Iowa State University
Digital Repository

Retrospective Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

1989

A two-dimensional boundary element groundwater model

Mark Allen Liebe
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Civil Engineering Commons

Recommended Citation
Liebe, Mark Allen, "A two-dimensional boundary element groundwater model " (1989). *Retrospective Theses and Dissertations*. 9216.
https://lib.dr.iastate.edu/rtd/9216

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

المنارة للاستشارات

www.manaraa.com

INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 8920161

A two-dimensional boundary element groundwater model

Liebe, Mark Allen, Ph.D.

Iowa State University, 1989

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A two-dimensional boundary element groundwater model

by

Mark Allen Liebe

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Department: Civil Engineering
Major: Water Resources

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1989

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Several techniques exist for the numerical modeling of groundwater flow. Both the Finite Difference Method (FDM) and the Finite Element Method (FEM) have been in use for many years and have gained wide acceptance. A newer modeling technique has of late received much attention due to several marked advantages. It is called the Boundary Element Method (BEM).

As micro-computers have become more affordable and accessible, their use for the solution of groundwater problems has become commonplace. One problem, however, has been the size or detail of the model which these smaller computers have been able to successfully analyze, particularly when the FDM or FEM are implemented. The BEM lends itself particularly well to use on small computer systems. This is due to the way in which the BEM can represent a particular groundwater problem and subsequently solve it. Because of this, the BEM is potentially capable of solving much larger and more complex groundwater systems using micro-computers when compared to either the FDM or FEM.

The founding theory behind the Boundary Element Method is relatively simple. There are certain aspects of the implementation of the method in a computer program, however, which become somewhat difficult. This dissertation shall point out some of these problems and clarify to the reader the methods the author used in developing a general purpose, two-dimensional groundwater model using the BEM, called GWBEM. The emphasis shall be to elucidate aspects of both the theory and actual programming of the method as well as present certain improvements in its implementation.

Modularity of GWBEM was important to ensure ease of understanding and future modifications. A widely accepted language which was block oriented and easily read was needed. Because of this, Pascal was chosen as the developmental language. It is a fairly transportable language from one machine to another.

The types of groundwater problems which GWBEM is capable of solving are two-dimensional, steady state, non-homogeneous domains with sources and sinks. A non-homogeneous domain for the purpose of this model consists of connected multiple zones, with each zone being homogeneous but not necessarily of the same conductive properties as adjacent zones. Sources and sinks may be defined in any zone, and may be designated as either specified potential or flow rate. Special flow situations such as cutoff walls and corner flow are also accommodated by the model. Unknown potential or flux values are calculated at all boundary points, and at selected interior points as well.

GWBEM was tested using known analytical solutions to various groundwater conditions and is also compared to results obtained by other researchers using various analytical techniques.

LITERATURE REVIEW

Through the use of numerical methods, researchers have been able to develop highly sophisticated models which simulate physical systems and allow for the solution of problems which are difficult or impossible to solve analytically. By using and validating these models, a greater understanding of a given physical system may be realized. Aided by the computer, these models have, with time, become larger and more complex and have thus been able to more completely incorporate finer details of the systems involved. Of great interest to those involved in water resources management has been the simulation of various groundwater scenarios as well as the inverse problem of defining aquifer properties from observable field data. Groundwater researchers have used various methods of formulating such problems to solve them numerically. The formulation methods used to solve these complex problems have evolved rapidly in the last three decades.

Numerical Methods

Throughout the 1960s, the method of choice for solving groundwater problems on computers was the finite-difference method (FDM). During the 1970s, popularity shifted from the FDM to the finite-element method (FEM). The FEM had several advantages over the FDM. First, boundary conditions were easier to apply using the FEM, and as such, universal computer codes could be written which could be used in most types of groundwater situations. Secondly, the actual geometry of a problem could be used with the FEM, whereas it is oftentimes "altered" to allow the use of the FDM (Liggett and Liu, 1983). Carr (1985) also points out the relative ease of use of the FEM over the FDM in three dimensional problems. Like the FDM, the FEM was a domain based formulation where the physical problem space was represented by a collection of nodes or elements (Pinder and Gray, 1977).

Despite these advantages of the FEM over the FDM, other methods were still pursued. At about the same time as the initial development of the FEM, another formulation called the boundary element method (BEM) started to emerge during the early 1970s. Despite its concurrent development with the FEM, the BEM's initial applications were somewhat limited. But, as its advantages over the FEM were eventually realized, its use spread. Today, it is used to solve problems of a wide variety, ranging from structural analysis to predicting wave action through off-shore drilling platforms (Brebbia, 1985).

The BEM's advantages over the FEM's are numerous. Brebbia (1985) and Liggett and Liu (1983) discuss several. First, most problems are reduced by one dimension when solved using the BEM. This is particularly advantageous when dealing with problems in three dimensions. The amount of data preparation required for the BEM is considerably less than with the FEM. Less data preparation means less errors in coding. The inter-element continuity requirements are also much less stringent with the BEM, which allows for more abrupt changes in element size over different areas of a model domain when compared to the FDM or FEM.

The location of internal points which require a solution may be specified using the BEM with those points being the only internal ones required. The FEM, by contrast, calculates solutions at all interior points which are required for grid definition, which results in unnecessary effort. The interior points are required of the FEM for proper solution, whereas the interior points specified in the BEM are only at points of interest. Problems of infinite domain can be accurately solved using the BEM using special elements. In contrast, the FEM requires the mesh to be truncated at some finite boundary. Finally, since the BEM reduces the dimension of a problem by one, the number of equations which must be solved is also reduced. This can result in substantial savings in both computer storage and run times, making the BEM highly desirable for micro-computer applications.

Development of the Boundary Element Method

The development of the BEM as applied to groundwater problems apparently originated in two camps. The initial groundwork was laid out by Kellogg (1929) who used the integral equation method for the analytical solution of Laplace type problems. The first to propose a numerical solution to problems using the BEM was Trefftz in 1926 (Brebbia and Chang, 1985). Unfortunately, his method was hindered by the lack of computers during his time. In Western Europe, groups of researchers started to explore the possibilities of using the BEM in the early 1970s. Brebbia (1978) published the first general text on the use of the BEM for engineers. Although the text was not limited to problems dealing with flow through porous media, it did contain discussions on LaPlace problems. In the United States, Liggett (1977) published a paper on determining the location of the free surface in porous media. This was probably the first paper which specifically addressed a groundwater problem using the BEM. Since that time, a plethora of research has been done on the BEM, with annual conferences being held (Brebbia, 1984). Banerjee (1979, 1982, and 1984) has been the principal editor of a number of volumes dealing with the latest developments in the BEM and its applications.

Applications to groundwater

The BEM's initial use in groundwater problems was somewhat limited as it was only able to solve steady-state problems with isotropic media. However, recent developments in the BEM have occurred which significantly broaden the scope of groundwater problems which may be solved. Banerjee et al. (1981) discussed the use the BEM for two dimensional problems with transient groundwater flow. Cheng (1984a) developed a method for calculating Darcy's flow with spatially varied permeability using the boundary integral equation method. His paper gave examples of Green's functions based on certain permeability distributions which could be used to fit field data and which would lead to the BEM solution. However, he only provided functions for one and two dimensional problems.

The application of the BEM to seepage problems in zoned anisotropic soils was presented by Brebbia and Chang (1985). Their method broke down permeability into orthogonal tensors for homogeneous zones. When several different homogeneous zones were present within one problem domain, the method called for the formation of separate subregions having the same properties. Continuity and equilibrium were then maintained at each boundary between the subregions. If there were a large number of zones, it could be more advantageous to use the FEM in this situation, although the authors claim to get better accuracy using the BEM. As the number of zones increases, the BEM becomes more like the FEM, with the "mesh" becoming finer to define the problem domain.

The use of the BEM with non-linear conditions, as would be encountered in unsaturated groundwater flow, are still being examined by several researchers. Bialecki and Nowak (1981) wrote on the use of non-linear material and boundary conditions in heat conduction problems, while Brebbia and Skerget (1984) discussed the use of Kirchhoff's transform to linearize non-linear material properties. The transform can be applied to both steady-state and transient conditions when only Neumann (specified flux) and Dirichlet (specified potential) boundary conditions are used. Rubin (1968) provided a similar use of the Kirchoff transform in an application to transient flow in partially saturated soils.

Extensions of the BEM method for groundwater problems have also increased in the last few years. Tolikas et al. (1983) combined the BEM with non-linear programming techniques to manage and optimize the operation of an aquifer in Greece. They reported excellent results and great efficiency in the case of steady-state flows and homogeneous aquifers, but felt more work was required for transient problems and nonhomogeneous media. Kemblowski (1984) provided a BEM solution to simulate salt-water upconing under the Smokey Hill River in Kansas. The model predicts the free-surface and the interface between fresh and saline waters due to changing boundary conditions.

Dillon and Liggett (1983) developed an ephemeral stream-aquifer model based on the BEM. It is a two dimensional vertical slice model capable of simulating a stream-aquifer system when they are hydraulically connected or disconnected and any transition between the two states. The model was successfully calibrated using data from a South Australian aquifer system. Shapiro and Andersson (1985) formulated a method for simulating steady-state flow in three-dimensional fracture networks using the BEM. The model treated the host rock as impervious and the fractures as surfaces where fluid movement was two-dimensional. Fracture intersections were modelled as one-dimensional conduits. As opposed to other models dealing with transport through fractured media which consider average characteristics, Shapiro and Andersson's model considered discrete fractures. Although their model would be cumbersome to use in highly fractured rock, its application to simple fracture systems would be advantageous due to its numerical efficiency.

Existing Computer Models

An effort was made to determine the existence of any BEM based computer models. A computer search conducted in 1986 through the Holcomb Research Institute of Indianapolis, Indiana produces four computer groundwater models based on the BEM (Holcomb Research Institute, 1986). The institute maintains a data base of known computer models which deal specifically with groundwater. Since that initial search, a recent reference on the BEM (Mackerle and Brebbia, 1988) has been published. In it, some 135 computer models which use the BEM to solve various types of problems are listed. As evident from these figures, the use of the BEM has increased considerably in recent years. Of that 135 however, only eleven were targeted for micro-computer use and of that eleven, only three dealt specifically with potential problems. Of those three, none listed any capabilities for including interior sources or sinks, multiple zones, or the solution of flux at interior points. Only one of the models listed the capability of solving for interior potentials. What is evident here is that only a small percentage of the programs available for micro-computer use are tailored for groundwater flow problems, and that their capabilities are limited. Although many capable potential flow programs exist on mainframe computers, the existence of comparable BEM programs on micro-computers is lacking.

Conclusions

As can been seen, the use of the boundary element method for the solution of groundwater problems is well established. Its advantages make it well suited to solving complex groundwater systems with a minimum of input data generation and computational effort. Although many main frame computer models based on the method exist, a relatively small number with limited function are available for micro-computer use. The need for a micro-computer program which is easy to use yet capable of handling a broader assortment of groundwater problems exists.

A second need also becomes evident with a review of the BEM literature. Unlike its numerical predecessors, the FDM and FEM, scant literature exists for the BEM which deals with the actual programming techniques required to use it. This occurs in spite of the large number of publications dealing with the BEM. Many references present the theory, but moving from theory to actual application is often difficult. The need exists, therefore, for a general purpose groundwater model which makes use of the BEM, but which can also be used as a tool in illuminating various means of implementing the BEM on a micro-computer.

MODEL DEVELOPMENT

This discussion presents the development of the BEM for steady-state flow in a saturated porous medium. It starts with a review of the basic equations for groundwater flow. Following this will be an overview of the basic theories behind the boundary element method (BEM) for potential problems. Finally, the details of applying the BEM to groundwater problems are presented. This final section shall also discuss the actual implementation of the BEM using a micro-computer.

Basic Groundwater Equations

Darcy's law

Microscopically, the actual process of fluid flow through porous media is an extremely complex one. A fluid flowing in such a medium is forced through pores of varying diameter and connectivity. Because of this complexity, the actual determination of such flow usually requires that one look at the process macroscopically and ignore the microscopic details (Hillel, 1982).

During an investigation of fluid flow through sand filters, the French engineer Henri Darcy noted the following relationship:

$$v = - K \nabla\Phi \tag{1}$$

where

v	=	seepage velocity,
K	=	hydraulic conductivity,
∇	=	gradient operator,
Φ	=	potential function.

The potential function, Φ is defined as:

$$\Phi = \frac{p}{\rho g} + z. \tag{2}$$

where

p	=	pore water pressure,
ρ	=	fluid density,
z	=	elevation above some datum.

<u>Conservation of mass</u>

Conservation of mass for steady-state porous media flow necessitates that the rate of fluid flow into any saturated volume be the same as the flow rate out (Freeze and Cherry, 1979). By assuming that the compressibility of the medium and the fluid are relatively small the equation for the conservation of mass can be stated mathematically as:

$$\nabla \cdot v = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} = 0. \tag{3}$$

Substitution of Darcy's Equation 1 into Equation 3 yields the flow equation for anisotropic porous media flow:

$$\nabla \cdot (K \nabla \Phi) = \frac{\partial (K_x \frac{\partial \Phi}{\partial x})}{\partial x} + \frac{\partial (K_y \frac{\partial \Phi}{\partial y})}{\partial y} + \frac{\partial (K_z \frac{\partial \Phi}{\partial z})}{\partial z} = 0. \tag{4}$$

If the medium is isotropic, then $K_x = K_y = K_z$. Also, if the medium is homogeneous, then $K(x,y,z)$ = constant. Equation 4 then reduces to Laplace's equation, or:

$$\nabla^2\Phi = \frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2} + \frac{\partial^2\Phi}{\partial z^2} = 0. \tag{5}$$

Laplace's equation is valid for steady flow in either confined or unconfined aquifers. By the use of adequate boundary conditions, it can satisfactorily model many groundwater situations.

<u>Sources and sinks</u>

For large scale groundwater modeling, wells are often idealized as sources or sinks (Liggett and Liu, 1983). Fluid enters or leaves the medium from a point. To incorporate this, the equation of conservation of mass (Equation 3) is modified to include a source term:

$$\nabla\cdot v = -\sum_{k=1}^{N_s} Q_k \delta(x-x_k)\ \delta(y-y_k)\ \delta(z-z_k) \tag{6}$$

where

N_s	=	number of sources,
Q_k	=	flow rate at kth source (out = positive),
(x_k,y_k,z_k)	=	coordinate of kth source, and
$\delta(p)$	=	Dirac delta function, zero for $p\neq0$ and one for $p=0$.

<u>Inter-zonal compatibility</u>

Whenever two zones with differing hydraulic properties meet along a common boundary, certain conditions exist along that boundary. First, the potential Φ at any point along the boundary between two zones is the same for either zone. Second, continuity across the boundary is maintained. The flow $-K\partial\Phi/\partial n$ across a common boundary leaving one zone is the same as the flow entering the adjacent zone (Cheng, 1984b). Taking advantage of these relationships allows one to solve for boundary values on the interface which would otherwise be indeterminant.

Development of the Boundary Element Method

To apply the BEM to groundwater problems, two concepts must be introduced. They are Green's second identity and the concept of fundamental solution. This development follows that of Liggett and Liu (1983) and Brebbia (1978).

Green's second identity

Green's second identity is the primary foundation for the BEM. For an understanding of this identity, one must start with the divergence theorem. This theorem states that:

$$\int_{\Omega} (\nabla \cdot \mathbf{V}) \, d\Omega = \int_{\Gamma} \mathbf{V} \cdot \mathbf{n} \; d\Gamma \tag{7}$$

where

$\mathbf{V}$	=	some differentiable vector function,
Ω	=	domain of integration (volume in 3D, area in 2D),
$\mathbf{n}$	=	outward unit normal vector, and
Γ	=	domain boundary (area in 3D, line in 2D).

It should be noted that $\nabla \cdot \mathbf{V} = \text{div } \mathbf{V}$. Another way of looking at the divergence theorem is to consider some fixed volume within a porous media. Imagine that fluid is either entering or leaving this volume through its boundary and that the density of the fluid within that volume is changing accordingly. With this in mind, one can regard the left-hand side of Equation 7 as the rate of change in fluid density within the volume while the right-hand side is the amount of fluid mass per unit time which is passing through the boundary of the volume to effect that density change (Kaplan, 1984).

To derive Green's second identity from the divergence theorem, one defines V as A∇B, where A and B are twice differentiable scalar functions in the domain Ω. Substituting this into Equation 7 produces:

$$\int_{\Omega} (\nabla A \cdot \nabla B + A\nabla^2 B) d\Omega = \int_{\Gamma} A\nabla B \cdot n d\Gamma. \quad (8)$$

Redefining V as B∇A produces:

$$\int_{\Omega} (\nabla B \cdot \nabla A + B\nabla^2 A) d\Omega = \int_{\Gamma} B\nabla A \cdot n d\Gamma. \quad (9)$$

Finally, subtracting Equation 9 from Equation 8 forms Green's second identity, or:

$$\int_{\Omega} (A\nabla^2 B - B\nabla^2 A) d\Omega = \int_{\Gamma} (A\nabla B - B\nabla A) \cdot n d\Gamma. \quad (10)$$

Fundamental solution

The final step in applying Green's second identity to saturated porous media flow problems requires that the functions A and B satisfy Laplace's equation, i.e. $\nabla^2 A = \nabla^2 B = 0$. Since the product of the potential function and constant conductivity for isotropic media $K\Phi$ already satisfies Laplace's equation (Equation 5), it can be assigned to A. The function B is assigned a fundamental solution of Laplace's equation. A fundamental solution is simply some function which satisfies the field equation. In potential theory the fundamental solution is called a free space Green's function (Alarcon et al., 1979) and will be denoted as ψ. This function satisfies Laplace's equation everywhere but at a singular point $S(\mathrm{x})$, where it goes to infinity. Substituting ψ and $K\Phi$ into Equation 10 and noting that $\nabla^2\Phi = \nabla^2\psi = 0$ leaves:

$$\int_\Gamma (K\Phi\nabla\psi - \psi\nabla K\Phi)\cdot\mathrm{n}d\Gamma = 0. \tag{11}$$

The dot product $\nabla K\Phi\cdot\mathrm{n}$ represents the flow velocity normal to the boundary, or in differential notation $\partial K\Phi/\partial\mathrm{n}$. The dot product $\nabla\psi\cdot\mathrm{n}$ represents the normal derivative of the fundamental solution at the boundary . A shorthand notation for these normal boundary derivatives shall be Φ' and ψ' for the potential function derivative and the fundamental derivative, respectively. It is important to note that the conductivity K is included in the normal derivative for Φ, such that the flux boundary shall be in terms of normal flow and not normal flux. With this notation, Equation 11 becomes:

$$\int_\Gamma (K\Phi\psi' - \psi\Phi')d\Gamma = 0. \tag{12}$$

The form of the free space Green's function, or ψ, varies depending upon the dimensionality of the problem. For two-dimensional problems, $\psi = \ln r$, where r is the distance from the singular point S to some other point on the boundary. In three-dimensions, $\psi = 1/r$. For a complete derivation of the two and three dimensional fundamental solutions, the reader is referred to either Brebbia (1978) or Liggett and Liu (1983).

