/* POINTER TO NEXT SEGMENT      */
                                                /*-------------------------------*/
        };                                      /*   END OF SEGMENT DEFINITION   */
                                                /*-------------------------------*/

typedef struct segment_struct seg;
seg            *seg_head, *seg_tail;
int            n_segs;
```

FIGURE 3.3   Segment's data structure.

stored in seg _a -> number. If we are interested in the type of the next segment after seg _a, then it is found in: seg _a -> seg _ next -> type. These concepts make the programmer's life easier, and will be discussed in detail in Section 3.2.

2. Another important advantage of the "C" language is the transportability. FIDES' solver is written in FORTRAN. While transporting the working version of FIDES from the VMS operating system to the UNIX operating system, and later on to the AEGIS operating system, the author had more problems with the FORTRAN part of the code than the "C" part of the code. Details will be given in Section 3.1.2.3.

3. "C" can call subroutines written in other languages very easily. This was one of the reasons to write the solver in FORTRAN. The author wanted to prove that pre- and post-processing capabilities in the "C" language can be added very easily to the large existing FORTRAN codes. The FORTRAN code does not even need to be recompiled. The only requirement is that the main program should be in the "C" language. It will be similar in structure to the one shown in Fig. 3.2. The switch statement in "C" is a special multi-way decision maker that tests whether an expression matches one of a number of constant values, and branches accordingly. The break statement causes an immediate exit from the switch. The switch statement allows a

very good structured programming style. An example is given in the main program of Fig. 3.2.

5. Another very important advantage of "C" is recursion, which means that a function may call itself either directly or indirectly. This can be used to write "search" subroutines through a tree. Recursive codes are not faster, but they are more compact and easier to write and understand [16].

6. "C" is very strong in character and bit manipulation [17]. Different examples are: the unary operator ~ which yields the one's complement of an integer, the shift operators << and >> that perform left and right shifts of their left operand by the number of bit positions given by the right operand, & which is a bitwise AND, and others such as: £ and ^.

7. "C" provides very useful language extensions by means of a simple macro pre-processor [16]. The # define capability is the most common of these extensions. It is shown in Fig. 3.2, where:

```
# define begin {
```

calls for a macro substitution of a simple kind: it replaces the left curly bracket { by the name begin. Similarly, the right curly bracket } is replaced by the name end. The author finds this notation much more programmer friendly.

The # include statement is also very commonly used. In Fig. 3.2, the line:

```
# include < stdio.h>
```

is replaced by the contents of the file "stdio", which in this case is the standard input/output library.

8. "C" has excellent debugging tools [17].
9. "C" provides additional operators such as the increment "++" and decrement "--" operators. These help in achieving shorter codes.
10. "C" is relatively small and can be described in a small space and learned quickly.
11. A compiler for "C" can be simple, compact, and easily written [16].
12. The arguments to functions are passed by copying the value of the argument and not the address, and it is impossible for the called function to change the actual argument in the caller. When it is desired to achieve "call by address", a pointer may be passed explicitly and the function may change the object to which the pointer points.
13. "C" is not a strongly typed language in the sense of PASCAL.

(c) Disadvantages of the "C" language

1. "C" being a "low-level" language does not have many high-level mechanisms. "C" does not provide operations to deal directly with composite objects such as sets, and lists of arrays considered as a whole. These must be provided by explicitly-called functions. To remedy this, several special purpose

libraries have been developed such as the math, input/output, and type's functions libraries. To be able to use a library function the library itself should be included in the code, by using the statement:

```
# include <library name> .
```

As an example, the functions getchar and putchar have been developed to read and write a character at a time. To use them one should have the following statement in the code:

```
# include stdio, on VMS, or
# include <stdio.h>, on UNIX.
```

2. "C" is not a block structured language, in that functions may not be defined within other functions.
3. Some of the "C" operators have the wrong precedence.
4. Being a relatively new language, only few commercially available codes are written in "C".

3.1.2.2 Database management concepts

The efficient use of data-structures can lead to very structured and easily understood programs. The same manipulations could be done using arrays, but it is much more tedious and can be very misleading if somebody wants to make any additions to the code. Furthermore, as outlined before, the arrays are restricted to homogeneous data and do not allow convenient handling of dynamic storage allocation for data management. The data structures' concept have been introduced in Section 3.1.2.1.

Let us consider the example of the segment data-structure shown in Fig. 3.3. It contains an integer that indicates the number of dots that have been used to define a segment. But suppose we are interested in knowing which dots are connected to a particular segment; how do we go about it? Figure 3.4 shows the dot's data structure. It is noticed that it only contains an integer for the dot's number, two double precision real numbers for the dot's coordinates, and a pointer to the next dot. The data structure for a dot is, therefore, very simplified and contains no reference to segments. Thus, the segment cannot be accessed through the dot data structure. Looking back at Fig. 3.3 one notices a pointer to a dot-list structure, which is shown in Fig. 3.5. This concept of list of data structures is one of the key elements used in developing the pre-and post-processors. It is used in the context of the segment data structure to point to a list of dots attached to this segment. The straight line, circular, and polynomial segments have 2,3 and n dots respectively. These will be discussed in detail in Section 3.2. The list of dots only contains a pointer to the dot and another pointer to the next dot. Thus the dots can be accessed through the segment, and this makes it very easy to keep track of the connectivity between dots and segments. Dots and segments are given as an example in here. The same procedure is used between surfaces, segments, and internal points. Surfaces have pointers to list of segments and lists of internal points as shown in the data-base in Appendix A. Similarly, elements used for discretization

```
                                            /*------------------------------*/
struct  dot_struct                          /*       GEOMETRICAL DOT        */
        {                                   /*------------------------------*/
        short          number;              /* DOT NUMBER                   */
        double         x, y;                /* ORIGINAL COORDINATES         */
        struct dot_struct *dot_next;        /* POINTER TO THE NEXT DOT      */
                                            /*------------------------------*/
        };                                  /*    END OF DOT DEFINITION     */
                                            /*------------------------------*/

typedef struct dot_struct dot;
dot             *dot_head, *dot_tail;
int             n_dots;
```

FIGURE 3.4   Dot data structure.

```
                                            /*------------------------------*/
struct  dot_list_struct                     /*         LIST OF DOTS         */
        {                                   /*------------------------------*/
        struct dot_struct *dot;             /* POINTER TO THE DOT           */
      struct dot_list_struct *next;         /* POINTER TO THE NEXT DOT_LIST */
                                            /*------------------------------*/
        };                                  /*  END OF DOT_LIST DEFINITION  */
                                            /*------------------------------*/

typedef struct dot_list_struct dot_list;
```

FIGURE 3.5   Dot-list data structure.

have pointers to lists of nodes. This concept of lists of structures is not restricted to the discretization of a geometry, but can be used in design of structures, where truss bars would point to lists of joints, beams would point to lists of nodes, frames would point to lists of beams, etc. It can also be used in material science for materials classification, and even in business or accounting. The main point is to differentiate between the pointers in the structure itself that point to properties, and the pointers in these lists that point to other structures. Looking back at Fig. 3.5 it is emphasized that the dot-list structure only contains pointers to other structures of type dot. Using this artifact, the code is much easier to write and much more organized. The whole picture is given in Fig. 3.6. As an example a segment #a is considered which is attached to two dots: #b and #c. Dots #b and #c can be accessed directly through the main list of dots by searching for their number. But if one is at the segment's level, and is interested to access the different dots connected to the segment, he can do so by using the dot-list data structure as shown in Fig. 3.6. Thus to find a dot with number "b", one can use:

```
     dot_b = find_dot(b);
```

where find_dot is a function, shown in Fig. 3.7, that takes the dot's numbers as an argument, and returns a pointer to the dot if it exists. Supposing we want to access dot #b through segment #a, then we would use:

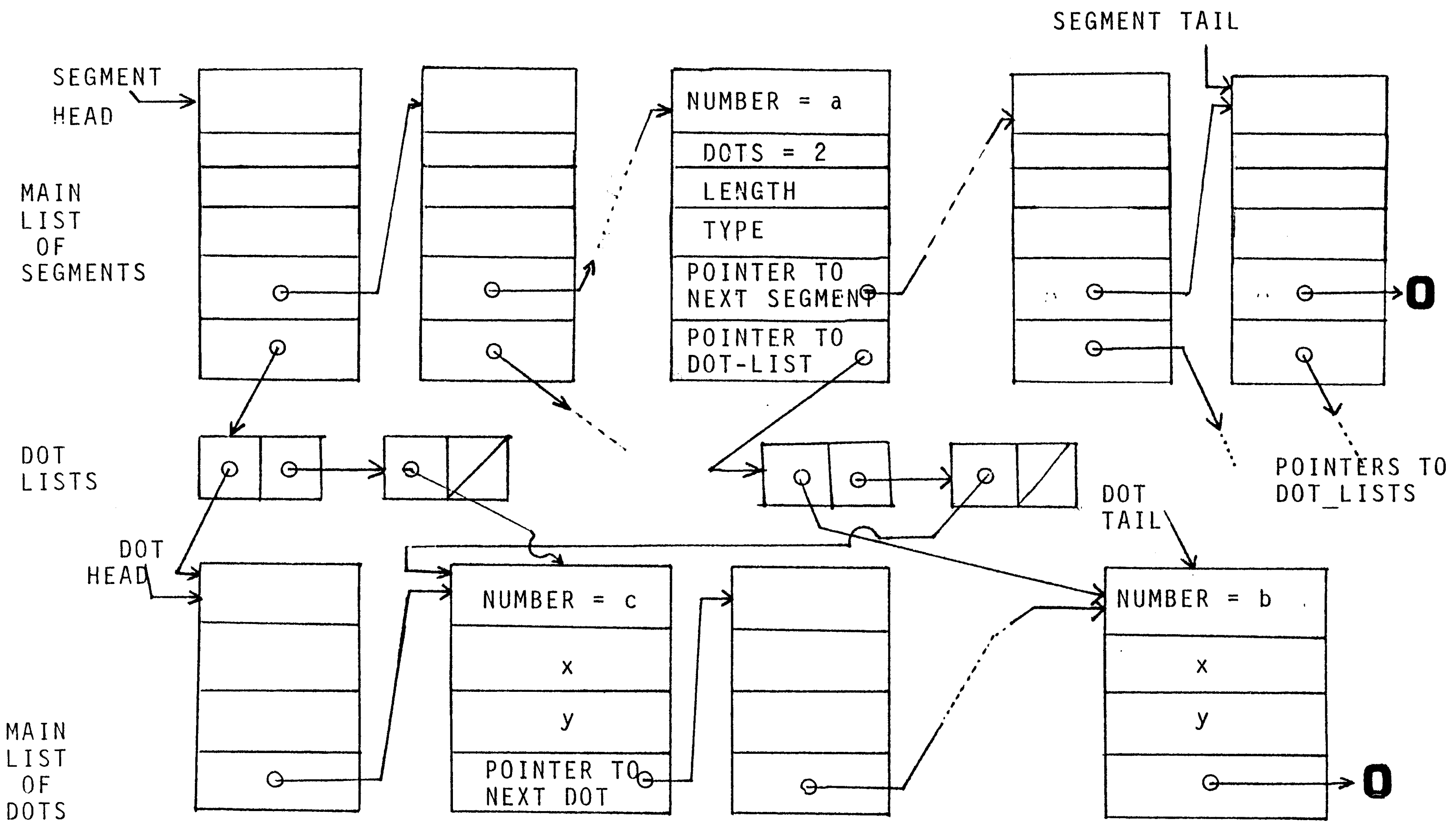


FIGURE 3.6 Relation between segment, dot and dot-list structures.

```
/*********************************************************************
*                         START  OF find_dot FUNCTION                *
*********************************************************************/



dot *find_dot(i)
int      i;

         begin

         extern int n_dots;
         extern dot      *dot_head;
         dot             *dot_b;
         int  j;

         dot_b = dot_head;
         for(j=0; j<n_dots; j++)
           begin
           if (dot_b->number == i) return(dot_b);
           dot_b = dot_b->dot_next;
         end
         return(0);
end


/*********************************************************************
*                END  OF find_dot FUNCTION                           *
*********************************************************************/
```

FIGURE 3.7   Listing of function "find_dot".

```
seg_a = find _seg(a);
dot_b = seg_a → dot-list → dot;
```

the first statement will find the segment which has a number equal to "a", and the second statement will return for dot_b a pointer to the desired dot-structure. This clearly is superior when we are not searching for a particular dot, but rather for the dots attached to a particular segment. These concepts will be explained in more detail in Section 3.2.

3.1.2.3 Different Operating systems facilities

(a) Systems' Facilities

FIDES' development began on the VMS operating system using a VAX 11-780 computer. The supported graphics package on the system was PENPLOT2. The graphics are displayed on TEKTRONIX 4025 terminals. Editing is very difficult on these terminals, since only line editing is provided. Most of the editing was done on VT 100 terminals where screen editing is possible. Due to several reasons, FIDES' development continued on the UNIX operating system using VAX 11-750 computers. The graphics used were a modified version of PENPLOT 2. Three main types of terminals were available: Digital's VT 100, VT 125, and PRØ350 terminals. Screen editing is possible on all three, but only the VT 125 and the PRØ350 support graphics. When used on the PRØ350, FIDES shows color graphics. As an example, in plotting the mesh discretization, the geometry plot is shown in red, the discretization in green, and the internal points in blue. VT 125 terminals do not support colored graphics, but have different shades of gray. A very powerful editor (EMACS) was provided, which offers a wider range of options than the VMS's EDT editor. Digital VAX station 100 terminals have been installed recently with UNIX operating systems. They support multi-windowing capabilities with high resolution graphics, contributing to a friendly environment. In the spring of the academic year 1984-85, FIDES was used for teaching of a discrete element course at MIT. For this purpose, it was installed on APOLLO computers, running on the AEGIS operating system. This was the ultimate as far as the user and programmer friendly environment is concerned. APOLLO supports a powerful window manager that allows multiple windowing, and provides a nice editing capability. Editing a file on APOLLO is as easy as writing on a very large paper with a pencil and an eraser. The AEGIS operating system offers very powerful programming development tools, such as multi-processing that allows the user to run several jobs in the foreground, virtual memory addressing, and efficient networking. The APOLLO's graphics are based on a primitive graphics package enhanced with a 2-D transformation module. The advantage of APOLLO is that it is intended to be a full power work station, in contrast to VAX 11-780 and VAX 11-750 which are intended to be time-sharing facilities.

(b) <u>Transportability Issues</u>

The "C" language's compiler on VMS is more flexible than on other operating systems. This is good for the programmer, but bad for transportability issues. While transporting FIDES from VMS to UNIX, substantial changes had to be made. The "C" version of UNIX is more strict than the VMS one. For example, attach_dot is a function that returns a pointer to a data structure of type dot. On VMS it is enough to declare it as: attach_dot(c). On UNIX, since it is returning a pointer to a dot data structure it should be declared as:

```
     dot * attach_dot().
```

In FIDES' code many simplifications have been made, and dot refers to a data structure due to the statement in the database (shown in Appendix A) that reads:

```
     typedef struct dot_struct dot;
```

This sort of statement among others made changes much easier.

Another difference was in the include statement. On the VMS system the declarations:

```
     # include stdio
     # include "ndatabase.dat"
```

include the standard input/output library, and the main database respectively, in the desired file. On UNIX these declarations are changed to:

```
     # include <stdio.h>
     # include "ndatabase.h" .
```

Although the changes are relatively simple, the debugging of the code to find them and correct them is time consuming. Using the "lint" program verifier on VMS would have made those changes much easier, since type errors are outlined. This deficiency was realized, and "lint" used consistently afterwards. Many other differences existed, but won't be discussed in detail since it is not the object of this thesis. The transportability of the "C" code from UNIX to AEGIS did not present any problems, because of the strict UNIX compiler, and of the consistent use of the "lint" program verifier.

As far as transportability is concerned, the FORTRAN part of the code did not require major changes from VMS to UNIX. But, upon switching from UNIX to AEGIS, many problems were encountered.

In general it is believed that "C", used along with the "lint" program verifier, gives less trouble than FORTRAN as far as transportability is concerned.

(c) Conclusions

The APOLLO computer offers a nicer environment than time sharing facilties. The use of the UNIX operating system is suggested, along with the use of the "C" language. It is recommended to debug the "C" code with the "lint" program verifier, to achieve easily transportable programs. It is believed that the combination of the "C" language, along with the "lint" program verifier, the UNIX operating system, and the APOLLO computers, forms an excellent user and programmer friendly environment. Specific attention is going to be given to these environments in Sections 3.1.3 and 3.1.4.

3.1.3 <u>The user-friendly environment</u>

User-friendly has become a key word in the world of software development nowadays. To achieve user-friendliness, one needs:

- Good graphics;
- A menu-driven system; and
- An interactive editor.

FIDES has all these features that will be discussed in detail in Sections 3.2 and 3.3 of this chapter relative to the pre- and post-processors.

The pre-processor's graphics allow the user to examine the geometry and the discretization of the problem, as it is being created. The post-processor's graphics allow the user to plot the results in the form of X-Y graphs, or to plot them directly on the discretization. He can, for example, plot normal fluxes on the boundary, or potential contour lines inside the domain. The user is given many options as far as choosing the different parts to be plotted. The plots can be redirected to other terminal screens on the VAX 11-750 computer, or to special files on APOLLO computers. A stack of plots can be kept on the APOLLO, showing the different stages in solving the problem. Color graphics can be obtained on PRO350 terminals connected to VAX 11-750 computers. More details will be given about the graphics capabilities, as well as examples of the different plots, in Sections 3.2 and 3.3 relative to the pre-and post-processors.

FIDES has been developed as a menu-driven system, so that the user does not need to refer to any manual to be able to run the program. This has been achieved through extensive testing of the program at the development stage by fellow graduate students. Any question or comment about the procedure to be used in running FIDES, gave rise to modifications in the existing menu. At this point it is claimed that the user does not need to read a user's manual, or go through a tutorial session to be able to use the program. Different menus were implemented for the VAX and APOLLO computers. A typical menu is the one for the discretization stage, that allows the user to create nodes and internal points, impose boundary conditions, display the discretization as printed output or in graphics form, edit what has been created, or exit. The menu used on the VAX computers is shown in Fig. 3.8. On the APOLLO, already developed menu facilities were used to lead the menu shown in Fig. 3.9.

Finally, the interactive editor allows the user to correct any mistake that he made at most of the development stages. It also allows him to create a problem, solve it, look at the results, and go back to the editor to change the discretization and run the problem again. All of this is done without exiting the system. This way one can solve many problems in a very short time.

These different features lead to a system that combines advanced display and editing capabilities with simplicity of

```
TO CREATE NODES , TYPE :NODE

TO INPUT LOADS , TYPE    :LOAD

FOR INTERNAL POINTS, TYPE:INT

TO OUTPUT INFORMATION , TYPE :OUT

TO EDIT INFORMATION , TYPE   :EDIT

TO PLOT THE PROBLEM , TYPE   :PLOT

TO EXIT THE MESH GENERATOR , TYPE :EX

ENTER YOUR COMMAND :
```

FIGURE 3.8 Discretization menu on VAX 11-75Ø with UNIX operating system.

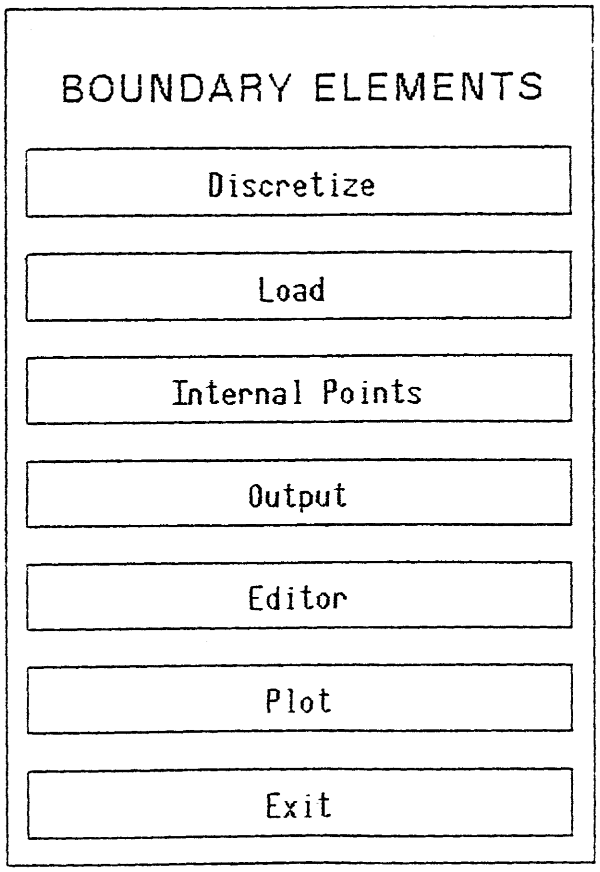


FIGURE 3.9 Discretization menu on APOLLO with AEGIS operating system.

execution. This will be clearer by examining the examples of Sections 3.2 and 3.3.

3.1.4 The programmer-friendly environment

The term programmer-friendly environment refers to two main desirable properties:

(i) Transparency of the code which allows a new programmer to be able to read it and understand it easily and quickly;

(ii) possibility of expansion which makes any additions or changes straightforward.

The "C" language contributes in making the FIDES system very easy to read and understand. Moreover, FIDES is written in a very modular form that allows changes to be made very easily, and very quickly. It also allows additions to be made as well. One can expand on the already existing modules without knowing what they contain. For expansion one only needs to take a look at the main data-structure shown in Appendix A. This is the strong point of the system, since all the useful information is stored in one place accessible to all users.

The code itself is very well documented. The programmer can know very quickly what the program is meant to do at any stage. This makes it very easy to find any mistakes, and also allows any additions to be made without any problem. This is the only way that big codes can be developed without confusion. The beginning and end of all "for" and "while" loops, "if" and "if-else" statements, "switch" statements, ..., are indicated.

Futhermore, spaces are left in between each important section of a function, and different titles are given to each of these sections. This will, hopefully, facilitate future work to be done in modifying and expanding FIDES.

3.2 PRE-PROCESSING PHASE

3.2.1 Summary of the Different Capabilities

At the main level of FIDES, the user has four different alternatives:

- Enter the pre-processor to create the geometry, discretization, and impose boundary conditions; or
- run the solver if the problem is already made; or
- access the post-processor to interpret the results; or
- exit FIDES.

These are shown in Fig. 3.1, and the APOLLO's menu for these options is shown in Fig. 3.10.

After entering the pre-processor the following choices are possible:

- create the geometry; or
- discretize the problem (imposing boundary conditions is included in the discretization) for boundary element methods; or
- mesh the problem for finite element methods; or
- exit the pre-processor.

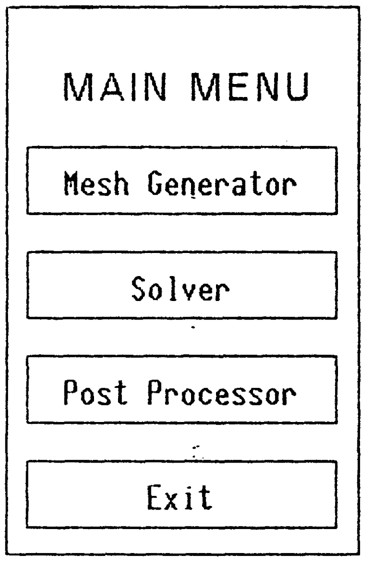


FIGURE 3.1Ø Main level menu.

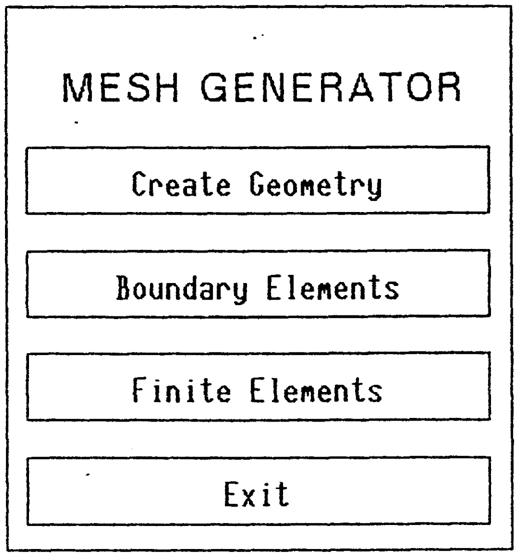


FIGURE 3.11 Pre-processor structure.

These options are shown in Fig. 3.11. The option to call a finite element mesh generator returns, at the present stage, a message informing the user that this is not developed yet. But it is included in FIDES to allow for an easier expansion in the future. As FIDES' name indicates, the program is thought of as a discrete element system, and not only as a boundary element system. Future expansion is planned to incorporate a finite element mesh generator, as well as a knowledge-based advisor for discretization. This advisor would, at the pre-processor level, recommend the best mesh to be used, or give a set of rules to guide in the creation of a good mesh, or at a more advanced stage of programming remesh the whole problem if the answers are not satisfactory. Some very simple accuracy advisors have already been designed, and will be discussed in Chapter 4. The pre-processor is completely menu-driven, with powerful editing and graphics display capabilities. It is claimed that no manual or tutorial what-so-ever is needed to run the pre-processor. The only point that is worth clarifying is the terminology used for the definition of the geometry and discretization, since there isn't one adopted convention for these. The geometrical boundary is defined by "segments" that can be straight lines, circles, or polynomials. "Dots" are points used to indicate the beginning and end of each "segment". Both "dots" and "segments" are created by the pre-processor for the geometry definition, and are shown in Fig. 3.12(a). The geometrical "segments" are discretized into boundary "elements". Then, depending on the type of expansion used for the "element", different number of

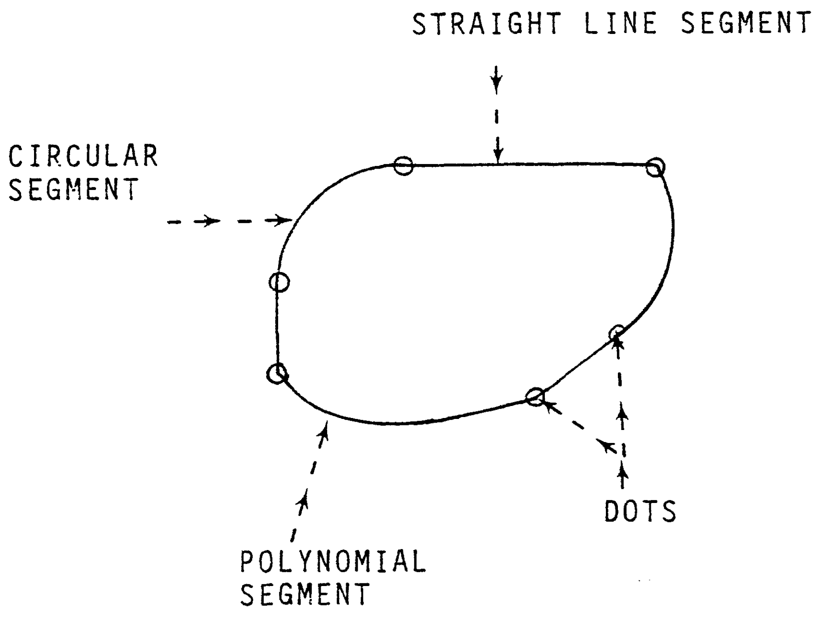


FIGURE 3.12 (a) Geometry definition.

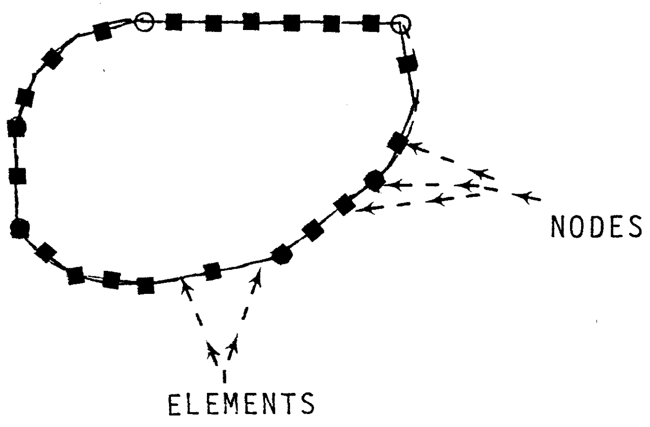


FIGURE 3.12 (b) Discretization definition.

FIGURE 3.12 FIDES' pre-processor notation.

"nodes" will be created, at which the boundary conditions will be specified. "Nodes" and "elements" used for discretization are shown in Fig. 3.12(b). The two main functions of the pre-processor are creating the geometry and discretizing the problem, and will be discussed next.

3.2.2 Creating the Geometry

The first step in defining any problem is the geometrical aspect. The user can input his boundary in the form of segments, display the current stage of the problem by either printing it on the screen or plot it interactively, or edit what has already been created. These options are shown in Fig. 3.13, and are discussed in Sections 3.2.2.1 through 3.2.2.3. It is noted that the three elements needed for user friendliness (graphics, editor capabilities, and menu-driven system) are present for the geometry definition.

3.2.2.1 Creation of segments and dots

Different types of segments are available. These include:

- straight lines;
- circular arcs; or
- polynomials.

The pre-processor creates surfaces by itself, and keeps track of the connectivity between surfaces and segments by using segment lists. If the segment to be created is the first one for a particular surface, then both the beginning and end dots are needed. Otherwise, only the end dot is required, since the starting dot of the segment under consideration is the same as

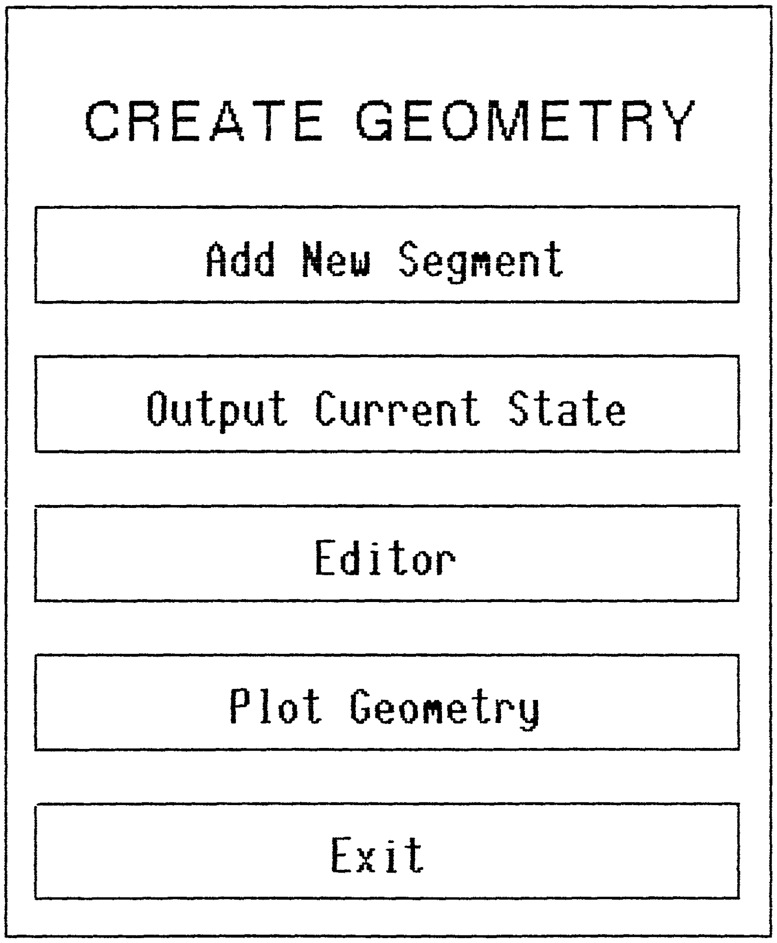


FIGURE 3.13 Menu for geometry creation.

the end dot of the previous segment. The concept of segment lists is similar to the concept of dot lists introduced in Section 3.1. The data structures for segments, dots, and dot-lists are shown in Figs. 3.3, 3.4, and 3.5 respectively. The way they all relate together is shown in Fig. 3.6. A schematic diagram of the hierarchy of all these different components is shown in Fig. 3.14.

Let us briefly describe the input needed for the creation of segments before examining the procedure used in the code itself.

To create a straight line segment one needs to input the beginning and end dot, or only the end dot, depending on whether the segment is the first one of a surface or not.

For circular segments the user needs to input the center on top of what is required for the straight line segment. Among other checks the system makes sure that the distance from the center to any of the two dots is the same. The user is also asked if his circle is counter-clockwise with respect to its center or not, as shown in Fig. 3.15. This introduces flexibility in the input, since the user does not have to worry about which point to enter first, as long as the direction of his circle is specified correctly. The circle's center is treated as a dot in the dot-list attached to the individual segment. But the center is not included in the overall list of dots. This is shown in Fig. 3.16.

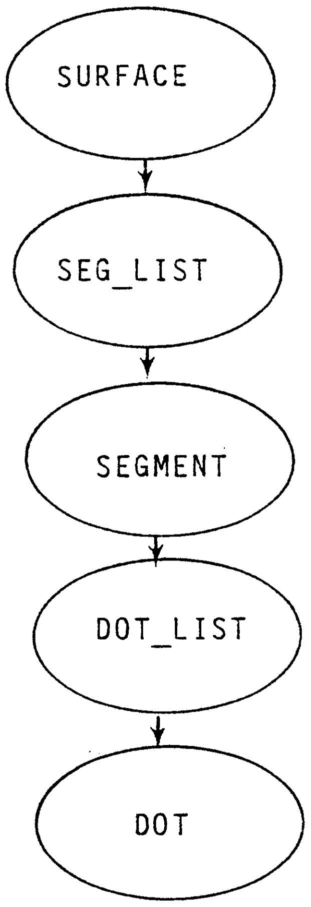


FIGURE 3.14 Interaction of the different geometry components.

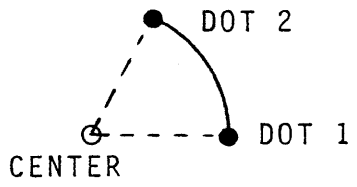


FIGURE 3.15 (a)

Counterclockwise direction.

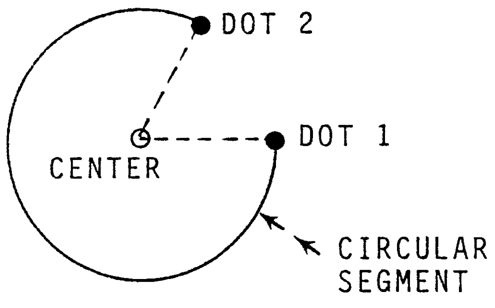


FIGURE 3.15 (b)

Clockwise direction.

FIGURE 3.15 Circle definition.

A straight line segment followed by a circular arc are defined in Fig. 3.16(a). The user has to input the coordinates of dot 1 and dot 2 to generate the straight line. Then, assuming he is still on the same surface, he will only need to specify the coordinates of dot 3 and of the center of the circular arc. In this case the circle's direction is counter-clockwise with respect to its center. The center dot is not a critical one for the discretization. Thus it is not included in the main list of dots, as shown in Fig. 3.16(b). But it is needed to calculate the circle's radius, and might be useful for later reference. Thus, it is included in the internal dot-list as shown in Fig. 3.16(c). Part of the module used for generating the dots and segments is shown in Appendix B . To access a particular dot two approaches are possible. It can be reached through the segment's internal dot-list, or by searching the main external list of dots. Thus to access dot 3 in Fig. 3.16(a), one can search through the main list of dots shown in Fig. 3.16(b), using:

```
dot_3 = find_dot (3);
```

where find_dot is a function that takes an integer as an argument, and returns a pointer to the dot which number it has been given. Or, one can make use of the fact that:

```
dot_3 = dot_head → next → next;
```

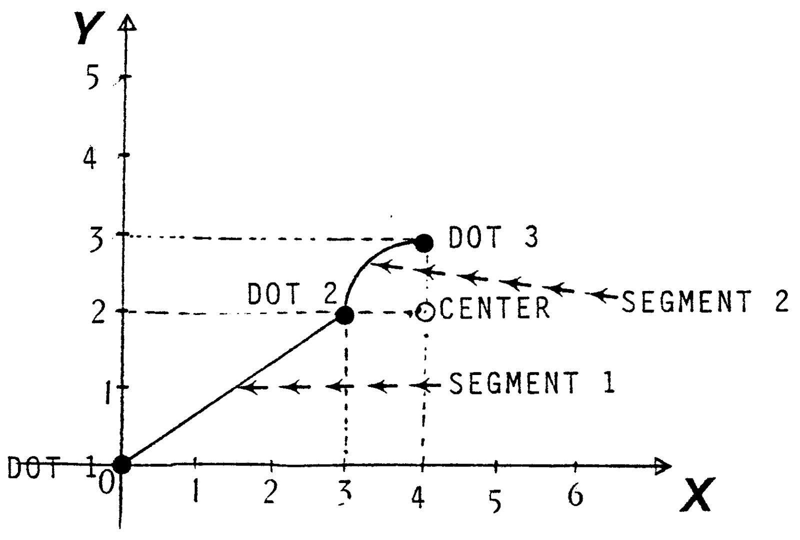


FIGURE 3.16 (a) Actual problem.

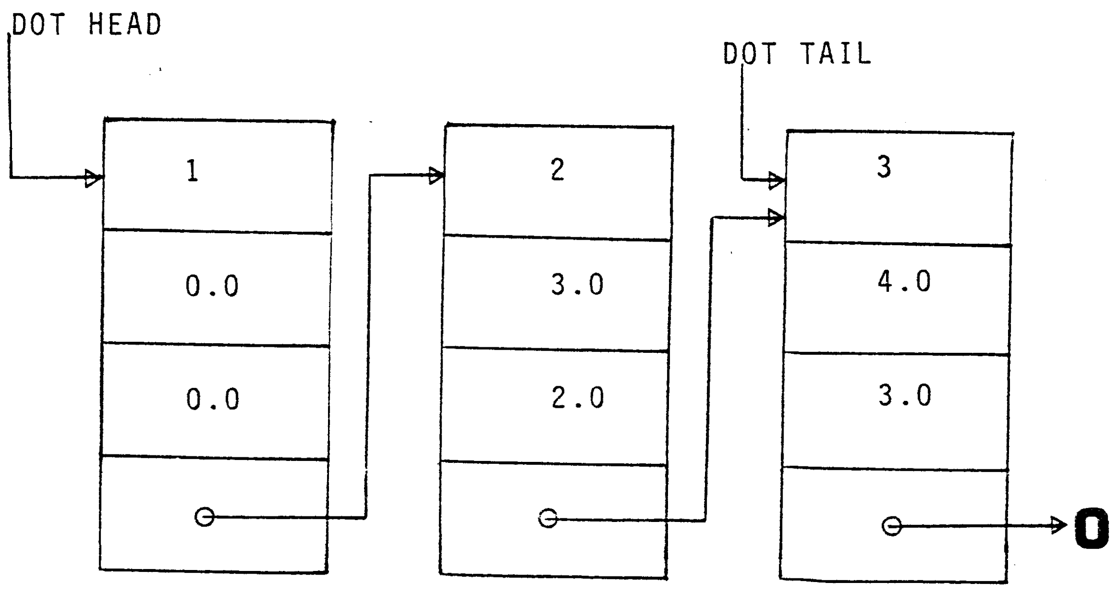


FIGURE 3.16 (b) Main list of dots.

FIGURE 3.16 Example of geometry definition.

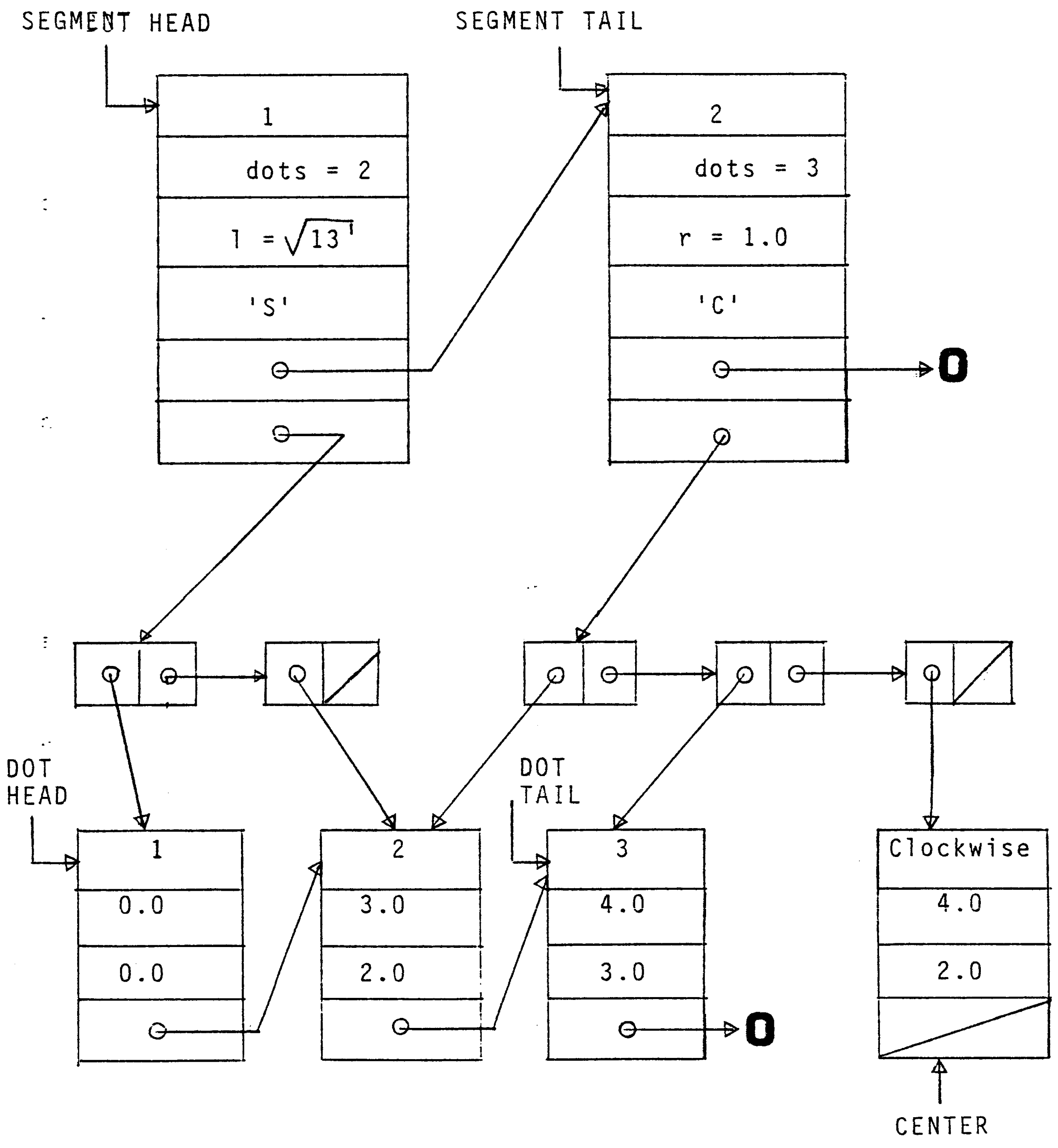


FIGURE 3.16 (c) Interaction of segments and dot_lists.

FIGURE 3.16 Example of geometry definition.

where dot_head is a pointer to the first dot in the list. This can only be done if the dots are assumed to be in the order shown in Fig. 3.16(b). The other way would be to access dot 3 through segment 2. Assuming seg_2 represents segment 2, then dot 3 can be reached by:

```
dot_3 = seg_2 -> dot_list -> dot -> next;
```

This is shown in Fig. 3.16(c). The pointer to segment 2 is found by searching through the external list of segments shown in Fig. 3.16(c).

A closer look at the different details involved in the creation of the circular arc segment of Fig. 3.16(a) is going to be taken. The code listing can be found in Appendix B .

Assume the user has already created the straight line segment, and that he specifies a circular arc to be created next. The program notices that this segment is not the first one on the surface, and thus, only asks for the coordinates of dot 3 and the center of the circle. After checking that the radius of the circle is correct, the system will allocate space for dot 3 and the circle's center, using the function malloc:

```
dot_a = (dot *) malloc [sizeof (dot)].
```

The function "sizeof" yields an integer equal to the size of the data structure dot. This size is given in unspecified units called "bytes", which are the same size as a char. The function "malloc" allocates space in memory for dot_a, equal to the integer returned by "sizeof". Then, once the dot is created, its coordinates are assigned using:

```
dot_b->x=x2

dot_b->y=y2
```

where x2 and y2 are specified by the user. The center dot does not have a number, but it is set to +1 or -1, depending on whether the circle is counterclockwise or clockwise with respect to its center. This is shown in the code by

```
if {toupper(counterclockwise [0]) == 'N'}
dot_c->number = -1;
else
dot_c->number = 1;
```

Dot 3 is going to be attached to the main list of dots using the following:

```
if (n_dots == 0)
     begin
     dot_head = dot_a;
     dot_tail = dot_a;
     dot_a->number = i;
end
```

```
     if (n_dots != 0)
          begin
          dot_tail->dot_next = dot_a;
          dot_a->dot_next = dot_a;
          dot_a->number = i;
          dot_tail = dot_a;
     end
```

Where the following macro substitutions for "begin" and "end have been used:

```
     #define begin     {
     #define end       ;}
```

These substitutions are used most of the time in the code. Their advantage is to facilitate the understanding of the program by achieving a better transparancy, and their disadvantage is that the programmer cannot use the UNIX feature that checks for missing brackets.

As already mentioned, the circle's center is not going to be attached to the main external list of dots.

The segment is created next along with its different properties. Its type is referred to as 'C' standing for circle, compared to 'S' for straight line. To be able to use a common data-structure, the double precision floating point for length is used to store the radius in the circle case. A note explaining this is found in the code to help the programmer realize it. The segment creation is shown below:

```
    /* CREATE SEGMENT */
    seg_w=attach_seg(ns++);
    seg_w->dots=3;
    seg_w->length = r;
    seg_w->type = 'C';
```

Finally, the dot-list is created and attached to the segment as follows:

```
    /*POINT TO THE THREE DOTS */
    dot_list_a = attach_dot_list();
    seg_w->dot_list = dot_list_a;
    dot_list_b = attach_dot_list();
    dot_list_a->next = dot_list_b;
    dot_list_c = attach_dot_list();
    dot_list_b->next = dot_list_c;
    dot_list_a->dot = dot_a;
    dot_list_b->dot = dot_b;
    dot_list_c->dot = dot_c;
```

In the case of the circular segment shown in Fig. 3.16(a), dot_a, dot_b, and dot_c refer to dot 2, dot 3 and the circle's center respectively, while seg_w refers to segment 2.

This covers in enough detail the creation of segments and dots.

The last type of segment is the polynomial one. It is being implemented but is not complete yet. Ideally the user can create any segment of the form:

$$y^n + a_1 y^{n-1} + \ldots + a_{n-1} y = b_1 x^\rho + b_2 x^{\rho-1} + \ldots$$
$$+ b_\rho x + c,$$

where n and ρ are the highest exponents of y and x respectively. There is $(n+\rho)$ unknown coefficients in general. The user should specify all the zero coefficients as well as the beginning and end points of the segment. The pre-processor will then require the coordinates of q points to solve for the q remaining unknown coefficients where:

$$q = n+\rho-2\text{-number of zero coefficients.}$$

Creating the boundary geometry is useful, but would not be user-friendly without the graphics and editing capabilities that will be discussed in the next sections.

3.2.2.2 Display of geometric input

The geometric input can be plotted on the screen, printed on the screen, or in a file. A graphic display gives the user a general idea of what has been created, and allows him to check quickly if the input is correct. The printed display provides more precise details about dot's and segment's properties and connectivity between them.

(a) Graphics display

The same module is used to plot the geometry and discretization. It is also employed at the post-processor stage to display the mesh. Different flags that refer to various objects and can be turned "on" or "off" exist. The different objects that can be plotted are:

- dots,
- segments,
- nodes,
- elements, and
- internal points.

Dots, nodes, and internal points are plotted as diamonds, squares, and X respectively. Segments and elements are represented by straight or curved lines. Dots and segments are used to plot the geometry; nodes and elements represent the discretization and lie on the boundary; while internal points are situated inside the domain.

The graphics menu is shown in Fig. 3.17(a). To plot the geometry the flags for dots and segments should be both "on", as shown in Fig. 3.17(b). A flag keeps its last state (i.e., "on" or "off") until it is changed again. Thus, if the user wants to plot all that has been created, the five different flags are left "on".

Examples of the pre-processor graphics are shown in Fig. 3.18 and 3.19. Both geometries are created using straight lines and circular segments. The graphics shown in Fig. 3.18 might not represent any practical situation, but is used to illustrate the different type of geometries that can be created, while the plot of Fig. 3.19 can represent a pipe's elbow. The dots' and segments' numbers are not drawn yet by the pre-processor graphics, and have been printed on the figures for clarity. Options are being developed to allow the user to plot dot's

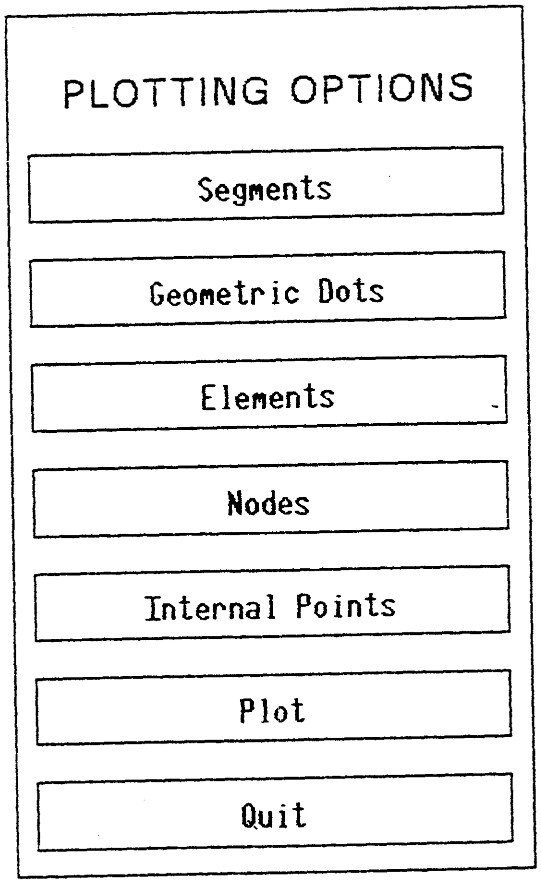


FIGURE 3.17 (a) Graphics menu.

```
PLOT OPTIONS:

 INT_PTS     DOTS     SEGMENTS     NODES     ELEMENTS
   OFF        ON         ON         OFF         OFF
```

FIGURE 3.17 (b) Flags set-up.

FIGURE 3.17 Graphics options.

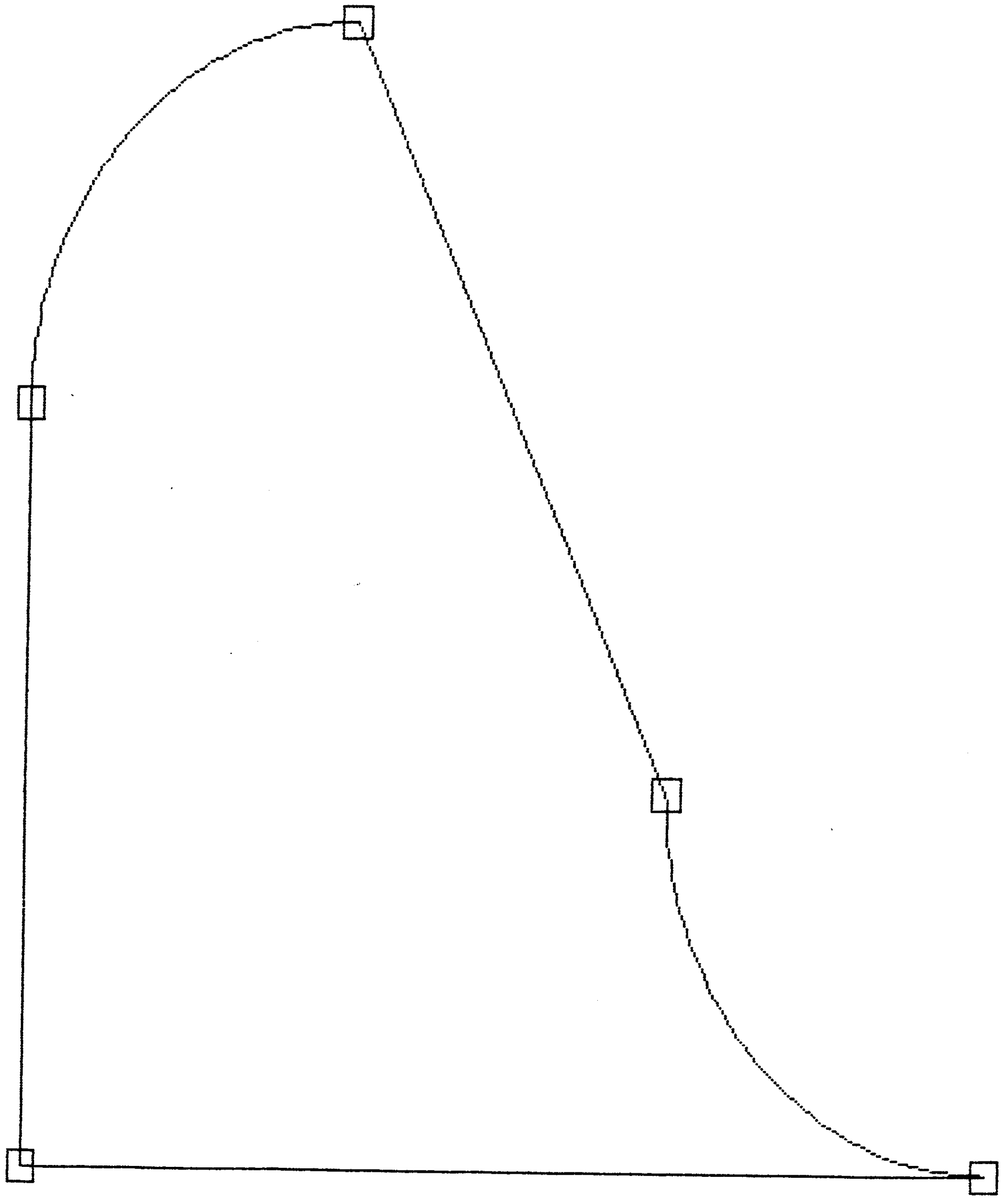


FIGURE 3.18 Pre-processor graphics' display of first geometry example.

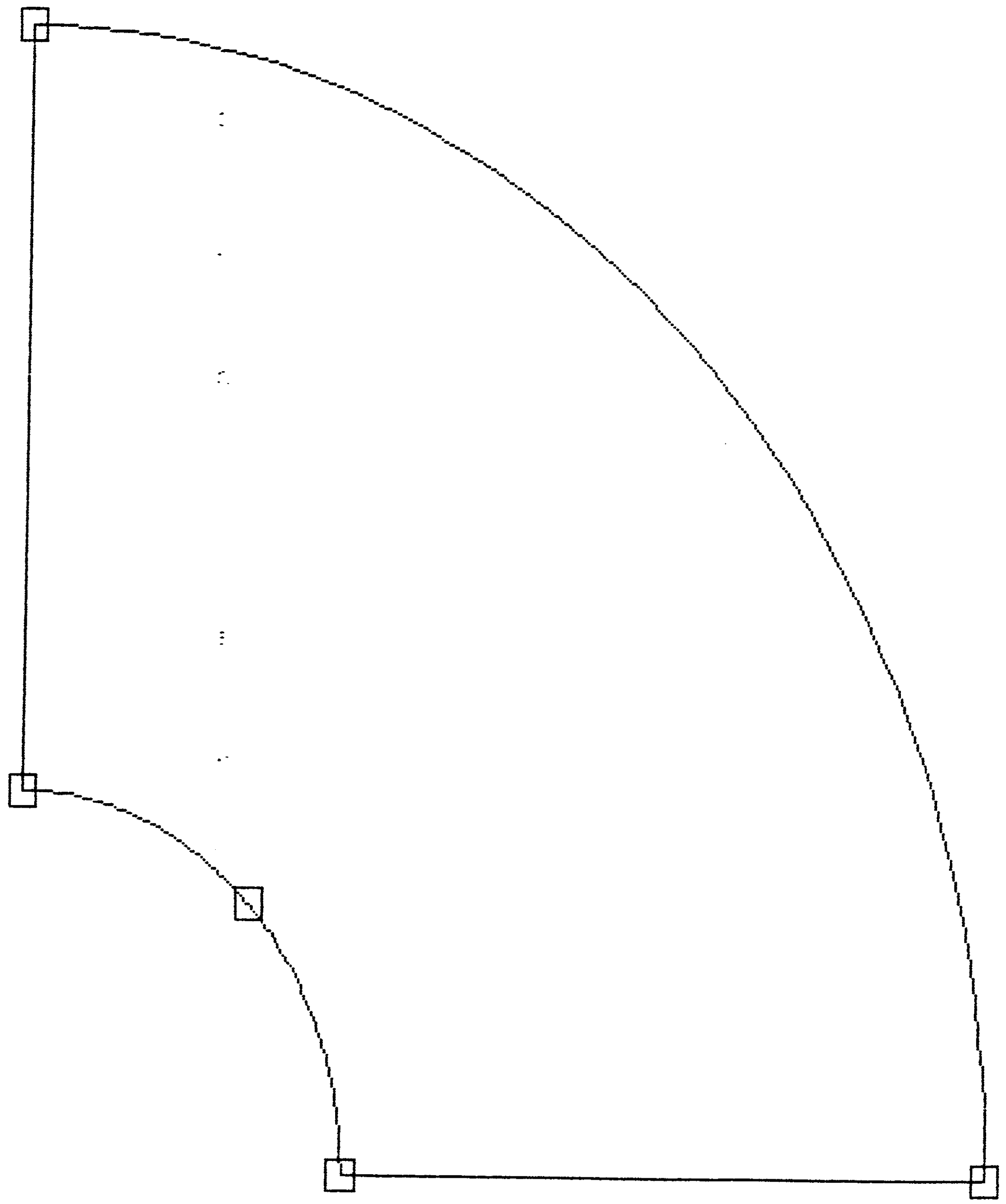


FIGURE 3.19 Pre-processor graphics display of second geometry example.

numbers only, segments' numbers only, both, or neither on the geometry.

The graphics shown look much nicer on the screen because the hard copy printer used did not give as good a resolution as expected. Furthermore, color graphics exist on the PRO-350, where the dots and segments representing the geometry are plotted in red, the nodes and elements used for discretization are drawn in green, and the internal points plotted in blue. These plots help the user in tracking errors very quickly. In order to obtain detailed information about the geometry, the printing option can be used.

(b) <u>Printing dots' and segments' properties</u>

The user can print what has been created directly on the screen. If no dots or segments have been created, the system returns the following error message:

"SORRY! NO DOTS DEFINED YET!!"

Otherwise, it starts by printing the total number of dots created, followed by the list of dots, along with the numbers and coordinates.

The segment's output is divided into straight lines and circular arcs. For both types of segments, their dot connectivity is shown by printing the starting and ending dots. The segment's length is printed for straight lines, while the center's coordinates along with the radius are printed for circular arcs. Examples of printed output for the graphics plotted in Fig. 3.18, 3.19 are shown in Figs. 3.20, 3.21.

```
      LIST OF DOTS

    NUMBER OF DOTS = 6


   DOT  COORDINATES
   No.    X         Y
   ---  -----    -----

     1     0.00     0.00
     2    15.00     0.00
     3    10.00     5.00
     4     5.00    15.00
     5     0.00    10.00
     6     0.00     0.00




    NUMBER OF SEGS = 5


      LIST OF STRAIGHT LINE SEGMENTS

   SEG    DOT1     DOT2    LENGTH
   No.     No.      No.
   ----   ----     ----     ------

     1      1       2      15.000000
     3      3       4      11.180340
     5      5       6      10.000000



      LIST OF CIRCULAR SEGMENTS

    SEG    DOT1     DOT2    CENTER   CENTER   RADIUS
    No.     No.      No.      X        Y
   ----    ----     ----    ------   ------   -------

     2       2       3     15.000000  5.000000  5.000000
     4       4       5      5.000000 10.000000  5.000000
```

FIGURE 3.2Ø Printed output of geometry shown in Fig. 3.18.

```
 NUMBER OF DOTS = 6


DOT  COORDINATES
No.    X        Y
---  -----    -----


  1     1.00     0.00
  2     3.00     0.00
  3     0.00     3.00
  4     0.00     1.00
  5     0.71     0.71
  6     1.00     0.00



 NUMBER OF SEGS = 5


   LIST OF STRAIGHT LINE SEGMENTS

 SEG   DOT1    DOT2   LENGTH
 No.    No.     No.
----   ----    ----   ------


  1      1     2    2.000000
  3      3     4    2.000000



   LIST OF CIRCULAR SEGMENTS

 SEG   DOT1    DOT2   CENTER  CENTER  RADIUS
 No.    No.     No.     X       Y
----   ----    ----   ------  ------  -------


  2      2     3     0      0     3.000000
  4      4     5     0      0     1.000000
  5      5     6     0      0     1.000000
```

FIGURE 3.21 Printed output of geometry shown in Fig. 3.19.

3.2.2.3 Editing capabilities

A powerful editor that allows the user to modify the geometry and correct mistakes is available. Its main tasks are:

- add dots or segments;
- delete dots or segments;
- get informaton about dots or segments, or
- modify existing data for dots or segments.

A similar editor, using the same basic modules, is used for nodes and elements. More variables are involved in the definition of the discretization, and, thus, the editor for nodes and elements is more complex. To avoid redundancy, the reader interested in editing capabilities is referred to Section 3.2.3.5.

3.2.3 Discretization of the problem

After creating the geometry, the need to discretize it and impose boundary conditions arises. Upon entering the boundary element mesh generator the following options are available:

- create elements and nodes;
- create internal points;
- impose boundary conditions;
- plot the discretization;
- print the current stage;
- edit what has been created; and
- exit the mesh generator.

These are shown in Fig. 3.8, and details about each of them are discussed in the following sub-sections.

3.2.3.1 Generation of elements and nodes

Different types of elements and nodes are available. Each segment can be divided into elements of:

- equal length;
- graded increasing or decreasing size; and
- arbitrary length.

Two basic types of elements are available:

- discontinuous; or
- continuous.

At the present time three possible choices for elements' expansion exist:

- constant;
- linear; or
- quadratic.

All these different types of elements and nodes are available for both:

- straight lines, and
- circular segments.

The first step is to specify the different elements' length.

To discretize a segment into equal elements, the number of elements: n should be specified, and the program will create n elements of length equal to: (segment length/n). On the other hand, if graded elements are needed, the ratio of the last element length to the first element length should be specified. Depending on whether this ratio is greater or smaller than one, the elements length will be increasing or decreasing respectively. Given the number of elements and the ratio of the last to first element length, the program calculates the length of each element, and creates all the elements by itself. The "arbitrary length" option can also be used, where the size of each element is specified separately. This option is not expected to be used very often, because it would defeat the purpose of the pre-processor.

Once the element's length is chosen, the type of element, as well as the expansion for the different boundary variables have to be specified. The continuous elements are going to be discussed first.

The use of continuous elements implies that two adjacent elements have the same values for the boundary variables at their intersection. For linear and quadratic expansion, this means that adjacent elements have one node in common at their intersection. When a linear expansion is employed, there exists one node at the beginning and another one at the end of the element, as shown in Fig. 3.22(a). A quadratic expansion would need one more node in the middle of the element as in Fig. 3.22(b).

Discontinuous elements can have jumps (i.e. different values) in the boundary quantities at the ends. The advantage of using them is that nodes are created on the interior of the elements, avoiding corner problems. If a constant expansion is specified, one node would be created at the middle of each element as shown in Fig. 3.23(a). Linear expansion implies

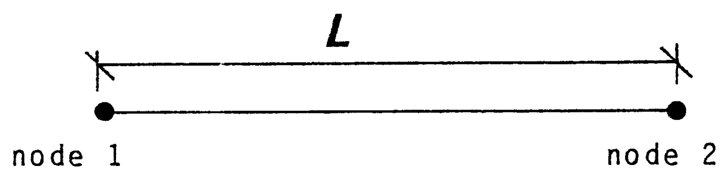


FIGURE 3.22 (a) Linear expansion.

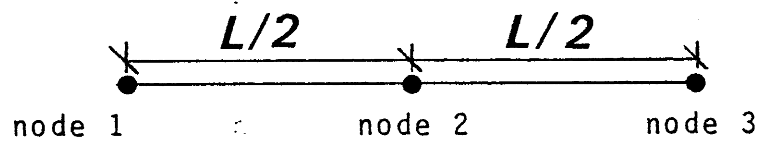


FIGURE 3.22 (b) Quadratic expansion.

FIGURE 3.22 Continuous elements.

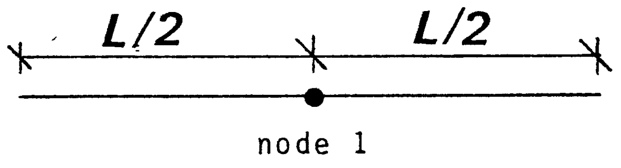


FIGURE 3.23 (a) Constant expansion.

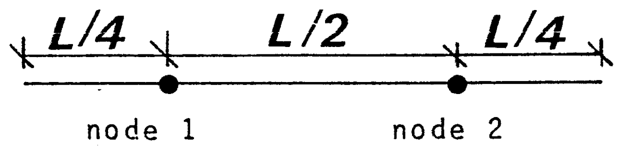


FIGURE 3.23 (b) Linear expansion.

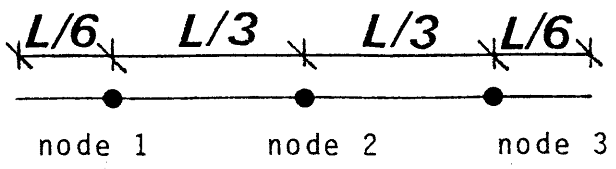


FIGURE 3.23 (c) Quadratic expansion.

FIGURE 3.23 Discontinuous elements.

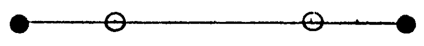
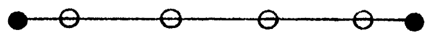
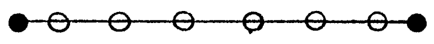
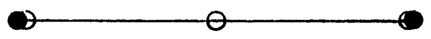
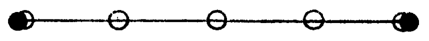
two nodes at a distance of L/4 from each end of the element (where L is the length of the element), as shown in Fig. 3.23(b). While for quadratic expansion two nodes at a distance of L/6 from the element's ends and one node in the middle are created, as shown in Fig. 3.23(c).

It is interesting to show examples of adjacent elements to illustrate these concepts. Table 3.1 shows the different possible discretization of two adjacent equal elements, while Table 3.2 involves two and three graded adjacent elements. All the different elements and nodes have been shown for straight lines, but the same elements exist for circular curves.

As an example, the geometry shown in Fig. 3.18 has been discretized according to Table 3.3. The result outputted by the pre-processor is shown in Fig. 3.24. In this graphic both the dots and nodes are plotted. For continuous elements it is noticed that the dots and nodes overlap at the corners, while for discontinuous elements they do not.

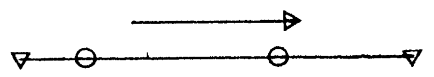
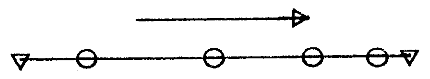
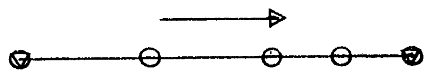
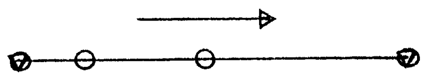
A discretization of the geometry of Fig. 3.19 is shown in Fig. 3.25, with three internal points. The arc of circle between dots 4 and 6 has been divided into two segments: 4 and 5, so that both segments can be graded decreasingly from dot 5 to dots 4 and 6 respectively.

A summary of the discretization for Fig. 3.25 is shown in Table 3.4. A very important remark is that the discretization should be done in the counter-clockwise direction for exterior surfaces, and in the clockwise direction for interior surfaces. This is outputted every time the discretization module is

ELEMENT'S SIZE	NUMBER OF ELEMENTS	RATIO OF LAST ELE--MENT'S LENGTH TO FIRST ONE	CONTINUOUS OR DISCONTI--NUOUS	EXPANSION USED	GRAPHICS
EQUAL	2	1	DISCONTI--NUOUS	CONSTANT	
EQUAL	2	1	DISCONTI--NUOUS	LINEAR	
EQUAL	2	1	DISCONTI--NUOUS	QUADRATIC	
EQUAL	2	1	CONTINUOUS	LINEAR	
EQUAL	2	1	CONTINUOUS	QUADRATIC	

● == DOTS; ○ == NODES.

TABLE 3.1 Equal elements' discretization.

ELEMENT'S SIZE	NUMBER OF ELEMENTS	RATIO OF LAST ELEMENT LENGTH TO FIRST ONE	CONTINUOUS OR DISCONTI- -NUOUS	EXPANSION USED	GRAPHICS
GRADED INCREASING	2	2.0	DISCONTI- -NUOUS	CONSTANT	
GRADED DECREASING	2	0.5	DISCONTI- -NUOUS	LINEAR	
GRADED DECREASING	2	0.5	CONTINUOUS	QUADRATIC	
GRADED INCREASING	3	2.0	CONTINUOUS	LINEAR	

▽ == DOTS; ○== NODES.

TABLE 3.2 Graded element's discretization.

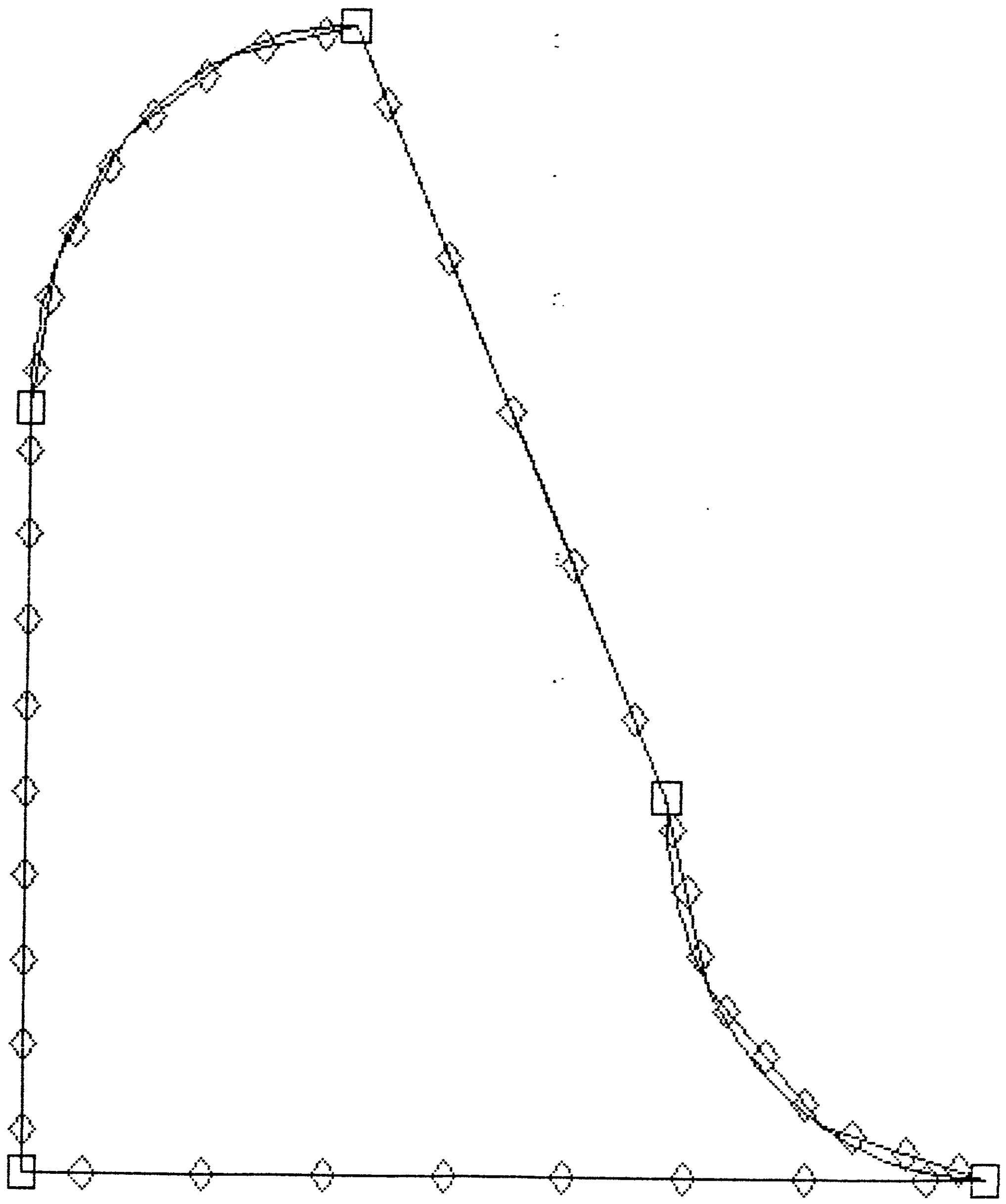


FIGURE 3.24 Discretization of geometry of Fig. 3.18 according to Table 3.3.

SEGMENT'S NUMBER	NUMBER OF ELEMENTS	SIZE OF THE ELEMENTS	CONTINUOUS OR DISCONTINUOUS	EXPANSION USED
1	4	Equal	Continuous	Linear
2	3	Equal	Continuous	Quadratic
3	5	Equal	Discontinuous	Constant
4	4	Equal	Discontinuous	Linear
5	3	Equal	Discontinuous	Quadratic

TABLE 3.3 Discretization outline for Fig. 3.24.

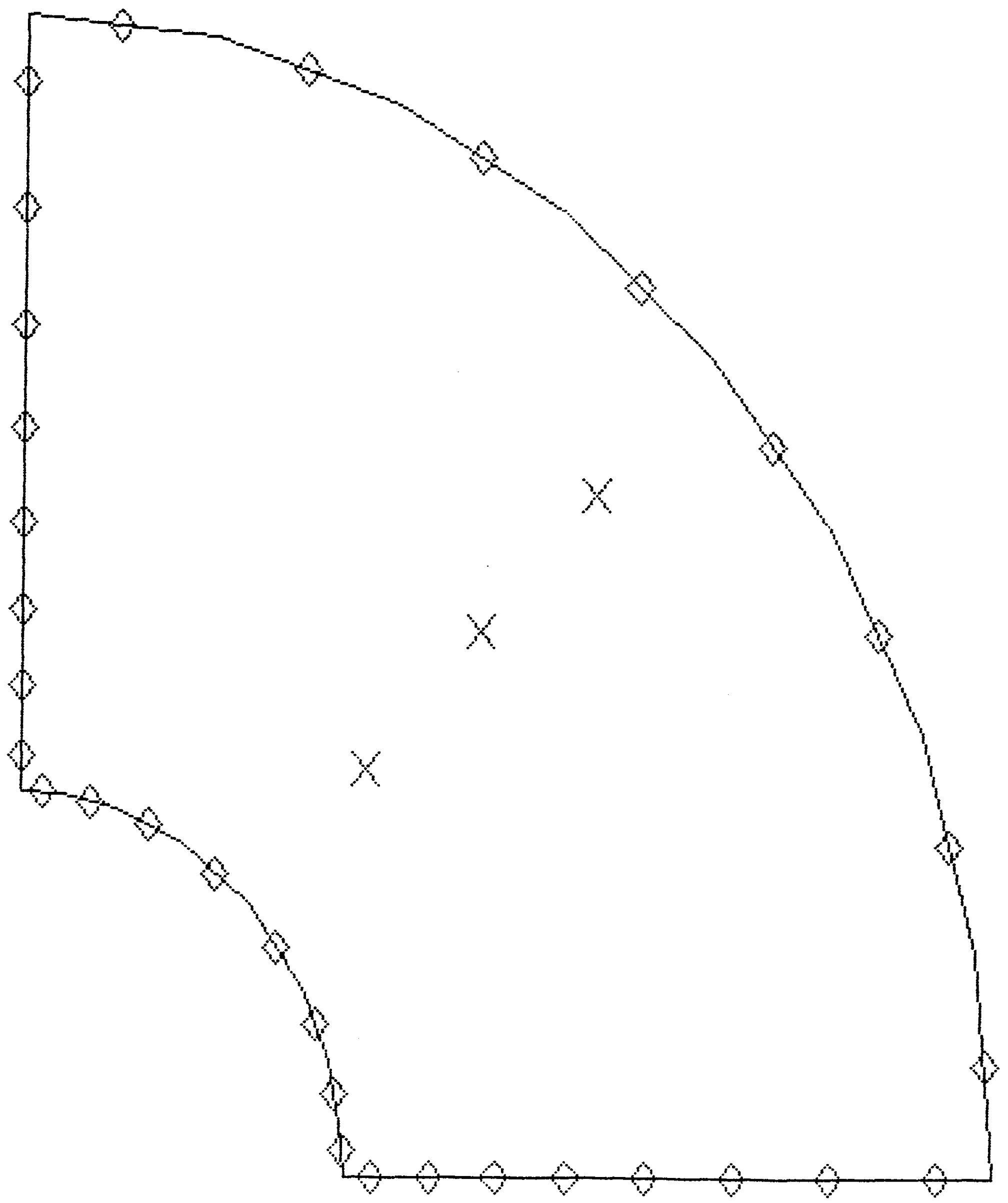


FIGURE 3.25 Discretization of geometry of Fig. 3.19 according to Table 3.4.

SEGMENT'S NUMBER	NUMBER OF ELEMENTS	SIZE OF THE ELEMENTS	CONTINUOUS OR DISCONTINUOUS	EXPANSION USED
1	8	Graded increasing Ratio = 2.0	Discontinuous	Constant
2	8	Equal	Discontinuous	Constant
3	8	Graded decreasing Ratio = 0.5	Discontinuous	Constant
4	4	Graded increasing Ratio = 2.0	Discontinuous	Constant
5	4	Graded decreasing Ratio = 0.5	Discontinuous	Constant

TABLE 3.4 Discretization outline for Fig. 3.25.

entered, and is shown in Fig. 3.26. This is another example of a friendly feature since the user is not required to remember this kind of detail. More details about the graphics will be discussed in Section 3.2.3.4.

The procedures to create elements and nodes are very similar to the ones discussed for segments and dots. The elements' and nodes' data structures are shown in Fig. 3.27 and 3.28 respectively. The element's data structure contains integers for its number, and the quantity of nodes that exist as well as double precision floating point for the original and final length and the coordinates of its ends. In potential problems, only the original quantities are used, but the final variables are added for possible use in elasticity problems, so that the same database can be used for both problems. In fact, the pre-processor is used to create a discretization for both types of problems. It is only at the solver stage that an expansion is needed. Also found in the element's data-structure are single byte characters referring to the type and expansion used. The type can either be a "S" for a straight lines or a "C" for a curved isoparametric element. The expansion can take the letters "C", "L" or "Q" to represent constant, linear, or quadratic respectively. Similarly to the segment's data structure, the element's one contains pointers to the next element and to a node_list data structure. Connectivity between elements and nodes is established using this node_list, which only has pointers to the node and to the next node_list. Data structure for the node_list is shown in Appendix A. As shown in

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
EXTERIOR BOUNDARY SHOULD BE COUNTERCLOCKWISE.
   INTERIOR BOUNDARY SHOULD BE CLOCKWISE.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

FIGURE 3.26 Pre-processor warning for discretization direction.