For the remainder of this discussion however, only the two-dimensional case shall be considered.

Since the fundamental solution ψ is singular at the point S, the point S must be excluded from the domain Ω of the problem in order to carry out the integration in Equation 12. To do this, the singular point S is surrounded by some infinitesimally small "shell" of radius r_o isolating it from the rest of the domain, thus creating a multiply connected domain. Equation 13 shows this as:

$$\int_\Gamma (K\Phi\frac{\partial}{\partial n}(\ln r) - (\ln r)\ \Phi')d\Gamma + \lim_{r_o \to 0}\int_\Gamma (K\Phi\frac{\partial}{\partial n}(\ln r_o) - (\ln r_o)\ \Phi')d\Gamma = 0. \quad (13)$$

Graphically, this can be seen in Figure 1.

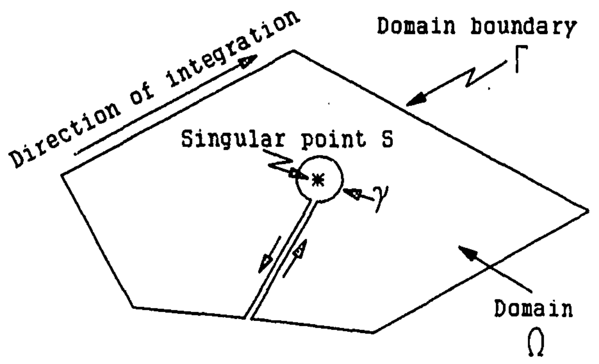

Figure 1. Boundary integration of domain Ω with singular point S

The two portions of the boundary integral running into the domain and connecting the actual boundary Γ and the infinitesimally small shell γ surrounding point S cancel one another out. They therefore do not appear in Equation 13. The limit of the second term of Equation 13 as r_o goes to 0 is $-2\pi K\Phi$. Equation 13 then becomes:

$$\int_\Gamma (K\Phi\frac{\partial}{\partial n}(\ln r) - (\ln r)\ \Phi')d\Gamma = 2\pi K\Phi(S) \qquad (14)$$

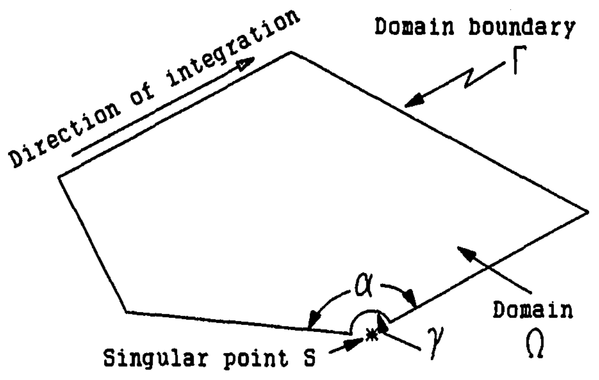

Figure 2. The singular point S moved to the boundary Γ

The singular point S can be anywhere in the problem domain Ω or on the boundary Γ. In moving S to the boundary, one still excludes it from the integration by an infinitesimally small shell γ. However, instead of being a circle, as in Figure 1, it becomes an arc whose subtended angle (α) is determined by the geometry of the surrounding boundary Γ as in Figure 2. The multiplier 2π in the right-hand side of Equation 14 is replaced by α or:

$$\int_\Gamma (K\Phi\frac{\partial}{\partial n}(\ln r) - (\ln r)\ \Phi')d\Gamma = \alpha K\Phi(S) \qquad (15)$$

The value of Φ can be calculated at the point S using either Equation 14 or 15, depending upon the location of S. However, in their present form, neither equation can be used directly in the calculation. This is because the necessary values of Φ or Φ' are not known everywhere on the boundary in a well posed problem. These missing boundary values must be calculated before either equation can be applied. Also, the form of Equations 14 and 15 requires analytical integration around the boundary, which for most problems is impossible. These problems are addressed in the next section. However, once all of the boundary conditions are known everywhere on Γ, Equation 14 can be then be used to find the values of Φ or its directional derivatives anywhere in the problem domain Ω.

General solution technique using the BEM

Since in a well posed problem neither Φ or Φ' are known everywhere on the boundary Γ, a means must be available for calculating them before interior values in the domain can be found. The BEM provides just such means through Equation 15. This process involves moving the singular point S to the boundary and applying the equation at various points around Γ. In order to use Equation 15, however, certain assumptions must be made about the boundary and its condition.

<u>Discretization</u> First, the behavior of the functions which make up the boundary conditions for the problem must be defined. This involves identifying where and what type of boundaries exist in a specific problem. The boundaries most commonly encountered in porous media flow problems are Dirichlet, or specified potential, and Neumann,

or specified flow, boundaries. These boundaries shall be denoted by Γ_1 and Γ_2, respectively. The boundary must be discretized into elements which must be placed so as to adequately describe the boundary geometry and conditions. This discretization oftentimes occurs at changes in a boundary type, or possibly at a geometric transition or corner.

These boundary elements may be defined by either single or multiple points, called nodes, along the boundary. The nodes may be either in the interior of an element or at its ends. The number of nodes which are required to completely define a boundary element depend upon the type of element which is being used. What is important at this point is to realize that instead of applying Equation 15 over the entire boundary analytically, the boundary will be segmented and the equation applied to each segment. The specifics of boundary discretization will be discussed later.

Shape functions Once the boundary is suitably discretized into elements, an estimate of the behavior of the boundary functions across each element is made. This approximation of boundary function behavior is called an shape function, or interpolation function, and will be denoted as M. Each node in an element is assigned a shape function which relates its nodal value to all other nodal values for the element. These shape functions vary as to order (i.e., constant, linear, quadratic, etc.). It is this order which determines the number of nodes which an element needs to be totally described. The types of elements may be seen in Figure 3.

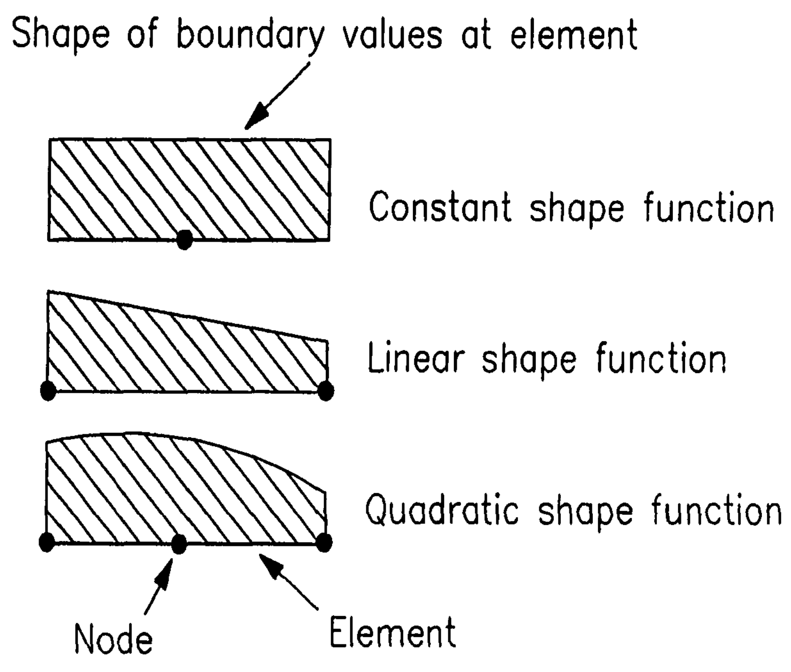

Figure 3. Examples of element shape functions

The elements themselves may be straight or curved, in order to best describe the geometry of a particular problem while the shape functions may be of any order which adequately describes the behavior of the boundary conditions across that element. In a groundwater problem, these boundary conditions are usually either prescribed potential (Φ) or flow (Φ'). Figure 3 shows straight elements with three different orders of shape functions, represented by the shaded areas above each straight element. The locations of nodes shown in each element in Figure 3 are not fixed. As long as there are enough nodes for the type of element being prescribed, and as long as the location of the nodes is accounted for in the definition of the shape function, the nodes may be placed anywhere in the element.

The standard element used for the remainder of this discussion and which is the basis for the computer model GWBEM will be a straight, two-node boundary element using a linear shape function. The reason for this is two-fold. First, the remainder of the derivation of the BEM using linear shape functions is complete enough to cover the important aspects of the technique without hindering the novice with excessive details. Secondly, the quality of data most often available for groundwater problems is insufficient to warrant the use of more complicated elements.

Boundary solution To calculate the unknown boundary data using the BEM, the boundary must be completely discretized into elements as in

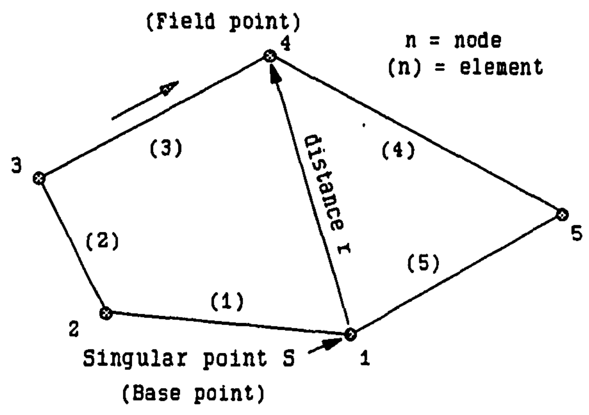

Figure 4. Example of boundary discretization with linear elements and base and field points

Figure 4. Equation 15 is then applied at each point around the boundary. The *r* is the distance from a "base" point *S* on the boundary to some "field" point. Each node on the boundary serves in turn as a base point, with the remaining nodes serving as field points for each base point. This generates a set of equations. With each of the resulting equations

relating Φ to Φ' at every node on the boundary, and with either Φ or Φ' being known at each boundary node, there are as many equations as boundary unknowns. This creates a solvable system of equations whose solutions are the unknown boundary values at each point on the boundary.

A problem develops during the integration around the boundary when the base point and the field point coincide. Since $r = 0$ in this case, the first term of the left-hand side of Equation 15 must receive special treatment. With the base point isolated from the domain by a small circular segment γ of radius r_o, as in Figure 2, the first term of Equation 15 becomes:

$$\int_\gamma \Phi \frac{\partial}{\partial n}(\ln r)\ d\Gamma = \int_\gamma \frac{\Phi}{r}\frac{\partial r}{\partial n}\ d\Gamma = \int_0^\alpha \frac{\Phi}{r_o}(-1)r_o d\theta = -\alpha\Phi. \tag{16}$$

This is obtained by differentiating the ln r term with respect to n and transforming the integral to polar coordinates. This means that during the assembly of the system of equations, which will be discussed in further detail in the next section, the contribution of the base point to itself is the value of $-\alpha$ at the base point. For a smooth section of boundary, this value is $-\pi$. A smooth boundary section is simply a straight line through the point.

Two-dimensional BEM with Linear Elements

This portion of the discussion of the BEM parallels that of Liggett and Liu (1983). Their approach was the most workable of all the methods investigated, both in terms of illuminating the intricacies of the BEM and in making the programming of the method more intuitive. Extensions to their method were required at various steps in the programming process, which shall be discussed as they appear.

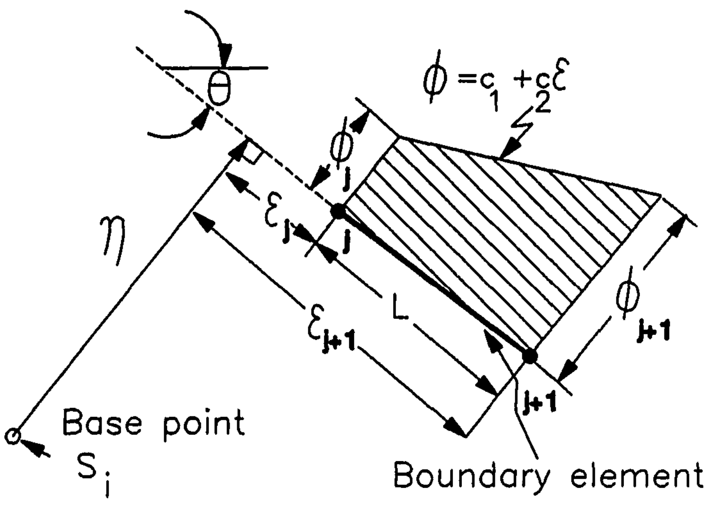

Figure 5. Linear element showing local coordinate system ξ and η and boundary value ϕ

Linear elements

A typical linear element is shown in Figure 5. The element is made up of two nodes located a distance *L* apart. For the purposes of this discussion, the nodes shall be located at the ends of the element, although as noted previously, this is not required. The nodes are denoted as *j* and *j*+1 respectively. The local coordinate system for the boundary element consists of two components: η, which is the normal distance to the element from the base point *S*; and ξ, which is as shown in Figure 5. The θ shown in the figure is the angle the element makes with the global x direction. A general scalar boundary function ϕ varies linearly across the element. Each nodal value of ϕ is identified by a subscript, with the intra-element values of ϕ being defined by the equation shown in the figure.

To derive the shape functions M_j and M_{j+1} for a linear element, the unknown coefficients c_1 and c_2 need to be found in terms of the nodal ϕ values and the local coordinate ξ. Segerlind (1984) developed these coefficients as:

$$c_1 = \frac{\phi_j \xi_{j+1} - \phi_{j+1} \xi_j}{\xi_{j+1} - \xi_j} \qquad c_2 = \frac{\phi_{j+1} - \phi_j}{\xi_{j+1} - \xi_j} \tag{17}$$

In Segerlind's derivation, these coefficients were substituted into the equation for ϕ in Figure 5, L was substituted for $\xi_{j+1} - \xi_j$, and terms were rearranged producing:

$$\phi = \left[\frac{\xi_{j+1} - \xi}{L}\right]\phi_j + \left[\frac{\xi - \xi_j}{L}\right]\phi_{j+1} \tag{18}$$

In Equation 18, each nodal value of ϕ is multiplied by a linear function of ξ. These functions are the shape functions M_j and M_{j+1} for the nodes j and j+1, respectively. To produce the proper value of ϕ everywhere on the element, the shape functions must have certain properties. First, the shape functions for each element node must sum to one at any location ξ on the element. Second, the value of a shape function must be unity at its respective node and zero at any other nodes on the element. Both of the multipliers shown in Equation 18 exhibit these properties.

Equations for boundary solution

For any linear boundary element with nodes at the ends, Liggett and Liu (1983) used a rearranged form of Equation 18 to describe the linear approximation for the boundary potential Φ as:

$$\Phi = \frac{\left|(\Phi_{j+1} - \Phi_j)\xi + (\xi_{j+1}\Phi_j - \xi_j\Phi_{j+1})\right|}{(\xi_{j+1} - \xi_j).} \tag{19}$$

The approximate normal derivative Φ' is:

$$\Phi' = \frac{\left|(\Phi'_{j+1} - \Phi'_j)\xi + (\xi_{j+1}\Phi'_j - \xi_j\Phi'_{j+1})\right|}{(\xi_{j+1} - \xi_j).} \tag{20}$$

The integral equation for each element, based on Equation 15, is of the form:

$$I_e = \int_{\xi_j}^{\xi_{j+1}} \left[\frac{K\Phi}{r_i}\ \frac{\partial r_i}{\partial \eta} - \Phi' \ln r_i\right] d\xi \tag{21}$$

Substituting the linear approximations for Φ and Φ' from Equations 19 and 20 into Equation 21, a series of integrations are performed which produce a set of equations relating the base points S_i on the boundary Γ to each pair of field points j and j+1 defining a boundary element. The terms associated with the nodal values of Φ and Φ' are collected to form:

$$I_e = \left|K_1^e\right| \begin{Bmatrix} \Phi_j \\ \Phi_{j+1} \end{Bmatrix} - \left|K_2^e\right| \begin{Bmatrix} \Phi'_j \\ \Phi'_{j+1} \end{Bmatrix} \tag{22}$$

where:

$$\left|K_1^e\right| = [-I_{11}+\xi_{j+1}I_{12} \quad I_{11}-\xi_j I_{12}]$$
$$\left|K_2^e\right| = [-I_{21}+\xi_{j+1}I_{22} \quad I_{21}-\xi_j I_{22}]. \tag{23}$$

The I terms found in Equation 23 are integrals and are discussed in the section on the micro-computer implementation of the BEM. The K^e terms in Equations 22 and 23 should not be confused with the K term used for hydraulic conductivity. By applying Equation 22 to each element on the boundary and summing up all the element integrals for each base point, one obtains a set of simultaneous linear equations of the form:

$$\sum_{j=1}^{N} R_{i,j}\, K\Phi_j = \sum_{j=1}^{N} L_{i,j}\, \Phi'_j \tag{24}$$

where:

$$R_{i,j} = [(K_1^e)_{i,j} - \delta_{i,j}\alpha_i] \quad \delta_{i,j} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$
$$L_{i,j} = (K_2^e)_{i,j} \tag{25}$$

The R coefficients shown in Equation 24 relate to the Γ_1 boundary conditions and the L coefficients are associated with the Γ_2 boundary conditions. These coefficients can be assembled into a system of linear equations with the knowns on one side and the unknowns on the other, forming:

$$\mathbf{A}\mathbf{x} = \mathbf{B} \tag{26}$$

where:

A	=	coefficient matrix of unknown boundary values from each boundary node,
x	=	unknown boundary conditions (Φ or Φ') for each boundary node, and
B	=	summation of product of integral coefficients and known boundary values for each node on the boundary.

<u>System assembly and solution</u>

<u>Equation assembly</u> The placement of the coefficients *R* and *L* from Equation 24 into the system matrices depends upon the type of boundary encountered at each element. If the known boundary condition at an element is a Dirichlet boundary (Γ_1), then the integral coefficients *R* for that element are multiplied by the known nodal potential values Φ for that element and are placed into **B**. The integral coefficients *L* are then placed into **A** without being multiplied by any nodal values as these are unknown.

The reverse is true if an element is a Neumann boundary (Γ_2). In this case, the *L* coefficients for the element are multiplied by their respective nodal flow velocity values, Φ'. These products are placed into B while the *R* coefficients for the element are placed into **A**. **A** accumulates the coefficients *R* or *L* without any multiplication by boundary values. B, on the other hand, always receives the integral coefficients after they have been multiplied by some known boundary values.

Each row of A and B has a matching boundary node. The row index *i* refers to each boundary node as it serves as base point for boundary integration. Each column of A corresponds to an unknown boundary value, or degree of freedom (DOF). As each element is integrated using Equation 21, the node numbers *j* and *j*+1 of the nodes defining the element being integrated become the colımn indices of A since there is one DOF for each boundary node.

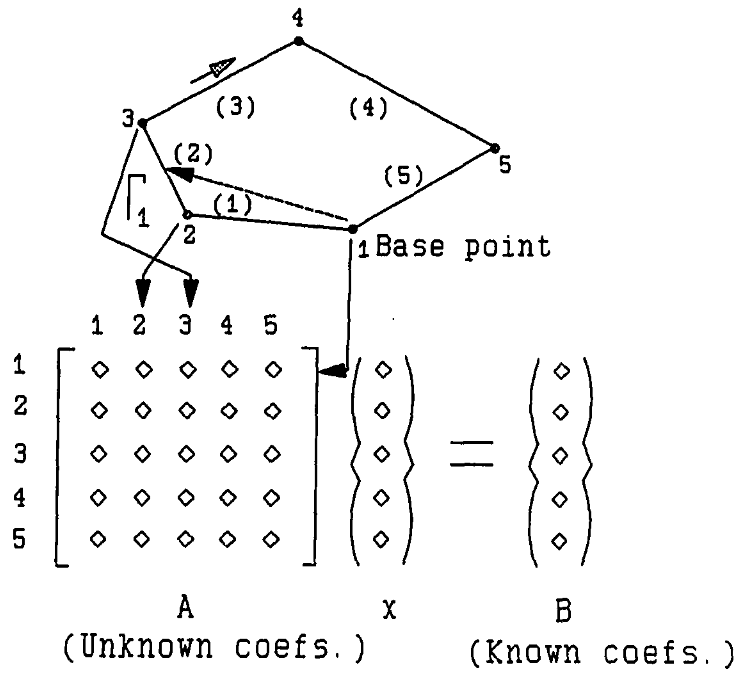

Figure 6. Placement of integral coefficients into system equations

Figure 6 depicts the assembly of an example system. In the figure, node 1 is the current base point for the discretized boundary integration. With node 1 as the base point, all of the integral coefficients $R_{1,n}$ and $L_{1,n}$ found using Equation 21 during the discretized boundary integration are placed somewhere in row 1 of either A or B. As is also shown in Figure 6, the integration is being performed on element 2, which is a Γ_1 boundary and whose end nodes are 2 and 3 respectively.

The integral coefficients L corresponding to the boundary DOFs at each node of element 2 are added into **A**. The coefficient for node 2 with node 1 as base point ($L_{1,2}$) is added into **A** at row 1 and column 2. The coefficient for node 3 with node 1 as base point ($L_{1,3}$) is added into **A** at row 1 and column 3. Consequently, the integral coefficient $R_{1,2}$ is multiplied by Φ_2, with the resultant product added to row 1 of **B**. The same thing is done for $R_{1,3}$, except that Φ_3 is the multiplier. This product is also added to row 1 of **B**.

Each column within **A** is accessed twice during the integration from each base point. This is because as the integration moves around the boundary, each boundary node serves first as the trailing node for one element and then as the leading node for the following element.

<u>Equation solution</u> Once a complete boundary integration is performed using each boundary node as base point and the integral coefficients from each integration have been properly assembled into the system equations, the system unknowns can be obtained using any standard equation solver. The system solver used in GWBEM is adapted from Forsythe, Malcolm, and Moler (1977). Their method contains two steps, decomposition and back substitution. The Pascal listing of this may be seen in the procedures DECOMP and SOLVE of unit B7SOLVER.PAS in Appendix A. The decomposition step performs the Gaussian elimination, which is dependent on the matrix only. This is advantageous as the system matrix is based solely on the problem geometry. The multipliers and pivot

information from the decomposition are saved and can be applied to any right-hand side, allowing for the solution of different sets of boundary conditions for a given geometry from only one decomposition. The decomposition routine also provides the condition number of the system matrix. The condition number is a measure of the singularity of the system matrix. The higher the condition number, the more ill-conditioned the system matrix.

Equations for interior solution

Once all the values on the boundary are known, it is then possible to obtain the solution at points in the interior of the domain Ω. Liggett and Liu (1983) also derived analytical integrations for the solution at interior points with the BEM, using Equation 14. To obtain the potential Φ at any interior point, one simply integrates around the boundary using the selected interior point as base point. However, since all of the boundary values are known at this point, one generates a single equation for the value of Φ at the base point based upon Equation 21. This may be repeated for as many interior points as desired.

The solution for flow velocity values at interior points is not as straight forward as those for potential at interior points. The flux value solutions involve differentiating Equation 14 with respect to the desired direction of flow. The equations for flow velocity in the x and y directions are:

$$\frac{\partial}{\partial x}\Phi(S) = \frac{1}{2\pi}\int_{\Gamma}\left[\frac{\sin\theta}{r^2}K\Phi + \frac{2\eta(\xi\cos\theta - \eta\sin\theta)}{r^4}K\Phi + \frac{\xi\cos\theta - \eta\sin\theta}{r^2}\Phi'\right]d\Gamma \qquad (27)$$

$$\frac{\partial}{\partial y}\Phi(S) = \frac{1}{2\pi}\int_{\Gamma}\left[-\frac{\cos\theta}{r^2}K\Phi + \frac{2\eta(\eta\cos\theta+\xi\sin\theta)}{r^4}K\Phi + \frac{\eta\cos\theta+\xi\sin\theta}{r^2}\Phi'\right]d\Gamma \qquad (28)$$

where:

η & ξ	=	local coordinates as defined in Figure 5,
θ	=	angle element makes relative to global x direction, and
r	=	distance from base point to field point.

The use of Equations 27 and 28 for the determination of interior flow velocity values is similar to that for interior potential values. For each interior point where the solution is desired, a boundary integration is performed using each interior point of interest as base point. The specific form of the integrals is somewhat different for the flow velocity values than it is for the potential solution. Their form is discussed in further detail in the next section.

Micro-computer Implementation

This section shall continue the derivation of the BEM by presenting the computer implementation of the methods discussed in the previous section. References will be made throughout to the Pascal listing of the computer program GWBEM found in Appendix A and which is the result of this work. All of the integrations used in this implementation are analytical. This was done to avoid excessive round off error found in many numerical integration methods and to avoid numerical complications encountered during the integration of certain singular integrals found throughout the BEM.

Boundary solution

To solve for the boundary unknowns, the I coefficients found in Equation 23 and subsequently the K^e terms in that equation need to be calculated. To derive the I_{1n} coefficients, one takes the linear approximation of Φ from Equation 19 and substitutes it into the first term of Equation 21. This produces:

$$\int_{\xi_j}^{\xi_{j+1}} \frac{\Phi}{r_i} \frac{\partial r_i}{\partial \eta} = \int_{\xi_j}^{\xi_{j+1}} \frac{\left| (\Phi_{j+1} - \Phi_j)\xi + (\xi_{j+1}\Phi_j - \xi_j\Phi_{j+1}) \right|}{(\xi_{j+1} - \xi_j)\ r_i} \frac{\partial r_i}{\partial \eta}\, d\xi . \tag{29}$$

Rearranging terms as in Equation 23 and substituting L for $\xi_{j+1} - \xi_j$, the I_{11} and I_{12} coefficients take on the form:

$$\begin{aligned} I_{11} &= \frac{1}{L} \int_{\xi_j}^{\xi_{j+1}} \frac{1}{r_i} \frac{\partial r_i}{\partial \eta}\, \xi\, d\xi \\ I_{12} &= \frac{1}{L} \int_{\xi_j}^{\xi_{j+1}} \frac{1}{r_i} \frac{\partial r_i}{\partial \eta}\, d\xi . \end{aligned} \tag{30}$$

Performing these integrations yields:

$$\begin{aligned} I_{11} &= \frac{\eta_i \ln(\eta_i^2 + \xi^2)}{2\ L} \Bigg|_{\xi_j}^{\xi_{j+1}} \\ I_{12} &= \frac{\tan^{-1}\left[\dfrac{\xi}{\eta_i}\right]}{L} \Bigg|_{\xi_j}^{\xi_{j+1}} \end{aligned} \tag{31}$$

where $(\eta^2 + \xi^2)^{1/2}$ has been substituted for r. The values of $K_1{}^e$ are then calculated from the I coefficients with the use of Equation 23.

The I_{2n} coefficients used to find the $K_2{}^e$ terms in Equation 23 are derived in a similar manner. Substituting the linear approximation of Φ' from Equation 20 into the second term of Equation 21 yields the I_{2n} coefficients as:

$$I_{21} = \frac{1}{L} \int_{\xi_j}^{\xi_{j+1}} \ln r_i \xi \, d\xi$$

$$I_{22} = \frac{1}{L} \int_{\xi_j}^{\xi_{j+1}} \ln r_i \, d\xi. \qquad (32)$$

The integrated form of these coefficients is:

$$I_{21} = \frac{\ln(\eta_i^2 + \xi^2)\left[\ln(\eta_i^2 + \xi^2) - 1\right]}{4\,L} \Bigg|_{\xi_j}^{\xi_{j+1}}$$

$$I_{22} = \frac{\xi \ln(\eta_i^2 + \xi^2) - 2\xi + 2\eta_i \tan^{-1}\left[\frac{\xi}{\eta_i}\right]}{2\,L} \Bigg|_{\xi_j}^{\xi_{j+1}}. \qquad (33)$$

Once the K^e terms are found for each element, they are assembled into the system of equations as discussed above. The Pascal listing for the integrations discussed above are contained in the procedure Integrate Boundary of the unit B7INT.PAS found in Appendix A.

What ultimately emerged while working with the equations for the BEM using linear elements was a recurrent pattern of basic formulas for both the boundary and interior solutions. Liggett and Liu (1983) provided many of the analytical integrations used with linear elements for the BEM. These integrals were checked and modified for use in GWBEM and are shown in Table 1. It should be noted that the integrations in the table were found by first substituting $(\eta^2 + \xi^2)^{1/2}$ for r in each

Table 1. Basic formulas encountered with the BEM using linear elements and adapted from Liggett and Liu (1983)

Form I:	$\int \frac{1}{r}\, d\xi = \frac{1}{\eta} \tan^{-1}\left[\frac{\xi}{\eta}\right]$
Form II:	$\int \frac{\xi}{r}\, d\xi = \frac{1}{2} \ln (\eta^2 + \xi^2)$
Form III:	$\int \frac{\xi^2}{r}\, d\xi = (\xi - \eta \tan^{-1}\left[\frac{\xi}{\eta}\right])$
Form IV:	$\int \frac{1}{r^2}\, d\xi = \frac{\xi}{2\eta^2(\eta^2 + \xi^2)} + \frac{1}{2\eta^3} \tan^{-1}\left[\frac{\xi}{\eta}\right]$
Form V:	$\int \frac{\xi}{r^2}\, d\xi = - \frac{1}{2(\eta^2 + \xi^2)}$
Form VI:	$\int \frac{\xi^2}{r^2}\, d\xi = - \frac{1}{2(\eta^2 + \xi^2)} + \frac{1}{2\eta} \tan^{-1}\left[\frac{\xi}{\eta}\right]$
Form VII:	$\int \frac{1}{2}\, \xi \ln r\, d\xi = \frac{1}{4} (\eta^2 + \xi^2)\, [\ln(\eta^2 + \xi^2) - 1]$
Form VIII:	$\int \frac{1}{2} \ln r\, d\xi = \frac{1}{2}[\xi \ln(\eta^2 + \xi^2) - 2\xi + 2\eta \tan^{-1}\left[\frac{\xi}{\eta}\right]$

equation. By utilizing the formulas shown in Table 1, the derivation of the other equations for the BEM becomes much simpler. These basic forms shall be used where applicable in the following derivations. Each of the formulas found in the table and which are used in subsequent equations shall be referred to by its Roman numeral listed in the table.

Interior solution

The equations used in the computer implementation of the BEM for the solution of Φ in the interior of the domain are identical to Equations 31 and 33, which generate the I coefficients used in Equation 22. Equation 22 is applied to every element on the boundary while the interior point of interest is used as base point.

For the calculation of the interior flow velocity values, one substitutes the linear approximation of Φ (Equation 19) and Φ' (Equation 20) into Equation 27. The terms can then be rearranged and integrated so that the form of the equation resembles Equation 22, with coefficients K^e are multiplied by the nodal values of Φ and Φ' on the boundary, or:

$$\frac{\partial\Phi}{\partial w_e} = \left|K'_1\right|\begin{Bmatrix}\Phi_j\\ \Phi_{j+1}\end{Bmatrix} + \left|K'_2\right|\begin{Bmatrix}\Phi_j\\ \Phi_{j+1}\end{Bmatrix} + \left|K'_3\right|\begin{Bmatrix}\Phi'_j\\ \Phi'_{j+1}\end{Bmatrix} \qquad (34)$$

where w is any direction in which the flow velocity is to be calculated and where the K' terms are defined as:

$$\begin{aligned}\left|K'_1\right| &= [-I_{11}+\xi_{j+1}I_{12};\ I_{11}-\xi_j I_{12}]\\ \left|K'_2\right| &= [-I_{21}+I_{23}+\xi_{j+1}(I_{22}-I_{24})\ ;\ I_{21}-I_{23}+\xi_j(-I_{22}+I_{24})] \qquad (35)\\ \left|K'_3\right| &= [-I_{31}+I_{33}+\xi_{j+1}(I_{32}-I_{34})\ ;\ I_{31}-I_{33}+\xi_j(-I_{32}+I_{34})]\end{aligned}$$

Using this form, the interior flow velocity values in any direction may be obtained by substituting the *I* terms with their proper values for that direction. For the flow velocity values in the x and *y* direction, the values for *I* are those shown in Table 2.

Table 2. *I* coefficients for flux in the *x* and *y* directions

I term	Flux x direction	Flux *y* direction
I_{11}	$\sin\theta$(II)	$-\cos\theta$(II)
I_{12}	$\sin\theta$(I)	$-\cos\theta$(II)
I_{21}	$2\eta\cos\theta$(VI)	$2\eta\sin\theta$(VI)
I_{22}	$2\eta\cos\theta$(V)	$2\eta\sin\theta$(V)
I_{23}	$2\eta^2\sin\theta$(V)	$2\eta^2\cos\theta$(V)
I_{24}	$\sin\theta$(III)	$\cos\theta$(III)
I_{31}	$\cos\theta$(III)	$\sin\theta$(III)
I_{32}	$\cos\theta$(II)	$\sin\theta$(II)
I_{33}	$\eta\sin\theta$(II)	$\eta\cos\theta$(II)
I_{34}	$\eta\sin\theta$(I)	$\eta\cos\theta$(I)

The Roman numerals found in Table 2 refer to the basic formulas listed in Table 1. As such, each formula is multiplied by the terms shown to produce the *I* coefficients used in Equation 35 and ultimately in Equation 34. The η in the table is the normal distance to each element from the base point, as defined in Figure 5. θ is the angle each element makes with the x axis. The Pascal listing of these equations is found in procedure Integrate Interior of Unit B7INT.PAS of the program GWBEM found in Appendix A.

Sources and sinks

Problems with wells require special techniques during the assembly of the equations for the boundary solution. As discussed previously, wells can be idealized as sources or sinks where the flow enters or leaves the domain through a point. Equation 6 shows the modification to Laplace's equation used to accommodate this.

Lafe et al. (1980) proposed a method of superposition to accommodate sources or sinks in the interior of a domain using the BEM. Unfortunately, the treatment allowed only for the solution of potential at a point with flow as the known value. It was desired that GWBEM allow either type of condition to be specified at a point. Therefore, a technique advanced by Radojkovic and Pecaric (1984) was used for the solution of domains with wells.

Their method involves including terms for each well point into the system of equations normally generated with the BEM for the boundary solution. The basic form of the terms added for each well point is $\ln(r_j)$, where r_j is the distance between the well node j and some other node, either on the boundary or in the interior of the problem domain. The placement of each term in the system of equations depends upon the type of condition specified for each well node.

If the flow rate Q_j is specified at a particular well j, then the product of $\dot{Q}_j \ln(r_j)$ is subtracted from the known vector B. The row of B corresponds to the base point from which the boundary integration is performed. Conversely, if the potential is specified at a well, the value $\ln(r_j)K$ is added to A at the row matching the base point of integration and the column coinciding with the node number of the current well node.

GWBEM considers well points as additional, yet separate, boundary nodes. For example, if a problem boundary was discretized using five boundary nodes and if two wells were found in the problem interior, there would be a total of seven boundary nodes defining the problem. The total size of A would then be 7 x 7. GWBEM also considers flow out of the domain as positive. Therefore, a pumping well would be defined positive, and an injection well defined as negative.

Boundary values   The inclusion of each well node into the system of equations during the solution of the unknown boundary values is carried out in procedure Integrate Boundary of GWBEM. In the loop which performs the integrations over the regular boundary using Equation 21, the presence of well nodes is checked. If well nodes are found in the problem domain, the $\ln(r)$ terms which account for the contribution of each well node are added to the system of equations at the end of each integration loop around the continuous boundary.

After all of the regular nodes on the continuous boundary have been used as base point for a boundary integration, another loop is entered which integrates from each well node to every other boundary node, including any other well nodes. During the integration from each well node to the nodes on the continuous boundary, the value of α for each well node is 2π and Equation 21 is applied as usual. While integrating from one well node to another, the values added to the system of equations are identical to the $\ln(r)$ terms. One exception should be noted which occurs when the well node acting as base point becomes the same point being integrated to. In this situation, r becomes the radius of the well rather than the distance to some other node. Since the wells are included as boundary nodes, solution of the resulting system of equations not only produces the unknowns on the continuous boundary, but also the unknowns at the well nodes.

Interior values Once all of the boundary unknowns are established, including the unknown well values, the interior values at selected points can be determined using a modified form of the interior solution using no wells presented previously. The potential values are modified by adding the product $Q_j\ln(r_j)$ for each well to the integral value for potential at each interior point found by the method described earlier for Equation 22. The variables of the additional product for each well are the same as for the boundary solution case. This operation is performed in the nested procedure AddSources in the global procedure Integrate Interior of unit B7INT found in Appendix A.

The values for flow velocity at interior points also require modification when wells are present. The values of $\partial\Phi/\partial x$ and $\partial\Phi/\partial y$ are first determined using the method discussed previously for Equation 34. The results are then added to the products $(r_xQ_j)/(r^2)$ and $(r_yQ_j)/(r^2)$

respectively. The values of r_x and r_y refer to the distance between the well and the selected interior point in the x and y directions. This summation is done for every interior point using all wells within the enclosed boundary.