```
                                              /*------------------------------*/
struct element_struct                         /*  ELEMENT FOR DISCRETIZATION  */
    {                                         /*------------------------------*/
    short                 number,             /* ELEMENT NUMBER               */
                          nodes;              /* NUMBER OF NODES IN ELEMENT   */
    double                length,             /* ORIGINAL LENGTH OF ELEMENT   */
                          lnew,               /* FINAL LENGTH OF ELEMENT      */
                          x1,y1,              /* ORIGINAL COORD. OF ELEM. START*/
                          x2,y2,              /* ORIGINAL COORD. OF ELEM. END*/
                          x1new,y1new,        /* FINAL COORD. OF ELEM. START  */
                          x2new,y2new;        /* FINAL COORD. OF ELEM. END    */
    char                  type,               /* TYPE OF ELEMENT (S,C,P)      */
                          expansion;          /* EXPANSION OF ELEMENT (C,L,Q) */
    struct node_list_struct *node_list;       /* POINTER TO LIST OF NODES     */
    struct element_struct *next;              /* POINTER TO NEXT ELEMENT      */
                                              /*------------------------------*/
    };                                        /* END OF ELEMENT DEFINITION    */
                                              /*------------------------------*/

typedef struct element_struct elem;
elem            *elem_head, *elem_tail;
int             n_elems;
```

FIGURE 3.27 Element's data structure.

Fig. 3.28, the node's data-structure is more complex than the dot's data structure. It contains an integer for the node's number, as well as double precision real numbers for the original and deformed coordinates, and the values of the different boundary conditions. Character bytes exist to indicate the kind of boundary conditions used. For potential problems these character bytes have not been used. They are added for a possible expansion to elasticity problems. They have not been used yet in the code, and their names will probably be changed to make them more programmer-friendly (i.e. more meaningful) upon utilization. Finally, a pointer to the next node is provided as well.

The code for discretization of the segments is too big to be included in this thesis. It is similar to the one shown for the creation of nodes and elements.

3.2.3.2 Addition of internal points

The internal point concept is a simple one. Its data structure, shown in Appendix A, only contains a short integer for its number, and double precision real numbers for its coordinates, potential, and flux. It also has a pointer to the next internal point. To create internal points, the user only needs to specify the number of internal points desired, and their coordinates. No boundary conditions are applied at the internal points, but the potential and flux are automatically calculated at those locations. Three internal points are shown in the discretization of Fig. 3.25.

```
                                                   /*------------------------------*/
struct node_struct                                 /*   NODE FOR DISCRETIZATION    */
        {                                          /*------------------------------*/
        short   number;                            /* NODE NUMBER                  */
        double  x, y,                              /* ORIGINAL COORDINATES         */
                dx, dy,                            /* DISPLACEMENTS                */
                xnew,ynew,                         /* DEFORMED SHAPE COORDINATES   */
                bc1,                               /* BOUNDARY CONDITION 1         */
                bc2,                               /* BONDARY CONDITION 2          */
                u,                                 /* POTENTIAL IN POTENTIAL PROBS.*/
                q;                                 /* FLUX IN POTENTIAL PROBS.     */
        char    kbc1,                              /* KIND OF BOUND. COND. 1       */
                kbc2;                              /* KIND OF BOUND. COND. 2       */
        struct node_struct *next;                  /* POINTER TO THE NEXT NODE     */
                                                   /*------------------------------*/
        };                                         /*    END OF NODE DEFINITION    */
                                                   /*------------------------------*/
```

FIGURE 3.28 Node's data structure.

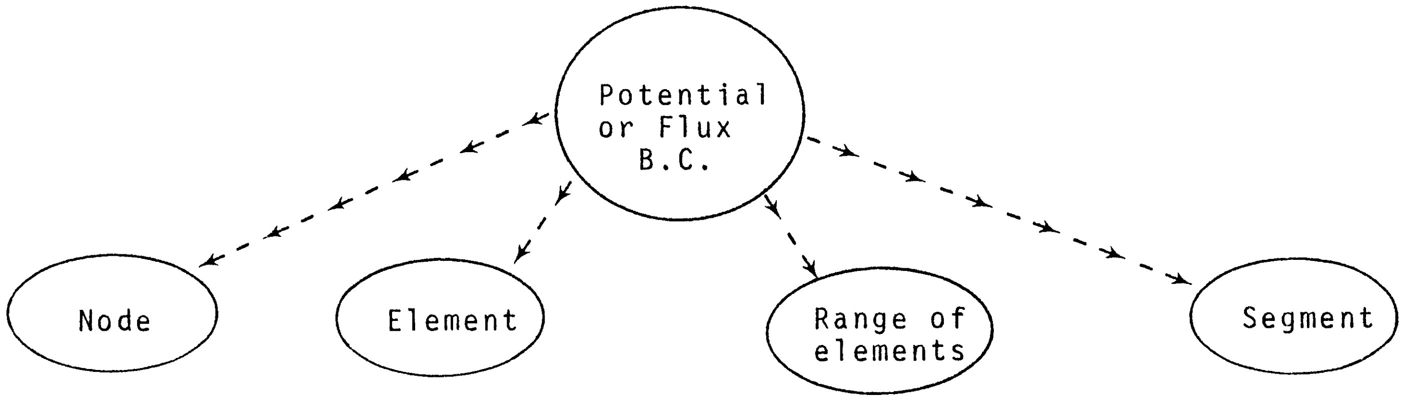


FIGURE 3.29 Imposing boundary conditions.

3.2.3.3 Introduction of boundary conditions

The next step is to impose boundary conditions. Presently the solver is programmed for potential problems only. The data structure is designed for elasticity problems as well, and the system has all the different functions set up for both potential and elasticity problems. One would only need to plug in the solver module for elasticity problems. If the user tries to access the elasticity module at the present time, the program will return a message saying that "the elasticity solver has not been developed yet".

For potential problems potential or potential derivative boundary conditions can be imposed. In fact, the term "potential" is used in a broad sense: in fluid flow that conforms with the Laplace's equation the user can also specify streamline and streamline derivative boundary conditions.

The boundary conditions can be specified for a node, an element, a range of elements or a segment, as shown in Fig. 3.29. Only the kind and value of the boundary conditions desired for the selected option need to be specified. The program will automatically assign those to all the nodes specified.

As long as the kind and value of the boundary conditions are not known, they will be taken as "?" and 0.0 value that will appear for the kind and value of boundary conditions in the output.

Printing of these boundary conditions at the nodes is discussed in Section 3.2.3.4 relative to the discretization display.

Notice that a positive flux means a flux inwards into the surface, and a negative flux means a flux outwards going out of the surface.

3.2.3.4 Display of discretization

The discretization can be either plotted on the screen, or printed on the screen or in a file.

(a) Graphics display

The same module is used for plotting both geometry and discretization. The reader is referred to Section 3.2.2.2 for a detailed description of the different available options. To be able to plot the discretization, the flags relative to the elements, nodes, and internal points shall be turned "on".

The flags corresponding to the segments and dots can either be "on" or "off". They can be turned "on" to check if the discretization is close enough to the geometry. As an example, Fig. 3.24 shows the discretization of the geometry of Fig. 3.18, according to Table 3.3. The segments' and dots' flags, as well as the elements' and nodes' flags are "on". This plots both geometry and discretization, and shows how the circular segments 2 and 4 are approximated by the continuous quadratic and discontinuous linear elements respectively. On the other hand, only the discretization of Fig. 3.19 is shown on Fig. 3.26. The different kinds of elements and nodes used are summarized in

Table 3.4. The segments and nodes are not drawn in this case, and this achieves a better clarity by putting more emphasis on the discretization.

As a last example of the graphics versatility, FIDES' name is plotted by the pre-processor, and the result is shown in Fig. 3.30. Only the exterior rectangle is discretized, and both segments' and nodes' flags are "on", while elements' and dots' flags are "off".

(b) Printed output

A printed output of the discretization can be obtained on the screen. After checking that nodes and elements exist, the system starts by listing nodes' properties. The total number of created nodes is printed first, followed by nodes' number, coordinates and boundary conditions. Then the total number of elements is shown along with elements' number, type, expansion, length and nodal connectivities. The user has the option to print elements' ends as well. Then if any internal points have been created, they are printed along with their coordinates.

The printed output for the discretizations of Fig. 3.24 and 3.25, with the applied boundary conditions of Table 3.5 and 3.6 respectively, are shown in Fig. 3.31 and 3.32.

3.2.3.5 Editing Options

Last, but not least, the user can edit the problem he just created. The editor is one of the most powerful features of FIDES. It allows mistake correction, and fast remeshing. The following options are available:

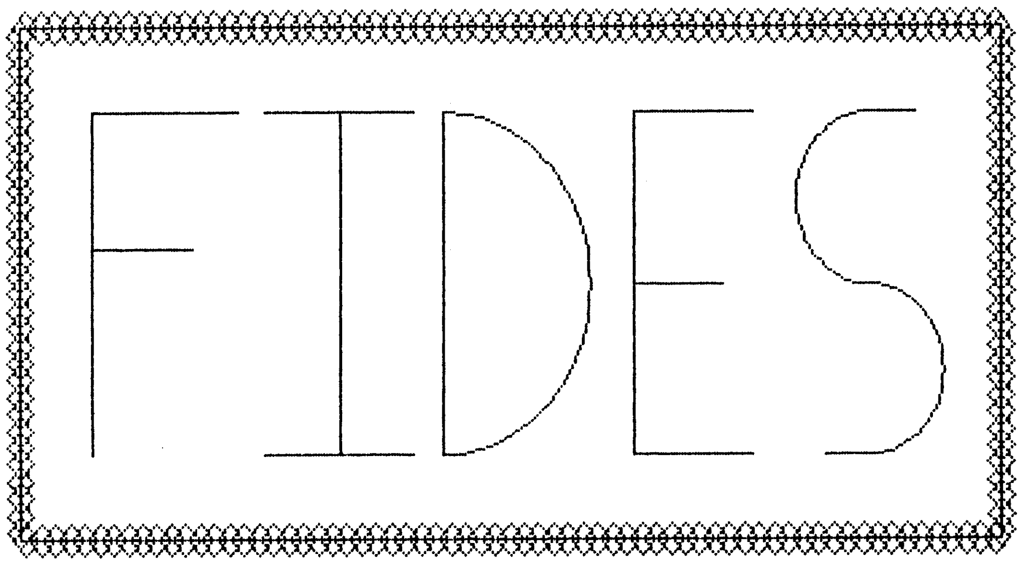


FIGURE 3.30 Pre-processor graphics' display of system's name.

SEGMENT'S NUMBER	KIND OF BOUNDARY CONDITION	VALUE OF B.C.
1	Potential	700.0
2	Potential	200.0
4	Potential derivative	50.0
5	Potential derivative	-30.0

TABLE 3.5 Boundary conditions imposed on the discretization of Fig. 3.24.

```
LIST OF ELEMENTS

ELEM.   TYPE   EXPANSION  NODE   NODE   NODE   LENGTH
 No.                       1      2      3
-----   -----  ---------  ----   ----   ----   ------
  1       S        C        1   CONS.  CONS.    0.17
  2       S        C        2   CONS.  CONS.    0.19
  3       S        C        3   CONS.  CONS.    0.21
  4       S        C        4   CONS.  CONS.    0.23
  5       S        C        5   CONS.  CONS.    0.26
  6       S        C        6   CONS.  CONS.    0.28
  7       S        C        7   CONS.  CONS.    0.31
  8       S        C        8   CONS.  CONS.    0.34
  9       S        C        9   CONS.  CONS.    0.59
 10       S        C       10   CONS.  CONS.    0.59
 11       S        C       11   CONS.  CONS.    0.59
 12       S        C       12   CONS.  CONS.    0.59
 13       S        C       13   CONS.  CONS.    0.59
 14       S        C       14   CONS.  CONS.    0.59
 15       S        C       15   CONS.  CONS.    0.59
 16       S        C       16   CONS.  CONS.    0.59
 17       S        C       17   CONS.  CONS.    0.34
 18       S        C       18   CONS.  CONS.    0.31
 19       S        C       19   CONS.  CONS.    0.28
 20       S        C       20   CONS.  CONS.    0.26
 21       S        C       21   CONS.  CONS.    0.23
 22       S        C       22   CONS.  CONS.    0.21
 23       S        C       23   CONS.  CONS.    0.19
 24       S        C       24   CONS.  CONS.    0.17
 25       S        C       25   CONS.  CONS.    0.13
 26       S        C       26   CONS.  CONS.    0.17
 27       S        C       27   CONS.  CONS.    0.21
 28       S        C       28   CONS.  CONS.    0.27
 29       S        C       29   CONS.  CONS.    0.27
 30       S        C       30   CONS.  CONS.    0.21
 31       S        C       31   CONS.  CONS.    0.17
 32       S        C       32   CONS.  CONS.    0.13

         NUMBER OF INTERNAL POINTS=3


         POINT     COORDINATES

          No.     X          Y
         ----  -------    -------

          1     1.41       1.41
          2     1.06       1.06
          3     1.77       1.77
```

FIGURE 3.31 Pre-processor output for the discretization of Fig. 3.24 and boundary conditions of Table 3.5.

```
NUMBER OF NODES=32

NODE        COORDINATES          B.C.'S

 No.     X          Y         B.C. 1  KIND
----  -------    -------     -------  ----

  1     1.09       0.00     1000.00    U
  2     1.27       0.00     1000.00    U
  3     1.47       0.00     1000.00    U
  4     1.69       0.00     1000.00    U
  5     1.93       0.00     1000.00    U
  6     2.20       0.00     1000.00    U
  7     2.50       0.00     1000.00    U
  8     2.83       0.00     1000.00    U
  9     2.97       0.29        0.00    Q
 10     2.86       0.87        0.00    Q
 11     2.63       1.41        0.00    Q
 12     2.31       1.89        0.00    Q
 13     1.89       2.31        0.00    Q
 14     1.41       2.63        0.00    Q
 15     0.87       2.86        0.00    Q
 16     0.29       2.97        0.00    Q
 17     0.00       2.83        0.00    U
 18     0.00       2.50        0.00    U
 19     0.00       2.20        0.00    U
 20     0.00       1.93        0.00    U
 21     0.00       1.69        0.00    U
 22     0.00       1.47        0.00    U
 23     0.00       1.27        0.00    U
 24     0.00       1.09        0.00    U
 25     0.07       1.00        0.00    Q
 26     0.22       0.97        0.00    Q
 27     0.40       0.91        0.00    Q
 28     0.60       0.79        0.00    Q
 29     0.79       0.60        0.00    Q
 30     0.91       0.40        0.00    Q
 31     0.97       0.22        0.00    Q
 32     1.00       0.07        0.00    Q
```

FIGURE 3.31 Pre-processor output for the discretization of Fig. 3.24 and boundary conditions of Table 3.5.

(Continued)

SEGMENT'S NUMBER	KIND OF BOUNDARY CONDITION	VALUE OF B.C.
1	Temperature	1000.00
2	Flux	0.00
3	Temperature	0.00
4	Flux	0.00
5	Flux	0.00

TABLE 3.6 Boundary conditions imposed on the discretization of Fig. 3.25.

```
NUMBER OF NODES=39

NODE        COORDINATES          B.C.'S

 No.      X          Y        B.C. 1  KIND
----   -------    -------    -------  ----

  1     0.94       0.00      700.00    U
  2     2.81       0.00      700.00    U
  3     4.69       0.00      700.00    U
  4     6.56       0.00      700.00    U
  5     8.44       0.00      700.00    U
  6    10.31       0.00      700.00    U
  7    12.19       0.00      700.00    U
  8    14.06       0.00      700.00    U
  9    14.58       0.11      200.00    U
 10    13.75       0.33      200.00    U
 11    12.92       0.56      200.00    U
 12    12.19       0.97      200.00    U
 13    11.58       1.58      200.00    U
 14    10.97       2.19      200.00    U
 15    10.56       2.92      200.00    U
 16    10.33       3.75      200.00    U
 17    10.11       4.58      200.00    U
 18     9.50       6.00        0.00    ?
 19     8.50       8.00        0.00    ?
 20     7.50      10.00        0.00    ?
 21     6.50      12.00        0.00    ?
 22     5.50      14.00        0.00    ?
 23     4.52      14.90       50.00    Q
 24     3.56      14.71       50.00    Q
 25     2.68      14.35       50.00    Q
 26     1.87      13.81       50.00    Q
 27     1.19      13.13       50.00    Q
 28     0.65      12.32       50.00    Q
 29     0.29      11.44       50.00    Q
 30     0.10      10.48       50.00    Q
 31     0.00       9.44      -30.00    Q
 32     0.00       8.33      -30.00    Q
 33     0.00       7.22      -30.00    Q
 34     0.00       6.11      -30.00    Q
 35     0.00       5.00      -30.00    Q
 36     0.00       3.89      -30.00    Q
 37     0.00       2.78      -30.00    Q
 38     0.00       1.67      -30.00    Q
 39     0.00       0.56      -30.00    Q
```

FIGURE 3.32 Pre-processor output for the discretization of Fig. 3.25 and boundary conditions of Table 3.6.

```
NUMBER OF ELEMENTS=19


LIST OF ELEMENTS

ELEM.   TYPE   EXPANSION  NODE  NODE  NODE  LENGTH
 No.                       1     2     3
-----   -----  ---------  ----  ----  ----  ------
  1       S        L        1     2    LIN.      3.75
  2       S        L        3     4    LIN.      3.75
  3       S        L        5     6    LIN.      3.75
  4       S        L        7     8    LIN.      3.75
  5       S        Q        9    10    11        2.59
  6       S        Q       12    13    14        2.59
  7       S        Q       15    16    17        2.59
  8       S        C       18   CONS.  CONS.     2.24
  9       S        C       19   CONS.  CONS.     2.24
 10       S        C       20   CONS.  CONS.     2.24
 11       S        C       21   CONS.  CONS.     2.24
 12       S        C       22   CONS.  CONS.     2.24
 13       S        L       23    24    LIN.      1.95
 14       S        L       25    26    LIN.      1.95
 15       S        L       27    28    LIN.      1.95
 16       S        L       29    30    LIN.      1.95
 17       S        Q       31    32    33        3.33
 18       S        Q       34    35    36        3.33
 19       S        Q       37    38    39        3.33
```

FIGURE 3.32 Pre-processor output for the discretization of Fig. 3.25 and boundary conditions of Table 3.6.

(Continued)

- get information about elements or nodes;
- modify existing information about elements or nodes;
- add elements or nodes;
- delete elements or nodes;
- weld two adjacent nodes; and
- exit the discretization editor.

This is shown in Fig. 3.33.

Let us discuss the different options available for the elements first.

(a) Elements' editor

Using the command "get information", one can examine the elements' properties. The user needs to give the element's number, and can ask the program to output different element's properties, such as:

- its number;
- its nodal connectivity (the nodes' numbers to which the element is connected will be outputted).
- its type (straight line or curved arc);
- its expansion (constant, linear, or quadratic);
- its length; or
- the coordinates of its start and end.

These can be seen in the total output as well, but this is more practical if one is interested in one particular value.

As far as changes are concerned, the following quantities can be modified for an element:

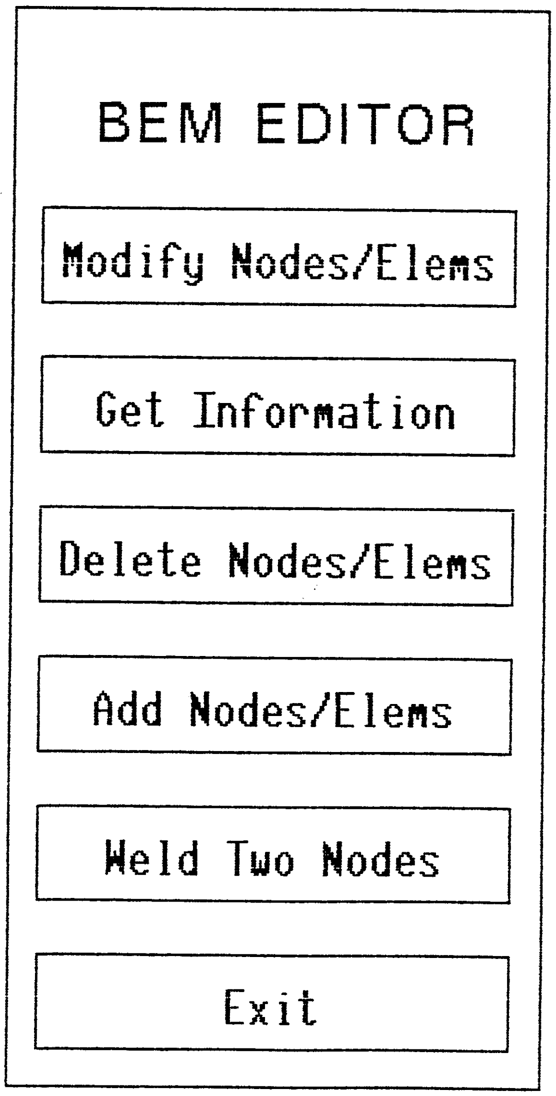


FIGURE 3.33 Discretization editor's menu.

- its number;
- its nodes' number; or
- its expansion.

To modify the number of an element, its new number is needed, which should be different from any of the existing elements' numbers. Similar procedure is used to change the nodes' numbers to which the element is connected. Requiring a change of expansion, deletes the dot_lists associated to the element. The dots, to which the dot_lists point to, are deleted if and only if they are not connected to any other element. Then, a new expansion is specified, and new nodes are connected to the element accordingly. This last option is very useful. If the user is not satisfied by the results obtained, he can use a higher expansion where needed. This means that he can either specify a change in expansion in the region where higher accuracy is desired, or at all the elements. This is very useful for accuracy studies where different expansions are tried.

Another very powerful capability is the possibility of deleting and adding elements and nodes. Elements should always be deleted before the nodes. If a mistake is made in creating the discretization, then this option allows to delete all the "undesirable" elements. Their space in memory will be made empty, by using the function "free". Since deleting elements is an important step, the system checks again that this is the user's real intention before proceeding. The nodes are not automatically deleted, since they might belong to another element. Future development focuses on automatically deleting nodes belonging to discontinuous elements, upon deletion of the corresponding elements; since in the discontinuous case no single node is shared by two elements. Figure 3.34 shows the discretization of Fig. 3.25 with elements 1 through 4 that have been deleted. The associated nodes have not been deleted yet. The user is also allowed to add elemlents; but this option is not expected to be used frequently, since the mesh generator can do this as easily, if not better.

(b) Nodes' editor

Similarly to the elements' editor, any information can be obtained about the nodes, such as:

- their number;
- their coordinates; and
- the kind and value of the applied boundary conditions.

All of the above properties can be modified. As an example of a change of coordinates, node 1 in Fig. 3.25 has been moved to the middle of the domain, as shown in Fig. 3.35. This option can also be used to move nodes along the element, and to check how the node's position affects the result. The most useful option in the "modify-element" module is the alteration of the boundary conditions. This allows the user to run several problems, involving the same discretization but different boundary conditions in a minimal amount of time. This can be done without ever exiting the system. This capability also allows the user to correct any mistakes made while imposing

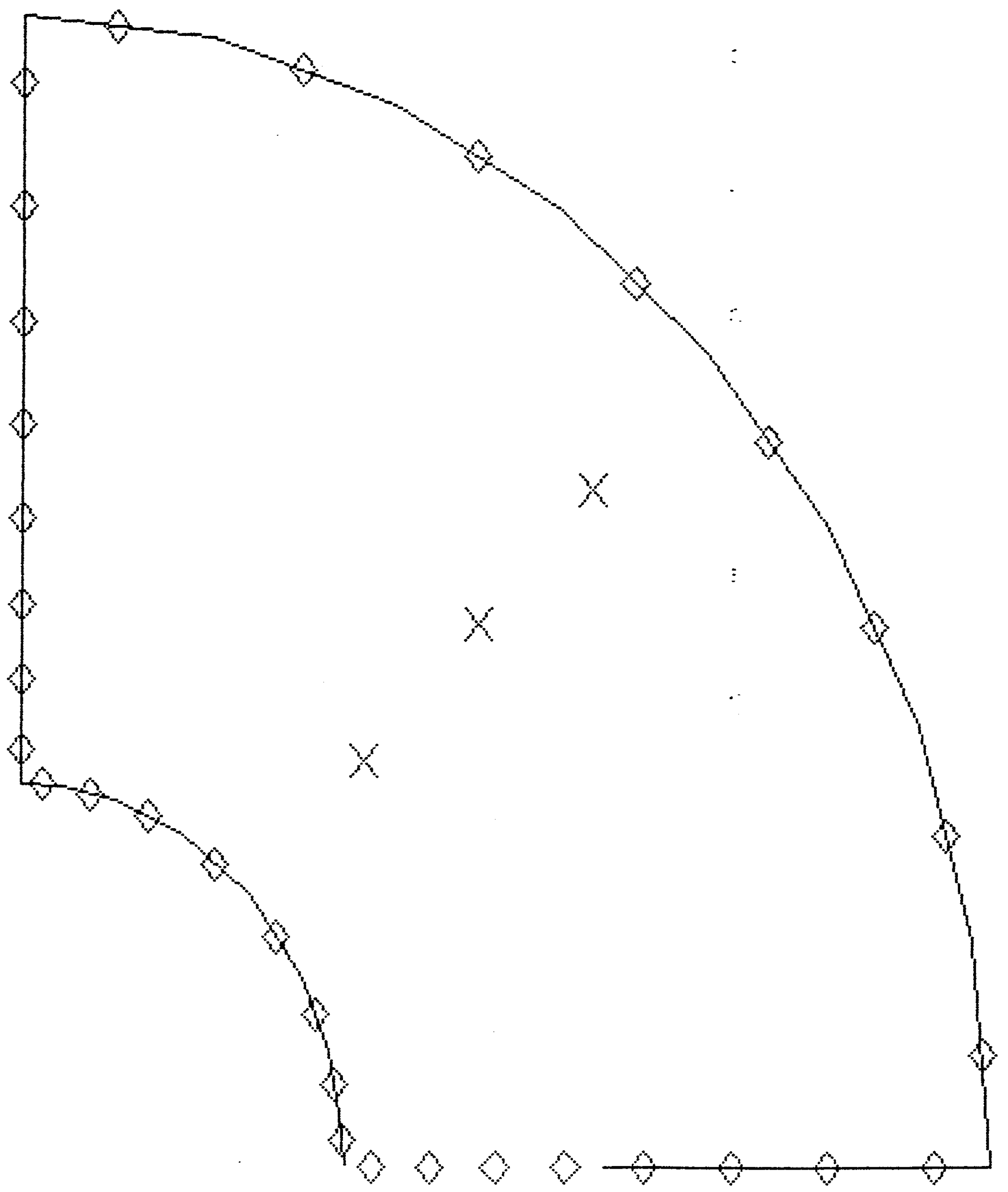


FIGURE 3.34 Deleting elements using the editor.

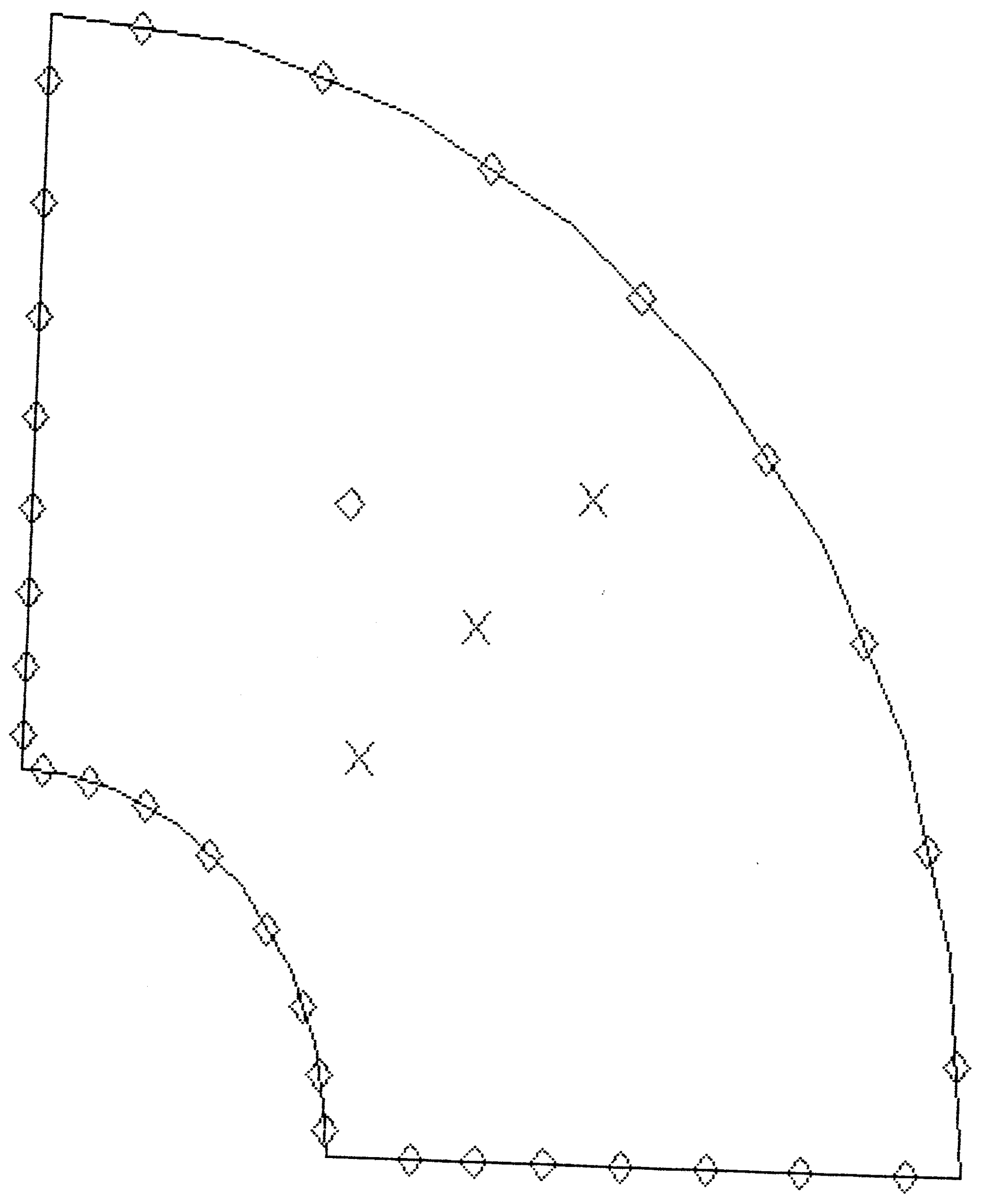


FIGURE 3.35 Modifying node's coordinates using the editor.

boundary conditions.

The choice of deleting and adding nodes is also given. Figure 3.36 is the result of deleting nodes 1 through 4 in Fig. 3.34. The corresponding elements had already been deleted.

The last option is to "weld" two nodes. This means that two nodes having the same coordinates can be turned into one node. The elements pointing to the two nodes will, therefore, be pointing to the same single node. Situations where the two nodes can be located near each other arise when continuous elements are used to discretize two different segments on both sides of a corner. This is shown in Fig. 3.27.

Thus, the editor's capabilities are quite numerous and can be used to achieve efficient remeshing and fast correction of mistakes.

Along with graphics and menu-driven capabilities, the editor is the third element of the triad needed for user-friendliness.

3.3 POST-PROCESSING PHASE

The post-processor allows the user to examine the results by either plotting them or printing them. Different print options are available, such as printing the:

- geometry,
- discretization, or
- results.

These are shown in Fig. 3.38.

The graphics are more elaborate, and the various capabilities available include plotting:

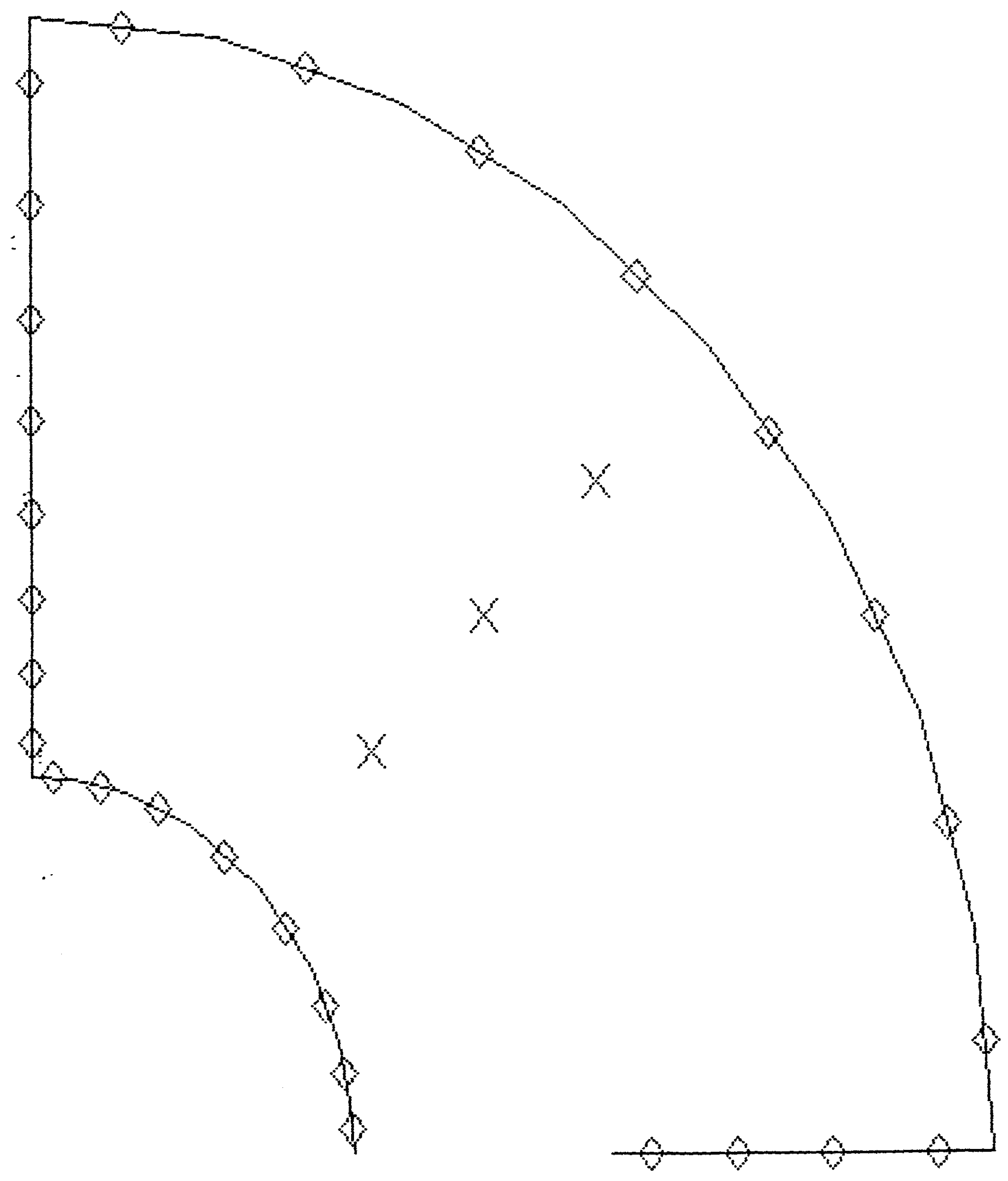


FIGURE 3.36 Deleting nodes using the editor.

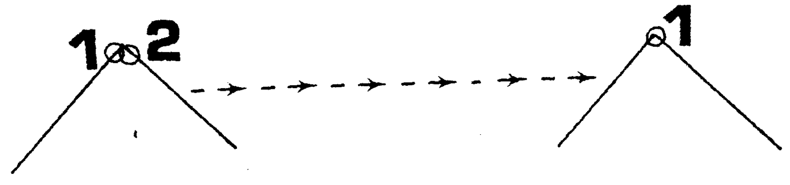


FIGURE 3.37 Welding nodes using the discretization editor.

- the original or final shape of the geometry and discretization;
- X-Y graphs to show the results for potential and potential derivative, at both the nodes and internal points;
- the normal flux on the boundary by arrows showing its magnitude and direction; and
- contour lines of equal potential values in the domain.

The menu for these different options is shown in Fig. 3.39.

3.3.1 Print Options

The print options for geometry and discretization are similar to the one used by the pre-processor. They are discussed in Sections 3.2.2.2 and 3.2.3.4 respectively.

The results of the analysis can be outputted on the screen for both nodes and internal points. The node or internal point number is printed along with the coordinates, the potential, and the flux. The results can be printed for a node, an element, a range of elements, a segment, or for the whole problem. The results for the discretization shown in Fig. 3.25, with the applied boundary conditions of Table 3.6 are shown in Fig. 3.40 for the whole problem. The advantage of the post-processor is that it allows the user to examine specific values without having to look at the complete stack of results.

But the real power of the post-processor is in the graphics capabilities to be discussed next.

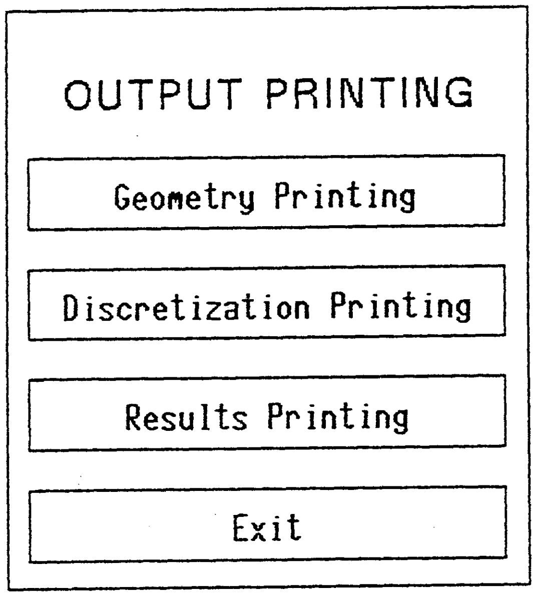


FIGURE 3.38 Post-processor's printing options.

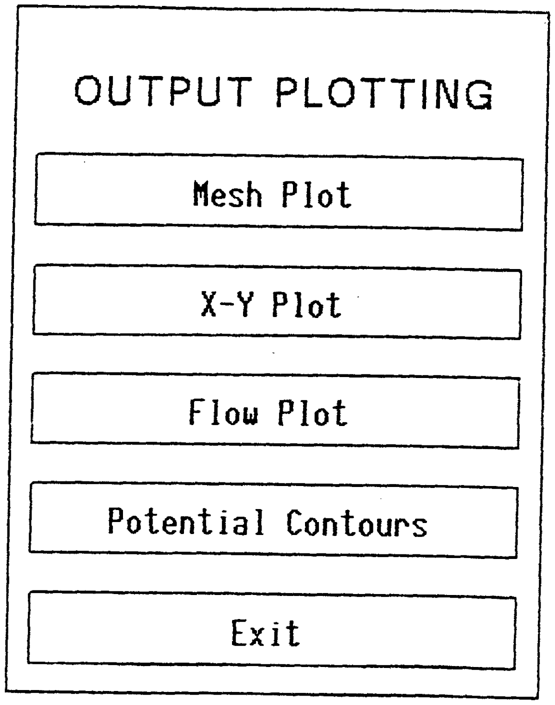


FIGURE 3.39 Post-processor's plotting options.