Multi-zone solutions

The BEM described up to this point is capable of solving homogeneous groundwater problems. Many groundwater problems are, however, far from homogeneous. Brebbia (1978) and several others have discussed methods of analyzing non-homogeneous media using the BEM. Their approach has been to divide the media up into zones of differing characteristics. In the case of groundwater flow, these zones would differ as to their hydraulic conductivity. The program GWBEM approaches non-homogeneous problems in a similar manner. Figure 7 shows a two zone system with a shared boundary.

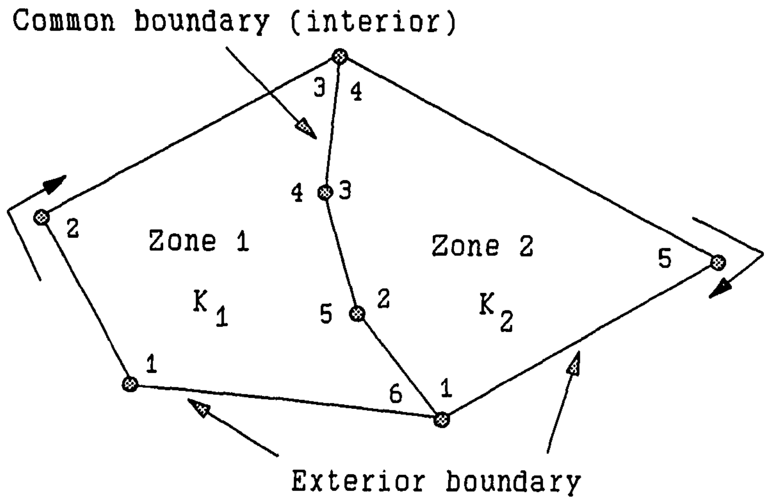

Figure 7. Example problem with two zones of differing conductivities with common interior boundary showing zonal node numbering

The shared boundary between the two zones is designated as an interior boundary while the unshared boundaries for each zone are denoted as exterior boundaries. Each zone is treated as a separate domain whose boundary is discretized and integrated. Each zone is then coupled to the other zones to form one system of equations using the equilibrium and compatibility conditions across shared zone boundaries.

Unfortunately, to maintain the advantages of the BEM over other analytical techniques, each of the different zones must be homogeneous within themselves. If any zones are non-homogeneous, then integration must be performed over the entire domain for each non-homogeneous zone rather than just over their boundaries. A problem may have as many connected zones as desired, but each zone must be uniform throughout. Although simple in theory, the actual computer implementation of the BEM for multi-zone problems was found to be a formidable task for a number of reasons.

First, in solving for a single zone system with the BEM discussed up to this point, there are as many boundary equations as boundary unknowns. However, with multi-zones sharing common boundaries, as in Figure 7, there are two unknowns for every point on the interior boundary, namely Φ and Φ'. If the boundary for each zone is integrated in the normal fashion, there will be more unknowns than equations because each node on the interior boundary will serve as base point once for each zone even though there are two unknowns associated with that node.

To resolve this situation, what is usually done is to apply the equilibrium and continuity conditions for potential and normal flow at the interior boundary nodes. Liggett and Liu's (1983) application of these conditions generates two additional equations for each interior node, one for potential compatibility and another for normal flow equilibrium. This creates a solvable system, but at the cost of generating a larger system of equations. Since GWBEM is intended for microcomputer use where memory is at a premium, the use of another method was preferable. An approach explored by Brebbia and Chang (1985) was therefore utilized in GWBEM.

System assembly In their method, the compatibility and equilibrium equations are accounted for by condensing the system equations during assembly. This involves having fixed columns in the system of equations for each shared unknown boundary DOF and having the integral coefficients K^e accumulate in those fixed columns for each adjacent zone during the matrix assembly. The conditions for equilibrium and compatibility are utilized at this point, depending upon the type DOF being operated on.

Figure 8 shows the system matrix for the simple two zone system of Figure 7. As an example, node 5 of zone 1 is the same as node 2 of zone 2. By the use of the compatibility condition, the integral coefficients from each zone for the Φ DOF at this shared node are placed into the system matrix in column 5 after being multiplied by that zone's conductivity. The row each set of integral coefficients is placed into depends upon the base point and zone which is being integrated. By the use of the equilibrium condition, the integral coefficients for the Φ' DOF at this node are placed in column 6. Since the normal flow out of zone 1 is

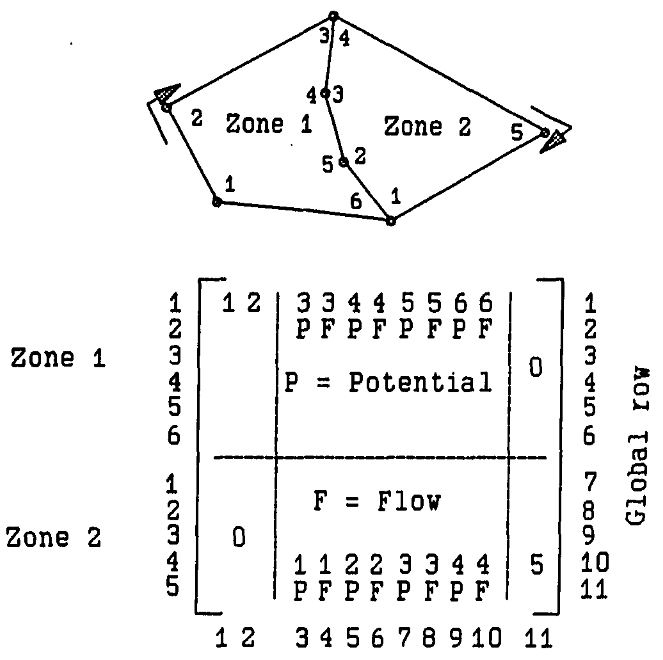

Figure 8. Assembly of system matrix for simple two zone system with shared boundary

the same as the normal flow into zone 2 at this common node, the sign of the integral coefficients placed into column 6 from zone 1 is opposite that of zone 2. In other words, one of the two zones has a multiplier of -1 applied to its integral coefficients for the flow DOFs at the common interior nodes. The resulting system matrix then has a row for every base point in each zone and a column for each boundary DOF. This system is now in solvable form.

<u>Discontinuous elements</u> Unfortunately, the problem of assembling multi-zone systems is not the only obstacle to solving them. Another problem occurs during the calculation of the unknown flow values on interior boundaries. When the intersection of two interior boundary elements form a straight line at a common node as in the upper half of

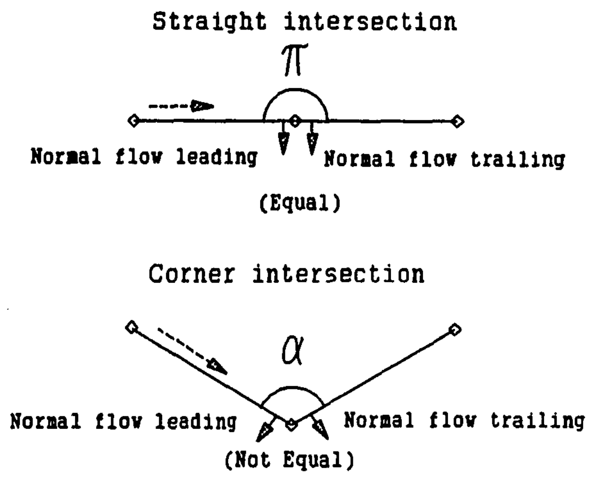

Figure 9. Normal flow at element intersection

Figure 9, the normal flow component at that node is the same on either side of the node. On the other hand, if the elements meet and form a corner as in the lower half of Figure 9, the normal flow components are not the same on either side of the common node for each zone. The normal flow component at the corner is actually not defined. This same problem of ambiguous normal flow also occurs when two Γ_1 boundary elements meet at a corner on an external boundary.

These types of element intersections at corners develop problems during the boundary integration and system assembly. Normally, the integral coefficients from the leading and trailing elements for the flow DOF at a node (K_2) are placed in the same column of the system matrix because they represent the same flow DOF. However, with a corner node, these coefficients from the leading and trailing elements cannot be placed into the same DOF, or column, of the system matrix because they actually represent different flow DOF. One could lump the coefficients from each side of the node into one DOF, but this creates significant numerical errors in the vicinity of the corner (Patterson and Sheikh, 1981).

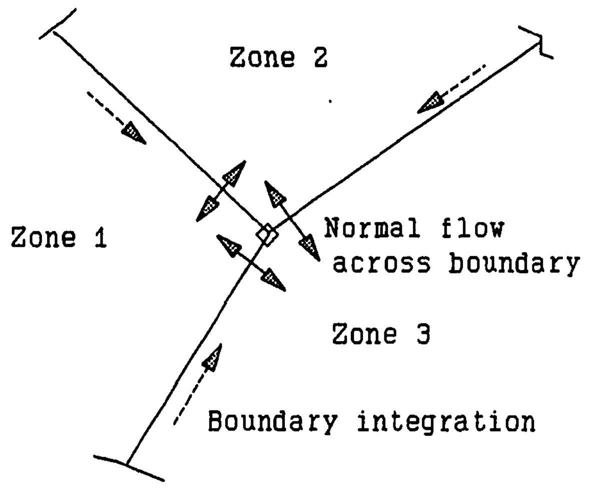

Figure 10. Junction of three zones meeting at common node and normal flows across zone boundary

Figure 10 shows another problem which is encountered during the assembly of multi-zone systems. When several zones meet at a common node, the normal flow for the junction node is not defined nor is it equal as one moves along the boundary of any zone and across the junction node. The assembly of the flow integral coefficients for each zone is also made more difficult by now having to determine which flow DOFs match across zone boundaries. A similar situation exists when moving from an exterior zone boundary to an interior boundary or vice versa. The normal flow at the junction node can be defined for several different directions at once.

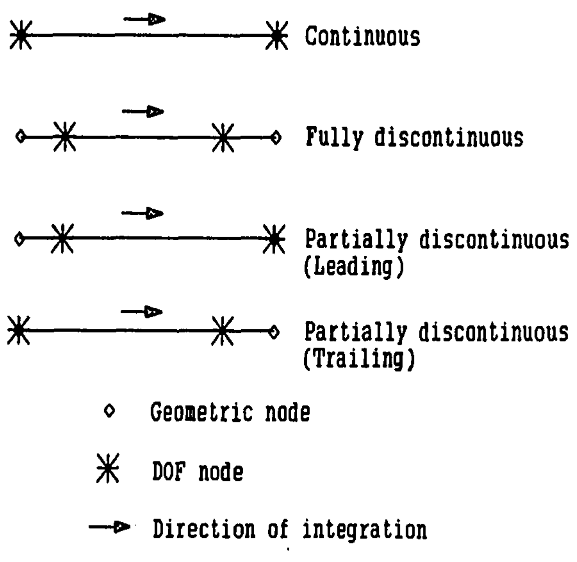

Figure 11. Continuous and discontinuous two-node linear elements

Several methods were evaluated to resolve this problem of ambiguous normal flow at a boundary corner. The method implemented in GWBEM was that of Patterson and Sheikh (1984), which make use of discontinuous elements. A discontinuous element is defined as one where the geometric nodes defining the end points of an element and DOF nodes defining the boundary values do not coincide. Figure 11 shows different types of discontinuous linear elements and compares them with a continuous linear element. These elements allow for the normal boundary flow to be discontinuous across an element intersection, something which continuous elements do not allow. A discontinuous element, then, solves the problem of ambiguous flow at a corner, but with a price. Because of the extra DOF needed to define them, their use requires another equation for each discontinuous element in a boundary. Also, the linear shape functions used to define the behavior of the boundary values for each element must be modified to accommodate the new interior location of the DOF nodes for the element.

Patterson and Sheikh present linear shape functions for fully discontinuous and partially discontinuous elements. Unfortunately, their development is couched in terms of normalized natural element coordinates, which is different then the coordinate system discussed previously for Figure 5. Normalized element coordinates have the origin at the center of the element and the end nodes at $\pm$ 1. This type of coordinate system lends itself very well to numerical integration routines. GWBEM, however, uses analytical integrations and unnormalized element coordinates. The shape functions provided by Patterson and Sheikh had to be modified for use in GWBEM. After performing a coordinate transformation on the shape functions of Patterson and Sheikh and then substituting these transformed shape functions into Equation 21, new K terms for Equation 22 were found for each of the three types of discontinuous elements from Figure 11. These are seen in Equation 36.

The I_1 and I_2 terms for these equations are the same as those defined in Equations 30 and 32. The L is the element length. The use of these new integral coefficients for discontinuous elements arises when elements are adjacent to flow ambiguities, i.e., corners or junctions. If an element has an ambiguous flow DOF at only one of its nodes, then it becomes a partially discontinuous element depending upon which end the indefinite node is on, and the proper K coefficients are then substituted for the regular ones used during matrix assembly. If both element nodes are ambiguous, then the element becomes fully discontinuous. During the boundary integration, the DOF nodes become the base (collocation) points rather than the geometric nodes, as is the case for continuous elements. The limits of integration used in all of the integral equations for each element are still based on the element end points, however.

(Fully discontinuous element)

$$\left|K_1^e\right| = [-\frac{2}{L}I_{11} + (1.5 + \frac{2}{L}\xi_j)I_{12} \; ; \; \frac{2}{L}I_{11} - (\frac{1}{2} + \frac{2}{L}\xi_j)I_{12}]$$

$$\left|K_2^e\right| = [-\frac{2}{L}I_{21} + (1.5 + \frac{2}{L}\xi_j)I_{22} \; ; \; \frac{2}{L}I_{21} - (\frac{1}{2} + \frac{2}{L}\xi_j)I_{22}]$$

(Leading discontinuous element)

$$\left|K_1^e\right| = [-\frac{4}{3L}I_{11} + (\frac{4}{3} + \frac{4}{3L}\xi_j)I_{12} \; ; \; \frac{4}{3L}I_{11} - (\frac{1}{3} + \frac{4}{3L}\xi_j)I_{12}] \tag{36}$$

$$\left|K_2^e\right| = [-\frac{4}{3L}I_{21} + (\frac{4}{3} + \frac{4}{3L}\xi_j)I_{22} \; ; \; \frac{4}{3L}I_{21} - (\frac{1}{3} + \frac{4}{3L}\xi_j)I_{22}]$$

(Trailing discontinuous element)

$$\left|K_1^e\right| = [-\frac{4}{3L}I_{11} + (1 + \frac{4}{3L}\xi_j)I_{12} \; ; \; \frac{4}{3L}I_{11} - (\frac{4}{3L}\xi_j)I_{12}]$$

$$\left|K_2^e\right| = [-\frac{4}{3L}I_{21} + (1 + \frac{4}{3L}\xi_j)I_{22} \; ; \; \frac{4}{3L}I_{21} - (\frac{4}{3L}\xi_j)I_{22}]$$

Model input

A great amount of effort was devoted to make GWBEM as easy as possible to use, particularly for those unfamiliar with the BEM. This involved keeping the data input to a minimum while allowing for an adequate problem description. An attempt was also made to keep the output from the program as concise yet meaningful as possible. The program's utility as a tool in understanding groundwater flow becomes most apparent in a learning environment, as the normally cumbersome details of defining a problem for computer solution are performed by the program itself rather than by the user. This is in contrast to finite difference and finite element models where the entire domain requires discretization and where special techniques are often required, as in modeling the flow in the vicinity of a well. GWBEM frees the user to concentrate on the details of the results rather than the details of the input.

Several different methods of describing the geometry and connectivity of a multi-zone groundwater problem for GWBEM were tested in terms of ease of input and programming. The ultimate method used by GWBEM was developed from one advanced by Rudolphi (1988). The method consists of two tiers, a global tier and a local, or zone tier. All nodes and elements which define a problem boundary for all zones are specified on the global tier. At this level, the coordinates of all nodes making up the boundary are input and a unique global node number is assigned to each boundary node. The elements are defined by specifying which global nodes correspond to the end nodes of each element and which type of boundary is found at each element, either Γ_1, Γ_2, or interior. Each element is also given a unique global element number. There is an implied direction for each element which comes from the order in which the global nodes for each element are specified. If an element is defined as having global nodes 5 and 6 as end points in that order, then the implied direction for that element is from global node 5 to node 6.

Once all of the boundary nodes and elements are defined globally, the problem definition moves to the local, or zone level. At this level, the elements which make up each zone boundary are input. An ordered list is generated for each zone. This list contains the global element numbers defining a zone boundary. The order of the elements matches the order the elements come in as one moves around each zone boundary. Any element may be the starting point for the list as long as the succeeding elements are listed in order around the boundary. The sign on each element number in a zone list compares the relative direction of the clockwise integration around the zone boundary with the implied direction for each global element. If the direction of integration and the implied element direction are the same, then the sign on the global element for that zone list is positive. If the directions are opposite, then the sign on the global element is negative. This scheme provides the simplest means of describing a problem geometry as well as accounting for any connectivity between zones. By using global elements, any shared boundaries are easily identified in the computer program.

Other problem characteristics are also defined at the zone level. These include the hydraulic conductivity of each zone, the location and type of any wells contained in a zone, and the location of any interior points where the solution is desired. An example input listing for GWBEM may be found in Appendix B.

Model output

The input and output for the model is file based. This allows for the modification of output using any text editor and eases the importation of the model output into other analysis programs such as spreadsheets or contouring programs. It has been the author's experience that programs which prevent this type of modification to their output to be of limited use.

The output from the model consists of several blocks. The main output block is made up of several distinct sections. The first section is a formatted form of the input is repeated to allow for a check of the data. Following this, the boundary of each zone is traversed element by element. For each element, the coordinates and solution at the end nodes is listed. After the boundary node solutions are given, the solution at interior nodes for each zone is written. This information includes the local zone numbering of each interior node and the potential and flow vector values in the local x and y directions. All of this output block may be directed to either a printer or a disk file.

Other output blocks are contained in text files. These include a listing of the system matrix generated during the boundary integration and separate files for the potential and flow values in the x and y directions at all interior nodes. Only the potential values for the boundary nodes are output in these files.

Model features

All of the routines and data structures used in the model GWBEM are unique and have been tailored to allow the most efficient use of the micro-computer environment. This was done with a mix of fixed and dynamic data structures. The model makes extensive use of linked lists and virtual disk arrays. Model size is therefore limited mostly by available disk space, although with the advent of large mass storage devices, this constraint will become less significant in the future. Currently, the model is capable of running problems with approximately 4000 boundary nodes and 1400 boundary elements on a machine equipped with 640K of core memory, although future improvements in micro-computer operating systems and operating speeds could increase this amount dramatically.

Manually determining which elements are discontinuous and which type of discontinuous element an element should be becomes a cumbersome task for large, multi-zone systems. The process usually is to analyze the problem boundary and manually designate each discontinuous element. Once the discontinuous elements are selected, several things must be done to each. First, the boundary values normally defined for each element end point must be altered to reflect the position of the new DOFs on the interior of the element. Then each new DOF must be included in the boundary integration. The assignment of the proper DOFs during the assembly process becomes very complex as the number of boundary nodes and the number of zones increase for a given problem. This process takes a large amount of time and is a source of considerable error during problem input.

One significant feature of the model is found in the routine Prep System of the unit B7PREP.PAS. This routine automatically checks each zone boundary and determines which elements require conversion to discontinuous elements. It decides which type of discontinuous element should be used in each case, calculates the new boundary values at any interior DOFs assuming linear behavior, and assigns a proper DOF number for correct assembly of the system equations. This provides for more fool-proof use of the model for multi-zone systems which would otherwise provide many difficulties to a BEM neophyte.

MODEL VERIFICATION

Test cases from several sources were used to verify GWBEM. These included simple problems where the analytical solutions were available and more complex problems where the FDM and FEM were used. Also, the results of problems solved using other models based on the BEM were compared with those from GWBEM. In each case, the results obtained using GWBEM were of equal or better quality than those from other methods or models when compared with theoretical results. Each of the test cases was run on an 8 MHz IBM AT compatible computer with 640K of memory, a math coprocessor, and a 32 megabyte hard disk. The time required to run the test cases is given for each.

The first test case used to verify the model GWBEM is shown in Figure 12. Several BEM researchers have used this simple problem as a measure of the accuracy of their particular BEM models; Mitra and Ingber (1987), Patterson and Sheikh (1984), and Brebbia (1978) to name a few. To carry on in this ritual, GWBEM was also applied to this problem. Brebbia's initial solution was greatly improved by the use of double nodes at the corners. Mitra and Ingber's solution used extra collocation points at the corners to resolve flow ambiguities there. Patterson and Sheikh's solution used their discontinuous elements, the same type of elements used in special cases by GWBEM. Interestingly enough, due to the way GWBEM defines a problem, this simple test case did not require the use of double nodes, extra collocation points, or discontinuous elements by GWBEM to obtain a solution. The solutions from all sources used the same boundary discretization of twelve elements.

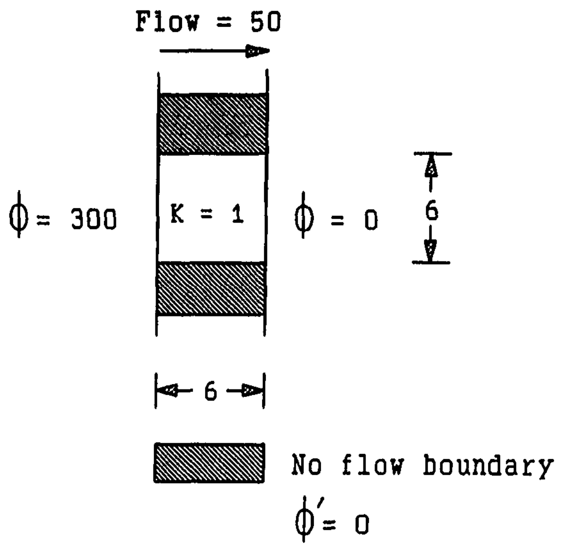

Figure 12: Test case 1 - Simple flow problem thru a rectangular prism (after Brebbia, 1978)

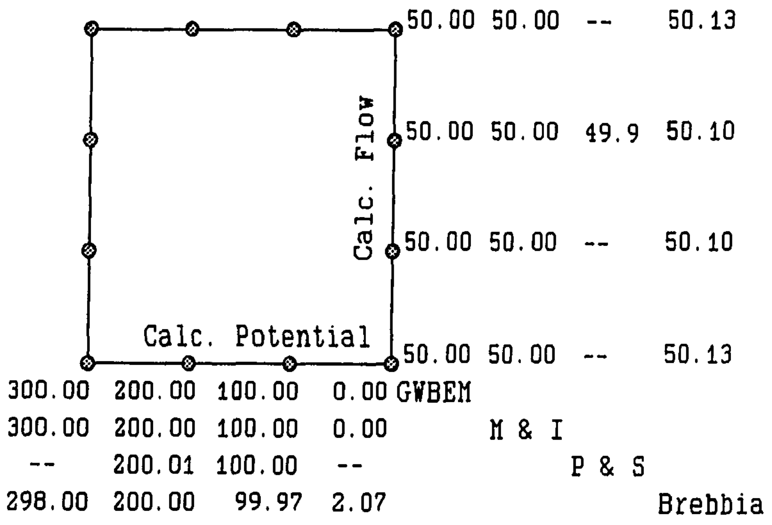
Calc. Flow
Calc. Potential
50.00 50.00 -- 50.13
50.00 50.00 49.9 50.10
50.00 50.00 -- 50.10
50.00 50.00 -- 50.13
300.00 200.00 100.00 0.00 GWBEM
300.00 200.00 100.00 0.00 M & I
-- 200.01 100.00 -- P & S
298.00 200.00 99.97 2.07 Brebbia

Figure 13: Comparison of numerical solution of test case 1 by various authors to that of GWBEM

Figure 13 compares the results obtained from each source. GWBEM provided better results than either Brebbia or Patterson and Sheikh, and comparable results to those of Mitra and Ingber. Mitra and Ingber's method involved extra collocation points at the corners, which resulted in a larger system to be solved than that generated by GWBEM. These extra collocation points are required to be outside of the problem domain and generate an extra equation for each extra point used. Also, good results using this method are very dependent upon the proper placement of these extra points, something which would be difficult to do in an automated fashion as was desired with GWBEM. Although the test case was small, if a larger problem were used, GWBEM would use a significantly less amount of computer memory than Mitra and Ingber's method for presumably similar results. The total time required to solve this problem was 2.7 seconds. Test case 1 was also solved by GWBEM using only 4 elements, or one element per side. The results were identical to those obtained from the twelve element discretization and matched the theoretical values exactly.

For this problem, the main difference between GWBEM and the other researchers' models is in the approach to the boundary unknowns at the corners. In the other methods, it was assumed that there were ambiguous flow definitions at the corners which resulted in two corner flow unknowns. What GWBEM does is account not only for the geometry at the corners but also the boundary conditions. For the sample problem, since the flow is known to one side of every corner node and since the potential is also defined at each corner node, there is in fact one unknown flow DOF at each corner. The system is assembled using this fact to produce an unknown vector which contains only those unknown flow DOFs at the corners.

Test case 1 showed the validity of GWBEM in providing boundary solutions for a simple flow problem. Larger, more complex test cases with different boundary conditions were used to further test GWBEM. Franke and Reilly (1987) tested the effects of applying different sets of boundary conditions to a groundwater flow system. The different flow systems are shown in Figure 14. System 1 has a constant head boundary specified at both ends of the domain, while the head along the upper and lower boundaries are specified as a linear variation from the left boundary to the right boundary. System 2 again has the left and right boundaries specified as constant head, while the upper and lower boundaries are specified as no flow boundaries. System 3 differs from system 2 in that the left boundary is a specified flow rather than a specified head boundary. These three boundary condition systems were analyzed by Franke and Reilly in three different experiments for a total of nine cases. Experiment A used a K of 2.0 ft/day, experiment B used a K of 4.0 ft/day, and experiment C used a K of 2.0 ft/day but with a discharge well located in the center of the domain. The flow rate of this well was 100 ft^3/day. In all cases, the flow was assumed confined and the medium was assumed to be isotropic and homogeneous. In all systems, the domain was 20 feet long and 8 feet wide.

Franke and Reilly used a finite-difference, square point-centered mesh with 81 × 33 nodes to solve the three groundwater systems, a total

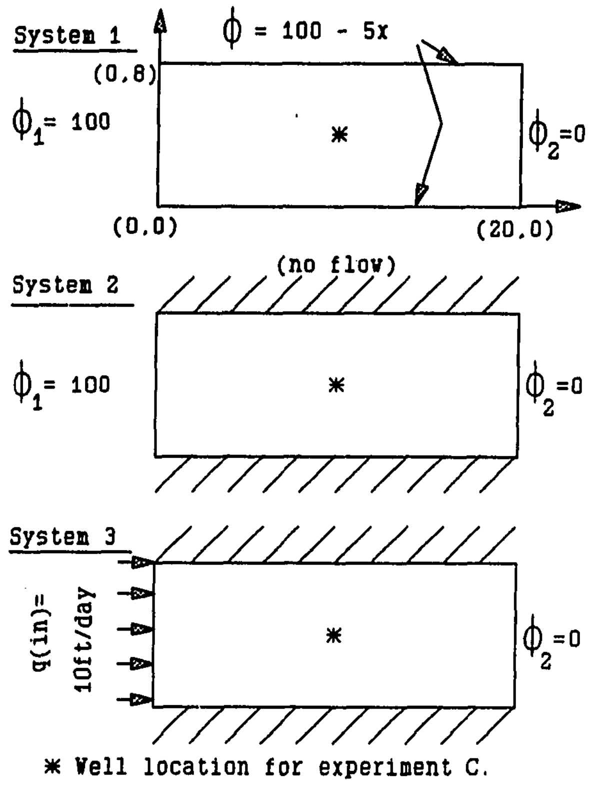

Figure 14: Three different flow systems investigated by Franke and Reilly (1987)

of 2673 nodes. To actually solve for the unknown boundary values using GWBEM, a total of 28 boundary nodes were spaced equally around the boundary every 2 feet. Sixty interior nodes were used to determine behavior of potential and flow in the interior with GWBEM. Including the well node for the C experiments, the total number of nodes required by GWBEM to obtain results of similar accuracy to those of Franke and Reilly was 89. Comparing the number of nodes required to adequately model the flow systems using the two methods (2673 versus 89) shows that GWBEM requires much less input than the finite-difference model used by Franke and Reilly. It took an average of 19.3 seconds to solve for all of the boundary and interior nodes for these problems on the test computer.

The results obtained from GWBEM for the nine cases are shown in Figures 15 thru 23. These figures show the potential surfaces calculated from the boundary and interior nodes and the flow vector calculated for each interior node. In each figure the direction for the flow vectors is from the asterisks to the triangles. The asterisks mark the location of the nodes used to calculate the interior values. The flow vectors in each plot are normalized so that the plots' maximum flow vectors are no longer than one tenth of the longest side of the flow domain. Based on the units of the problem, the units on the maximum flow vector shown at the top of each solution plot is feet/day. All other flow vectors are scaled accordingly. The value represented by the longest flow vector is given at the top of each figure. The potential value at each well is also listed in each of the figures for the C experiments.

The potential surfaces shown in Figures 15 thru 23 compared very closely with those of Franke and Reilly. Direct comparison of the actual numerical results was limited to certain potential and flow values on the boundary and the heads at the wells in the C experiments, as these were the only ones provided by Franke and Reilly. The direct comparisons which could be made are listed in Table 3. Another check on the validity of the results come from a simple mass balance of each. All cases for flow systems A and B balanced exactly as to inflow and outflow. The worst flow imbalance was found for the C flow systems and was 0.8

Table 3: Comparison of GWBEM and Franke and Reilly results

Exp.[a] No.	K (ft/d)	Head Left	Inflow Left	Head Right	Inflow Right	Inflow Top/Bot.	Head Well
FRA1	2	100.0[c]	80.0	b	b	b	
GWA1	2	100.0[c]	80.0	0.0	-80.0	0.0	
% Diff			0.0				
FRA2	2	100.0[c]	80.0	b	b	b	
GWA2	2	100.0[c]	80.0	0.0	-80.0	0.0[c]	
% Diff			0.0				
FRA3	2	100.0	80.0[c]	b	b	b	
GWA3	2	100.0	80.0[c]	0.0	-80.0	0.0[c]	
% Diff		0.0					
FRB1	4	100.0[c]	160.0	b	b	b	
GWB1	4	100.0[c]	160.0	0.0	-160.0	0.0	
% Diff			0.0				
FRB2	4	100.0[c]	160.0	b	b	b	
GWB2	4	100.0[c]	160.0	0.0	-160.0	0.0[c]	
% Diff			0.0				
FRB3	4	50.0	80.0[c]	b	b	b	
GWB3	4	50.0	80.0[c]	0.0	-80.0	0.0[c]	
% Diff		0.0					
FRC1	2	100.0[c]	82.5	b	-77.5	95.0	13.0[d]
GWC1	2	100.0[c]	82.7	0.0	-77.32	95.42	13.2
% Diff			0.24		-0.23	0.44	1.5
FRC2	2	100.0[c]	130.0	b	-30.0	b	-7. [d]
GWC2	2	100.0[c]	130.07	0.0	-29.93	0.0[c]	-6.9
% Diff			0.06		-0.25		1.4
FRC3	2	38. [d]	80.0[c]	b	20.0	b	-38. [d]
GWC3	2	37.4	80.0[c]	0.0	20.15	0.0[c]	-38.1
% Diff		1.6			0.74		0.26

a FRxx - Franke and Reilly results, GWxx - GWBEM results.

b not provided by Franke and Reilly.

c specified conditions.

d specified as approximate values by Franke and Reilly.