```
NUMBER OF NODES=32

NODE       COORDINATES          RESULTS

 No.      X          Y        POTENTIAL      FLUX
----   -------    -------    -----------  ----------

  1     1.09       0.00      1000.00         611.68
  2     1.27       0.00      1000.00         498.82
  3     1.47       0.00      1000.00         433.01
  4     1.69       0.00      1000.00         375.67
  5     1.93       0.00      1000.00         327.19
  6     2.20       0.00      1000.00         285.44
  7     2.50       0.00      1000.00         242.17
  8     2.83       0.00      1000.00         254.86
  9     2.97       0.29       942.58           0.00
 10     2.86       0.87       815.13           0.00
 11     2.63       1.41       688.91           0.00
 12     2.31       1.89       563.06           0.00
 13     1.89       2.31       437.29           0.00
 14     1.41       2.63       311.43           0.00
 15     0.87       2.86       185.23           0.00
 16     0.29       2.97        57.92           0.00
 17     0.00       2.83         0.00        -252.06
 18     0.00       2.50         0.00        -244.83
 19     0.00       2.20         0.00        -286.08
 20     0.00       1.93         0.00        -327.71
 21     0.00       1.69         0.00        -376.13
 22     0.00       1.47         0.00        -433.46
 23     0.00       1.27         0.00        -499.39
 24     0.00       1.09         0.00        -611.95
 25     0.07       1.00        40.51           0.00
 26     0.22       0.97       137.76           0.00
 27     0.40       0.91       260.03           0.00
 28     0.60       0.79       414.10           0.00
 29     0.79       0.60       586.08           0.00
 30     0.91       0.40       740.16           0.00
 31     0.97       0.22       862.39           0.00
 32     1.00       0.07       959.35           0.00

RESULTS AT INTERNAL POINTS FOLLOW :

NUMBER OF INTERNAL POINTS=3

POINT      COORDINATES      RESULTS

 No.      X          Y      POTENTIAL
----   -------    -------  -----------

  1     1.41       1.41     500.11
  2     1.06       1.06     500.09
  3     1.77       1.77     500.15
```

FIGURE 3.40 Post-processor's output of analysis' results for Fig. 3.25 and Table 3.6.

3.3.2 X-Y plots

A large flexibility is available for x-y graphs. The user can create about any plot he needs by combining the different options available. The y-abscissa can represent the potential or potential derivative, at the nodes or internal points. The x-abscissa shows the numbers of the nodes or internal points at which the desired variable is plotted.

For the nodes the results can be plotted at:

- a range of nodes,
- an element, or
- a range of elements

The "range of nodes" option is very practical, since the nodes do not need to be in order. The "-" is used to define ranges which are separated by spaces or commas. Typing the word "end" terminates the input. As an example, the following specification: "1,5-8 9,12,18-21 END" defines a range which consists of nodes 1, 5 through 8, 9,12 and 18 through 21. This is very useful since focus on the variation of a certain quantity at different nodes is made easier by looking at this type of x-y graph. Figure 3.41 represents a post-processor graph of the variation of flux at nodes 1 through 8 and 17 through 24. On the other hand, it is possible to examine the variation of q or u for a single element, as well as a range of elements. Figure 3.42 has been generatd by asking for a plot of the temperature for the range of elements 9 through 16. The user specifies the x and y labels himself.

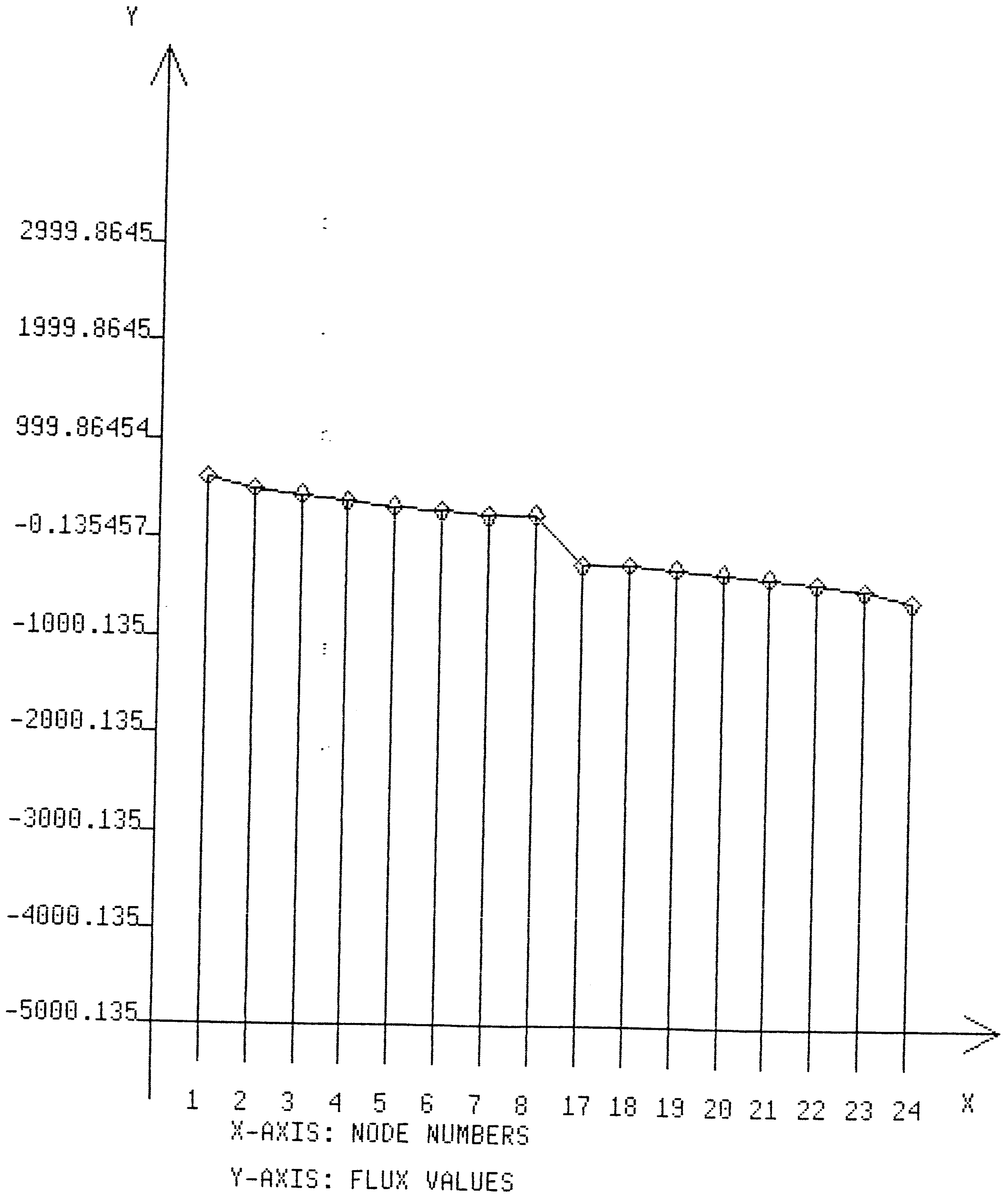


FIGURE 3.41 Post-processor's X-Y plot of flux at a range of nodes.

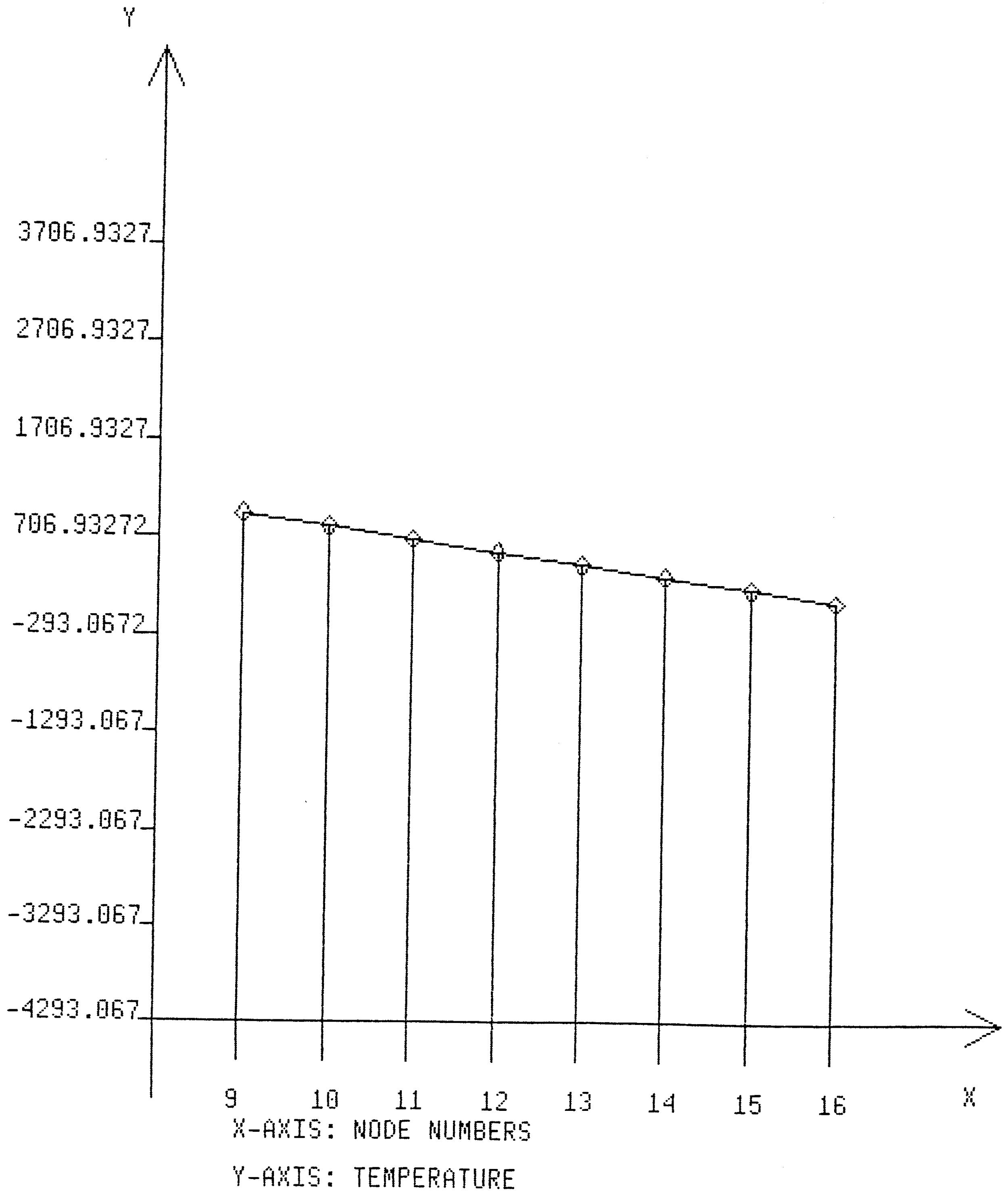


FIGURE 3.42 Post-processor's X-Y plot of temperature at a range of elements.

Similar options are available for internal points. This is shown in Fig. 3.43 where the temperature for the "range of internal points" 1 through 3 is plotted. On the VAX 11-750 computers running on the UNIX operating system, color plots are available. The frame is plotted in red, the x and y abscissas in blue, and the graph itself in green.

3.3.3 Graphics display of normal flux

Although x-y graphs are useful, it is much nicer to examine the results plotted directly on the boundary. This option is available for normal fluxes. They are plotted by arrows pointing in the direction of the flux, and having a magnitude proportional to the flux value. This allows an easier interpretation of the results. An arrow pointing inwards into the surface represents a positive flux, while the outward direction going away from the boundary stands for a negative flux. To illustrate this option the results of Fig. 3.41 for the fluxes are plotted directly on the discretization, as shown in Fig. 3.44.

3.3.4 Other features

The same module used to plot the geometry and discretization in the pre-processor is used again for the same purpose in the post-processor. It can plot both the original and final shapes. But, since the problem solved is of the potential type, the geometry and discretization are not going to change. This option will be very useful in elasticity problems, since displacements are one of the variables. It will allow the

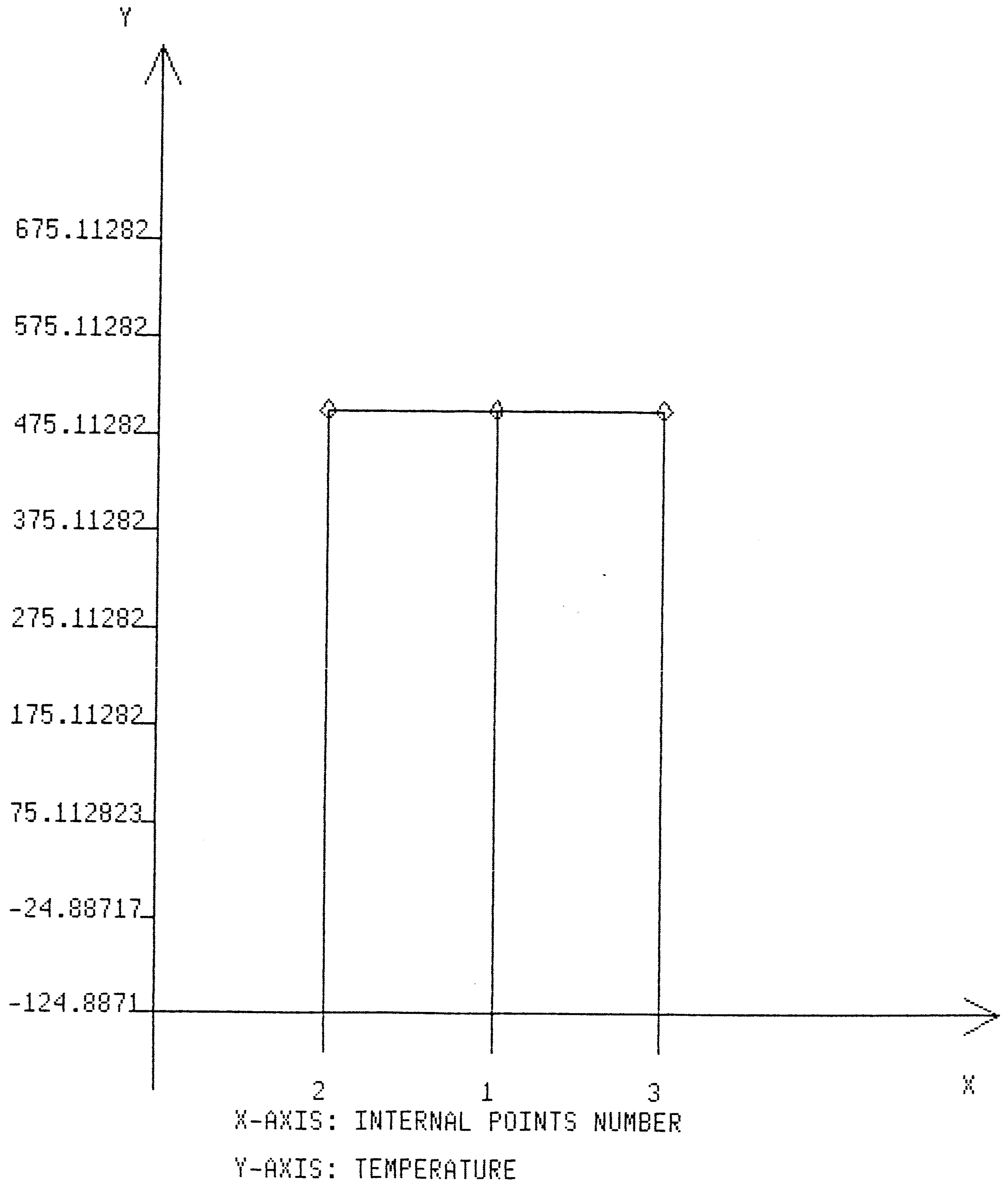


FIGURE 3.43 Post-processor's X-Y Plot of temperature at a range of internal points.

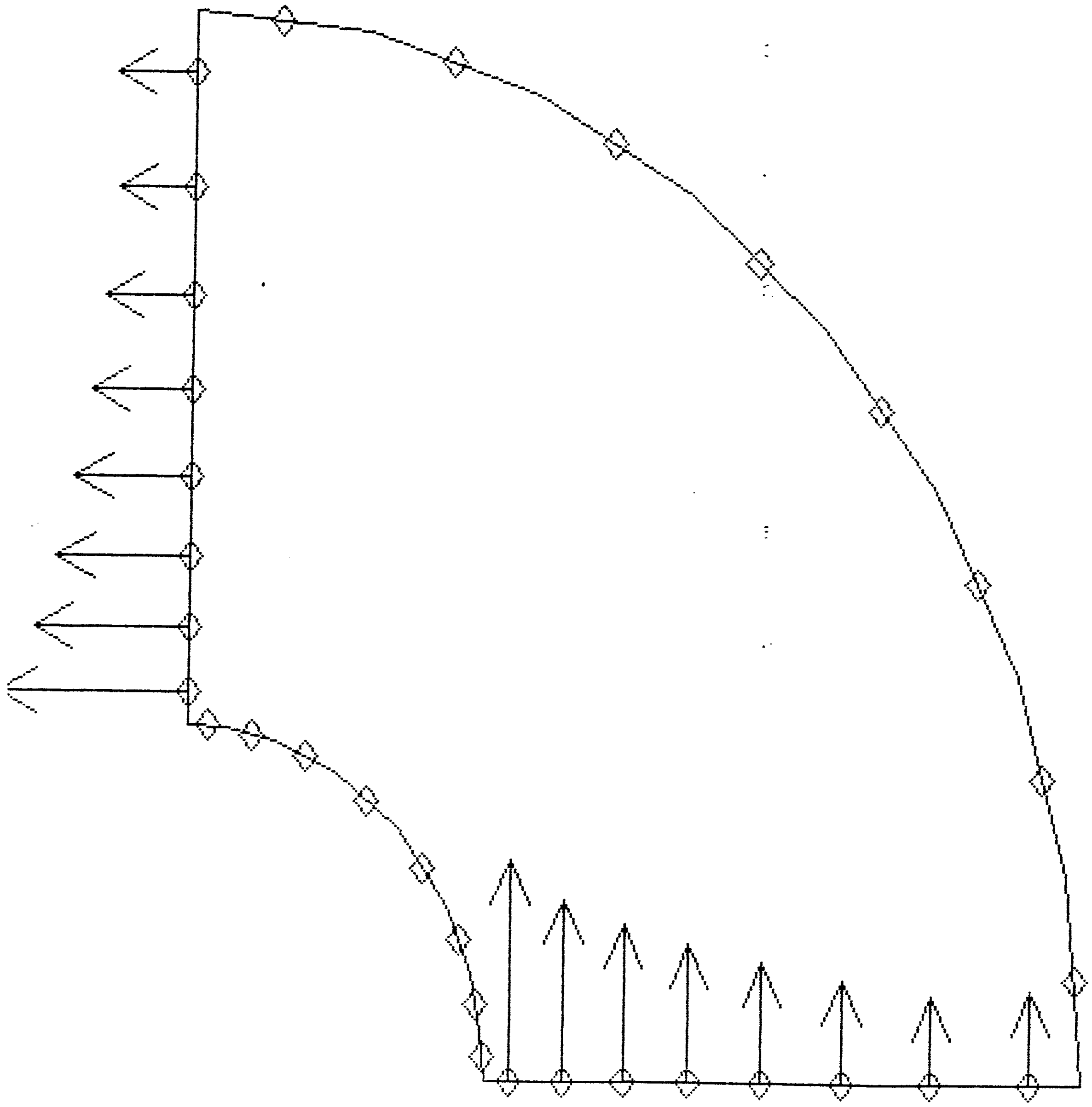


FIGURE 3.44 Post-processor's display of normal flux on the boundary.

user to examine the deformed configuration. With minor modifications this module can be changed to plot the deformations at different times, for time-dependent problems. For more details about the capabilities of this graphic module the reader is refered to Sections 3.2.2.2 and 3.2.3.4.

A capability under development is going to plot equi-potential contour lines inside the domain. This option along with the existing graphic display of normal flux, will allow the user to interpret his results very easily. The same module can be used to plot the variation of stresses and strains on the discretization. The only objective against drawing these contour lines in boundary element methods is that to achieve a good accuracy, values at many internal points are needed. These internal points will not be specified by the user, but created by the program itself. If a large number of internal points is used, then this would defeat the purpose of the boundary element method which tries to minimize the points inside the domain.

Color graphics are also available for these options on PRO-350 terminals.

3.4 SOLUTION PHASE

The present solver is written in FORTRAN. It reads its input from a translator that converts the data from the pre-processor's data structures of "C" to the array form of FORTRAN. A second module is used to translate the results from arrays and fill the corresponding post-processor's data structures. The solver evaluates the different matrices corresponding to the boundary unknowns u and q, introduces the boundary conditions for u and q, solves the system of equations for boundary values of u and q, and evaluates u, q_x, q_y at selected internal points. The solver is based on the boundary element theory presented in Chapter 2. The solver presents the options of different integration rules, continuous or discontinuous elements, and different expansions. But it is very rigid and difficult to expand in comparison with the "C" code. It is planned to rewrite the solver in "C", and create a new data base. It will contain data structures for integration rules, expansion types, and discontinuous elements. Continuous elements are not going to be given too much emphasis. The user will be able to mix different expansions and different integration rules. This will allow more flexibility. The code will also be more homogeneous, since it will be all written in "C". The translator modules will not be used anymore, and fastest execution will be achieved. It would also be interesting to add data structures for different fundamental solutions. This will allow the elasticity problem to be added without too many changes.

A rule-based advisor could also be used to choose the different integration rules and expansions, depending on the accuracy desired. This will be discussed in Chapter 4.

CHAPTER 4 - ACCURACY STUDIES

4.1 SYSTEM VERIFICATION

4.1.1 Potential Flow

A two-dimensional potential flow around a cylinder between parallel walls is considered as shown in Fig. 4.1[18]. A short review of the basic theory is presented first.

The potential value to be specified on the boundary (denoted by u in preceding chapters) can be taken as either the velocity potential, ϕ, or the stream function, ψ. Both depend on the coordinates x and y.

Let V_x, V_y represent the velocity components in the two-dimensional flow. They can be expressed as:

$$v_x = \frac{\delta\phi}{\delta x}, \quad v_y = \frac{\delta\phi}{\delta y} ;$$

or

$$v_x = \frac{\delta\psi}{\delta y}, \quad v_y = -\frac{\delta\psi}{\delta x} .$$

In potential flow both ϕ and ψ satisfy the Laplace's equation.

FIDES can be used efficiently to solve for this kind of problem. Two different options are possible for the choice of the unknown boundary quantities:

- the potential ϕ with the associated normal flow $\delta\phi/\delta n$, or
- the stream function ψ along with its derivative $\delta\psi/\delta n$.

On solid boundaries, ψ can be specified. At other points the velocity is known and is given in terms of a derivative of ψ. In the case of ϕ, it is not known on solid walls or other bodies in the flow; however, the flow $\delta\phi/\delta n$ normal to such surfaces is zero. It is thus more difficult to treat the problem in terms of ϕ.

Different meshes are tried, and as an illustration one of them is shown in Fig. 4.2. Only quarter of the geometry needs to be solved for because of symmetry considerations about axes a-a' and d-d' of Fig. 4.1. The stream function ψ is chosen as the main variable. Due to symmetry the boundary conditions:

$\psi = 0$ on lines a-b and b-c

can be imposed. Then, using the definition of the stream function gives:

$\psi = 2$ on d-e.

Along a-e, the nodal values of ψ vary linearly from $\psi = 0$ at a, to $\psi = 2$ at e. Finally, on c-d the vertical components of velocity must vanish. Thus:

$$v_y = -\frac{\delta\psi}{\delta x} = 0$$

But, since c-d is a vertical line, the normal n is parallel to the x-axis. Then,

$$\frac{\delta\psi}{\delta x} = \frac{\delta\psi}{\delta n} = 0 \ .$$

Once the boundary conditions for ψ and $\frac{\delta\psi}{\delta n}$ are applied, the problem can be solved. Results of the analysis for the stream function ψ on line c-d are compared to finite element analysis'

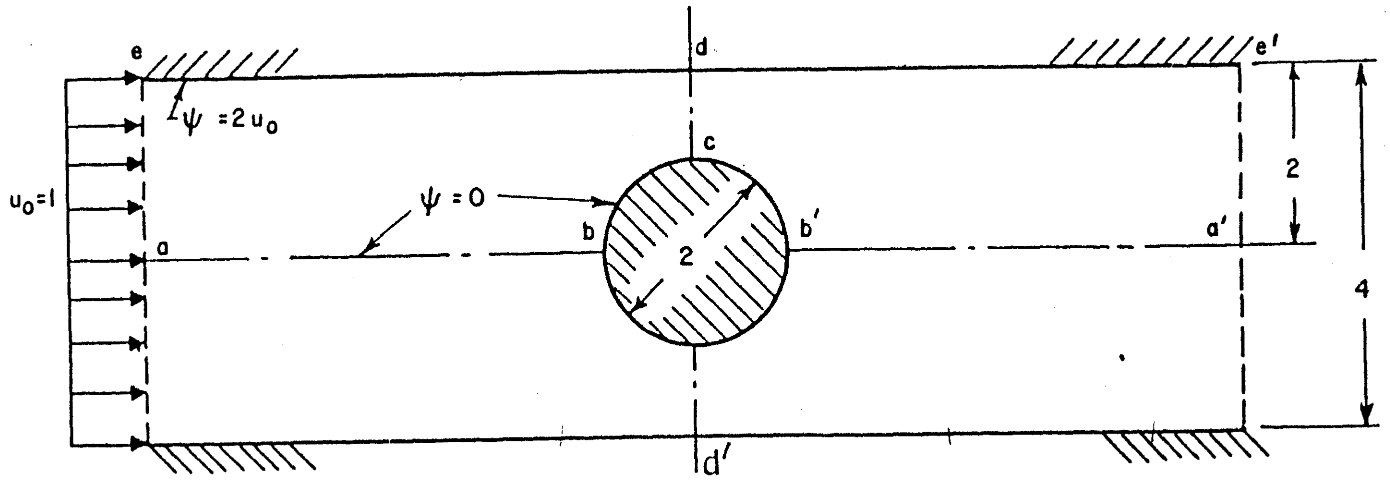


FIGURE 4.1 Flow around a cylinder between two parallel walls.

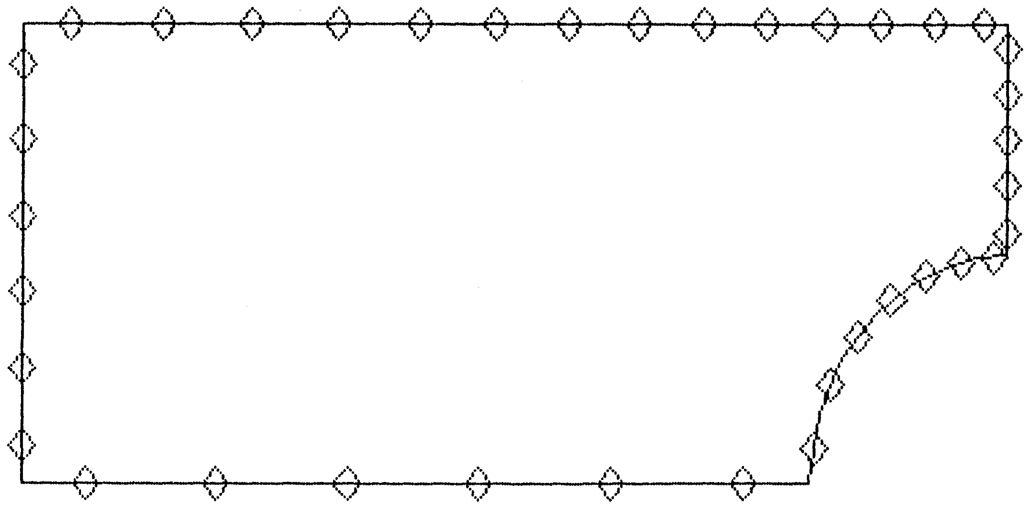


FIGURE 4.2 Boundary element discretization of potential flow problem.

runs[18], in Fig. 4.3. Different boundary element meshes were tried. The results of selected ones only are shown. The agreement between the two methods is very good. It is interesting to compare the boundary element discretization of Fig. 4.2 to the finite element one shown in Fig. 4.4 [18]. The same order of accuracy is obtained by both, although the boundary element mesh is much simpler to create and visualize. Generally, boundary element methods are preferred in linear static problems, and when the quantity of interest lies on the boundary. But they also give excellent results for complex problems as well. This will be the subject of the next section.

4.1.2 Thermal Conduction

The problem shown in Fig. 4.5 illustrates an infinitely long solid rod of radius, R, subjected to a surface temperature $T(R,\theta)$. This is a pathological problem since the boundary points represented by $\theta = 0$ and $\theta = \pi$ are singular points; the flux at these locations is theoretically infinite. One of the discretizations used in FIDES is shown in Fig. 4.6. The upper half of the rod is subjected to a temperature $T_o = 100$, and the lower half to a temperature of zero. The variation of temperature at the interior is of interest. The analytical solution gives:

$$T(r,\theta) = T_o \left[\frac{1}{2} + 2 \sum_{n=1}^{\infty} \frac{1}{n\pi} \left(\frac{r}{R}\right)^n \sin n\theta\right]$$

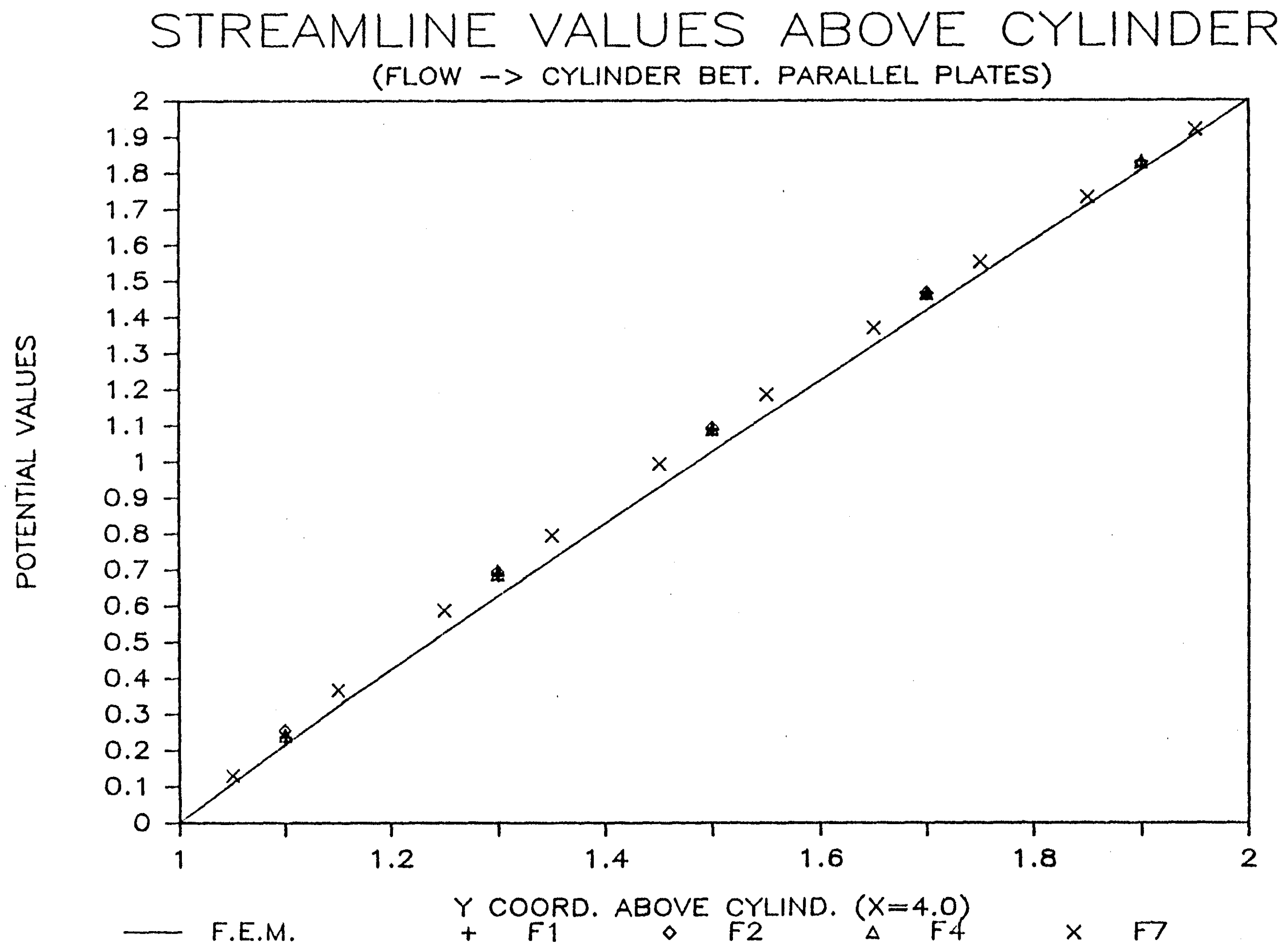


FIGURE 4.3 F.E.M. vs. B.E.M. results for streamline values above cylinder.

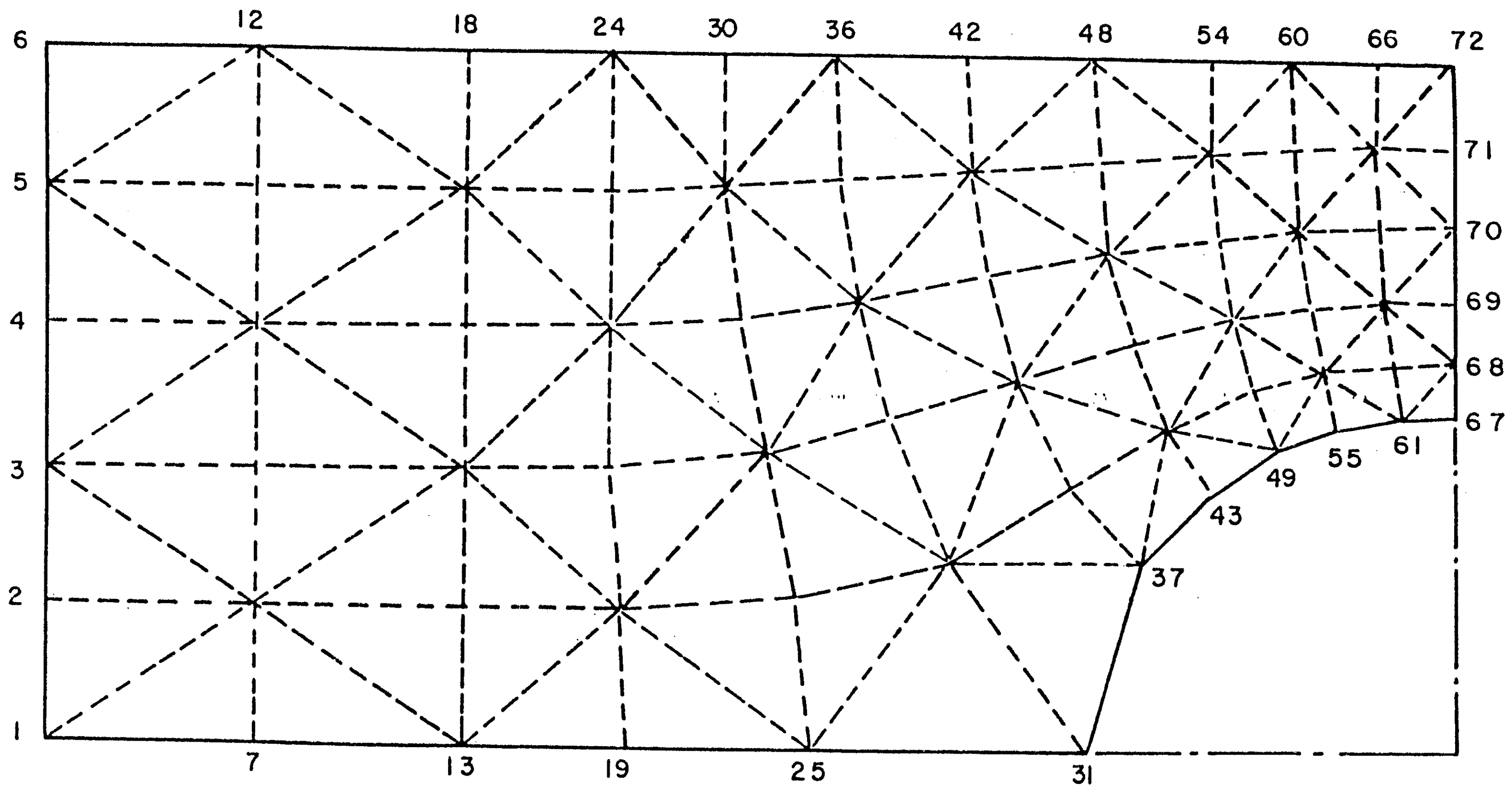


FIGURE 4.4 Finite element discretization of potential flow problem.

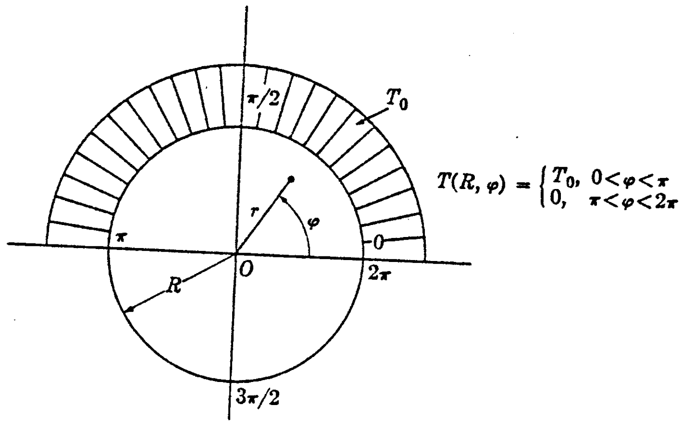

$\pi/2$
T_0
$T(R,\varphi) = \begin{cases} T_0, & 0<\varphi<\pi \\ 0, & \pi<\varphi<2\pi \end{cases}$
r
φ
π
0
O
R
2π
$3\pi/2$

FIGURE 4.5 Infinitely long solid rod subjected to two different temperatures.

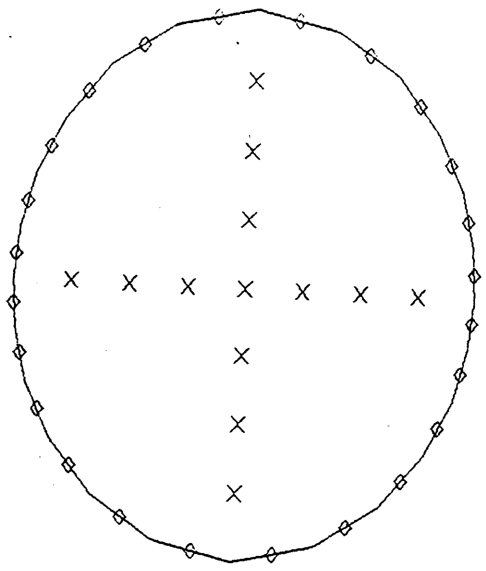


FIGURE 4.6 Boundary element discretization of heat conduction problem.

FIDES' results at seven horizontal and vertical internal points, shown in Fig. 4.6, are compared to the exact solution in Figs. 4.7, 4.8. The accuracy of the boundary element method results is excellent. For a mesh of only 4 constant elements per quarter of circle, the largest error is smaller than 2%. This proves that boundary element methods can be very efficient in solving for complex problems, as well as studying the behavior of a variable inside the considered domain. One also notices that using 30 elements per quarter circle is not really needed, since convergence can be achieved by a smaller number of elements. A set of rules to help the user in selecting the optimum number of elements is, therefore, needed. It is always better to try a coarse mesh first to get a feel for the problem. The variation of the fluxes is drawn in Fig. 4.9 by the post-processor. It is noticed that the fluxes' magnitude gets very large near the singular points.

4.2 ACCURACY ADVISOR

4.2.1 <u>Rule-Based Systems</u>

FIDES' name stands for Friendly Interactive Discrete Element Solver. The words "friendly" and "interactive" might seem redundant, since the interactiveness is really a part of the friendly attributes. But the name was chosen this way so that in the future it could be changed to: Friendly Intelligent Discrete Element System. As already discussed, the main task of this "intelligence" would be to play the role of an accuracy advisor. The user would have access through FIDES' pre-processor to this rule-based advisor.

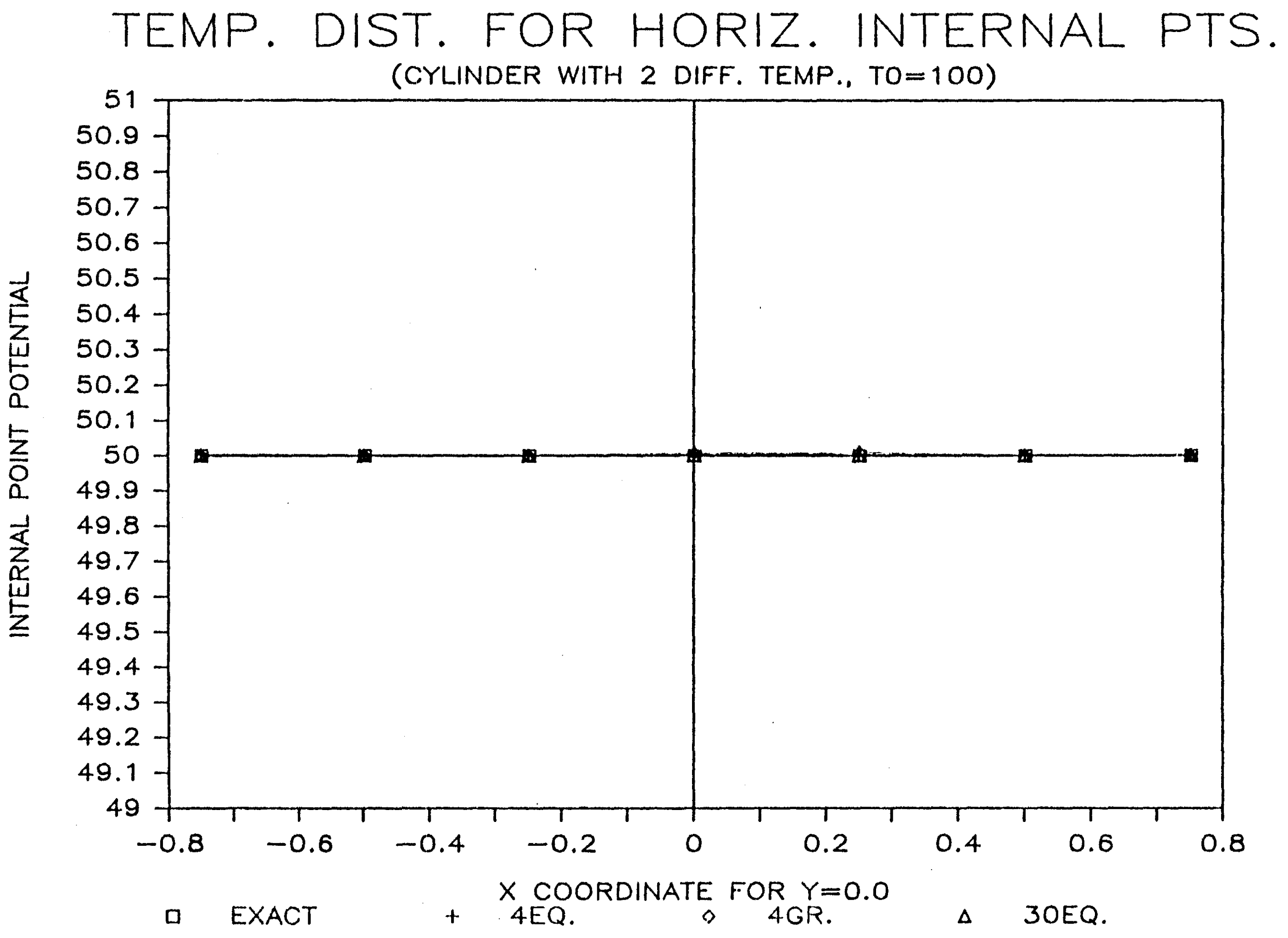


FIGURE 4.7 Exact vs. B.E.M. results for horizontal internal points.

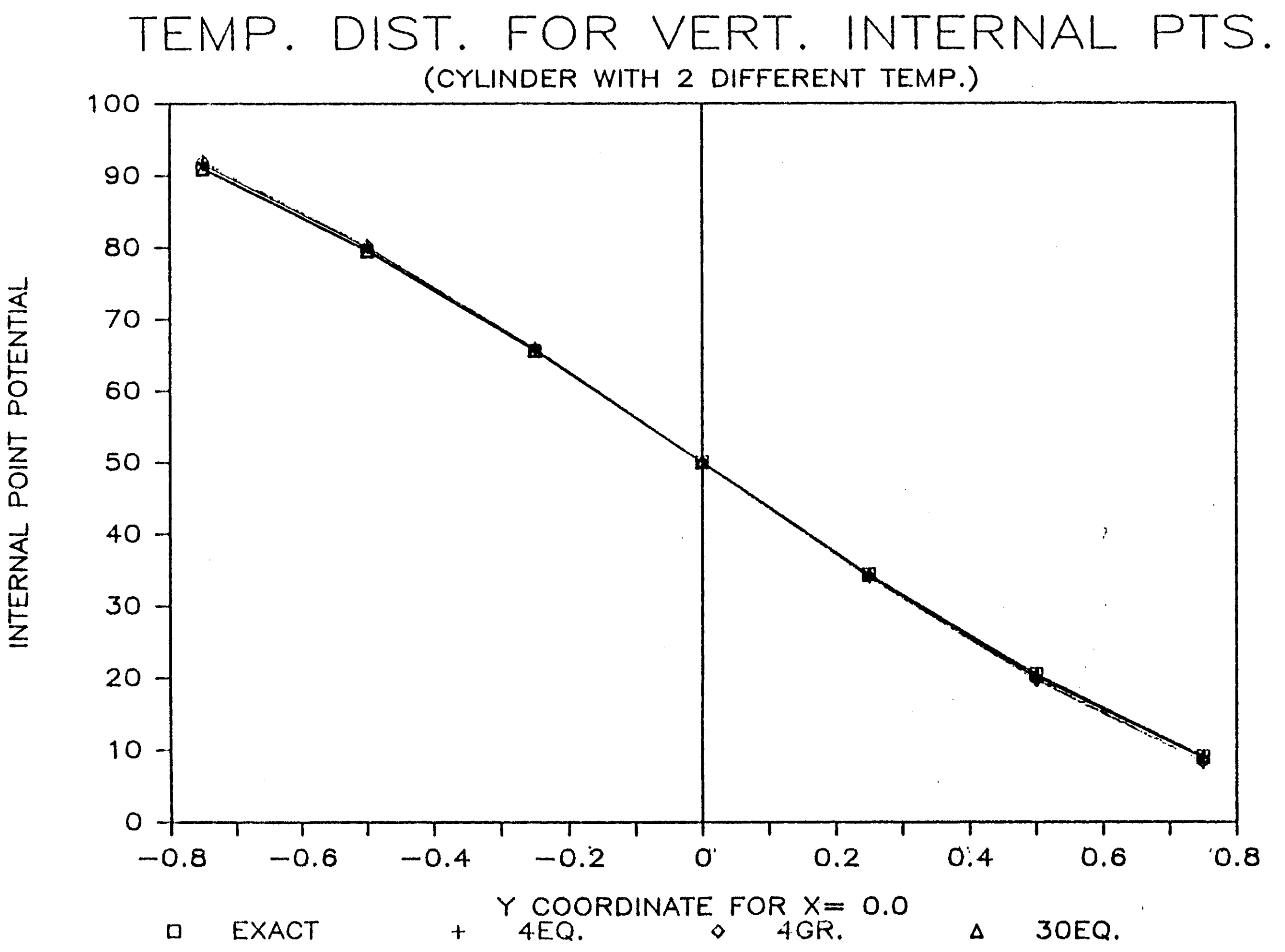


FIGURE 4.8 Exact vs. B.E.M. results for vertical internal points.

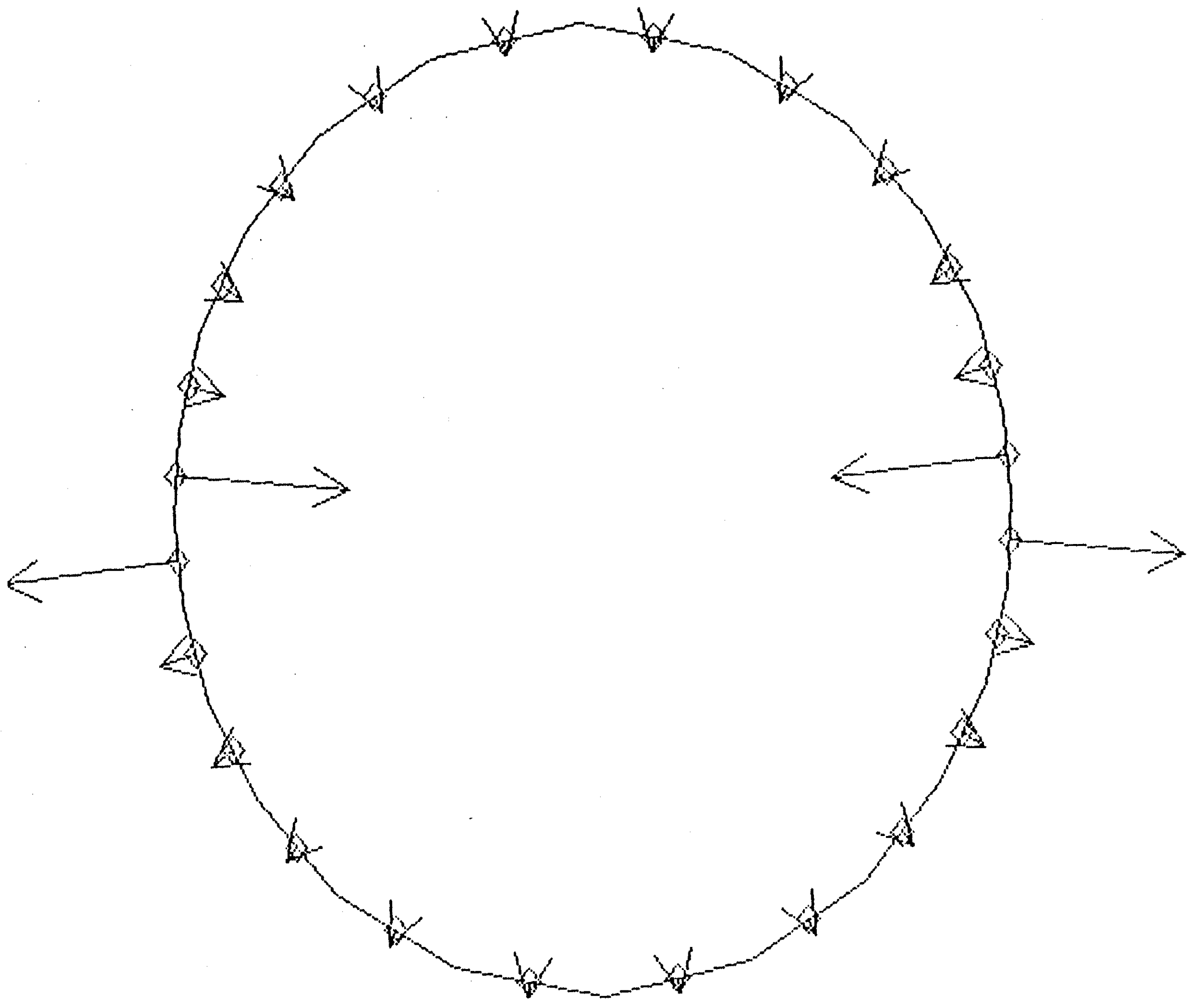


FIGURE 4.9 Graphic display of normal flux for heat conduction problem.

In rule-based systems, knowledge is represented using rules. These rules consist of small independent chunks of knowledge. Rule-based systems have been successfully used in practical situations, such as medical diagnosis, and computer configuration for VAX systems. One of the recent applications of these systems was as a finite element advisor: SACON.

Two issues are important in the operation of these systems:

- the knowledge representation, and
- the inferencing mechanism.

While the knowledge representation depends on the domain of application, the inferring mechanisms are more general and usually less important.

There are two types of knowledge:

- static (represented by facts), and
- procedural (consisting of rules).

The facts describe the domain under consideration. This might be the geometry definition in the pre-processor of FIDES. On the other hand, rules describe relations between facts. The rules used for knowledge representation can be generally divided into:

- rules of thumb, that express the "expert's" general feeling for the problem; and
- pattern matching where comparison with previous results may yield needed information.

There are two ways in which this rule-based system can be linked to FIDES. It can either control FIDES and drive it; or FIDES can send it information and expect feedback from it. FIDES' structures can be very easily modified to adopt an accuracy advisor. At the pre-processor stage, FIDES would send information about the geometry and boundary conditions, and the rule-based system would return informed opinions about the discretization. Of if the user of the boundary element system is more familiar with the method, he would create the discretization and get some feedback on how good it is. At the solver's level, the rule-based system might choose the order of the integration rule to be used. And finally, at the post-processor stage, results can be examined by the rule-based system and compared to previous runs to decide if they make sense.

The final goal, common to all cases, is to obtain "informed opinions" about accuracy issues. At the present time the preliminary advisor is independent of FIDES, and will be discussed in the next section.

4.4.2 A preliminary system

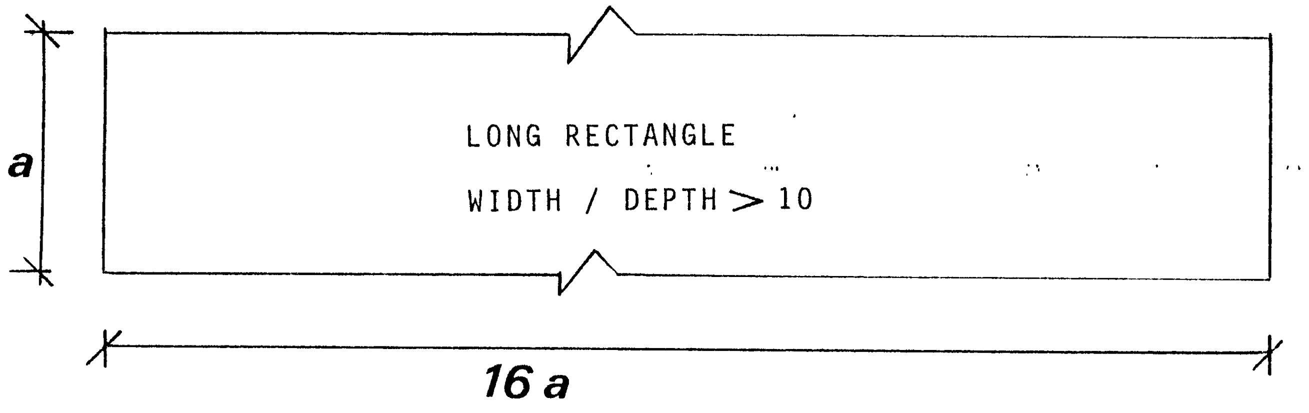
The development of this system is very preliminary. Its main purpose is to help the user in creating the mesh and interpreting the results. Thus, it addresses the same issues as pre- and post-processing techniques, namely, how to optimize the discretization and the accuracy, minimize the amount of time needed to solve a problem, and clarify ambiguities in the results. This requires different goals. If the static knowledge consists of the geometry description and boundary conditions, then the corresponding goal would be to define the discretization. Similarly, if the facts are computed results, then the goal would be to assess their validity and accuracy. At this stage, concern is focused on outlining different possible discretization schemes for a geometry and its associated boundary conditions. The rules needed for this task can be developed through interviews with various experts in boundary element methods and review of results for a broad range of problems.

Particular issues where rules are appropriate are:

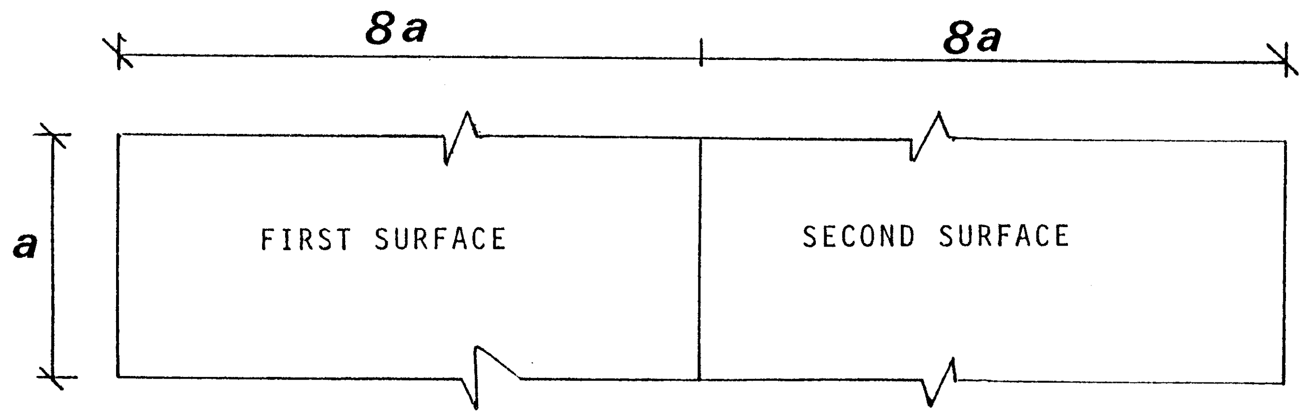
(a) Geometric considerations

Care should be taken in the discretization of odd geometric shapes.

- In the case of the rectangle shown in Fig. 4.10(a), the width to depth ratio is larger than 10. This is not advisable. It is better to split this long rectangle into different surfaces, with side's ratio smaller than 10, as shown in Fig. 4.10(b).
- It is not recommended to have two segments that are nearly parallel very close to each other. The notched specimen shown in Fig. 4.11(a) presents this problem. Due to symmetry only half of it needs to be analysed. Furthermore, it is split into two surfaces in order not to have the two sides of the notch belonging to the same surface, as shown in Fig. 4.11(b).

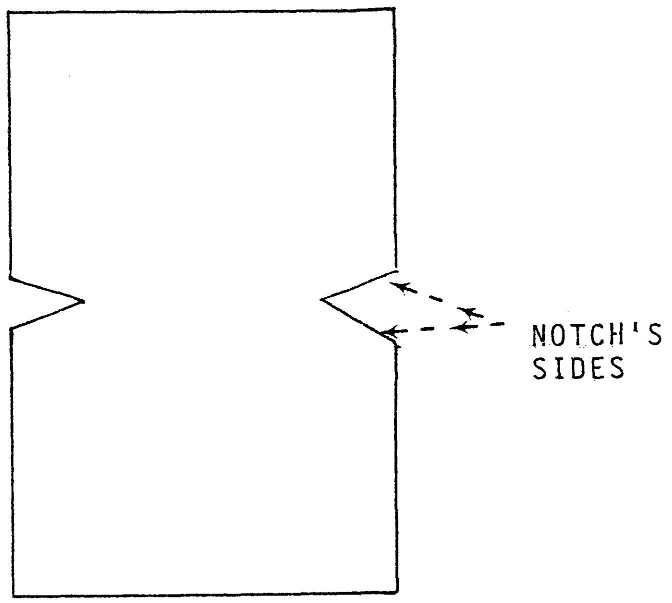


(a) Geometry.

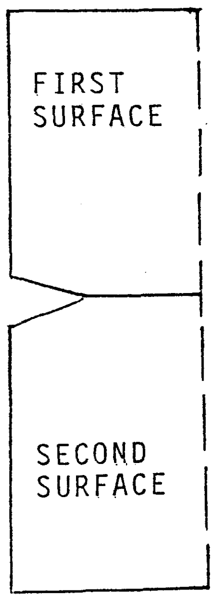


(b) Advisable discretization.

FIGURE 4.10 Problem of long rectangle.



(a) Geometry definition.



(b) Advisable discretization.

FIGURE 4.11 Notched specimen problem.

- Try not to locate any nodes at the corners. Continuous elements near the corners can disturb the accuracy. The use of discontinuous elements at the corners is preferred. This is illustrated in Figs. 4.12(a) and (b).

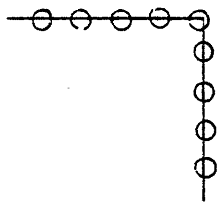
(b) Special boundary conditions

Care should be taken not to locate any node where a singularity in one of the variables is expected. Nodes can be placed on both sides of the singular point, and the accuracy diminishes as the node gets closer to the singular point. This is illustrated in Fig. 4.13, where only potential boundary conditions are applied. The two points that experience a jump in potential are also singular points for fluxes.

(c) Run number

It is always advisable to use a coarse mesh in the first run to get a feel for the problem, unless the user is very well experienced. Finer meshes can be used in the next runs.

These different rules have been used to generate a preliminary accuracy advisor. They have been implemented into GARFIELD, which is a knowledge-based system for the simulation of engineering problems using a rule format. It is being devloped at MIT. One of the powerful features of GARFIELD is the capability of easily interfacing with external programs. This makes it very attractice to use with general purpose programs like FIDES. GARFIELD is written in "C", and is an interactive, menu driven system. The GARFIELD environment


(a) Continuous elements.

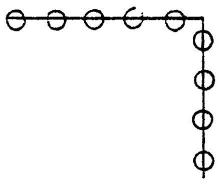

(b) Discontinuous elements.

FIGURE 4.12 Corner node.

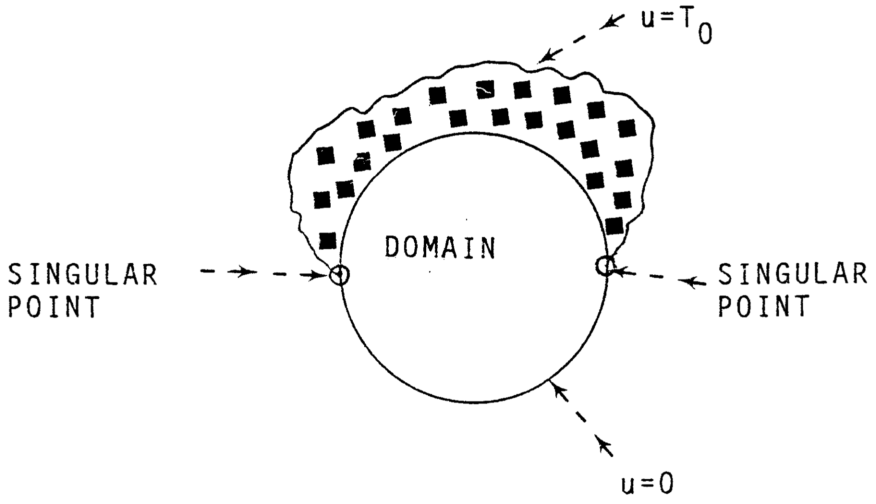


FIGURE 4.13 Flux singularity due to special potential boundary conditions.

consists of a user interface, an object base, a rule base, meta level rules, a goal stack, an inference engine, as well as a calculator and an object base language. Only few of the impressive capabilities of GARFIELD have been used. Objects and facts have been defined, as well as the goal to be reached. The rules shown above have been represented in GARFIELD's format. They are very general and can be adapted to any kind of geometry or discretization.

Particular rules concerning the problem at hand should, then, be specified. The inference engine can analyze all these facts and rules, and give advice about the discretization. These procedures are not explained in detail, since it is not the object of this thesis. Few examples of the objects and rules are going to be shown, but the reader is referred to Appendix E for more extensive information.

A fact is described by the triplet:

(attribute object value).

The different kind of objects used are: geometry, dot, problem, segment, etc. A description of the object dot is shown in Fig. 4.14. An object named dot is created first. It can either be a corner dot or not. The values of its attributes segment_left and segment_right are objects of the type segment. And, finally, each dot has a number. Further examples can be found in Appendix E. Rules consist of a name, an IF part, and a THEN part. Both IF and THEN parts are pointers to statements. If the statement in the IF part is correct, then the statement of the THEN part is executed.