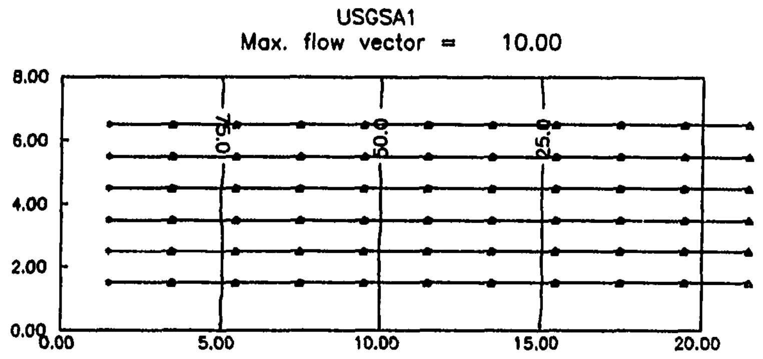

Figure 15: GWBEM results of USGS test case A1, K = 2

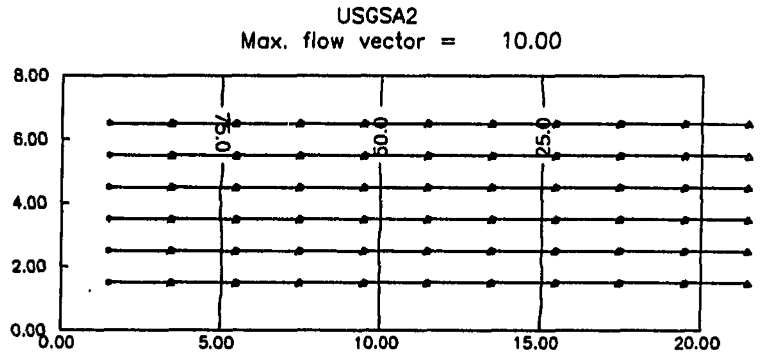

Figure 16: GWBEM results of USGS test case A2, K = 2

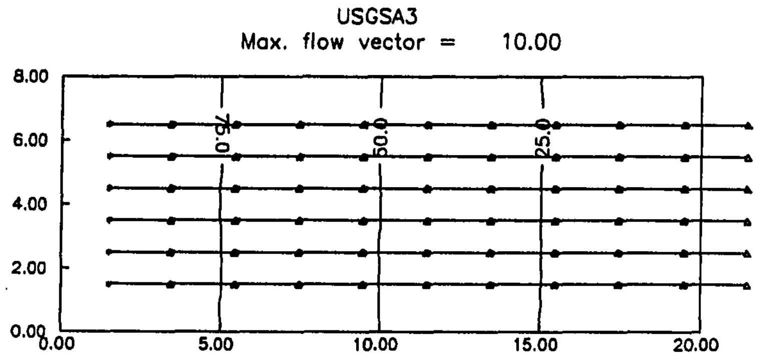

Figure 17: GWBEM results of USGS test case A3, K = 2

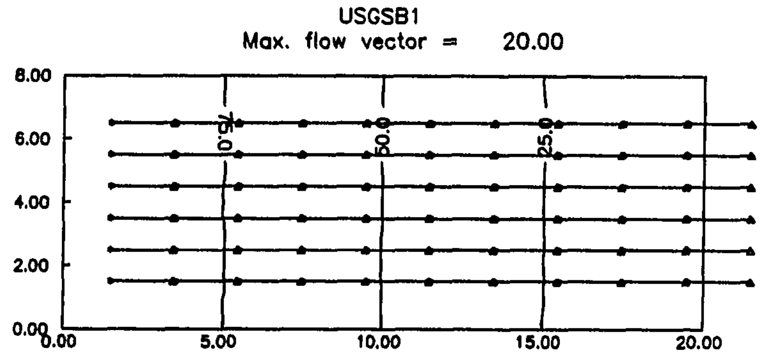

Figure 18: GWBEM results of USGS test case B1, K = 4

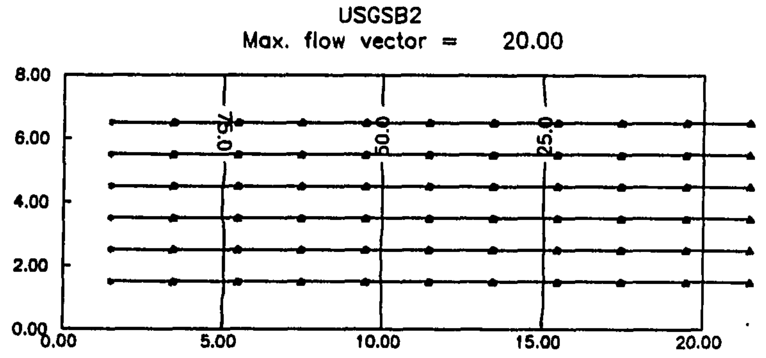

Figure 19: GWBEM results of USGS test case B2, K = 4

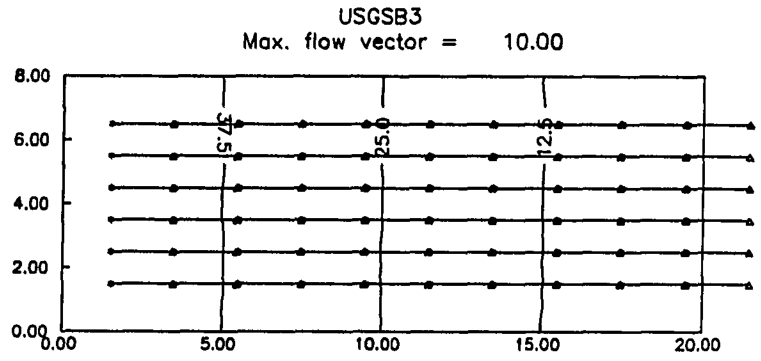

Figure 20: GWBEM results of USGS test case B3, K = 4

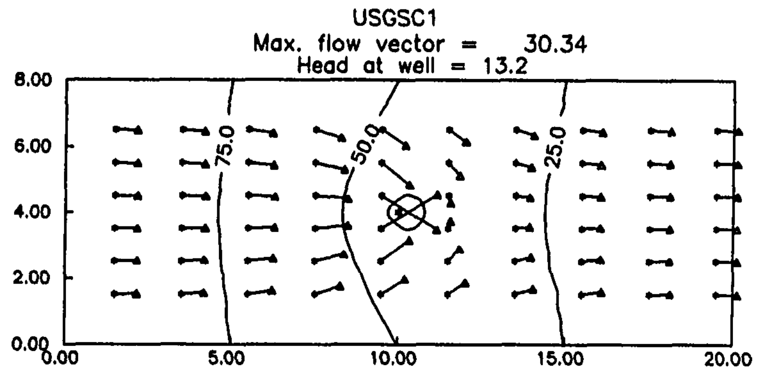

Figure 21: GWBEM results of USGS test case C1, K = 2

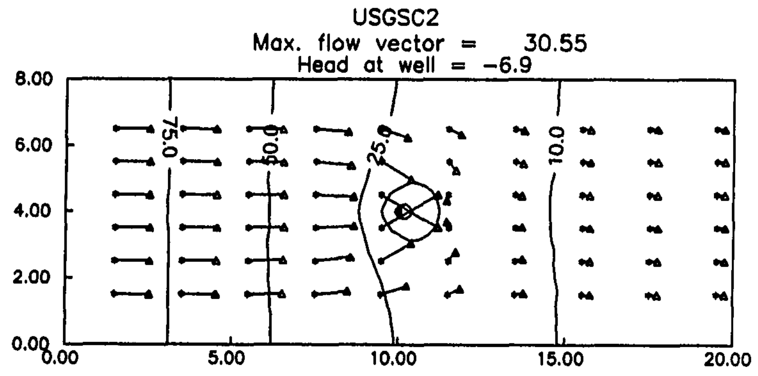

Figure 22: GWBEM results of USGS test case C2, K = 2

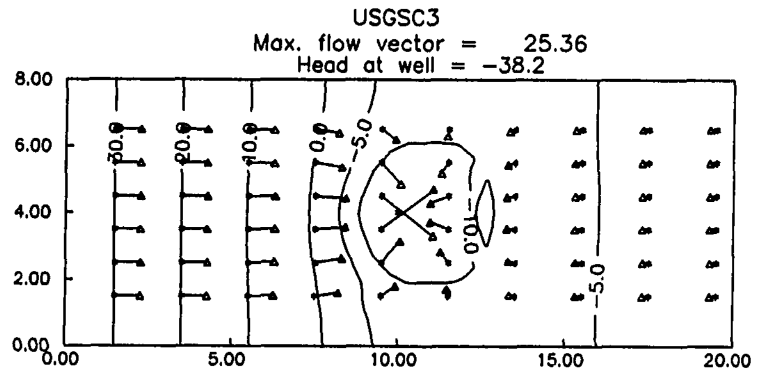

Figure 23: GWBEM results of USGS test case C3, K = 2

percent. The C1 experiment had a total calculated inflow of 100.8 cfs/day balanced against a specified well outflow of 100.0 cfs/day.

The interior flow vectors were calculated with GWBEM only, so that no direct comparison of the interior flow behavior could be made with the results of Franke and Reilly. However, the flow vectors found using GWBEM agree with the expected flow determined from the potential contours calculated with GWBEM. Overall, the numerical results which could be compared were identical. The C experiments were slightly different between sets, with the maximum difference being 1.5 percent. Many of the values provided by Franke and Reilly found in Table 3 were given as approximate only. The percent differences for these values are approximate and for rough comparison only.

Table 4: Comparison of drawdown results for different well flow rates for C experiments

	Well discharge (ft^3/d)	Drawdown (ft)		
		Flow system 1	Flow system 2	Flow system 3
FR	1	0.37	0.57	0.88
GW		0.37	0.57	0.88
FR	10	3.7	5.7	8.8
GW		3.7	5.7	8.8
FR	100	37	57	88
GW		37	57	88

Franke and Reilly ran another set of experiments based on the C experiments. In this set, they altered the flow rate of the well to determine what effects this had on the calculated drawdown at the well. These cases were also analyzed using GWBEM and the results are compared in Table 4. As can be seen, for the number of significant digits provided by Franke and Reilly, the results were identical.

Several different test cases were run to test the validity of GWBEM with multi-zone systems. The first test case run was a simple three zone system shown in Figure 24. The results from GWBEM are compared against theoretical values from Bolteus and Tullberg (1985) in Figure 25. They solved for the theoretical temperature profile using a one-dimensional system of equations. As can be seen in the figure, the results from GWBEM compared very well with the theoretical ones. The number of nodes used to model this problem with GWBEM was 11 nodes along the top and

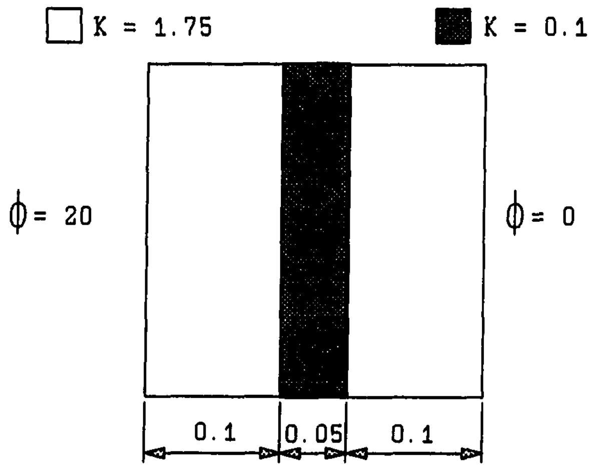

Figure 24: Simple multi-zone system

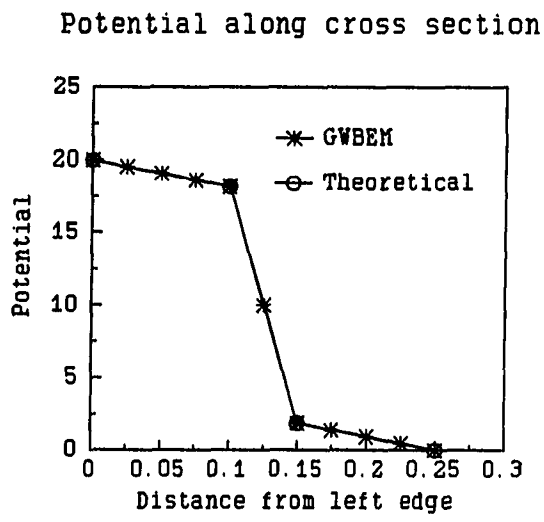

Figure 25: Comparison of theoretical and calculated results of simple multi-zone system

bottom and 10 nodes along each side and along the interior boundaries. The run time for this problem was 63 seconds.

Another multi-zone system presented by Brebbia and Chang (1985) was analyzed using GWBEM. This system is shown in Figure 26. It consists of three zones under a dam with sheet piles. Zones were used to aid in the modeling of the sheet piles, which was done by the use of special elements along the interior boundaries between the zones. These special elements had the flux across them set equal to zero, so that the potential was the only unknown. Brebbia and Chang used both a BEM model using 72 constant value boundary elements and an FEM model with 68 nodes and 95 elements. The system was solved with GWBEM using 96 global nodes with 72 global linear elements in approximately 3.7 minutes. The potential

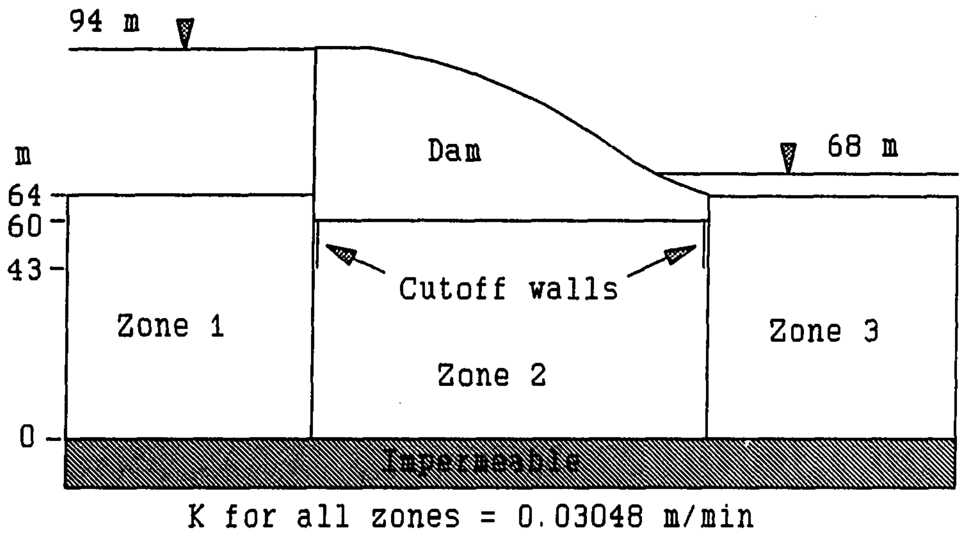

Figure 26: Three zone system with cutoff walls (Brebbia and Chang, 1985)

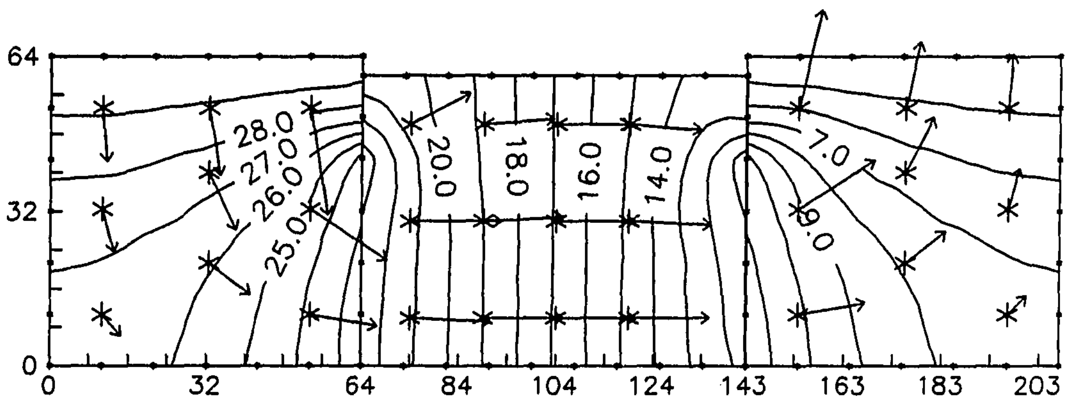

Figure 27: Potential values and flow vectors of three zone sheet pile dam problem calculated using GWBEM

contours and flow vectors from GWBEM for this problem are shown in Figure 27. This figure agreed very closely with that of Brebbia and Chang and the flow patterns are realistic for such a system.

Figure 28 compares the pressure head on the base of the dam as calculated with GWBEM and the constant element BEM model of Brebbia and Chang. As seen in the figure, the curvilinear behavior of the pressure

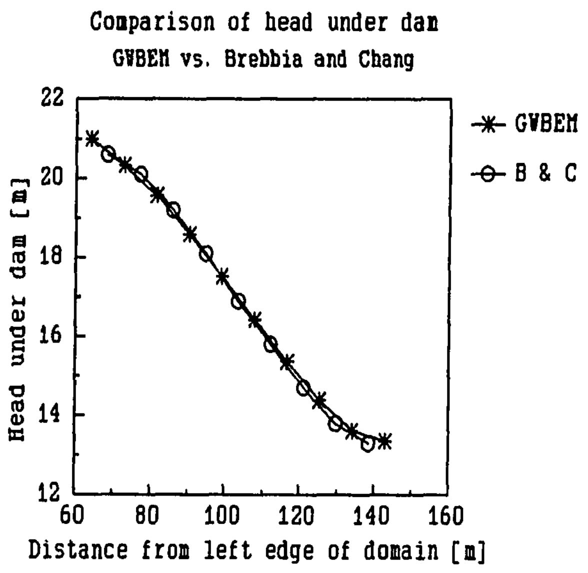

Figure 28: Comparison of GWBEM and Brebbia and Chang (1985) calculated heads under base of dam

distribution under the dam was properly simulated by GWBEM and verified by the Brebbia and Chang solution. Here is a case where the number of elements used to define a boundary was critical for the proper solution. If too few linear elements had been used, the curvilinear behavior of the pressure distribution beneath the dam would not have been suitably established. Conversely, to more precisely determine the pressure distribution, more linear elements could have been used along the boundary beneath the dam. More research is needed to establish how many elements would be needed to allow for the most efficient, yet accurate, solution for a given problem.

A final problem was used to further validate the numerical solutions found using GWBEM as well as to provide a test of the solution of problems with multiple wells in a domain. The problem consisted of a sub-irrigation system with an inflow and an outflow tile and a specified evaporation rate along the top boundary. An analytical solution to the problem had been worked out by Kirkham and Horton (1989) and agreed with the numerical results obtained from GWBEM. Several of the more exacting details of the problem solution compared very well between the analytical and numerical methods, and further validated both. Unfortunately, the results of the analytical solution had not been officially published at the time of this writing. To protect the interests of the authors of the analytical solution, neither the numerical nor the analytical results will be included. What is important, however, is that GWBEM properly solved the problem with a relatively small amount of effort.

Conclusions

The utilization of GWBEM to the various groundwater problems discussed here show the program to be a viable analysis tool for many groundwater flow situations. Its ability to calculate system response for multi-zone groundwater systems, flow systems with multiple wells, cutoff walls, and interior flow velocities and potentials have been verified. Given adequate discretization of a problem boundary, GWBEM has also been shown to accurately simulate non-linear behavior using linear elements.

CONCLUSIONS AND RECOMMENDATIONS

Conclusions

The details of a simple yet effective boundary element model for solving many groundwater flow problems involving non-homogeneous media with wells have been presented. The micro-computer program GWBEM has been shown to produce accurate results with a modest amount of user input, especially true when compared with the input requirements of other numerical techniques such as the finite-element and finite-difference methods.

The details furnished by the program listing provide a good foundation for the development of more advanced groundwater models along with information concerning the actual implementation of the BEM on computers. Unlike other numerical methods such as the FDM and FEM, such information was sorely lacking in the literature, at least at the outset of the model development. The use of Pascal, which is known for its readability and structured constructs, makes for relatively easy understanding and modification, as well as portability between different machines.

Several problems encountered with the application of the BEM, such as the coupling and assembly of multi-zone systems and the solution of ambiguous corners on a boundary have been dealt with and implemented in the model for an overall improvement of the method. An analytical integration scheme for linear elements which avoids the potential errors due to numerical integration has been used throughout the program, including the implementation of discontinuous elements. The use of higher order elements would necessitate numerical integration schemes, but the use of such elements is of questionable value considering the quality of most groundwater data.

Many of the more tedious details concerning the use of the BEM have been automated in the model, particularly those involving the numbering of nodal DOFs, use of discontinuous elements, and zonal connectivity. Through the use of dynamic memory structures and virtual arrays, large problems may be solved on machines with small core memory with little sacrifice in speed. The result is a simple to use model which frees the user to focus on the problem being modeled rather than the intricacies of the model used to solve the problem.

Recommendations

Several improvements could be made to the model in its current form. The most notable improvement would be the ability to solve for unsteady flow problems. Most approaches to solving these problems with the BEM require integration over the entire domain. This requirement diminishes one of the main advantages of the BEM, namely the reduction of a problem's dimension. Several researchers have proposed methods to move the domain integration to the boundary thereby maintaining an advantage of the BEM, but these methods are currently not implemented in this model.

The ability to model unsaturated flow would be valuable, but because of the variable conductivity which occurs with such flow, many problems would be encountered with a BEM solution. Further research is required. The application of the BEM to ephemeral stream-aquifer interaction would be useful, particularly during the transition from a connected stream to a disconnected one. A mechanism for the determination of state of the stream-aquifer connection would increase the model utility considerably.

REFERENCES

Alarcon, E., A. Martin, and F. Paris. "Boundary elements in potential and elasticity theory." Computers and Structures 10 (1979), 351-362.

Banerjee, P. K., ed. Developments in Boundary Element Methods, Vol. 1. Englewood Cliffs, New Jersey: Applied Science Publishers, Inc., 1979.

Banerjee, P. K., ed. Developments in Boundary Element Methods, Vol. 2. Englewood Cliffs, New Jersey: Applied Science Publishers, Inc., 1982.

Banerjee, P. K., ed. Developments in Boundary Element Methods, Vol. 3. Englewood Cliffs, New Jersey: Applied Science Publishers, Inc., 1984.

Banerjee, P. K., R. Butterfield, & G. R. Tomlin. "Boundary Element Methods for Two-Dimensional Problems of Transient Ground Water Flow." International Journal of Analytical and Numerical Methods in Geotech. 5 (1981), 15-31.

Bialecki, R. and A. Nowak. "Boundary Value Problems in Heat Conduction with Non-Linear Material and Non-Linear Boundary Conditions." Applied Mathematical Modelling 5 (1981), 417-421.

Bolteus, L. and O. Tullberg. "Boundary element method applied to two-dimensional heat conduction in non-homogeneous media." In Boundary element research. Ed. Carlos A. Brebbia. Southampton: Computational Mechanics, 1985.

Brebbia, Carlos A. The Boundary Element Method for Engineers. New York: Halstead Press, 1978.

Brebbia, Carlos A., ed. Boundary Elements VI. Proceedings of the Sixth International Conference on Boundary Element Methods on the Queen Elizabeth 2, July 1984. Berlin: Springer-Verlag, 1984.

Brebbia, Carlos A. "The Boundary Element Method in Engineering Practice." In Boundary Element Research, pp. 1-10. Southhampton, England: CML Publications, 1985.

Brebbia, Carlos A. and O. V. Chang. "Boundary Elements Applied to Seepage Problems in Zoned Anisotropic Soils." In Boundary Element Research, pp. 70-80. Southhampton, England: CML Publications, 1985.

Brebbia, Carlos A. and P. Skerget. "Diffusion-Convection Problems Using Boundary Elements." Advances in Water Resources 7 (1984), 50-57.

Carr, Robert S. "A Finite Element Stream-Aquifer Model." Ph.D. Dissertation, Iowa State University, Ames, Iowa, 1985.