```
 (create dot)

(add corner dot yes_no)

(add segment_left dot segment)

(add segment_right dot segment)

(add number dot value)
```

FIGURE 4.14 Description of object dot.

```
RULE "Recognize rectangle"

IF  (and (corners geometry 4)
         (FOR_ALL ?x
              (or (is? corner ?x no)
                  (and (is? corner ?x yes)
                       (perpendicular (get segment_left ?x)
                                      (get segment_right ?x))))))

THEN (shape geometry rectangle)
```

FIGURE 4.15 Recognizing a rectangle.

```
RULE " 10 < width/depth < 20 "

IF  (and (shape geometry rectangle)
         (>= (/ (get width (get shape geometry))
                (get depth (get shape geometry)))
             10)
         (<  (/ (get width (get shape geometry))
                (get depth (get shape geometry)))
             20))

THEN (and (print "Divide your rectangle into ")
          (print "\n two surfaces for your ")
          (print "\n   discretization !!"))
```

FIGURE 4.16 Advice for rectangle with 10 < width/depth < 20.

The rule to recognize a rectangle is shown in Fig. 4.15. If the system notices that the geometry has four corners, and that the segments to the left and right of each corner are perpendicular, then the geometry shape is a rectangle. Furthermore, if the geometry is a rectangle, and the ratio of width to depth is larger than 10 but smaller than 20, then it is advised to break the rectangle into two surfaces for discretization. This last rule is shown in Fig. 4.16. More details about the preliminary accuracy advisor are given in Appendix E.

CHAPTER 5 - CONCLUSIONS

5.1 SUMMARY

Discrete element methods are a practical way of solving continuum problems. The boundary element method is advantageous for problems with highly irregular boundaries, domain of infinite extent, linear elastic static behavior, incompressible material, and where the quantities of interest are on the boundary itself. The finite element method is very useful in problems involving non-homogeneous media, plasticity, time dependent analysis, and where the quantities of interest are in the interior of the domain. A review of boundary element methods, with emphasis on potential problems, is presented in Chapter 2.

Large discrete element codes exist that require long and time-consuming input, and print huge output files. Pre-and post-processing capabilities are needed to help in discretizing the problems, and presenting results in a readable form. A friendlier environment is necessary, as well, for the programmer concerned with maintaining the program.

An excellent computer environment is provided by the APOLLO system, together with the UNIX operation system, the "C" language, and the "lint" program verifier. Pre-and post-processors form the backbone of the FIDES system. They have been developed in the "C" language. The concept of lists of data structures has been used extensively. A user-friendly environment has been created due to menu-driven processors, graphics capabilities, and a powerful editor. Possibility of easy additions or changes to the system, along with transparency of the code lead to a programmer-friendly environment.

FIDES, at the present time, is an interactive system for discretization, solution, and presentation of results for potential problems using boundary elements.

5.2 RECOMMENDATIONS FOR FUTURE RESEARCH

The developed pre-and post-processing capabilities are designed to allow for flexible changes or additions. The boundary element numerical module for potential problems needs to be rewritten in the "C" language. Addition of higher order expansions, as well as different integration rules, would be useful. A solver module for elasticity problems would add flexibility to the system. The pre- and post-processors already contain developed facilities for elasticity problems, and thus would not require major changes.

A knowledge-based advisor is needed to help the engineer in choosing the discrete element method to be used (mainly boundary element vs. finite element). A rule-based advisor for accuracy issues should be developed, to help in the discretization of the problem and interpretation of the results. Rules can be generated by solving a large number of problems and considering the effect of different integration rules, various expansions, and different element sizes. Rules of thumb can also be collected and included in the system.

REFERENCES

1. Zienkiewicz, O.C., The Finite Element Method, 3rd Edition, McGraw-Hill Book Company (UK) Limited, 1977.

2. Segerlind, L.J., Applied Finite Element Analysis, John Wiley & Sons, 1976.

3. Bathe, K., Finite Element Procedures in Engineering Analysis, Prentice-Hall, Inc., 1982.

4. Jaswon, "Integral Equation Methods in Potential Theory, I," Procedures of the Royal Society, Ser. A, p. 273, 1963 .

5. Symm, G.T., "Integral Equation Methods in Potential Theory, II," Procedures of the Royal Society, Ser. A, p. 275, 1963.

6. Cruse, T.A., and F.J. Rizzo, "A Direct Formulation and Numerical Solution of the General Transient Elasto-Dynamic Problem, I," J. Math Analysis Applic., Vol. 22, 1968.

7. Watson, J., "Analysis of Thick Shells with Holes by Using Integral Equation Method," Ph.D. Thesis, University of Southampton, 1973.

8. Brebbia, C.A., The Boundary Element Method for Engineers, Pentech Press, London, 1978.

9. Brebbia, C.A. and S. Walker, The Boundary Element Technique in Engineering, Newnes-Butterworths, London, 1979.

10. Brebbia, C.A., New Developments in Boundary Element Methods, CML Publications, 1980.

11. Brebbia, C.A., Progress in Boundary Element Methods, Pentech Press, London, 1981.

12. Brebbia, C.A., J. Telles, and L. Wrobel, Boundary Element Methods - Theory and Applications in Engineering, Springer-Verlag, Berlin, 1983.

13. Floyd, C.G., The Determination of Stresses using a Combined Theoretical and Experimental Analysis Approach, Lloyd's Register of Shipping.

14. Reddy, J.N., An Introduction to the Finite Element Method, Mc-Graw Hill, 1984.

15. Brebbia, C.A., "The Simulation of Time Dependent Problems Using Boundary Elements," Presented at the Seminar on Advances in Finite Elements and Boundary Elements Methods, August 3, 1984, M.I.T.

16. Kernighan, B.W., D.M. Ritchies, The C Programming Language, Prentice-Hall, Inc., 1978.

17. Slater, J.H., "Alternate Languages and Data Structures for Effective Interactive Design," Proceedings of Offshore and Arctic Engineering, February 1985.

18. Martin, H.C., "Finite Element Analysis of Fluid Flows," Proceedings of the Second Conference on Matrix Methods in Structural Mechanics, December, 1969.

19. Arpaci, V.S., Conduction Heat Transfer, Addison Wesley, 1966.

20. Carslaw, H.S., J.C. Jaeger, Conduction of Heat in Solids, Oxford University Press.

21. Streeter, V.L., Handbook of Fluid Dynamics, Mc-Graw Hill, 1961.

22. Mironer, A., Engineering Fluid Mechanics, Mc-Graw Hill, 1979.

APPENDIX A

FIDES' DATABASE

This appendix contains the main data base of the FIDES' system. Most of the different data structures, as well as the concept of list of structures, are explained in Chapter 3.

```
/*************************************************************************
 *              START OF THE CENTRAL DATABASE OF FIDES                   *
 *************************************************************************/


                                               /*------------------------------*/
struct  dot_struct                             /*      GEOMETRICAL DOT          */
        {                                      /*------------------------------*/
        short           number;                /* DOT NUMBER                    */
        double          x, y;                  /* ORIGINAL COORDINATES          */
        struct dot_struct *dot_next;           /* POINTER TO THE NEXT DOT       */
                                               /*------------------------------*/
        };                                     /*    END OF DOT DEFINITION      */
                                               /*------------------------------*/

typedef struct dot_struct dot;
dot             *dot_head, *dot_tail;
int             n_dots;



                                               /*------------------------------*/
struct  dot_list_struct                        /*         LIST OF DOTS          */
        {                                      /*------------------------------*/
        struct dot_struct *dot;                /* POINTER TO THE DOT            */
      struct dot_list_struct *next;            /* POINTER TO THE NEXT DOT_LIST */
                                               /*------------------------------*/
        };                                     /*  END OF DOT_LIST DEFINITION  */
                                               /*------------------------------*/

typedef struct dot_list_struct dot_list;



                                               /*------------------------------*/
struct segment_struct                          /*    GEOMETRICAL SEGMENT        */
        {                                      /*------------------------------*/
        short           number,                /* SEGMENT NUMBER                */
                        dots;                  /* NUMBER OF DOTS IN SEGMENT     */
        double          length;                /* ORIGINAL LENGTH OF SEGMENT    */
        char            type;                  /* TYPE OF SEGMENT (S,C,P)       */
       struct dot_list_struct *dot_list;/* POINTER TO LIST OF DOTS             */
        struct segment_struct *seg_next;/* POINTER TO NEXT SEGMENT             */
                                               /*------------------------------*/
        };                                     /*   END OF SEGMENT DEFINITION  */
                                               /*------------------------------*/

typedef struct segment_struct seg;
seg             *seg_head, *seg_tail;
int             n_segs;



                                               /*------------------------------*/
struct seg_list_struct                         /*      LIST OF SEGMENTS         */
        {                                      /*------------------------------*/
        struct segment_struct *seg;            /* POINTER TO THE SEGMENT        */
        struct seg_list_struct *next;          /* POINTER TO THE NEXT SEG LIST */
```

```
                                           /*--------------------------------*/

typedef struct seg_list_struct seg_list;




                                           /*--------------------------------*/
struct  intpt_struct                       /*        INTERNAL POINT          */
        {                                  /*--------------------------------*/
        short           number;            /* INTERNAL POINT NUMBER          */
        double          x, y,              /* COORDINATES                    */
                        xnew,ynew,         /* NEW COORDINATES                */
                        u,                 /* POTENTIAL                      */
                        q;                 /* FLUX=du/dn                     */
        struct intpt_struct *next;         /* POINTER TO THE NEXT INT. PT.   */
                                           /*--------------------------------*/
        };                                 /*  END OF INT. PT. DEFINITION    */
                                           /*--------------------------------*/

typedef struct intpt_struct intpt;
intpt          *intpt_head, *intpt_tail;
int            n_intpts;



                                           /*--------------------------------*/
struct  intpt_list_struct                  /*      LIST OF INT. PTS.         */
        {                                  /*--------------------------------*/
        struct intpt_struct *intpt;        /* POINTER TO THE INT. PT.        */
      struct intpt_list_struct *next;      /* POINTER TO NEXT INTPT_LIST     */
                                           /*--------------------------------*/
        };                                 /*  END INTPT_LIST DEFINITION     */
                                           /*--------------------------------*/

typedef struct intpt_list_struct intpt_list;




                                           /*--------------------------------*/
struct surface_struct                      /*     GEOMETRICAL SURFACE        */
   {                                       /*--------------------------------*/
   short         number,                   /* SURFACE NUMBER                 */
                 segs    ;                 /* NUMBER OF SEGMENTS IN SURFACE*/
   struct seg_list_struct *seg_list;       /* POINTER TO LIST OF SEGMENTS    */
   struct intpt_list_struct *intpt_list;/* POINTER TO INT. PT. LIST          */
   struct surface_struct *next;            /* POINTER TO NEXT SURFACE        */
                                           /*--------------------------------*/
        };                                 /*  END OF SURFACE DEFINITION     */
                                           /*--------------------------------*/

typedef struct surface_struct surf;
surf           *surf_head, *surf_tail;
int            n_surfs;
```

```
struct node_struct                          /*   NODE FOR DISCRETIZATION     */
        {                                   /*-------------------------------*/
        short   number;                     /* NODE NUMBER                   */
        double  x, y,                       /* ORIGINAL COORDINATES          */
                dx, dy,                     /* DISPLACEMENTS                 */
                xnew,ynew,                  /* DEFORMED SHAPE COORDINATES    */
                bcl,                        /* BOUNDARY CONDITION 1          */
                bc2,                        /* BONDARY CONDITION 2           */
                u,                          /* POTENTIAL IN POTENTIAL PROBS.*/
                q;                          /* FLUX IN POTENTIAL PROBS.      */
        char    kbcl,                       /* KIND OF BOUND. COND. 1        */
                kbc2;                       /* KIND OF BOUND. COND. 2        */
        struct node_struct *next;           /* POINTER TO THE NEXT NODE      */
                                            /*-------------------------------*/
        };                                  /*    END OF NODE DEFINITION     */
                                            /*-------------------------------*/

typedef struct node_struct node;
node            *node_head, *node_tail;
int             n_nodes;



                                            /*-------------------------------*/
struct  node_list_struct                    /*         LIST OF NODES         */
        {                                   /*-------------------------------*/
        struct node_struct *node;           /* POINTER TO THE NODE           */
      struct node_list_struct *next;        /* POINTER TO THE NEXT NODE_LIST*/
                                            /*-------------------------------*/
        };                                  /*  END OF NODE_LIST DEFINITION */
                                            /*-------------------------------*/

typedef struct node_list_struct node_list;



                                            /*-------------------------------*/
struct element_struct                       /*  ELEMENT FOR DISCRETIZATION  */
    {                                       /*-------------------------------*/
    short               number,             /* ELEMENT NUMBER                */
                        nodes;              /* NUMBER OF NODES IN ELEMENT    */
    double              length,             /* ORIGINAL LENGTH OF ELEMENT    */
                        lnew,               /* FINAL LENGTH OF ELEMENT       */
                        x1,y1,              /* ORIGINAL COORD. OF ELEM. START*/
                        x2,y2,              /* ORIGINAL COORD. OF ELEM. END*/
                        x1new,y1new,        /* FINAL COORD. OF ELEM. START   */
                        x2new,y2new;        /* FINAL COORD. OF ELEM. END     */
    char                type,               /* TYPE OF ELEMENT (S,C,P)       */
                        expansion;          /* EXPANSION OF ELEMENT (C,L,Q) */
    struct node_list_struct *node_list; /* POINTER TO LIST OF NODES      */
    struct element_struct *next;            /* POINTER TO NEXT ELEMENT       */
                                            /*-------------------------------*/
    };                                      /* END OF ELEMENT DEFINITION     */
                                            /*-------------------------------*/

typedef struct element_struct elem;
elem            *elem_head, *elem_tail;
int             n_elems;
```

```
struct  plot_struct                              /* PLOTTING OPTIONS                */
    {                                            /*--------------------------------*/
    short                dots:1,                 /* PLOT DOTS                      */
                         nodes:1,                /* PLOT NODES                     */
                         segments:1,             /* PLOT SEGEMENTS                 */
                         elements:1,             /* PLOT ELEMENTS                  */
                         int_pts:1;              /* PLOT INTERNAL POINTS           */
} plot_options;                                  /*********************************/


/***********************************************************************
*              END OF FIDES' CENTRAL DATABASE DEFINITION.              *
***********************************************************************/
```

APPENDIX B

EXTRACTS FROM FIDES' CODE

This appendix contains some of FIDES' functions. The system consists of roughly 9000 lines of code (about 150 pages of program listings), and is too large to be included in this thesis. For any information about the FIDES's system, or about some specific functions, please contact the author.

```
/*************************************************************************
 *               START of create_boundary FUNCTION                       *
 *************************************************************************/


#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include "ndatabase.h"


#define begin            {
#define end              ;}
#define toupper(c)       (islower(c)?((c)-('a'-'A')):(c))

/* DEFINE MASTER I/O FILES */
FILE *master, *prompt;


create_boundary()
begin                                            /*BEGIN CREATE_BOUNDARY*/


int             i, j, nd, ns;
extern int      n_surfs;
double          x1, y1, x2, y2, x3, y3, l2, r1, r2, r;
char            module[9],check1[9],check2[9],answer[9],segtype[9];
char            direction[10];
dot             *dot_a, *dot_b, *dot_c, *attach_dot();
dot_list        *dot_list_a,*dot_list_b,*dot_list_c,*attach_dot_list();
seg             *seg_w, *seg_b, *attach_seg();
extern FILE     *master, *prompt;
extern dot      *dot_head;
extern seg      *seg_head;
extern int      n_dots, n_segs;

answer[0]='Y';
check1[0]='N';
check2[0]='Y';

while(strcmp(answer,"n",1)&&strcmp(answer,"N",1))
begin                                            /*BEGIN WHILE LOOP*/

   nd = n_dots + 1;
   ns = n_segs + 1;

   fprintf(prompt,"\nTO CREATE A SEGMENT, TYPE: S");
   fprintf(prompt,"\nTO     OUTPUT      , TYPE: O");
   fprintf(prompt,"\nTO     EDIT        , TYPE: E");
   fprintf(prompt,"\nTO PLOT THE GEOMETRY,TYPE: P");
   fprintf(prompt,"\nENTER YOUR COMMAND :");
   fscanf(master,"%s",module);

   switch(toupper(module[0]))
   begin                                         /*BEGIN FIRST SWITCH LOOP*/

     case 'S':                          /*BEGIN CASE S IN FIRST SWITCH*/

      if(n_surfs == 0)
      n_surfs = 1;

      while(strcmp(check2,"n",1)&&strcmp(check2,"N",1))
      begin                                      /*BEGIN WHILE LOOP*/
```

```
fprintf(prompt,"\nTO CREATE: -A STRAIGHT LINE, TYPE: S\n");
fprintf(prompt,"    -A CIRCLE          ,        C\n");
fprintf(prompt,"    -A POLYNOMIAL      ,        P\n");
fprintf(prompt,"ENTER YOUR COMMAND: ");
fscanf(master,"%s",segtype);

switch(toupper(segtype[0]))
begin                                         /*BEGIN SECOND SWITCH*/

case 'S':

 fprintf(prompt,"\n\nTO CREATE A STRAIGHT LINE YOU NEED TO INPUT :");
 fprintf(prompt,"\n TWO POINTS FOR THE FIRST SEGMENT OF A SURFACE,");
 fprintf(prompt,"\n ONE POINT FOR EACH SUBSEQUENT SEGMENT.");

 /* IN CASE IT IS THE FIRST DOT OF THE SURFACE*/
 if (check1[0]=='N' || check1[0]=='n')
 begin                                        /*BEGIN IF*/

  /* CREATE FIRST DOT OF SEGMENT */
  dot_a = attach_dot(nd++);
  fprintf(prompt,"\n\nGIVE COORDINATES OF FIRST POINT : x1,y1\n");
  fscanf(master,"%f %f",&x1,&y1);
  dot_a->x = x1;
  dot_a->y = y1;

  /* CREATE SECOND DOT OF SEGMENT */
  dot_b = attach_dot(nd++);
  fprintf(prompt,"\n\nGIVE COORDINATES OF SECOND POINT : x2,y2\n");
  fscanf(master,"%f %f",&x2,&y2);
  dot_b->x=x2;
  dot_b->y=y2;

 end                                          /*END IF*/

 /* IN CASE IT IS NOT THE FIRST DOT */
 else
 begin                                        /*BEGIN ELSE*/

  /* FIRST DOT = PREVIOUS SECOND DOT */
  dot_a = dot_b;
  x1 = dot_a->x;
  y1 = dot_a->y;

  /* CREATE SECOND DOT OF SEGMENT */
  dot_b = attach_dot(nd++);
  fprintf(prompt,"\n\nGIVE COORDINATES OF SECOND POINT : x2,y2\n");
  fscanf(master,"%f %f",&x2,&y2);
  dot_b->x=x2;
  dot_b->y=y2;

 end                               /*END OF ELSE*/

 /* CREATE SEGMENT */
 seg_w=attach_seg(ns++);
 seg_w->dots=2;
 l2=(x1-x2)*(x1-x2)+(y1-y2)*(y1-y2);
 seg_w->length = sqrt(l2);
 seg_w->type = 'S';

 /* POINT TO THE TWO DOTS */
 dot_list_a = attach_dot_list();
 seg_w->dot_list = dot_list_a;
 dot_list_b = attach_dot_list();
 dot_list_a->next = dot_list_b;
 dot_list_b->dot = dot_b;

 break;

case 'C':

 fprintf(prompt,"\n\nTO CREATE A CIRCULAR SEGMENT YOU NEED TO INPUT :");
 fprintf(prompt,"\n2 POINTS AND A CENTER, FOR THE FIRST SEGMENT OF A SURF
 fprintf(prompt,"\n1 POINT AND A CENTER, FOR EACH SUBSEQUENT SEGMENT.");

 /* IN CASE IT IS THE FIRST DOT OF THE SURFACE*/
 if (check1[0]=='N' || check1[0]=='n')
 begin                                        /*BEGIN IF*/

  /*ASK FOR DATA*/
  fprintf(prompt,"\n\nGIVE COORDINATES OF FIRST POINT : x1,y1\n");
  fscanf(master,"%f %f",&x1,&y1);
  fprintf(prompt,"\n\nGIVE COORDINATES OF SECOND POINT : x2,y2\n");
  fscanf(master,"%f %f",&x2,&y2);
  fprintf(prompt,"\n\nGIVE COORDINATES OF THE CENTER : x3,y3\n");
  fscanf(master,"%f %f",&x3,&y3);
  fprintf(prompt,"\nIS YOUR CIRCLE COUNTERCLOCKWISE W.R.T. ITS CENTER (Y/
  fscanf(master,"%s",direction);


  /*CHECK RADIUS*/
  r1=(x1-x3)*(x1-x3)+(y1-y3)*(y1-y3);
  r2=(x3-x2)*(x3-x2)+(y3-y2)*(y3-y2);
  if (fabs(r1-r2)>0.00001)
  begin                                       /*BEGIN IF*/
    fprintf(prompt,"Pt.1 AND Pt.2 ARE NOT AT SAME DISTANCE FROM CENTER.")
    fprintf(prompt,"YOUR RADIUS IS WRONG. CHECK DATA AND TRY AGAIN.");
    break;
  end                                         /*END IF*/
  r = sqrt(r1);

  /* CREATE FIRST DOT OF SEGMENT */
  dot_a = attach_dot(nd++);
  dot_a->x = x1;
  dot_a->y = y1;

  /* CREATE SECOND DOT OF SEGMENT */
  dot_b = attach_dot(nd++);
  dot_b->x=x2;
  dot_b->y=y2;

  /* CREATE CENTER OF SEGMENT */
  dot_c = (dot *) malloc(sizeof(dot));
  dot_c->x=x3;
  dot_c->y=y3;
  if (toupper(direction[0])=='N')
    dot_c->number = -1;
  else
    dot_c->number = 1;

 end                                          /*END IF*/

 /* IN CASE IT IS NOT THE FIRST DOT */
 else
 begin                                        /*BEGIN ELSE*/

  /* FIRST DOT = PREVIOUS SECOND DOT */
  dot_a = dot_b;
  x1 = dot_a->x;
  y1 = dot_a->y;
  /*ASK FOR DATA*/
  fprintf(prompt,"\n\nGIVE COORDINATES OF SECOND POINT : x2,y2\n");
  fscanf(master,"%f %f",&x2,&y2);
  fprintf(prompt,"\n\nGIVE COORDINATES OF THE CENTER : x3,y3\n");
  fscanf(master,"%f %f",&x3,&y3);
  fprintf(prompt,"\nIS YOUR CIRCLE COUNTERCLOCKWISE W.R.T. ITS CENTER (Y/
  fscanf(master,"%s",direction);

  /*CHECK RADIUS*/
  r1=(x1-x3)*(x1-x3)+(y1-y3)*(y1-y3);
  r2=(x3-x2)*(x3-x2)+(y3-y2)*(y3-y2);
  if (fabs(r1-r2)>0.00001)
  begin                                  /*BEGIN IF*/
    fprintf(prompt,"Pt.1 AND Pt.2 ARE NOT AT SAME DISTANCE FROM CENTER.")
    fprintf(prompt,"YOUR RADIUS IS WRONG. CHECK DATA AND TRY AGAIN.");
    break;
  end                                    /*END IF*/
  r = sqrt(r1);

  /* CREATE SECOND DOT OF SEGMENT */
  dot_b = attach_dot(nd++);
  dot_b->x=x2;
  dot_b->y=y2;

  /* CREATE CENTER OF SEGMENT */
  dot_c = (dot *) malloc(sizeof(dot));
  dot_c->x=x3;
  dot_c->y=y3;
  if (toupper(direction[0])=='N')
    dot_c->number = -1;
  else
    dot_c->number = 1;

 end                            /*END OF ELSE*/

 /* CREATE SEGMENT */
 seg_w=attach_seg(ns++);
 seg_w->dots=3;
 seg_w->length = r;
 seg_w->type = 'C';

 /* POINT TO THE THREE DOTS */
 dot_list_a = attach_dot_list();
 seg_w->dot_list = dot_list_a;
 dot_list_b = attach_dot_list();
 dot_list_a->next = dot_list_b;
 dot_list_c = attach_dot_list();
 dot_list_b->next = dot_list_c;
 dot_list_a->dot = dot_a;
 dot_list_b->dot = dot_b;
 dot_list_c->dot = dot_c;

 break;


case 'P':
 fprintf(prompt,"\n POLYNOMIAL BEING IMPLEMENTED !\n");
 break;

default:
 fprintf(prompt,"NO SUCH SEGMENT TYPE.");
 break;

end                                     /*END SECOND SWITCH*/
```

```
      fprintf(prompt,"\nARE YOU STILL ON THE SAME SURFACE (Y/N)?\n");
      fscanf(master,"%s",check1);

      check2[0] = check1[0];

      if(check1[0] == 'N' || check1[0] == 'n')
      n_surfs++;

      end                                       /*END WHILE LOOP*/

    break;                              /*END S CASE IN FIRST SWITCH*/

    case 'O':                   /*BEGIN CASE O IN FIRST SWITCH*/

     output_geometry();
     break;                                     /*END CASE O IN FIRST SWITCH*/

    case 'P':                           /*BEGIN CASE P IN FIRST SWITCH*/

     be_plot();
     break;                                     /*END CASE P IN FIRST SWITCH*/

    case 'E':                           /*BEGIN CASE E IN FIRST SWITCH*/

     edit_create();
     break;                                     /*END CASE E IN FIRST SWITCH*/

    default:

     fprintf(prompt,"\n\nUNRECOGNIZED COMMAND!...\n\n\n");
     break;


    end                                         /*END FIRST SWITCH LOOP*/

    check2[0]='Y';

    fprintf(prompt,"\nDO YOU WANT TO CONTINUE CREATING ?");
    fprintf(prompt,"\nENTER YES OR NO :");
    fscanf(master,"%s",answer);
end                                             /*END OF WHILE LOOP*/

return;

end                                             /*END OF CREATE_BOUNDARY*/

/**************************************************************************
*                END of create_boundary FUNCTION                         *
**************************************************************************/
```

APPENDIX C

DISCRETIZATION RUN ON UNIX/VAX-750 SYSTEM

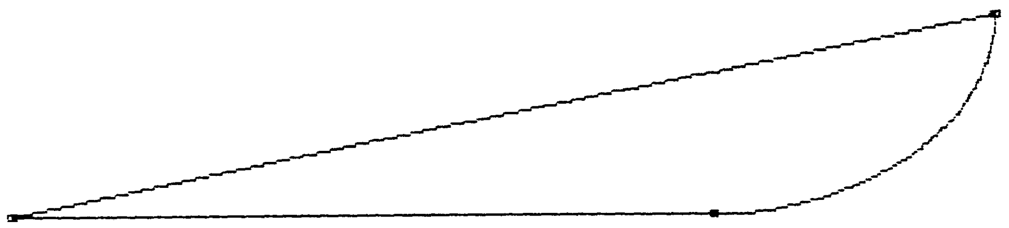
This appendix contains an example of the discretization of the problem shown in Fig. C.1, using the UNIX/VAX-750's version of FIDES. It will be limited to a mesh generation, to avoid redundancy with Appendix D which contains a complete illustration of the system. In this appendix the editor facility will be domonstrated as well as the graphics capabilities. The menu driven option is illustrated better by the AEGIS/APOLLO's version of FIDES.

To obtain a copy of FIDES executable file, please contact the author.

```
H     H     EEEEEE     L          L          0000000
H     H     E          L          L          0     0
H     H     E          L          L          0     0
H     H     E          L          L          0     0
HHHHHHH     EEEE       L          L          0     0
H     H     E          L          L          0     0
H     H     E          L          L          0     0
H     H     E          L          L          0     0
H     H     EEEEEE     LLLLLLL    LLLLLLL    0000000


YOU ENTERED THE MAIN PROGRAM

COMMANDS ARE : MESH, SOLVE, POST, EXIT

ENTER YOUR COMMAND :  m


YOU ENTERED THE DISCRETE ELEMENT MESH GENERATOR

COMMANDS ARE : CREATE, BEMESH, FEMESH, EXIT

ENTER YOUR COMMAND :  c

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
EXTERIOR BOUNDARY SHOULD BE COUNTERCLOCKWISE.
   INTERIOR BOUNDARY SHOULD BE CLOCKWISE.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

TO CREATE A SEGMENT, TYPE: S
TO     OUTPUT      , TYPE: O
TO     EDIT        , TYPE: E
TO PLOT THE GEOMETRY,TYPE: P
ENTER YOUR COMMAND :s

TO CREATE: -A STRAIGHT LINE, TYPE: S
           -A CIRCLE       ,       C
           -A POLYNOMIAL   ,       P
ENTER YOUR COMMAND: s


TO CREATE A STRAIGHT LINE YOU NEED TO INPUT :
 TWO POINTS FOR THE FIRST SEGMENT OF A SURFACE,
 ONE POINT FOR EACH SUBSEQUENT SEGMENT.
```

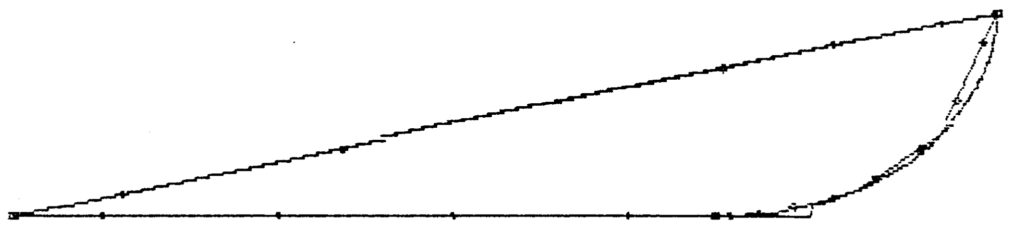
```
GIVE COORDINATES OF FIRST POINT : x1,y1
0 0

GIVE COORDINATES OF SECOND POINT : x2,y2
5 0

ARE YOU STILL ON THE SAME SURFACE (Y/N)? y


TO CREATE: -A STRAIGHT LINE, TYPE: S
           -A CIRCLE       ,       C
           -A POLYNOMIAL   ,       P
ENTER YOUR COMMAND: c


TO CREATE A CIRCULAR SEGMENT YOU NEED TO INPUT :
2 POINTS AND A CENTER, FOR THE FIRST SEGMENT OF A SURFACE,
1 POINT AND A CENTER, FOR EACH SUBSEQUENT SEGMENT.

GIVE COORDINATES OF SECOND POINT : x2,y2
7 2

GIVE COORDINATES OF THE CENTER : x3,y3
5 2

IS YOUR CIRCLE COUNTERCLOCKWISE W.R.T. ITS CENTER (Y/N)? y


ARE YOU STILL ON THE SAME SURFACE (Y/N)? y

TO CREATE: -A STRAIGHT LINE, TYPE: S
           -A CIRCLE       ,       C
           -A POLYNOMIAL   ,       P
ENTER YOUR COMMAND: s

TO CREATE A STRAIGHT LINE YOU NEED TO INPUT :
 TWO POINTS FOR THE FIRST SEGMENT OF A SURFACE,
 ONE POINT FOR EACH SUBSEQUENT SEGMENT.

GIVE COORDINATES OF SECOND POINT : x2,y2
0 0

ARE YOU STILL ON THE SAME SURFACE (Y/N)? n


DO YOU WANT TO CONTINUE CREATING ?y

ENTER YES OR NO :
TO CREATE A SEGMENT, TYPE: S
TO     OUTPUT      , TYPE: O
TO     EDIT        , TYPE: E
TO PLOT THE GEOMETRY,TYPE: P
ENTER YOUR COMMAND : q
```

```
 NUMBER OF DOTS = 4


DOT  COORDINATES
No.    X        Y
---  -----    -----

  1      0.00      0.00
  2      5.00      0.00
  3      7.00      2.00
  4      0.00      0.00



 NUMBER OF SEGS = 3


   LIST OF STRAIGHT LINE SEGMENTS

 SEG    DOT1     DOT2    LENGTH
 No.     No.      No.
----    ----     ----     ------

  1       1       2     5.000000
  3       3       4     7.280110



   LIST OF CIRCULAR SEGMENTS

 SEG    DOT1     DOT2    CENTER   CENTER   RADIUS
 No.     No.      No.      X        Y
----    ----     ----    ------   ------   -------

  2       2       3       5.000    2.000    2.000


TO CREATE NODES , TYPE :NODE

TO INPUT LOADS , TYPE    :LOAD

FOR INTERNAL POINTS, TYPE:INT

TO OUTPUT INFORMATION , TYPE :OUT

TO EDIT INFORMATION , TYPE   :EDIT

TO PLOT THE PROBLEM , TYPE   :PLOT

TO EXIT THE MESH GENERATOR , TYPE :EX

ENTER YOUR COMMAND : p
```



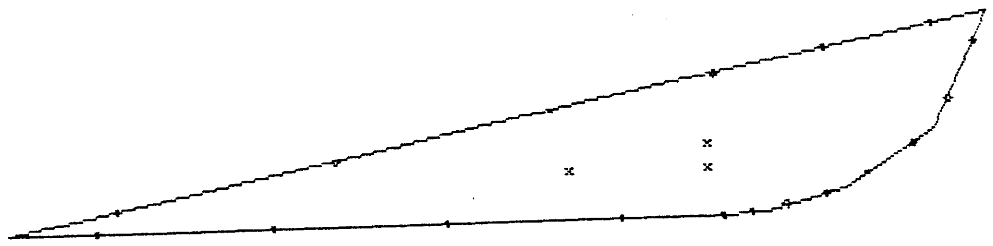
```
   DO YOU WANT TO CONTINUE CREATING ?
   ENTER YES OR NO : n


COMMANDS ARE : CREATE, BEMESH, FEMESH, EXIT

ENTER YOUR COMMAND : b

ARE YOU SOLVING A 2D POTENTIAL (P)
OR 2D ELASTICITY PROBLEM (E)?
p

DO YOU WANT CONTINUOUS (C) OR
DISCONTINUOUS (D) NODES ?
d


TO CREATE NODES , TYPE :NODE

TO INPUT LOADS , TYPE    :LOAD

FOR INTERNAL POINTS, TYPE:INT

TO OUTPUT INFORMATION , TYPE :OUT

TO EDIT INFORMATION , TYPE   :EDIT

TO PLOT THE PROBLEM , TYPE   :PLOT

TO EXIT THE MESH GENERATOR , TYPE :EX

ENTER YOUR COMMAND : n
ENTER THE SEGMENT NUMBER : 1

YOU CAN DIVIDE YOUR SEGMENT INTO :
EQUAL SIZE ELEMENTS BY TYPING :EQ
GRADED SIZE ELEMENTS BY TYPING :GR
ARBITRARY SIZE ELEMENTS BY TYPING :AR
ENTER THE DIVISION DESIRED : e

ENTER THE NUMBER OF ELEMENTS
4

ENTER THE TYPE OF EXPANSION DESIRED .
FOR CONSTANT DISPLACEMENT EXPANSION TYPE : C
FOR LINEAR DISPLACEMENT EXPANSION TYPE : L
FOR QUADRATIC DISPLACEMENT EXPANSION TYPE : Q
c

TO CREATE NODES , TYPE   :NODE

TO INPUT LOADS , TYPE    :LOAD

FOR INTERNAL POINTS, TYPE:INT

TO OUTPUT INFORMATION , TYPE :OUT

TO EDIT INFORMATION , TYPE   :EDIT

TO PLOT THE PROBLEM , TYPE   :PLOT

TO EXIT THE MESH GENERATOR , TYPE :EX

ENTER YOUR COMMAND : n

ENTER THE SEGMENT NUMBER : 2

YOU CAN DIVIDE YOUR SEGMENT INTO :
EQUAL SIZE ELEMENTS BY TYPING :EQ
GRADED SIZE ELEMENTS BY TYPING :GR
ARBITRARY SIZE ELEMENTS BY TYPING :AR
ENTER THE DIVISION DESIRED : g

ENTER THE NUMBER OF ELEMENTS : 4

ENTER RATIO OF LAST ELEMENT TO FIRST ELEMENT : 3

ENTER THE TYPE OF EXPANSION DESIRED .
FOR CONSTANT DISPLACEMENT EXPANSION TYPE : C
FOR LINEAR DISPLACEMENT EXPANSION TYPE : L
FOR QUADRATIC DISPLACEMENT EXPANSION TYPE : Q
l
```

```
 TO CREATE NODES , TYPE   :NODE

 TO INPUT LOADS , TYPE    :LOAD

 FOR INTERNAL POINTS, TYPE:INT

 TO OUTPUT INFORMATION , TYPE :OUT

 TO EDIT INFORMATION , TYPE   :EDIT

 TO PLOT THE PROBLEM , TYPE   :PLOT

 TO EXIT THE MESH GENERATOR , TYPE :EX

 ENTER YOUR COMMAND : n

ENTER THE SEGMENT NUMBER : 3

YOU CAN DIVIDE YOUR SEGMENT INTO :
EQUAL SIZE ELEMENTS BY TYPING :EQ
GRADED SIZE ELEMENTS BY TYPING :GR
ARBITRARY SIZE ELEMENTS BY TYPING :AR
ENTER THE DIVISION DESIRED : g

ENTER THE NUMBER OF ELEMENTS : 2

ENTER RATIO OF LAST ELEMENT TO FIRST ELEMENT : 2

ENTER THE TYPE OF EXPANSION DESIRED .
FOR CONSTANT DISPLACEMENT EXPANSION TYPE : C
FOR LINEAR DISPLACEMENT EXPANSION TYPE : L
FOR QUADRATIC DISPLACEMENT EXPANSION TYPE : Q
q


TO CREATE NODES , TYPE   :NODE

TO INPUT LOADS , TYPE    :LOAD

FOR INTERNAL POINTS, TYPE:INT

TO OUTPUT INFORMATION , TYPE :OUT

TO EDIT INFORMATION , TYPE   :EDIT

TO PLOT THE PROBLEM , TYPE   :PLOT

TO EXIT THE MESH GENERATOR , TYPE :EX

ENTER YOUR COMMAND : p
```



```
HOW MANY INTERNAL POINTS DO YOU WANT?

 3

ENTER THE COORDINATES (X,Y) OF POINT: 1

 5.00      0.50

ENTER THE COORDINATES (X,Y) OF POINT: 2

 5.00      0.75

ENTER THE COORDINATES (X,Y) OF POINT: 3

 4.00      0.50
```

```
TO CREATE NODES , TYPE   :NODE

TO INPUT LOADS , TYPE    :LOAD

FOR INTERNAL POINTS, TYPE:INT

TO OUTPUT INFORMATION , TYPE :OUT

TO EDIT INFORMATION , TYPE   :EDIT

TO PLOT THE PROBLEM , TYPE   :PLOT

TO EXIT THE MESH GENERATOR , TYPE :EX

ENTER YOUR COMMAND : p
```



```
TO CREATE NODES , TYPE :NODE

TO INPUT LOADS , TYPE    :LOAD

FOR INTERNAL POINTS, TYPE:INT

TO OUTPUT INFORMATION , TYPE :OUT

TO EDIT INFORMATION , TYPE   :EDIT

TO PLOT THE PROBLEM , TYPE   :PLOT

TO EXIT THE MESH GENERATOR , TYPE :EX

ENTER YOUR COMMAND : ex

DO YOU WANT TO STAY IN BEMESH ?
ENTER YES or NO: n

COMMANDS ARE : CREATE, BEMESH, FEMESH, EXIT

ENTER YOUR COMMAND :ex
end of mesh in demg
DO YOU WANT TO STAY IN DEMG ?
ENTER YES or NO: n
MESH END

COMMANDS ARE : MESH, SOLVE, POST, EXIT

ENTER YOUR COMMAND : ex



        BBBBBBB        Y         Y      EEEEEEEE
        B      B        Y       Y       E
        B       B        Y     Y        E
        B       B         Y   Y         E
        B      B           Y Y          E
        BBBBBBBB            Y           EEEEEEEE
        B      B            Y           E
        B       B           Y           E
        B        B          Y           E
        B       B           Y           E
        B      B            Y           E
        BBBBBBB             Y           EEEEEEEE
```

APPENDIX D

COMPLETE RUN ON AEGIS/APOLLO SYSTEM

This appendix contains an example run using the AEGIS/ APOLLO's version of FIDES for the problem outlined in Fig. D.1. The three different stages of the system:

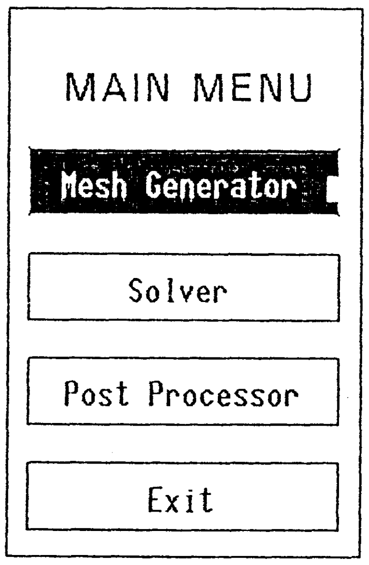
- pre-processor,
- solver, and
- post-processor

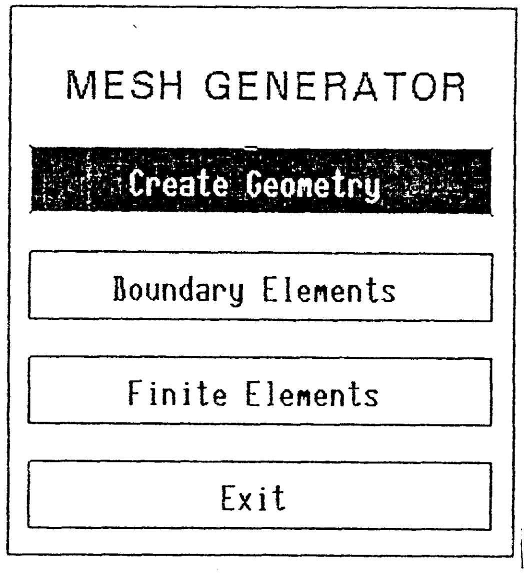
are used in illustrating this problem.

The different menus on APOLLO are shown for each stage. The dark option indicates the mouse's selection. For similar steps, menus are not shown again to avoid repetition. As the reader can notice, no explanation is necessary to be able to use the program. Although this gives a taste of the system's capabilities, it is not as impressive as a real demonstration. To obtain a copy of the executable file, please contact the author.

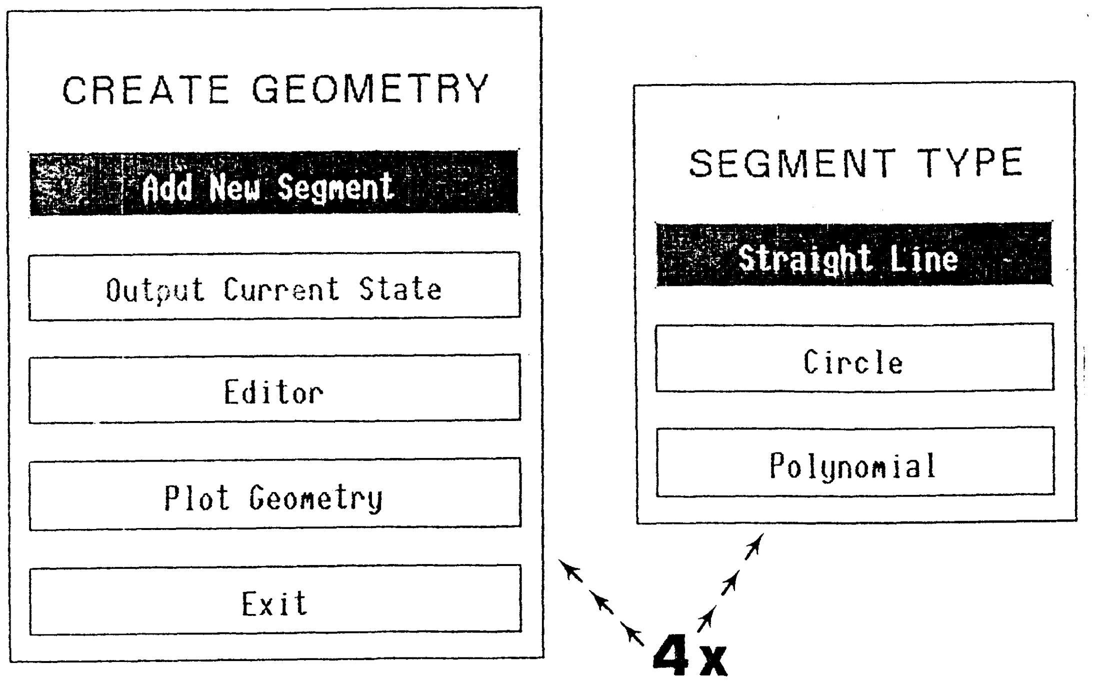
```
********************   HELLO  *************************
*                                                     *
*                  WELCOME  TO :                      *
*                                                     *
*                  **************                     *
*                  *            *                     *
*                  *    FIDES   *                     *
*                  *            *                     *
*                  **************                     *
*                                                     *
*                                                     *
*    The Friendly Interactive Discrete Element Solver .*
*                                                     *
*                                                     *
*             Developed by :   FOUAD G. TAMER         *
*                                                     *
*                                                     *
*    Under the guidance of :   Prof. JEROME J. CONNOR *
*                                                     *
*******************************************************

 TYPE <RETURN> TO CONTINUE
```

```
*******************************************************
*  NOTE : Thanks are due to :                         *
*                                                     *
*              FADI A. CHEHAYEB                       *
*   for his help with graphics, and the menus on Apollo*
*                                                     *
*                                                     *
*           and ROBERT B. PETROSSIAN                  *
*   for transfering this program on Apollo.           *
*                                                     *
*******************************************************

 TYPE <RETURN> TO CONTINUE
```



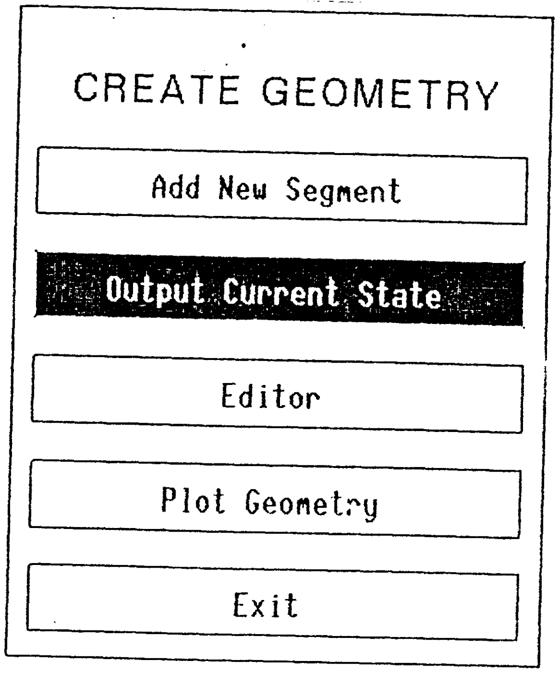


```
YOU ENTERED THE DISCRETE ELEMENT MESH GENERATOR

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
EXTERIOR BOUNDARY SHOULD BE COUNTERCLOCKWISE.
   INTERIOR BOUNDARY SHOULD BE CLOCKWISE.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

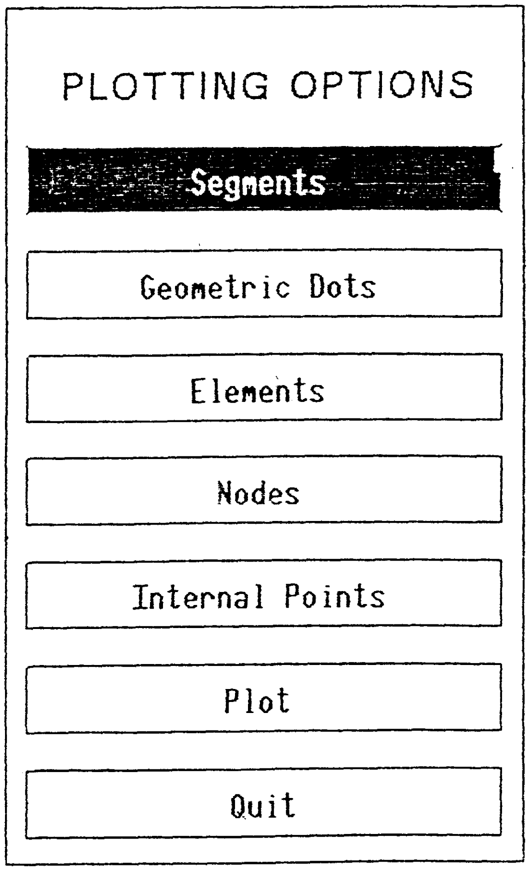


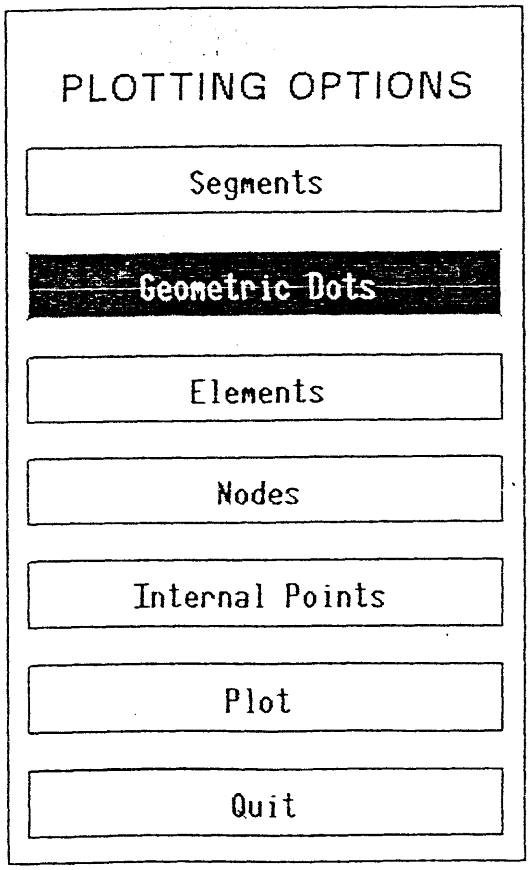
```
TO CREATE A STRAIGHT LINE YOU NEED TO INPUT :
 TWO POINTS FOR THE FIRST SEGMENT OF A SURFACE,
 ONE POINT FOR EACH SUBSEQUENT SEGMENT.

GIVE COORDINATES OF FIRST POINT : x1 y1



SPACE COORDINATES BY SPACES OR <RETURN>
0 0


GIVE COORDINATES OF SECOND POINT : x2 y2


SPACE COORDINATES BY SPACES OR <RETURN>
5 0

ARE YOU STILL ON THE SAME SURFACE (Y/N)?
y


TO CREATE A STRAIGHT LINE YOU NEED TO INPUT :
 TWO POINTS FOR THE FIRST SEGMENT OF A SURFACE,
 ONE POINT FOR EACH SUBSEQUENT SEGMENT.

GIVE COORDINATES OF SECOND POINT : x2 y2


SPACE COORDINATES BY SPACES OR <RETURN>
5 5

ARE YOU STILL ON THE SAME SURFACE (Y/N)?
y


TO CREATE A STRAIGHT LINE YOU NEED TO INPUT :
 TWO POINTS FOR THE FIRST SEGMENT OF A SURFACE,
 ONE POINT FOR EACH SUBSEQUENT SEGMENT.

GIVE COORDINATES OF SECOND POINT : x2 y2


SPACE COORDINATES BY SPACES OR <RETURN>
0 5

ARE YOU STILL ON THE SAME SURFACE (Y/N)?
y
```

```
TO CREATE A STRAIGHT LINE YOU NEED TO INPUT :
 TWO POINTS FOR THE FIRST SEGMENT OF A SURFACE,
 ONE POINT FOR EACH SUBSEQUENT SEGMENT.

GIVE COORDINATES OF SECOND POINT : x2 y2


SPACE COORDINATES BY SPACES OR <RETURN>
0 0

ARE YOU STILL ON THE SAME SURFACE (Y/N)?
n
```