Cheng, Alexander H.-D. "Darcy's Flow with Variable Permeability: A Boundary Integral Solution." Water Resources Research 20 (1984a), 980-984.

Cheng, Alexander H.-D. "Heterogeneities in Flows Through Porous Media by the Boundary Element Method." In Topics in Boundary Element Research. Ed. C. A. Brebbia. Berlin: Springer-Verlag, 1984b.

Dillon, P. J. and J. A. Liggett. "An Ephemeral Stream-Aquifer Interaction Model." Water Resources Research 19 (1983), 621-626.

Forsythe G. E., M. A. Malcolm and C. E. Moler. Computer Methods for Mathematical Computations. Englewood Cliffs, N.J: Prentice-Hall, Inc., 1977.

Franke, O Lehn and Thomas E. Reilly. "The effects of boundary conditions on the steady-state response of three hypothetical ground-water systems - results and implications of numerical experiments." U.S.G.S. Water-Supply Paper No. 2315. U.S. Department of the Interior, U.S. Geologic Survey, Washington D.C., 1987.

Freeze, R. Allan and John A. Cherry. Groundwater. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979.

Hillel, Daniel. Introduction to Soil Physics. New York: Academic Press, 1982.

Holcomb Research Institute. Letter. Holcomb Research Institute, Indianapolis, Indiana, June 10, 1986.

Kaplan, Wilfred. Advanced calculus. 3rd ed. Reading, Massachusetts: Addison-Wesley Publishing Co., 1984.

Kellogg, Oliver D. Foundations of Potential Theory. Berlin: Verlag Von Julius Springer, 1929.

Kemblowski, Marian. "Salt-water Upconing Under a River - a Boundary-Element Solution." In Boundary Elements VI:. Proceedings of the Sixth International Conference on Boundary Element Methods on the Queen Elizabeth 2, July 1984. Edited by C.A. Brebbia, 4:29-4:39. Berlin: Springer-Verlag, 1984.

Kirkham, Don and Bob Horton. Personal communication. Agronomy Department, Iowa State University, Ames, Iowa, 1989.

Lafe, Olurinde E., J. Sergio Montes, Alexander H.D. Cheng, James Liggett, and Philip Liu. "Singularities in Darcy Flow through Porous Media." ASCE Journal of the Hydraulics Division, 106 (1980), 977-997.

Liggett, James A. "Locations of Free Surface in Porous Media." ASCE Journal of the Hydraulics Division 103 (1977), 353-365.

Liggett, James A. and Philip L-F. Liu. The Boundary Integral Equation Method for Porous Media Flow. London: George Allen & Unwin, 1983.

Mackerle, J. and C. A. Brebbia. The Boundary Element Reference Book. Berlin: Springer-Verlag, 1988.

Mitra, Ambar K. and Marc S. Ingber. "Resolving difficulties in the BIEM caused by geometric corners and discontinuous boundary conditions." Iowa State University Engineering Research Institute Preprint 87238. Iowa State University, Ames, Iowa, 1987.

Patterson, C. and M. A. Sheikh. "Discontinuous boundary elements for heat conduction." In Numerical Methods in Thermal Problems - II. Ed. R. Lewis, W. Morgan, and B. A. Schrefler. Swansea, U.K.: Pineridge Press, 1981.

Patterson, C. and M. A. Sheikh. "Interelement continuity in the boundary element method." In Topics in Boundary Element Research, Vol. I. Ed. C. A. Brebbia. Berlin: Springer-Verlag, 1984.

Pinder, G. F. and W. G. Gray. Finite Element Simulation in Surface and Sub-Surface Hydrology. New York: Academic Press, 1977.

Radojkovic, Miodrag and Josip Pecaric. "Boundary element analysis of flow in aquifers." Finite Elements in Water Resources. Ed. J.P. Laible, C.A. Brebbia, W. Gray, and G. Pinder. Berlin: Springer-Verlag, 1984.

Rubin, J. "Theoretical Analysis of Two-Dimensional, Transient Flow of Water in Unsaturated and Partly Unsaturated Soils." Soil Science Society of America Proceedings, 32 (1968), 607-15.

Rudolphi, Thomas. Personal communication. Department of Engineering Science and Mechanics, Iowa State University, Ames, Iowa, 1988.

Segerlind, Larry J. Applied Finite Element Analysis. New York: John Wiley and Sons, 1984.

Shapiro, Allen M. and Johan Andersson. "Simulation of Steady-State Flow in Three-Dimensional Fracture Networks Using the Boundary-Element Method." Advances in Water Resources 8 (1985), 106-110.

Tolikas, D., P. Latinopoulos and J. Ganoulis. "Boundary Elements and Non-Linear Programming in Aquifer Management." In _Boundary Elements V. Proceedings of the Fifth International Conference on Boundary Element Methods, Hiroshima, Japan, November 1983_. Edited by C. A. Brebbia, T. Futagami and M. Tanaka, 95-101. Berlin: Springer-Verlag, 1983.

ACKNOWLEDGEMENTS

Financial support for this research came from the U.S. Department of Agriculture National Needs Fellowship program and the Iowa State Water Resources Research Institute of Iowa State University. Additional support came from the Engineering Research Institute of Iowa State University.

Many thanks are due to Dr. T. Al Austin, who oversaw this project and offered assistance along the way. His continual support, both scholastically and financially, were above and beyond the call and smoothed many of the rough spots. His efforts were greatly appreciated. I am grateful to Dr. Robert Baumann, who also provided financial support at the end of this venture.

I would like to express my esteem for the late Dr. Paul W. Peterson and his wife Mary. Better friends are rarely found. Much appreciation goes to Dave Schoeller for his help and most of all for his companionship. My new son Kyle provided the requisite hours of diversion in the course of completing this project.

Finally, I would like to thank my wife Marcia, who always knew when I needed a break. She put up with many a long day and distracted weekend during my graduate work . . . although she has no one to blame but herself since she didn't shoot me when I came back for more, as she said she would the last time we went through this.