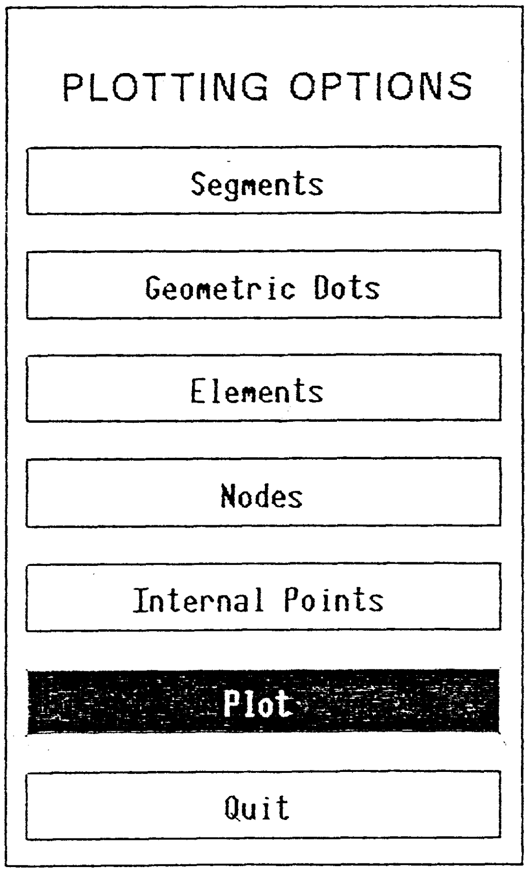


```
NUMBER OF DOTS = 5


 DOT   COORDINATES
 No.     X          Y
 ---   -----     -----

   1      0.00       0.00
   2      5.00       0.00
   3      5.00       5.00
   4      0.00       5.00
   5      0.00       0.00
 NUMBER OF SEGS = 4


   LIST OF STRAIGHT LINE SEGMENTS

 SEG    DOT1     DOT2    LENGTH
 No.     No.      No.
----    ----     ----    ------


  1       1        2    5.000000
  2       2        3    5.000000
  3       3        4    5.000000
  4       4        5    5.000000


PLOT OPTIONS:

 INT_PTS    DOTS    SEGMENTS    NODES    ELEMENTS
   OFF      OFF       OFF        OFF        OFF
```

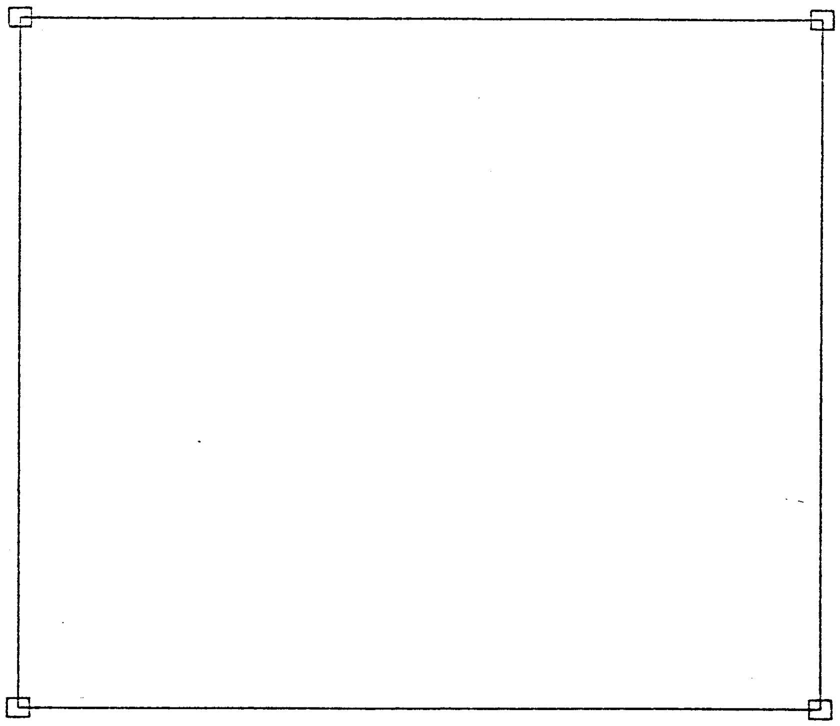


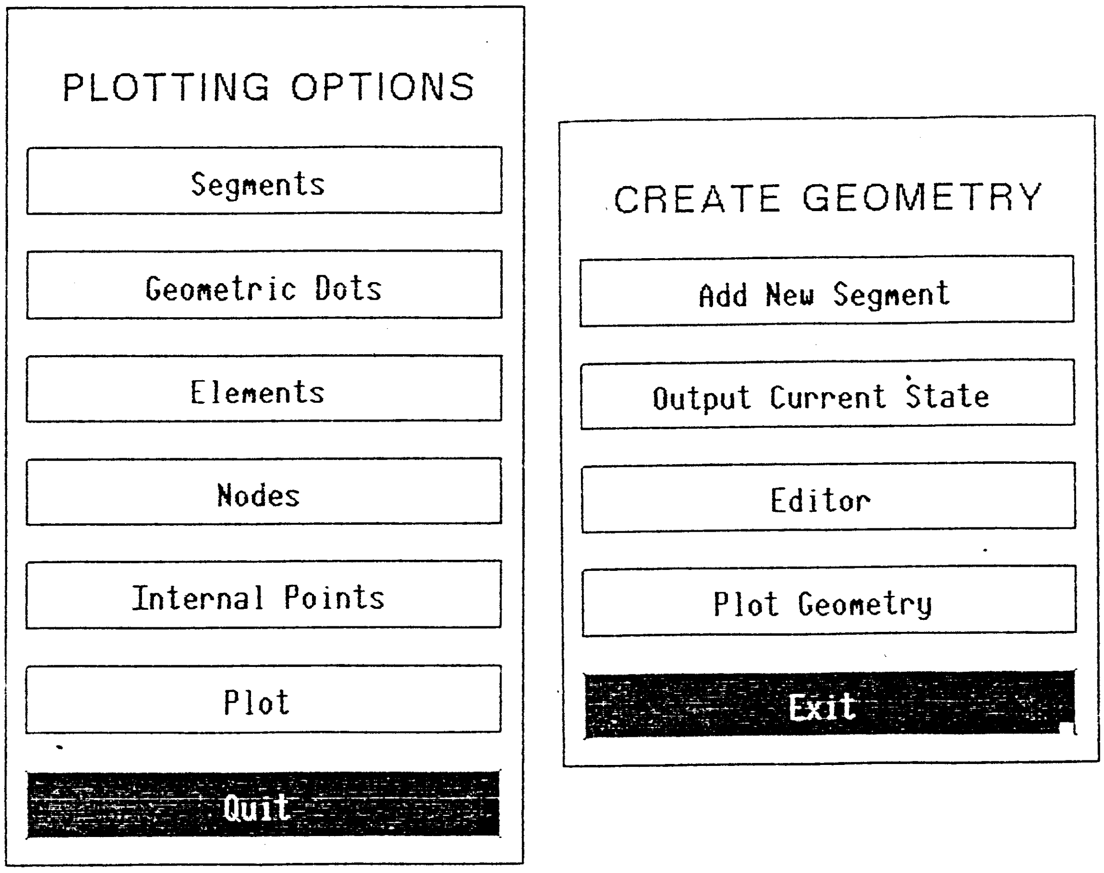


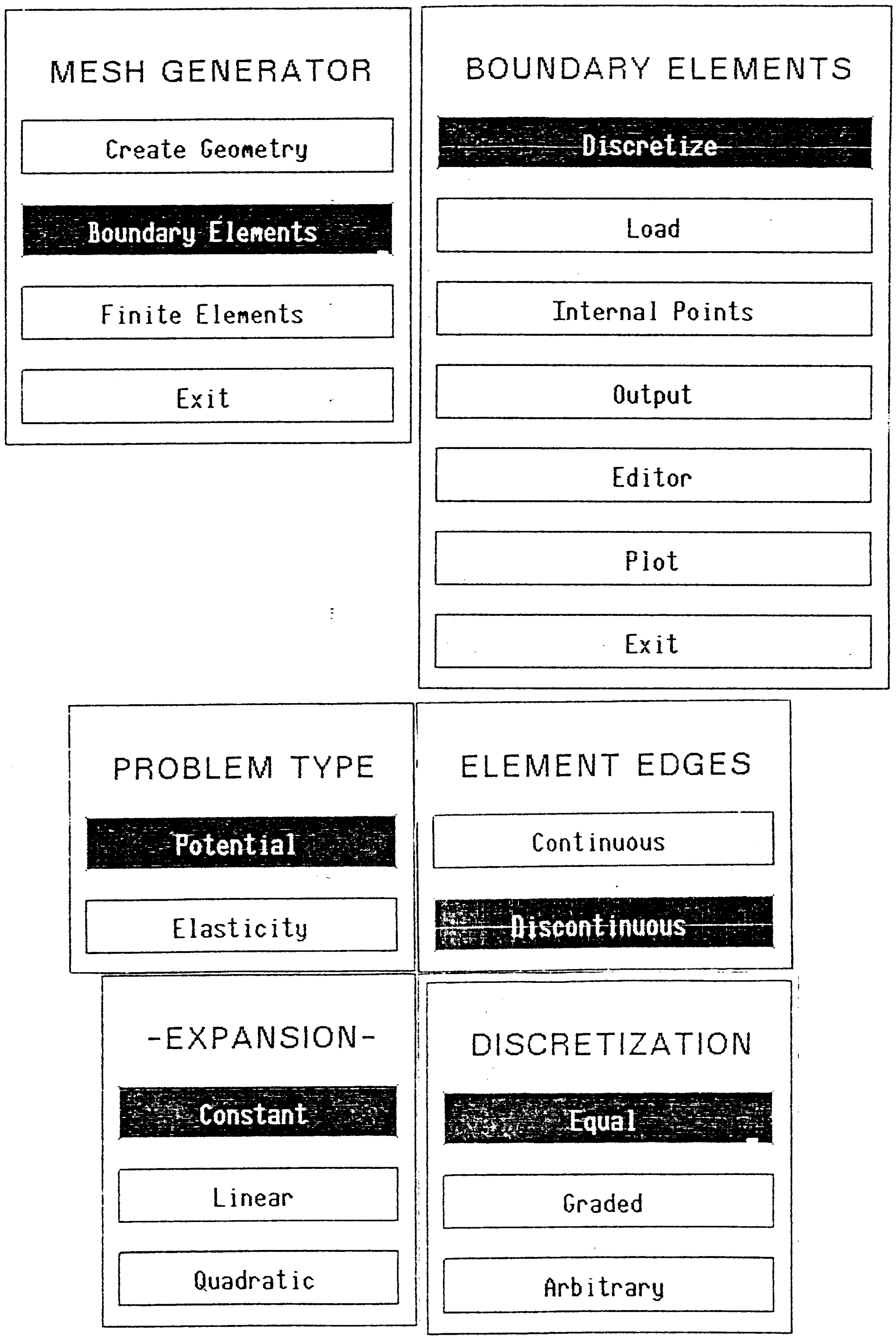
```
PLOT OPTIONS:

  INT_PTS    DOTS    SEGMENTS    NODES    ELEMENTS
    OFF       OFF       ON         OFF       OFF

PLOT OPTIONS:

  INT_PTS    DOTS    SEGMENTS    NODES    ELEMENTS
    OFF       ON        ON         OFF       OFF
```



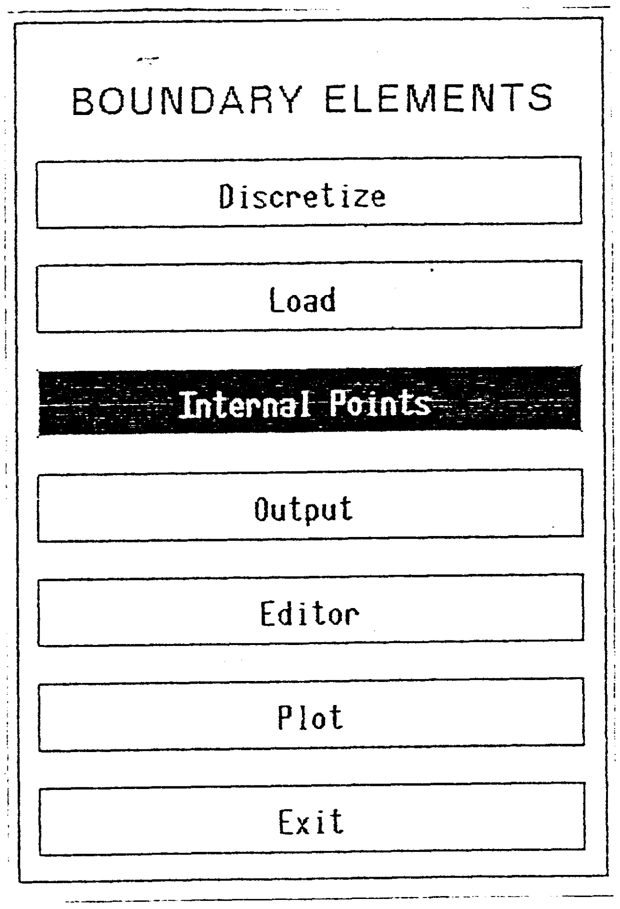
P.T.O.



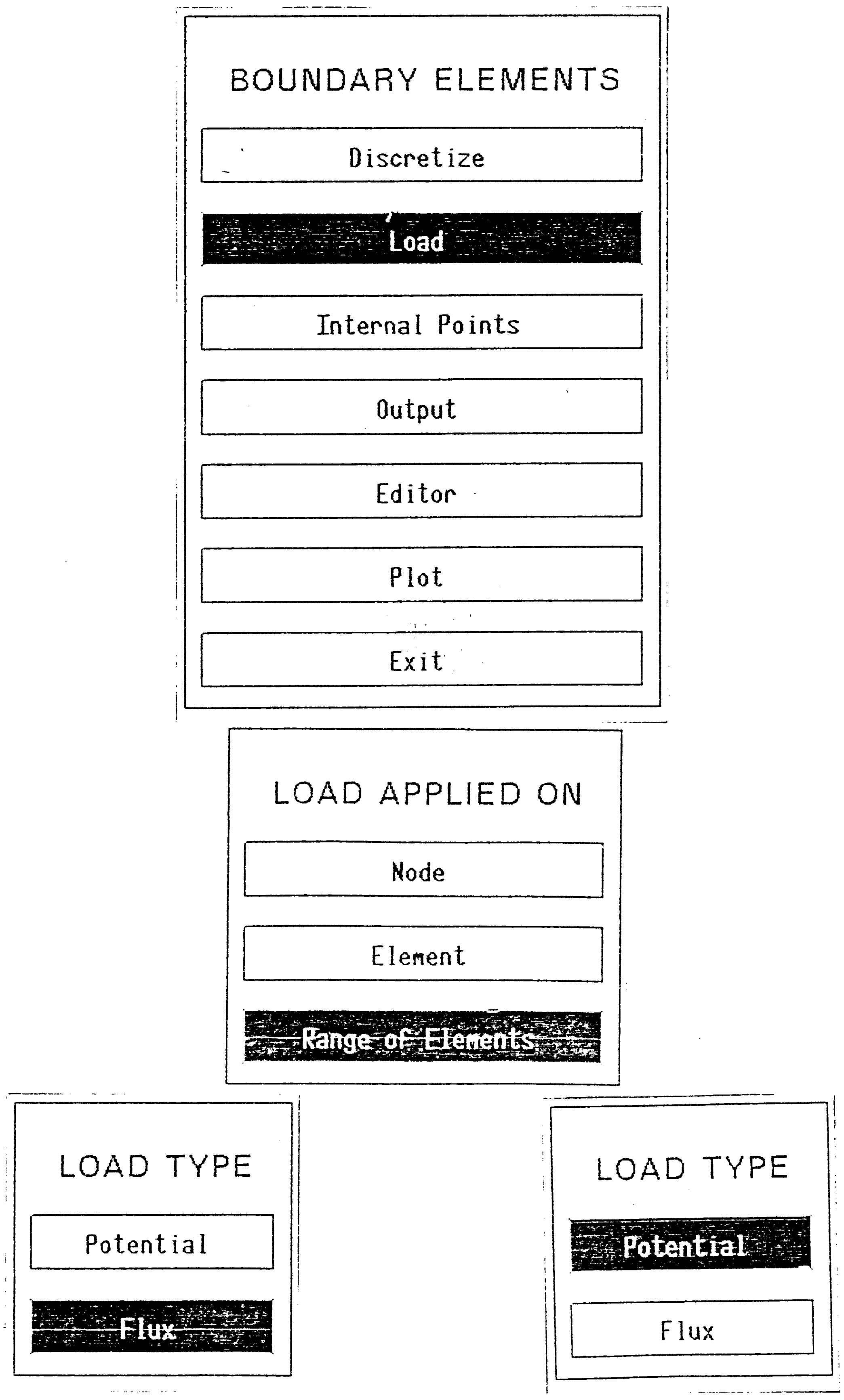

PLOTTING OPTIONS
Segments
Geometric Dots
Elements
Nodes
Internal Points
Plot
Quit
CREATE GEOMETRY
Add New Segment
Output Current State
Editor
Plot Geometry
Exit


MESH GENERATOR
Create Geometry
Boundary Elements
Finite Elements
Exit
BOUNDARY ELEMENTS
Discretize
Load
Internal Points
Output
Editor
Plot
Exit
PROBLEM TYPE
Potential
Elasticity
ELEMENT EDGES
Continuous
Discontinuous
-EXPANSION-
Constant
Linear
Quadratic
DISCRETIZATION
Equal
Graded
Arbitrary

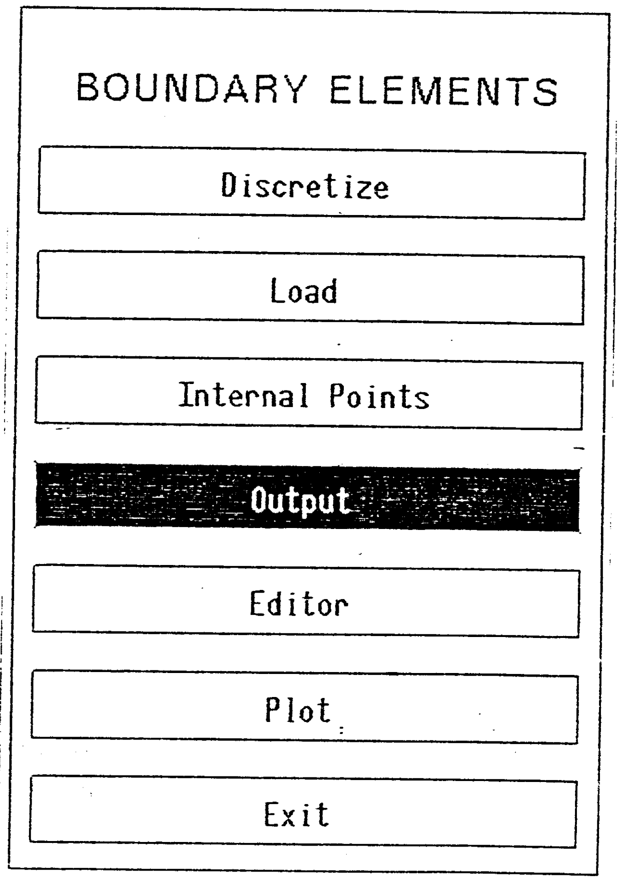
```
ENTER THE SEGMENT NUMBER :
1
YOU CAN DIVIDE YOUR SEGMENT INTO :
EQUAL SIZE ELEMENTS
GRADED SIZE ELEMENTS
ARBITRARY SIZE ELEMENTS

ENTER THE NUMBER OF ELEMENTS
5
ENTER THE TYPE OF EXPANSION DESIRED .
CONSTANT DISPLACEMENT EXPANSION
OR LINEAR DISPLACEMENT EXPANSION
OR QUADRATIC DISPLACEMENT EXPANSION

ENTER THE SEGMENT NUMBER :
2
YOU CAN DIVIDE YOUR SEGMENT INTO :
EQUAL SIZE ELEMENTS
GRADED SIZE ELEMENTS
ARBITRARY SIZE ELEMENTS


ENTER THE NUMBER OF ELEMENTS
5
ENTER THE TYPE OF EXPANSION DESIRED .
CONSTANT DISPLACEMENT EXPANSION
OR LINEAR DISPLACEMENT EXPANSION
OR QUADRATIC DISPLACEMENT EXPANSION

ENTER THE SEGMENT NUMBER :
3
YOU CAN DIVIDE YOUR SEGMENT INTO :
EQUAL SIZE ELEMENTS
GRADED SIZE ELEMENTS
ARBITRARY SIZE ELEMENTS

ENTER THE NUMBER OF ELEMENTS
5
ENTER THE TYPE OF EXPANSION DESIRED .
CONSTANT DISPLACEMENT EXPANSION
OR LINEAR DISPLACEMENT EXPANSION
OR QUADRATIC DISPLACEMENT EXPANSION

ENTER THE SEGMENT NUMBER :
4
YOU CAN DIVIDE YOUR SEGMENT INTO :
EQUAL SIZE ELEMENTS
GRADED SIZE ELEMENTS
ARBITRARY SIZE ELEMENTS

ENTER THE NUMBER OF ELEMENTS
5
ENTER THE TYPE OF EXPANSION DESIRED .
CONSTANT DISPLACEMENT EXPANSION
OR LINEAR DISPLACEMENT EXPANSION
OR QUADRATIC DISPLACEMENT EXPANSION
```



```
HOW MANY INTERNAL POINTS DO YOU WANT?4

ENTER THE COORDINATES (X,Y) OF POINT: 1
1 2.5

ENTER THE COORDINATES (X,Y) OF POINT: 2
2 2.5

ENTER THE COORDINATES (X,Y) OF POINT: 3
3 2.5

ENTER THE COORDINATES (X,Y) OF POINT: 4
4 2.5
```


BOUNDARY ELEMENTS
Discretize
Load
Internal Points
Output
Editor
Plot
Exit
LOAD APPLIED ON
Node
Element
Range of Elements
LOAD TYPE
Potential
Flux
LOAD TYPE
Potential
Flux

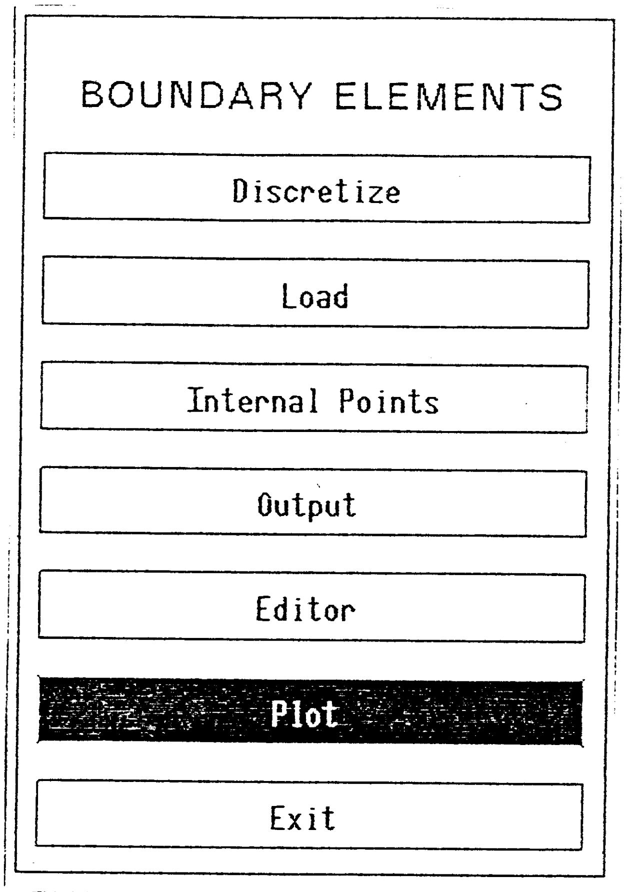
```
ENTER THE FIRST ELEMENT NUMBER:1

ENTER THE SECOND ELEMENT NUMBER:5

ENTER THE FLUX VALUE FROM ELEMENT 1 TO ELEMENT5 :0.0


ENTER THE FIRST ELEMENT NUMBER:6

ENTER THE SECOND ELEMENT NUMBER:10

ENTER THE POTENTIAL VALUE FROM ELEMENT6 :TO ELEMENT10 :0.0


ENTER THE FIRST ELEMENT NUMBER:11

ENTER THE SECOND ELEMENT NUMBER:15

ENTER THE FLUX VALUE FROM ELEMENT 11 TO ELEMENT15 :0.0


ENTER THE FIRST ELEMENT NUMBER:16

ENTER THE SECOND ELEMENT NUMBER:20

ENTER THE POTENTIAL VALUE FROM ELEMENT16 :TO ELEMENT20 :500.0
```

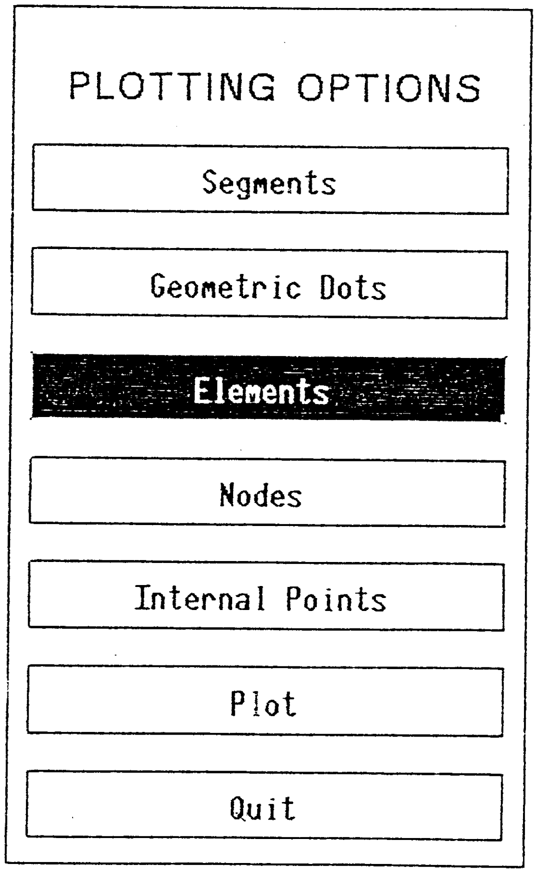


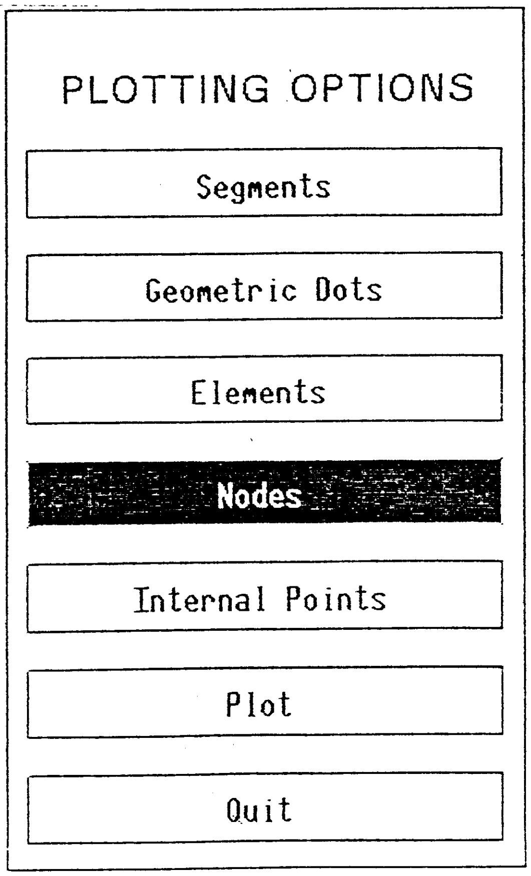
```
NUMBER OF NODES=20

NODE       COORDINATES          B.C.'S

 No.     X          Y        B.C. 1  KIND
----  -------    -------    -------  ----

  1     0.50      0.00        0.00    Q
  2     1.50      0.00        0.00    Q
  3     2.50      0.00        0.00    Q
  4     3.50      0.00        0.00    Q
  5     4.50      0.00        0.00    Q
  6     5.00      0.50        0.00    U
  7     5.00      1.50        0.00    U
  8     5.00      2.50        0.00    U
  9     5.00      3.50        0.00    U
 10     5.00      4.50        0.00    U
 11     4.50      5.00        0.00    Q
 12     3.50      5.00        0.00    Q
```

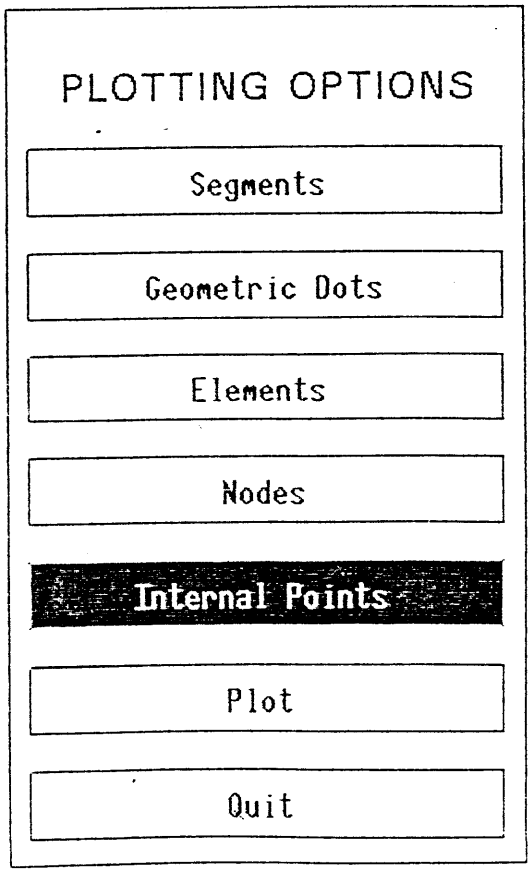
```
      NODE        COORDINATES          B.C.'S

       No.      X          Y         B.C. 1   KIND
      ----   -------    -------     -------   ----


       13     2.50       5.00         0.00     Q
       14     1.50       5.00         0.00     Q
       15     0.50       5.00         0.00     Q
       16     0.00       4.50       500.00     U
       17     0.00       3.50       500.00     U
       18     0.00       2.50       500.00     U
       19     0.00       1.50       500.00     U
       20     0.00       0.50       500.00     U




NUMBER OF ELEMENTS=20


LIST OF ELEMENTS

ELEM.   TYPE   EXPANSION  NODE  NODE  NODE  LENGTH
 No.                        1     2     3
-----   -----  ---------  ----  ----  ----  ------
  1       S        C        1   CONS.  CONS.    1.00
  2       S        C        2   CONS.  CONS.    1.00
  3       S        C        3   CONS.  CONS.    1.00
  4       S        C        4   CONS.  CONS.    1.00
  5       S        C        5   CONS.  CONS.    1.00
  6       S        C        6   CONS.  CONS.    1.00
  7       S        C        7   CONS.  CONS.    1.00
  8       S        C        8   CONS.  CONS.    1.00
  9       S        C        9   CONS.  CONS.    1.00
 10       S        C       10   CONS.  CONS.    1.00
 11       S        C       11   CONS.  CONS.    1.00
 12       S        C       12   CONS.  CONS.    1.00
 13       S        C       13   CONS.  CONS.    1.00
 14       S        C       14   CONS.  CONS.    1.00
 15       S        C       15   CONS.  CONS.    1.00
 16       S        C       16   CONS.  CONS.    1.00
 17       S        C       17   CONS.  CONS.    1.00
 18       S        C       18   CONS.  CONS.    1.00
 19       S        C       19   CONS.  CONS.    1.00
 20       S        C       20   CONS.  CONS.    1.00

DO YOU ALSO WANT TO OUTPUT ELEMENT ENDS (Y/N)?

y
```

```
ELEMENT ENDS ARE:

ELEM.        START                  END
No.      X1        Y1        X2        Y2
-----  --------  --------  --------  --------
  1      0.00      0.00      1.00      0.00
  2      1.00      0.00      2.00      0.00
  3      2.00      0.00      3.00      0.00
  4      3.00      0.00      4.00      0.00
  5      4.00      0.00      5.00      0.00
  6      5.00      0.00      5.00      1.00
  7      5.00      1.00      5.00      2.00
  8      5.00      2.00      5.00      3.00
  9      5.00      3.00      5.00      4.00
 10      5.00      4.00      5.00      5.00
 11      5.00      5.00      4.00      5.00
 12      4.00      5.00      3.00      5.00
 13      3.00      5.00      2.00      5.00
 14      2.00      5.00      1.00      5.00
 15      1.00      5.00      0.00      5.00
 16      0.00      5.00      0.00      4.00
 17      0.00      4.00      0.00      3.00
 18      0.00      3.00      0.00      2.00
 19      0.00      2.00      0.00      1.00
 20      0.00      1.00      0.00      0.00


NUMBER OF INTERNAL POINTS=4


POINT     COORDINATES

 No.      X          Y
----  -------    -------

  1     1.00       2.50
  2     2.00       2.50
  3     3.00       2.50
  4     4.00       2.50
```

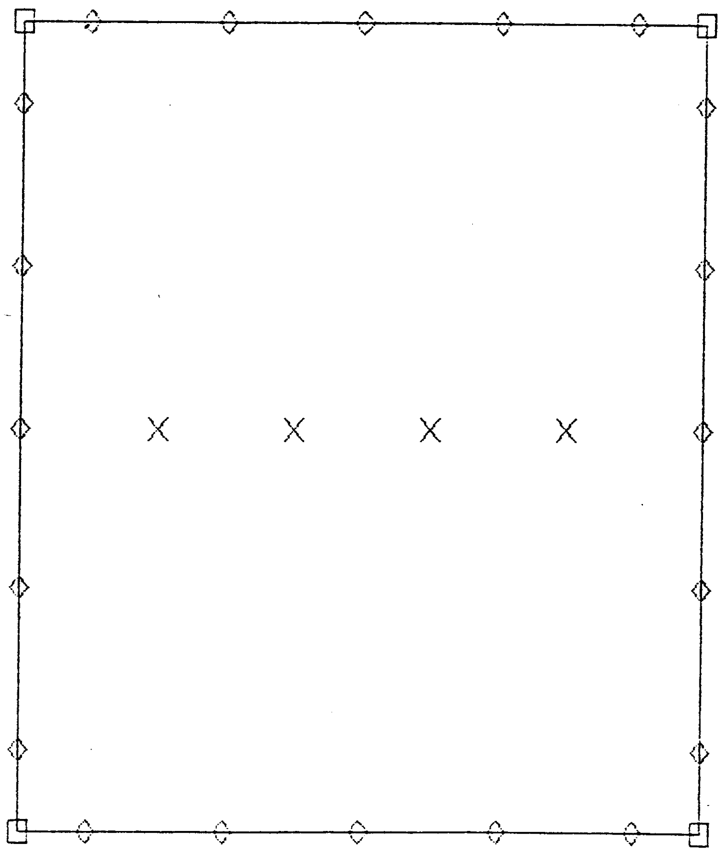

BOUNDARY ELEMENTS
Discretize
Load
Internal Points
Output
Editor
Plot
Exit

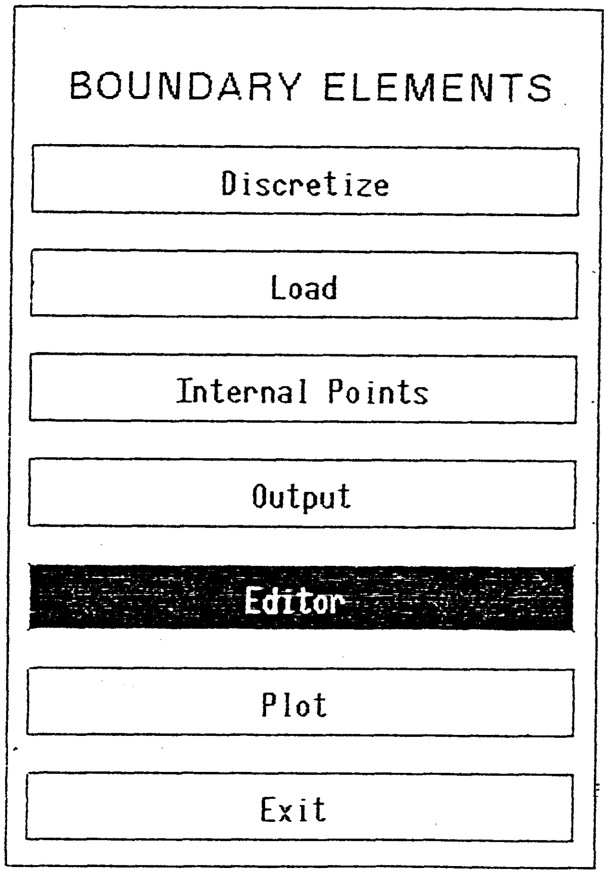

PLOTTING OPTIONS
Segments
Geometric Dots
Elements
Nodes
Internal Points
Plot
Quit

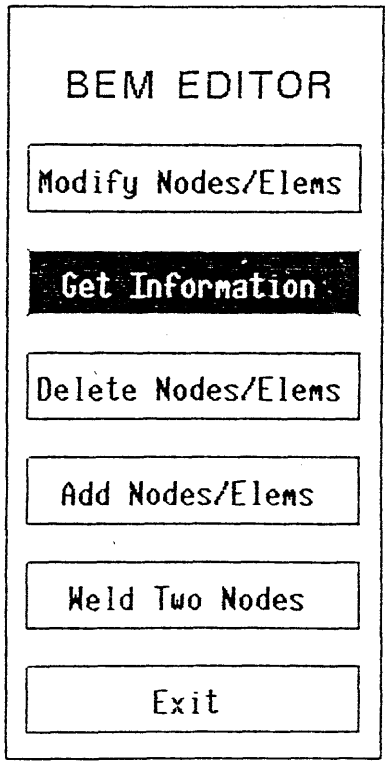

PLOTTING OPTIONS
Segments
Geometric Dots
Elements
Nodes
Internal Points
Plot
Quit

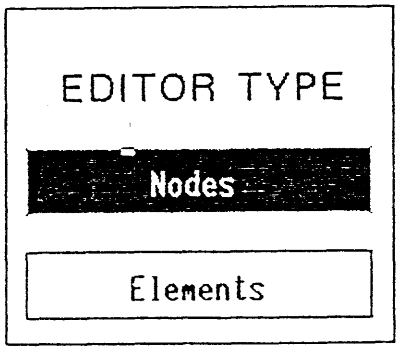

PLOTTING OPTIONS
Segments
Geometric Dots
Elements
Nodes
Internal Points
Plot
Quit

```
PLOT OPTIONS:

  INT_PTS     DOTS     SEGMENTS     NODES     ELEMENTS
    OFF        ON         ON         OFF        OFF

PLOT OPTIONS:

  INT_PTS     DOTS     SEGMENTS     NODES     ELEMENTS
    OFF        ON         ON         OFF        ON

PLOT OPTIONS:

  INT_PTS     DOTS     SEGMENTS     NODES     ELEMENTS
    OFF        ON         ON         ON         ON

PLOT OPTIONS:

  INT_PTS     DOTS     SEGMENTS     NODES     ELEMENTS
    ON         ON         ON         ON         ON
```



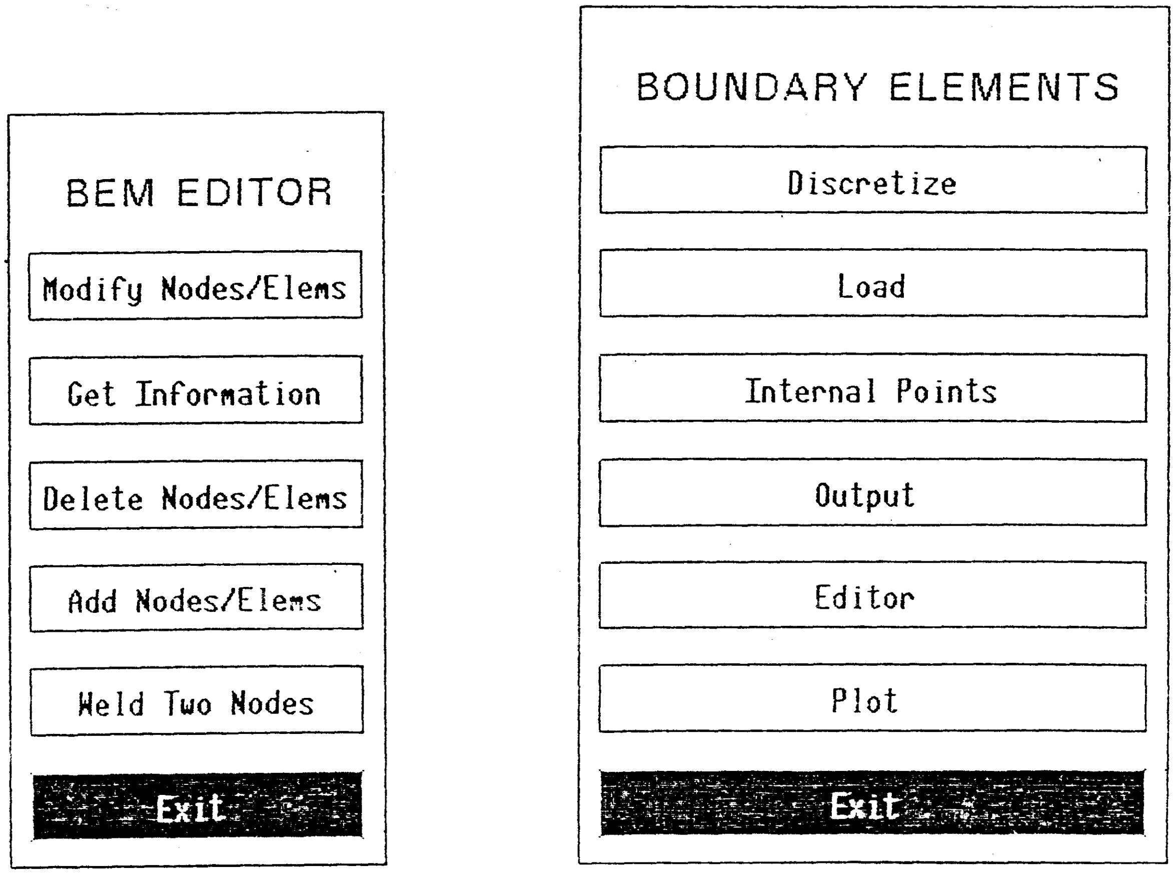


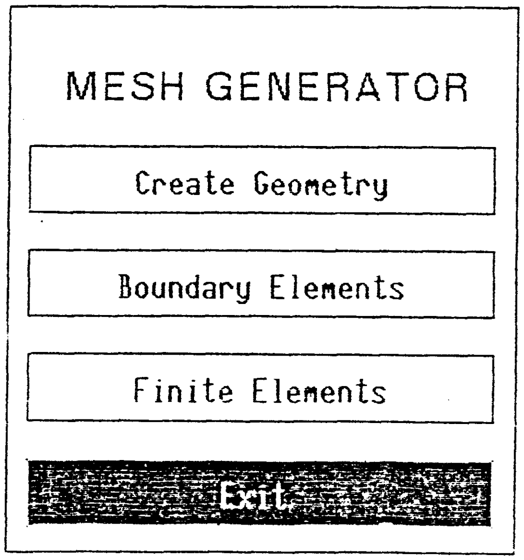


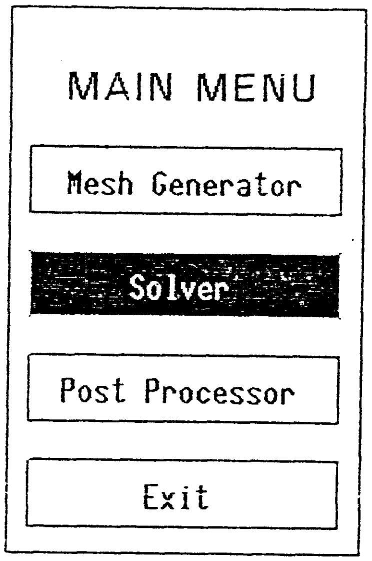


```
ENTER NODE NUMBER : 16

GET ONE OF THE FOLLOWING : NUMBER, COORD, TYPE_OF_BC, VALUE_OF_BC
GET NODE 16 >

type_of_bc


CURRENT TYPE OF BC IS POTENTIAL
```

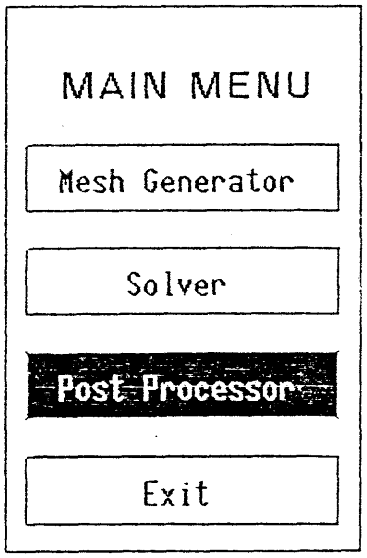
```
ENTER NODE NUMBER : 16

GET ONE OF THE FOLLOWING : NUMBER, COORD, TYPE_OF_BC, VALUE_OF_BC
GET NODE 16 >

value_of_bc


CURRENT VALUE OF BC = 500.000000
```

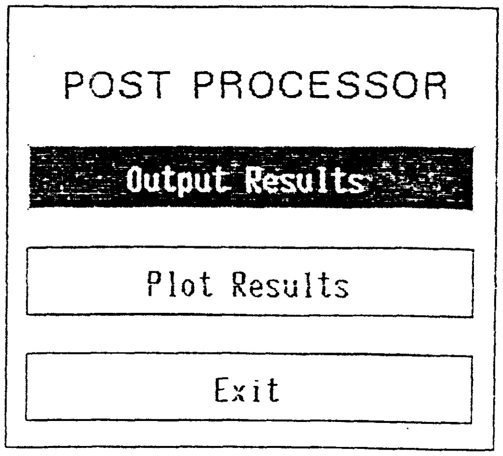


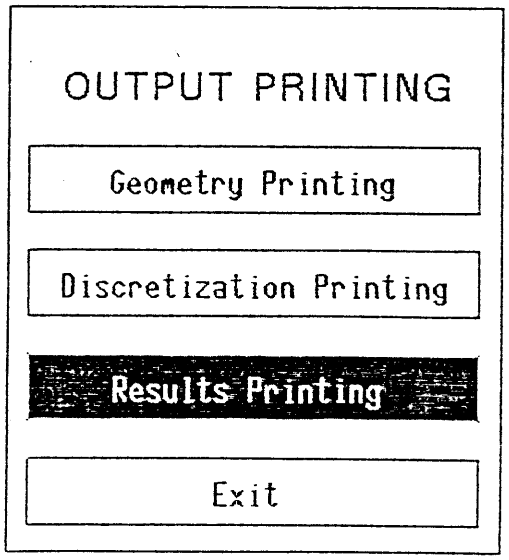




```
I AM SOLVING: PLEASE WAIT !
TRANSLATEI: Number of nodes = 20


I AM NEARLY DONE: PLEASE WAIT !

SOLUTION PHASE COMPLETED SUCCESSFULLY !!
```

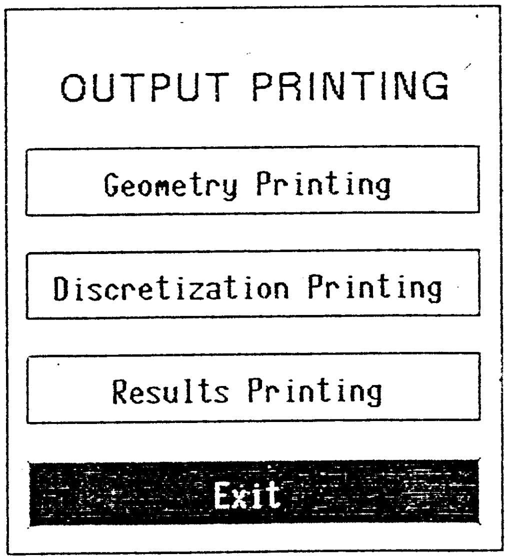


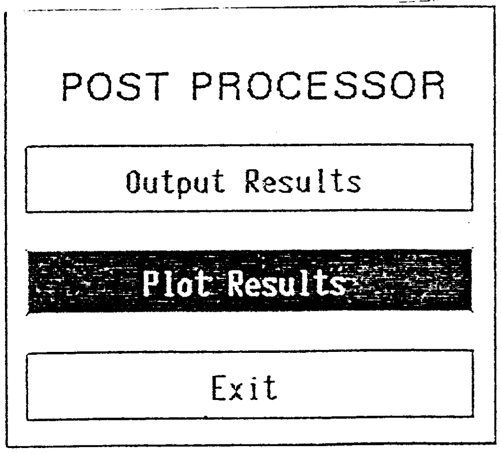


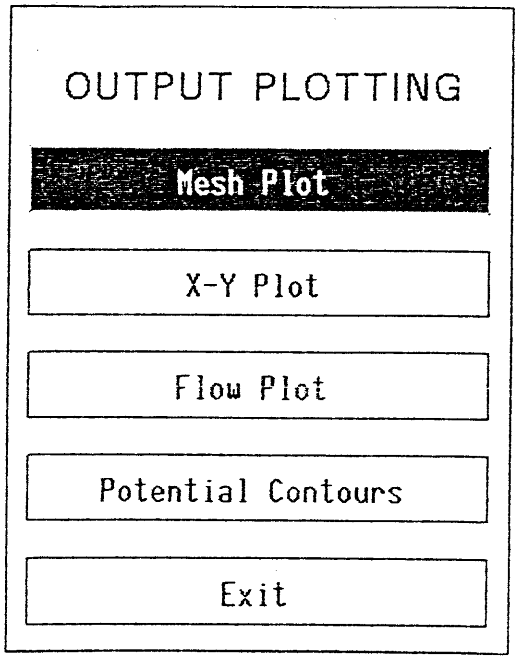


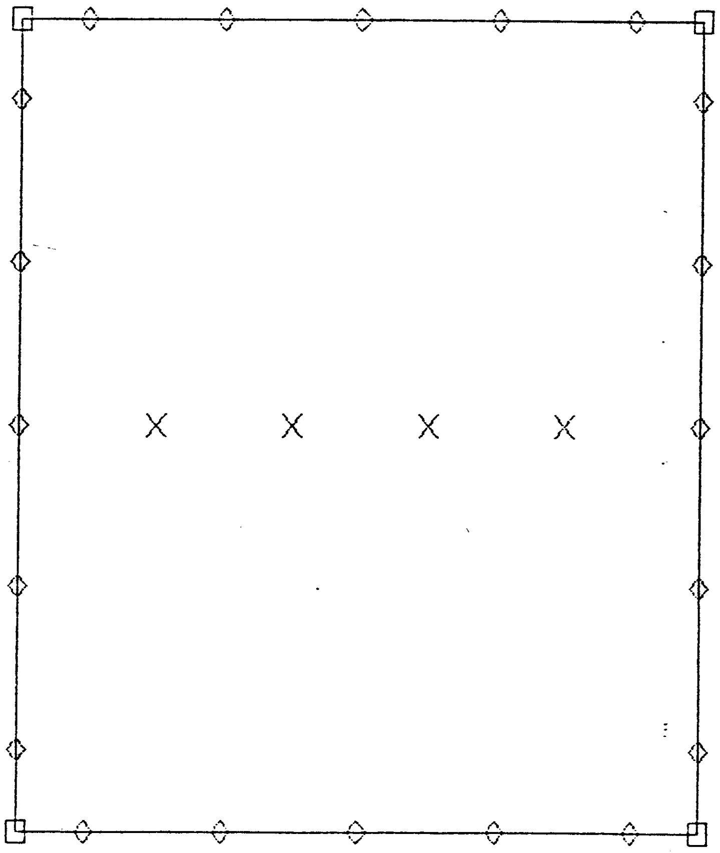
```
RESULTS AT THE NODES ARE :

NUMBER OF NODES=20


NODE       COORDINATES          RESULTS

 No.     X          Y        POTENTIAL     FLUX
----  -------    -------    -----------  ----------

  1    0.50       0.00      452.59          0.00
  2    1.50       0.00      351.10          0.00
  3    2.50       0.00      250.02          0.00
  4    3.50       0.00      148.95          0.00
  5    4.50       0.00       47.40          0.00
  6    5.00       0.50        0.00       -105.33
  7    5.00       1.50        0.00        -98.39
  8    5.00       2.50        0.00        -99.50
  9    5.00       3.50        0.00        -98.39
 10    5.00       4.50        0.00       -105.33
 11    4.50       5.00       47.40          0.00
 12    3.50       5.00      148.95          0.00
 13    2.50       5.00      250.02          0.00
 14    1.50       5.00      351.10          0.00
 15    0.50       5.00      452.59          0.00
 16    0.00       4.50      500.00        105.39
 17    0.00       3.50      500.00         98.33
 18    0.00       2.50      500.00         99.47
 19    0.00       1.50      500.00         98.33
 20    0.00       0.50      500.00        105.39
```

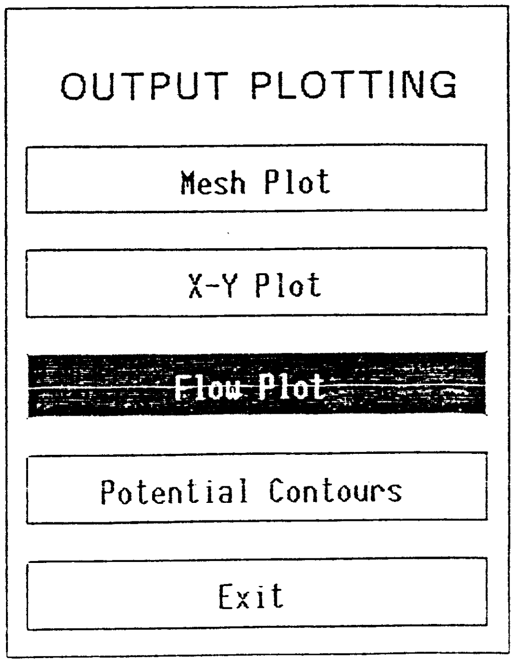
```
RESULTS AT INTERNAL POINTS FOLLOW :

NUMBER OF INTERNAL POINTS=4

POINT     COORDINATES       RESULTS

 No.      X          Y       POTENTIAL
----  -------   -------   -----------

  1     1.00      2.50     400.17
  2     2.00      2.50     300.11
  3     3.00      2.50     199.91
  4     4.00      2.50      99.84
```

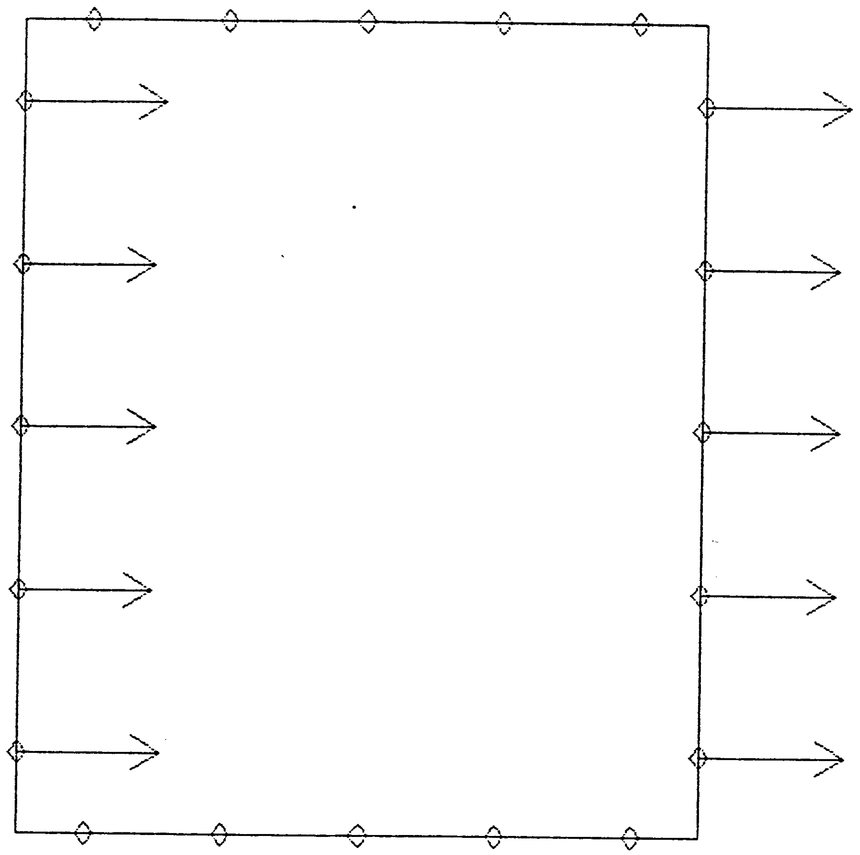


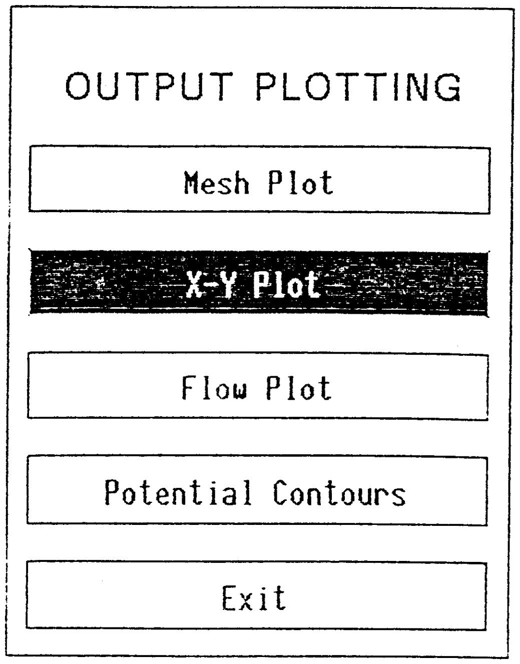


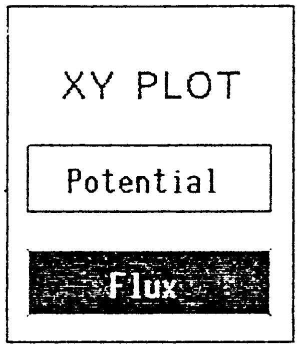


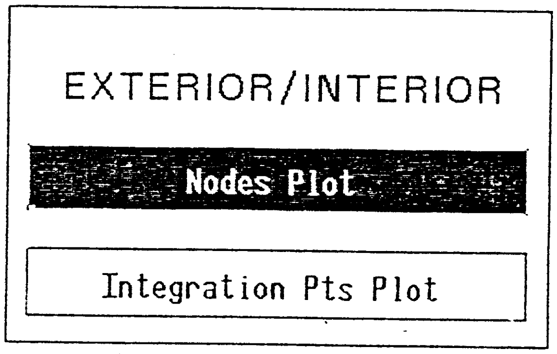


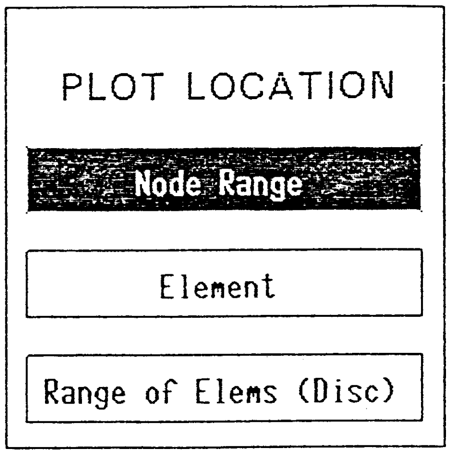

OUTPUT PLOTTING
Mesh Plot
X-Y Plot
Flow Plot
Potential Contours
Exit



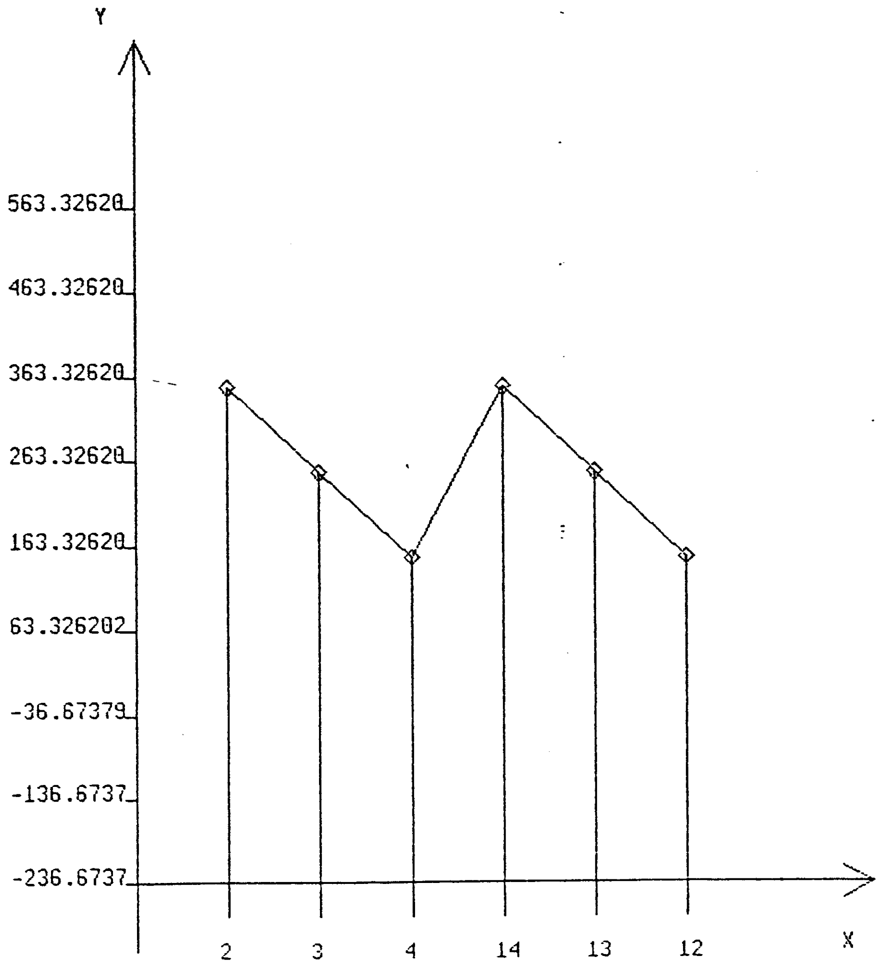

OUTPUT PLOTTING
Mesh Plot
X-Y Plot
Flow Plot
Potential Contours
Exit





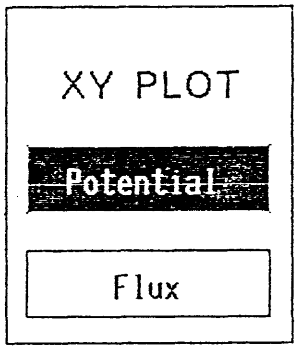


```
SEPARATE NODE NUMBERS BY COMMAS OR SPACES
TYPE END TO TERMINATE INPUT
AS AN EXAMPLE : 1 2,3,8 14,20 END IS O.K.
ENTER YOUR RANGE :  2 3 4 14 13 12 END


ENTER X-AXIS LABEL:  NODES NUMBER


ENTER Y-AXIS LABEL: FLUX VALUES
```



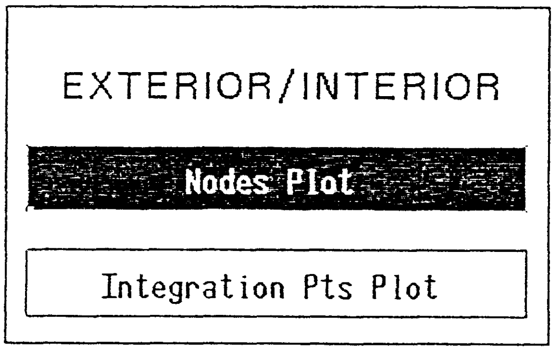
X-AXIS: NODE NUMBERS

Y-AXIS: POTENTIAL VALUES


XY PLOT

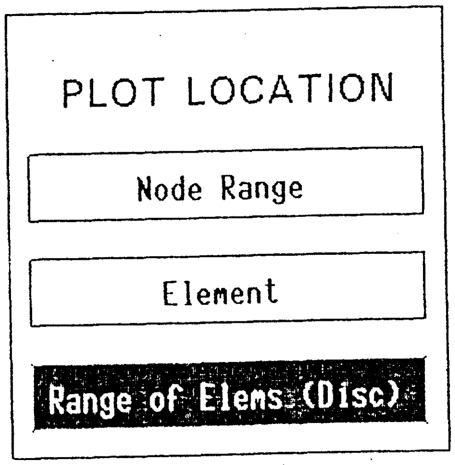
[✓ Potential]

Flux


EXTERIOR/INTERIOR

[✓ Nodes Plot]

Integration Pts Plot

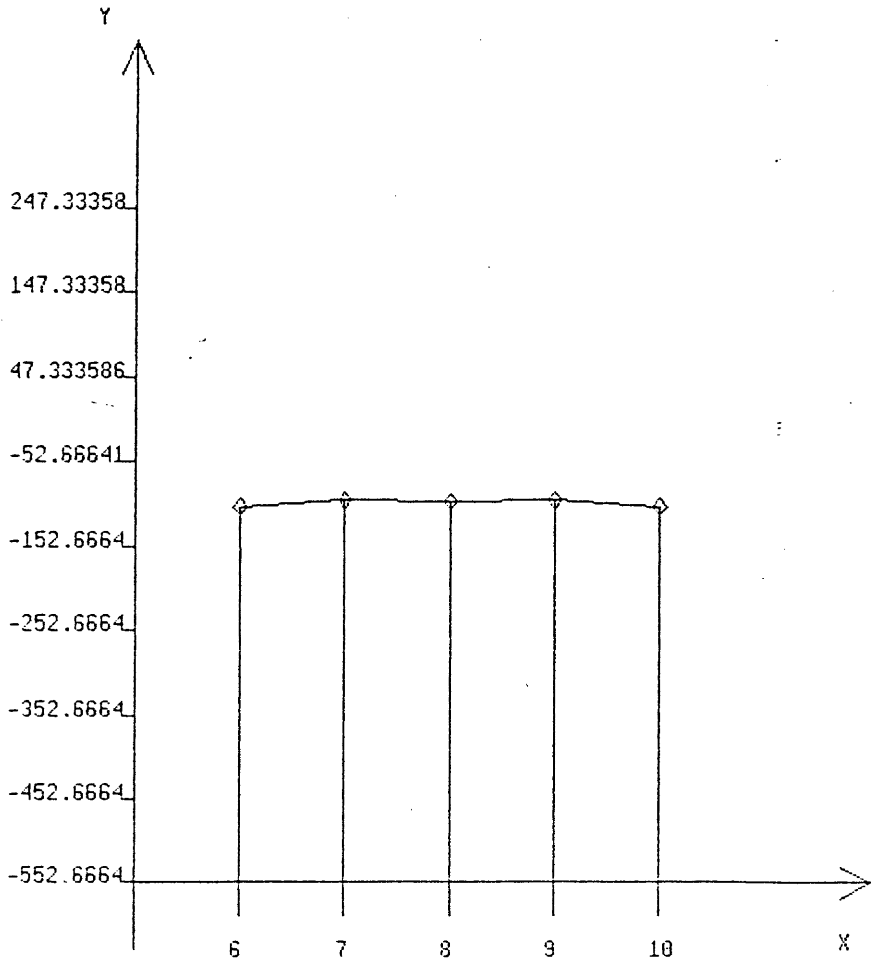

PLOT LOCATION

Node Range

Element

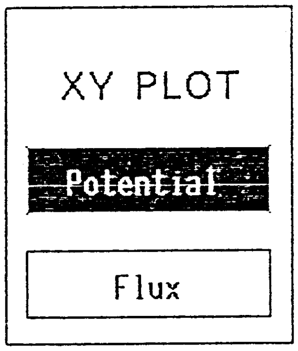
[✓ Range of Elems (Disc)]

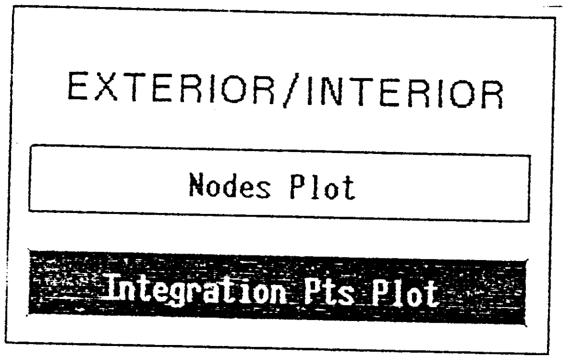
```
ENTER THE FIRST ELEMENT NUMBER: 6

ENTER THE SECOND ELEMENT NUMBER:10


ENTER X-AXIS LABEL: NODE NUMBERS


ENTER Y-AXIS LABEL: POTENTIAL VALUES
```

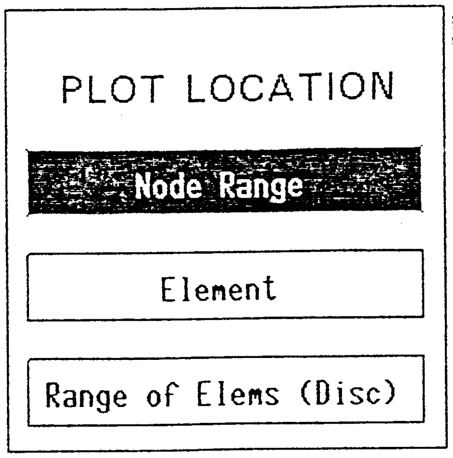


X-AXIS: NODES NUMBER

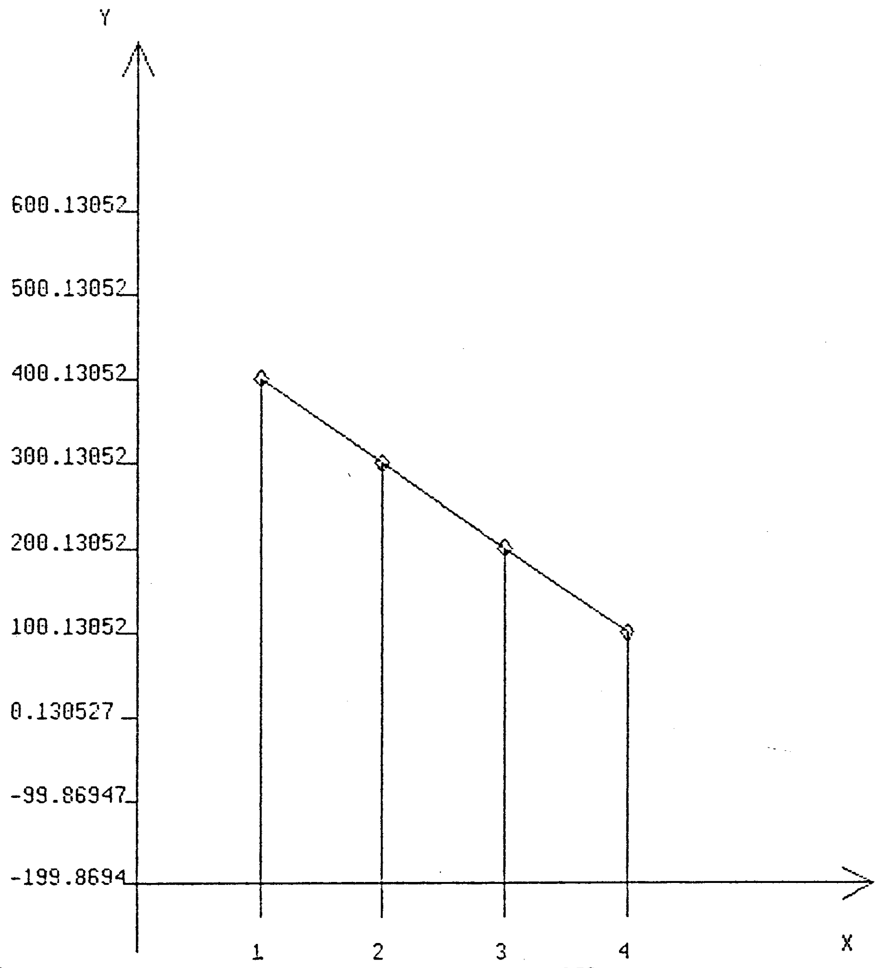
Y-AXIS: FLUX VALUES





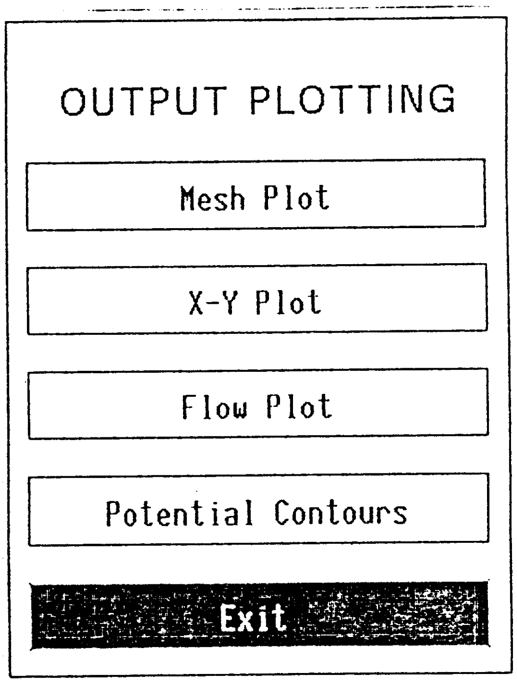


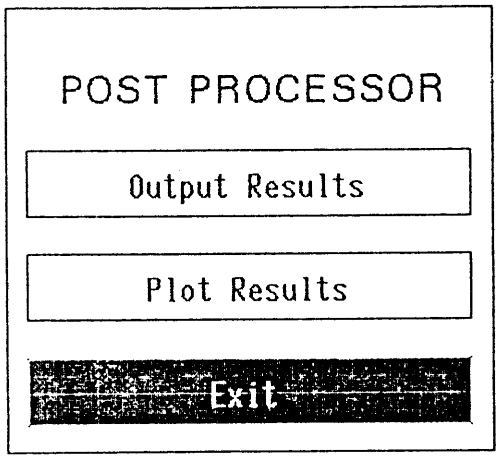
```
YOU CAN SPECIFY ANY INTERIOR POINT RANGE
SEPARATE NUMBERS BY COMMAS OR SPACES
TYPE END TO TERMINATE INPUT
AS AN EXAMPLE : 1 3,8 14 20,30 END IS O.K.
ENTER YOUR RANGE :  1 2 3 4 END


ENTER X-AXIS LABEL: INTERNAL POINTS NUMBER


ENTER Y-AXIS LABEL: POTENTIAL VALUE
```

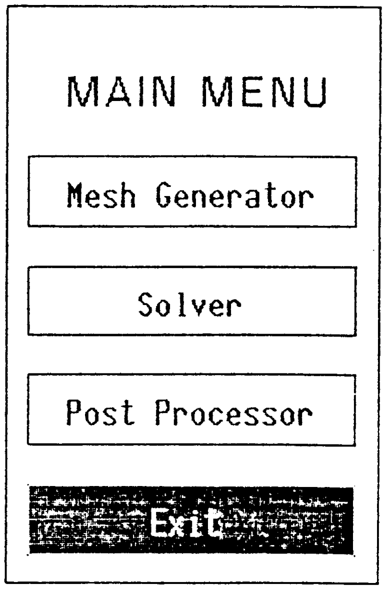


X-AXIS: INTERNAL POINTS NUMBER

Y-AXIS: POTENTIAL VALUE


OUTPUT PLOTTING
Mesh Plot
X-Y Plot
Flow Plot
Potential Contours
Exit


POST PROCESSOR
Output Results
Plot Results
Exit


MAIN MENU
Mesh Generator
Solver
Post Processor
Exit

APPENDIX E

PRESENTATION OF ACCURACY ADVISOR IMPLEMENTED ON GARFIELD

This appendix contains the general objects and rules for the discretization advisor implemented on GARFIELD. Particular rules for a special problem are also shown.

```
 (create geometry)
(add shape geometry description)
(add corners geometry number)

 (create rectangle)
(add width rectangle value)
(add depth rectangle value)

 (create dot)
(add corner dot yes_no)
(add segment_left dot segment)
(add segment_right dot segment)
(add number dot value)

 (create problem)
(add run problem number)

 (create segment)
(add boundary_condition segment kind)
(add value_bc segment value)
```

```
RULE "Recognize rectangle"

IF  (and (corners geometry 4)
         (FOR_ALL ?x
              (or (is? corner ?x no)
                  (and (is? corner ?x yes)
                       (perpendicular (get segment_left ?x)
                                      (get segment_right ?x))))))

THEN (shape geometry rectangle)




RULE " 10 < width/depth < 20 "

IF  (and (shape geometry rectangle)
         (>= (/ (get width (get shape geometry))
                (get depth (get shape geometry)))
             10)
         (<  (/ (get width (get shape geometry))
                (get depth (get shape geometry)))
             20))

THEN (and (print "Divide your rectangle into ")
          (print "\n two surfaces for your ")
          (print "\n   discretization !!"))


RULE   " width/depth > 20 "

IF  (and (shape geometry rectangle)
         (>= (/ (get width (get shape geometry))
                          (get depth (get shape geometry)))
             20))

THEN (and (print "Divide your rectangle into")
          (print "\n different surfaces with")
          (print "\n width/depth ratios")
          (print "\n smaller than 10 !!"))



RULE  "First run"

IF  (is? run problem 1)

THEN  (and (print " Do not use a fine mesh ")
           (print "\n for the first run. Try a")
           (print "\n coarse one first to coarse")
           (print "\n a feel for the results. "))
```

```
RULE "corner node"

IF  (corner ?x yes)

THEN   (and (print " Try not to locate nodes at ")
            (print "\n corner #")
            (print (get number ?x))
            (print "\n use discontinuous elements"))



RULE  "Singular point"

IF  (and (is? boundary_condition (get segment_left ?x) potential)
         (is? boundary_condition (get segment_right ?x) potential)
         (!= (get value_bc (get segment_left ?x))
             (get value_bc (get segment_right ?x))))

THEN   (and (print "Do not locate nodes at dot #")
            (print (get number ?x)))
```

```
(create dot_1)

(create dot_2)

(create dot_3)

(create dot_4)

(create dot_5)

(corner dot_1 yes)

(corner dot_2 yes)

(corner dot_3 yes)

(corner dot_4 no)

(corner dot_5 yes)

(number dot_1 1)

(number dot_2 2)

(number dot_3 3)

(number dot_4 4)

(number dot_5 5)

(run problem 1)

(corners geometry 4)

(width rectangle 15)

(depth rectangle 1)

(create segment_1)

(create segment_2)

(segment_left dot_4 segment_1)

(segment_right dot_4 segment_2)

(boundary_condition segment_1 potential)

(boundary_condition segment_2 potential)

(value_bc segment_1 4)

(value_bc segment_2 4)
```