APPENDIX A: PASCAL LISTING OF GWBEM

```
program GWBEM;
  {----------------------------------------------------

                Program GWBEM - Main module

    A general purpose groundwater model based on the boun-
    dary element method.  Capable of solving two-
    dimensional, steady-state problems.  Non-homogeneous
    domains dealt with by defining multiple zones of dif-
    fering but homogeneous hydraulic conductivities.  Also
    allows for the inclusion of wells as point sources of
    sinks, with either specified head or drawdown.  Cutoff
    walls accommodated by use of special elements.


    Copyright (c) 1989, Mark A. Liebe and Iowa State
    University

    ALL RIGHTS RESERVED

    This program is intended for non-commercial use only,
    and may not be used for any other purpose without the
    expressed written consent of the author and Iowa State
    University.

    Language:  Turbo Pascal V5.0.

    Last modified : 4/15/89

  ----------------------------------------------------}

Uses Crt,                                        { System unit }
     B7DEF,                                      { GWBEM unit  }
     B7File,                                     { GWBEM unit  }
     B7Utils,                                    { GWBEM unit  }
     B7Prep,                                     { GWBEM unit  }
     B7Int,                                      { GWBEM unit  }
     B7Solver,                                   { GWBEM unit  }
     B7Error;                                    { GWBEM unit  }

Procedure Initialize_Boundary_Arrays;
var Zone : byte;
BEGIN
   New(GNode);
   Fillchar(GNode^,Sizeof(GNode^),0);
   New(GElem);
   Fillchar(GElem^,Sizeof(GElem^),0);
   FillChar(ZoneD, sizeof(ZoneD),0);
   New(NodeF);
   Fillchar(NodeF^,Sizeof(NodeF^),0);
   for Zone := 1 to Max_Zones do
   with ZoneD[Zone] do
   begin
     Elements := nil;
     Walls := nil;
     IntNodes := nil;
     TempElList := nil;
   end;
END;

procedure writeGridFile;
  {- prints out solution grid files for contouring }
var
   Phifile,
   DPXfile,
   DPYfile : text;
   PhiFileName,
   DPXFileName,
   DPYFileName: string[30];
begin
   { write out phi solution }
   PHIFileName := ForceExtension(OutFileName,'PHI');
   assign(PHIFile, PHIFileName);
   rewrite(PHIFile);
   For J := 1 To Num_INodes do Writeln(PhiFile,INode^[J].X:9:4,' ',
                                        INode^[J].Y:9:4,' ',
                                                INode^[J].Phi:9:4);
    For J := 1 To Num_BNodes do Writeln(PhiFile,BNode^[J].X:9:4,' ',
                                                BNode^[J].Y:9:4,' ',
                                                BNode^[J].Phi:9:4);
    For J := 1 To Num_SNodes do Writeln(Phifile,SNode^[J].X:9:4,' ',
                                                SNode^[J].Y:9:4,' ',
                                                SNode^[J].Head:9:4);
    Close(PHIFile);

    if Num_INodes > 0 then
    begin
      DPXFileName := ForceExtension(OutFileName,'DPX');
      DPYFileName := ForceExtension(OutFileName,'DPY');
      assign(DPXFile, DPXFileName);
      rewrite(DPXFile);
      assign(DPYFile, DPYFileName);
      rewrite(DPYFile);
      { write out DPhiX solution }
      For J := 1 To Num_INodes do Writeln(DPXfile,INode^[J].X:9:4,' ',
                                                INode^[J].Y:9:4,' ',
                                                (-INode^[J].DPhiX*Conductivity):9:4);
      { write out DPhiY solution }
      For J := 1 To Num_INodes do Writeln(DPYfile,INode^[J].X:9:4,' ',
                                                INode^[J].Y:9:4,' ',
                                                (-INode^[J].DPhiY*Conductivity):9:4);

      Close(DPXFIle);
      Close(DPYFile);
    end;
  end; { of WritegridSol }

  procedure Solve_System;
  {- calls virtual array solver }
  var Condition_Num : float;
  begin
    if not Solver(DOFCount, K, F, Condition_Num) then
      ShowError('System Singular', True);
    writeln(OutFile,'***>>> CONDITION NUMBER : ',Condition_Num);
  end; { proc Solve_System }

  {---->>>Main<<<-----------------------------------------------------------------}
  begin
     Clrscr;
     HandleInputParams;
     Open_Text_file(Infile, Infilename, Rd);
     Open_Text_File(OutFile, Outfilename, Wrt);
     Initialize_Boundary_arrays;
     StartTimer('Getting data from input file');
     Get_Data;
     StopTimer('get data from input file');
     StartTimer('Preparing system for integration');
     Prep_System;
     StopTimer('prepare system for integration');

     { Solve for unknown boundary conditions }
     StartTimer('Integrating boundary equations');
     Integrate_Boundary;
     StopTimer('integrate boundary equations');

     Solve_System;
     PlaceSolution;
     WriteBoundarySolution;
     WriteSourceSolution;
     { Solve for interior unknowns }
     StartTimer('Integrating for interior node solutions');
     Integrate_Interior;
     StopTimer('integrate for interior node solutions');
     WriteInteriorSolution;
     DisposeWorkArrays(True);
     WriteGridFile;
     Close(Infile);
     Close(Outfile);
  end. { of Program GWBEM <******************************<< }
```

```
Unit B7DEF;
  {----------------------------------------------------


                    Program GWBEM - Unit B7DEF

    Contains global variable definitions and types for
    GWBEM.


    Copyright (c) 1989, Mark A. Liebe and Iowa State
    University

    ALL RIGHTS RESERVED

    TPRArray and TPVArray units copywrite (c) 1987 by
    TurboPower Software.  Part of Turbo Professional
    Programmer's Toolbox V4.0.  For information, contact:

        TurboPower Software
        3109 Scotts Valley Drive, Suite 122
        Scotts Valley, CA  95066
        (408) 438-8608

    last modified : 12/12/88 11:14 AM

  -----------------------------------------------------}

interface

uses TPRArray,
     TPVArray;

Const  Max_BNodes = 4000;
       Max_Elements = 1400;
       Max_Zones = 20;
       ZeroTol = 5.0e-8;
Type
   Float = double;
   Coordinate = single;
   Direction = (x,y);
   GlobalDOF = word;
   NodeNumber = word;
   ElementNumber = integer; { want to be +/- here }

   NodeType = (Boundary,
               Interior,
               Source);
   BNodeType = (Phi, dPhi, Intr, Wall);  { initial node assignments }

   ElementSFType = (Reg,       { regular element shape function }
                    Disc,      { full discontinuous element shape function }
                    LDisc,     { discontinuous leading element shape func }
                    TDisc);    { discontinuous trailing element shape func }

   CoordinatePair = array[direction] of Coordinate;
   CoordPtr     = ^CoordList;
   CoordList    = array[1..Max_BNodes] of CoordinatePair;
   NodeFlags    = ^FlagList;
   Flaglist     = array[1..Max_BNodes] of Boolean;

   ElementNode = record
                   Node : NodeNumber;
                   Dof : GlobalDof;
                   Phi, DPhi : float;
                   NType : BNodeType;
                 end;

   ElementType = record      { record setup for elements - linear for now}
                   A,B : ElementNode;
                   ElSFType : ElementSFType;
                 end;
   ElementPtr  = ^ElementList;
   ElementList = array[1..Max_Elements] of ElementType;

   TempNode = record  { record for nodal values of temp element list for each zone }
```

```
                 Coord : CoordinatePair;
                 KnownVal   : float;
               end;

    TempEl = record
               A, B : TempNode;
               ElType : BNodeType;
             end;

    SNodeType = (SFlow, SHead);
    SourceNode = record          { source node record }
                   Coord : CoordinatePair;
                   Dof : GlobalDof;
                   Radius,
                   Head,
                   Flow         : float;
                   SourceType   : SNodeType;
                 end;

    InteriorNode = record          { interior node record }
                    Coord : CoordinatePair;
                    Phi, DPhiX, DPhiY : float;
                   end;

    ZoneRec = record
                Kx, Ky, ThetaX : float;
                NumElems, NumWells, NumInt, StartRow, NumDOFs : word;
                Elements,  { tparray of Element Numbers }
                Wells,     { tparray of Sourcenode }
                IntNodes,  { tparray of InteriorNode }
                TempElList:{ tparray of TempElements }
                           TPRArray.TpArray; { RAM based dynamic arrays }
              end;
    ZoneList    = array[1..Max_Zones] of ZoneRec;

    File_Name   = String[40];
    File_Ext    = string[3];
    Titlestring = String[80];


 const BTypeStr :array[BNodeType] of string[4] = ('Phi ','DPhi','Intr','Well');
       STypeStr :array[SNodeType] of string[4] = ('Flow','Head');
       ElSFTypeStr:array[ElementSFType] of string[5]    = ('Reg  ','Disc ',
                                                         'LDisc','TDisc');
       HMatExtStr : string[3] = 'HMT';
       FVecExtStr : string[3] = 'FVC';

 var
   GNode : CoordPtr;   { RAM pointer array of global node coordinates }
   GElem : ElementPtr; { RAM pointer array of global element definitions }
   ZoneD : ZoneList;   { RAM array of zone definitions }
   NodeF : NodeFlags;  { RAM array of node flags }

   H        : TPVArray.TPArray;  { full matrices }
   F,
   RHS      : TPVArray.TPArray;  { vectors }
   SinPsi,
   CoSinPsi : float;
   Conductivity  : array[1..Max_Zones] of float;
   I,
   J,
   Zone,
   Num_Zones,
   Num_BNodes,           { total global boundary nodes }
   Num_Boundaries: word; { total master boundary elements }
   DOFCount : GlobalDof; { Global DOF counter }
   Title,
   Param1,
   Param2   : string[79];
   NumParams : byte;
   Outfile,
   InFile : text;
   InFileName,
   OutFileName : File_Name;
{=============================================================================}
Implementation
```

```
begin
  TPVArray.RangeCheck := True;
  TPRArray.RangeCheck := True;
end. ( of unit B7DEF <*****************************<< }
Unit B7Utils;
  {------------------------------------------------------

                  Program GWBEM - Unit B7UTILS

    Contains miscellaneous routines for use with GWBEM.

    Copyright (c) 1989, Mark A. Liebe and Iowa State
    University

    ALL RIGHTS RESERVED

    TPString, TPDos, TPCrt, TPWindow units copywrite (c)
    1987 by TurboPower Software.  Part of Turbo Profes-
    sional Programmer's Toolbox V4.0.  For information,
    contact:

        TurboPower Software
        3109 Scotts Valley Drive, Suite 122
        Scotts Valley, CA  95066
        (408) 438-8608

   last modified : 12/01/88  9:56 AM
  ------------------------------------------------------}

interface

uses B7Def,
     Trig;

procedure HandleInputParams;
  {- manipulates input params and sets infile & outfile names }

function ATAN2 ( Y, X : float) : float;
  {- Function which returns the properly signed value of the angle given by  }
  {  the slope provided.  Comparable to the FORTRAN external ATAN2.  10/8/87 }

function radius(X1, Y1, X2, Y2: float) : float;
   {- routine to calculate distances between given points }

Function Pow(Base,Exponent : float):float;
  {- returns Base^Exponent }

procedure GetLocalCoords(Source,          { Source point }
                         Field1,          { Field Points }
                         Field2: CoordinatePair;
                     var  Normal, Local1, Local2, SinPsi, CoSinPsi : float);
  {- routine to calculate local coordinates.  Clockwise is + }

Function Krndlt(I, J : Integer): single;
  {- Kronecker delta function for two indices }

Function WrapWord(MaxIndex: word; Index : word) : word;
  {- returns word index wrapped properly given Maxindex }

function WrapEIndex(Index, TotalElems: ElementNumber) : ElementNumber;
  {- returns proper index of element number if at either end of index list }

Procedure Sign(Var A, B : float);
  {- anelogous to FORTRAN sign routine }

Procedure GetAlpha(L, M, H: CoordinatePair; Var Alpha : float);
  {- returns value of angle subtended by line L-M-H }

function CheckZeroFloat(value : float): float;
  {- zero's out near zero values }

procedure StartTimer(Message : TitleString);
  {starts timer and write message to screen}

procedure StopTimer(Message : TitleString);
  {stops timer and writes time elapsed & message to output file}

{=========================================================================}

implementation
```

```
uses TPString,
     TPDos,
     TPCrt,
     TPWindow,
     B7Error;

const MessageAttr = $1E;

Type   Component = (i,j,k);
       Vector = array[i..k] of float;

var StartTime, StopTime : longint;
    MesAttr : byte;
    MesWindow : WindowPtr;

procedure HandleInputParams;
  {- manipulates input params and sets infile & outfile names }
begin
  Case ParamCount of
  0  : begin
         Write('Name of input file : ');
         Readln(Infilename);
         Writeln;
         Write('Name of Output file : ');
         Readln(Outfilename);
       end;
  1  : begin
         InFileName := DefaultExtension(ParamStr(1),'dat');
         OutFileName := ForceExtension(InFileName,'out');
       end;
  2  : begin
         InFileName := DefaultExtension(ParamStr(1),'dat');
         OutFileName := ForceExtension(ParamStr(2),'out');
       end;
  end;
end; { proc HandleInputParams }

{---->>>ATAN2.INC<<<-------------------------------------------------------}
{    Function which returns the properly signed value of the angle given by  }
{    the slope provided.  Comparable to the FORTRAN external ATAN2.  10/8/87 }
{---------------------------------------------------------------------------}
   function ATAN2 ( Y, X : float) : float;
   const Zero = 1.0E-8;

   var flag : byte;
       temp, sign : float;
   begin
      if abs(X) < Zero then X := 0.0;
      if abs(Y) < Zero then
      begin
         Y := 0.0;
         sign := 1.0;
      end
      else sign := Y/abs(Y);
      if x = 0.0 then temp := (PI / 2) * sign
      else
      begin
         temp := arctan(Y/x);
         if Y <> 0.0 then
         begin
            if X < 0.0 then temp := PI * sign + temp;
         end
         else
            if X < 0.0 then temp := PI;
      end;
      ATAN2 := temp;
   end; { ATAN2 function }

{->>>Radius<<<--------------------------------------------------------------}
{  Calculates the distance between two points in 2 dimensional space. 10/14/87}
{---------------------------------------------------------------------------}
   function radius(X1, Y1, X2, Y2: float) : float;
   {  routine to calculate distances between given points }
   var X_Diff, Y_Diff : float;
   begin
```

```
      X_Diff := X2 - X1;
      Y_Diff := Y2 - Y1;
      radius := sqrt( X_Diff * X_Diff + Y_Diff * Y_Diff);
   end; { of radius function }

{---->>>GetlCoord.INC<<<-------------------------------------------------}
{    Note : Assumes positive local direction is CLOCKWISE around boundary   }
{    Modified 09/28/88.                                                     }
{-----------------------------------------------------------------------}
procedure GetLocalCoords(Source,          { Source point }
                         Field1,          { Field Points }
                         Field2: CoordinatePair;
                    var  Normal, Local1, Local2, SinPsi, CoSinPsi : float);
  {- routine to calculate local coordinates.  Clockwise is + }
var  Element_Length : float;
begin
  Element_Length := radius(Field2[X],Field2[Y],Field1[X],Field1[Y]);
  CoSinPsi := (Field2[X] - Field1[X])/Element_Length;
  SinPsi := (Field2[Y] - Field1[Y])/Element_Length;
  Local1 := (Field1[Y] - Source[Y]) * SinPsi + (Field1[X] - Source[X]) * CoSinPsi;
  Local2 := (Field2[Y] - Source[Y]) * SinPsi + (Field2[X] - Source[X]) * CoSinPsi;
  Normal := Abs((Source[Y]-Field1[Y]) * CoSinPsi - (Source[X] - Field1[X]) * SinPsi);
end; { GetLocalCoords }


Function Pow(Base,Exponent : float):float;
  {- returns Base^Exponent }
var sign : integer;
BEGIN
   IF Exponent = 0.0 then Pow := 1.0
   Else
   begin
      IF Base = 0.0 then Pow := 0.0
      ELSE
      begin
        sign := round(abs(base)/base);
        if (sign < 0) and (Int(Exponent) <> Exponent) then
        begin
          write('nice try - bad expotentiation');
          Halt;
        end;
        base := abs(base);
        Base := Exp(Ln(Base) * Exponent);
        Pow := sign * Base;
      end;
   end;
END;

Function Krndlt(I, J : Integer): single;
  {- Kronecker delta function for two indices }
begin
   If I = J then Krndlt := 1.0
   ELSE KrnDlt := 0.0;
End;

Function WrapWord (MaxIndex: word; Index : word) : word;
  {- returns index wrapped properly given Maxindex }
begin
   If Index > MaxIndex then WrapWord := 1
   else
   If Index < 1 then WrapWord := MaxIndex
   else WrapWord := Index;
end;

function WrapEIndex(Index, TotalElems: ElementNumber) : ElementNumber;
  {- returns proper index of element number if at either end of index list }
begin
  WrapEIndex := Index;
  if Index < 1 then WrapEIndex := TotalElems
  else if Index > TotalElems then WrapEIndex := 1;
end;

Procedure Sign(Var A, B : float);
  {- analogous to FORTRAN sign routine }
Begin
   A := Abs(A);
```

```
    If B < 0.0 then A := - A;
 End;

 Function DotProduct(A, B: Vector): float;
   {- returns DotProduct of vectors A & B }
 Var Temp : float;
     Comp : Component;
 Begin
   Temp := 0.0;
   For Comp := i to k do
     Temp := Temp + A[Comp] * B[Comp];
   DotProduct := Temp;
 End;

 Procedure CrossProduct(A, B: Vector; Var C : Vector);
   {- returns cross product of vectors A & B in C }
 Var Comp : Component;
 Begin
   C[i] := A[j] * B[k] - B[j] * A[k];
   C[j] := -(A[i] * B[k] - B[i] * A[k]);
   C[k] := A[i] * B[j] - B[i] * A[j];
 End;

 Procedure GetAlpha(L, M, H: CoordinatePair; Var Alpha : float);
   {- returns value of angle subtended by line L-M-H }
 Var    A, B, C : Vector;
 Begin
   A[i] := L[X] - M[X];
   A[j] := L[Y] - M[Y];
   A[k] := 0.0;
   B[i] := H[X] - M[X];
   B[j] := H[Y] - M[Y];
   B[k] := 0.0;
   Alpha := ArcCos(Dotproduct(A,B)/Sqrt(Dotproduct(A,A) * Dotproduct(B,B)));
   CrossProduct(A,B,C);
   If C[k] < 0.0 then Alpha := 2.0 * Pi - Alpha;
 End;

 function CheckZeroFloat(value : float): float;
    {- zero's out near zero values }
 begin
   if abs(Value) < ZeroTol then CheckZeroFloat := 0.0
   else CheckZeroFloat := Value;
 end; { func CheckZero }

 procedure MakeMessageWin;
   {- puts up message window }
 begin
   HiddenCursor;
   FrameChars := '┌└┐┘─│';
   if not MakeWindow(MesWindow,5,14,75,18,True,True,False,MesAttr,MesAttr,MesAttr,'')
     then ErrorMem;
   if Not DisplayWindow(MesWindow) THEN ErrorMem;
 end; { proc ShowError }

 procedure ClearMessageWin;
   {- disposes of message window }
 begin
   if MesWindow <> nil then DisposeWindow(EraseTopWindow);
   MesWindow := nil;
 end;

 procedure ShowMessage(Mes : TitleString);
   {- puts up Message window }
 begin
   FastWriteWindow(Center(Mes,69),2,1,MesAttr);
 end; { proc ShowError }

 procedure PrintMessage(Mes : TitleString);
   {- writes message to output file }
 begin
   writeln(Outfile);
   writeln(Outfile,Mes);
   writeln(Outfile);
 end;  { proc printmessage }
```

```
procedure StartTimer(Message : TitleString);
  {starts timer and writes message to screen}
begin
  HiddenCursor;
  if MesWindow = nil then MakeMessageWin;
  StartTime := TimeMs;
  ShowMessage(Message);
end; { proc StartTimer }

procedure StopTimer(Message : TitleString);
  {stops timer and writes time elapsed & message to output file}
begin
  StopTime := TimeMS;
  PrintMessage('===> '+Form('###.####',(StopTime-StartTime)/1000.0) +
               ' seconds to '+ Message + '<===');
  ClearMessageWin;
  NormalCursor;
  Clrscr;
end; { proc StopTimer }


begin
  MapColors := True;
  MesAttr := MapColor(MessageAttr);
  MesWindow := nil;
end. { of Unit B7UTILS <*****************************<< }
```

```
Unit B7Prep;
  {-----------------------------------------------------

                 Program GWBEM - Unit B7PREP

     Contains routines for automatic generation of global
     DOFs and assignment of discontinuous elements for
     multi-zone flow system assembly.

     Copyright (c) 1989, Mark A. Lisbe and Iowa State
     University

     ALL RIGHTS RESERVED


     TPRArray, TPVArray, and TPString units copywrite (c)
     1987 by TurboPower Software.  Part of Turbo Profes-
     sional Programmer's Toolbox V4.0.  For information,
     contact:

         TurboPower Software
         3109 Scotts Valley Drive, Suite 122
         Scotts Valley, CA  95066
         (408) 438-8608

     last modified : 12/01/88  9:48 AM
  -----------------------------------------------------}

interface

uses B7Def,
     B7Utils,
     TPRArray,
     TPVArray,
     TPString,
     TPArr,
     B7Data;

procedure Prep_System;
  {- main routine to Prep unit.  Finds discontinuous elements and ID's nodes }

procedure MakeElList(Zone : byte);
  {- generates temp list of element coordinates and interpolated known bndy vals }

procedure ClearElList(Zone : byte);
  {- clears temp list of element coordinates and interpolated known bndy vals }

procedure DisposeWorkArrays(DeleteFile : Boolean);
  {- flushes and closes work arrays }

procedure PlaceSolution;
  {- places system results into proper DOF locations }

{=====================================================================}
implementation

const
  ElMult1 = 0.25;  { natural coordinate locations within elements for }
  ElMult2 = 0.75;  { interior freedom DOF for discontinuous elements  }


var Zone : byte;
    RowCount : word;
    Node, Well : NodeNumber;
    Element : ElementNumber;
    CurrENum, PrevENum : ElementNumber;
    CurrEl, PrevEl : ElementType;

procedure DisposeWorkArrays(DeleteFile : Boolean);
  {- flushes and closes work arrays }
begin
  TPVArray.DisposeA(H, DeleteFile);
  H := nil;
  TPVArray.DisposeA(F, DeleteFile);
  F := nil;
end;
```

```
 procedure MakeWorkArrays;
   {- creates work arrays H[DOF,DOF] & F[DOF] }
 var Zero : float;
     FRam : longint;
 begin
   Zero := 0.0;
   { note: TPVarray error handling is set to on for now.  write own handler later }
   if (DOFCount * DOFCount * sizeof(float)) < (MemAvail div 2) then
     FRAM := DOFCount * DOFCount * sizeof(float)
   else FRAM := memAvail div 2;
   TPVArray.MakeA(H, DOFCount, DOFCount, sizeof(float),
                  ForceExtension(OutFileName,HMatExtStr),
                  MemAvail div 2);
   TPVArray.ClearA(H, Zero, TPVArray.FastInit);
   if (DOFCount * sizeof(float)) < (MemAvail div 2) then FRAM := succ(DOFCount * sizeof(float))
   else FRAM := memAvail div 2;
   TPVArray.MakeA(F, DOFCount, 1, sizeof(float),
                  ForceExtension(OutFileName,FVecExtStr),
                  FRAM);
   TPVArray.ClearA(F, Zero, TPVArray.FastInit);
 end; { of MakeWorkArrays }

procedure Prep_System;
    {- main routine to Prep unit.  Finds discontinuous elements and Global DOFs}
var TempWell : SourceNode;
    NumNodes : word;
    Alpha    : float;
    FirstDOF : GlobalDOF;
    RegCount,
    TLDiscCount : word;

  function DOFSinNode( NType : BNodeType) : GlobalDOF;
    {- returns number of DOFS at node, depending on Interior of not }
  begin
    if NType = Intr then DOFSInNode := 2
    else DOFSInNode := 1;
  end;

begin
  DOFCount := 1;  { initialize DOF counter }
  RowCount := 1;
  for Zone := 1 to Num_Zones do
    with ZoneD[Zone] do
    begin
      StartRow := RowCount;  { start of equations for zone n }
      { determine if elements need to be discontinous }
      for Element := 1 to NumElems do
      begin
        CurrENum := GetElementNum(Elements,Element);
        PrevENum := GetElementNum(Elements,wrapEIndex(pred(Element),NumElems));
        GetElement(CurrENum,CurrEl);
        GetElement(PrevENum,PrevEl);
        with CurrEl do
        begin
{$undef ALLDISC}
{$ifdef ALLDISC }
          ElSFType := Disc;  { force all elements to disc for now }
{$else}
          case ElSFType of
            Reg :
            begin { check if current element should be discontinuous }
              { force interior-exterior junctions to be discontinous }
              if (PrevEl.B.NType = Intr) or (PrevEl.B.NType = Wall)
                then ElSFType := LDisc
              else  { check leading node boundary conditions and geometry }
                case A.NType of
                  Phi :  begin
                           if PrevEl.B.NType = Phi then
                           begin { check angle between elements }
                             GetAlpha(GNode^[PrevEl.A.Node],
                                      GNode^[PrevEl.B.Node],
                                      GNode^[CurrEl.B.Node],
                                      Alpha);
                             if abs(abs(Alpha) - Pi) > ZeroTol then
                             begin { change ElSFType of elements, if necessary }
```

```
                              ElSFType := LDisc; { at least }
                              if PrevEl.ElSFType = LDisc then PrevEl.ElSFType := Disc
                              else PrevEl.ElSFType := TDisc;
                            end;
                          end;
                        end; { Phi case }
                    Intr,
                    Wall: begin
                            ElSFType := LDisc; { at least }
                            if PrevEl.ElSFType = LDisc then PrevEl.ElSFType := Disc
                            else PrevEl.ElSFType := TDisc;
                          end; { Intr/Wall case }
                  end; { case }
                  if (B.NType = Intr) or (B.NType = Wall) then { check trailing node }
                  case ElSFType of
                    Reg :  ElSFType := TDisc;
                    LDisc: ElSFType := Disc;
                  end;
                end; { Reg }

                TDisc :
                begin
                  if (PrevEl.B.NType = Intr) or (PrevEl.B.NType = Wall) then
                    ElSFType := Disc
                  else
                    case A.NType of
                      Phi : begin
                              if PrevEl.B.NType = Phi then
                              begin { check angle between elements }
                                GetAlpha(GNode^[PrevEl.A.Node],
                                         GNode^[PrevEl.B.Node],
                                         GNode^[CurrEl.B.Node],
                                         Alpha);
                                if abs(abs(Alpha) - Pi) > ZeroTol then
                                begin { change ElSFType of elements, if necessary }
                                  ElSFType := Disc;
                                  if PrevEl.ElSFType = LDisc then PrevEl.ElSFType := Disc
                                  else PrevEl.ElSFType := TDisc;
                                end;
                              end;
                            end; { Phi case }
                      Intr,
                      Wall: begin
                              ElSFType := Disc;
                              if PrevEl.ElSFType = LDisc then PrevEl.ElSFType := Disc
                              else PrevEl.ElSFType := TDisc;
                            end; { Intr - Wall case }
                    end; { case }
                    if (B.NType = Intr) or (B.NType = Wall) then { check trailing node }
                    case ElSFType of
                      Reg :  ElSFType := TDisc;
                      LDisc: ElSFType := Disc;
                    end;
                end; { TDisc }

              { pick up lead in elements to int-ext junction; force all el's disc }
              else if ((A.NType = Intr) or (A.NType = Wall)) and (PrevEl.ElSFType<>Disc) then
                if PrevEl.ElSFType = LDisc then PrevEl.ElSFType := Disc
                else PrevEl.ElSFType := TDisc;
              end; { case }
{$endif}
              PutElement(CurrENum,CurrEl);
              PutElement(PrevENum,PrevEl);
            end; { CurrEl do }
          end; { element loop }

          { find global DOF numbers for zone DOFs }

          { take care of 1st node in zone first }
          CurrENum := GetElementNum(Elements,1);
          GetElement(CurrENum,CurrEl);
          if CurrEl.A.DOF = 0 then    { if not defined yet }
          begin
            CurrEl.A.DOF := DOFCount;
            FirstDOF := DOFCount;
            inc(DOFCount, DOFSInNode(CurrEl.A.NType));
```

```
    CurrEl.B.DOF := DOFCount;
    PutElement(CurrENum,CurrEl);
  end
  else
  begin
    FirstDOF := CurrEl.A.DOF;
    dec(DOFCount, DOFSInNode(CurrEl.B.NType));
  end;
  PrevEl := CurrEl;
  { take care of all other nodes in zone }
  for Element := 2 to NumElems do
  begin
    CurrENum := GetElementNum(Elements,Element);
    GetElement(CurrENum,CurrEl);
    if Currel.A.DOF = 0 then  { not assigned yet }
    with CurrEl do
    begin
      if (PrevEl.ElSFType = TDisc) or
         (CurrEl.ElSFType=LDisc) or (CurrEl.ELSFType=Disc) then
        begin
          inc(DOFCount, DOFSInNode(PrevEl.B.NType));
        end;
      A.DOF := DOFCount;
      if Element < NumElems then
      begin
        inc(DOFCount, DOFSInNode(CurrEl.A.NType));
        B.DOF := DOFCount;
      end
      else
        case ElSFType of
          TDisc, Disc : begin
                          inc(DOFCount, DOFSInNode(CurrEl.A.NType));
                          B.DOF := DOFCount;
                        end;
          else B.DOF := FirstDOF;
        end;
    end;
    PutElement(CurrENum,CurrEl);
    PrevEl := CurrEl;
  end;

  DOFCount := succ(DOFCount);

  { Calculate number of rows in boundary }
  RegCount := 0;          { these count up the number of REG elements }
  TLDiscCount := 0;       { these count up the number of T/LDISC elements}
  for Element := 1 to NumElems do
  begin
    CurrENum := GetElementNum(Elements,Element);
    GetElement(CurrENum,CurrEl);
    case CurrEl.ElSFType of
      Reg : begin
              inc(RowCount, 2);
              Regcount := succ(RegCount);
            end;
      LDisc,
      TDisc: begin
               inc(RowCount, 2);
               TLDiscCount := succ(TLDiscCount);
             end;
      Disc : inc(RowCount, 2);
      end;
  end;
  RowCount := RowCount - RegCount - (TLDiscCount div 2) - 1;
  { Note:  should ALWAYS have an even number of T/LDisc element types }
  {        for any zone boundary                                       }

  { account for wells in DOF and row count }
  if NumWells > 0 then
    for Well := 1 to NumWells do
    begin
      GetWell(Zone,Well,TempWell);
      TempWell.DOF := DOFCount;
      PutWell(Zone,Well,TempWell);
      DOFCount := succ(DOFCount);
      RowCount := succ(RowCount);
```

```
          end;

        NumDOFs := DOFCount - StartRow;  { store NumDOFs in zone }
        RowCount := succ(RowCount);
      end; { ZoneD do }
    DOFCount := pred(DOFCount);
    MakeWorkArrays;
  end;

  procedure GetElFreedoms(var TElement : TempEl);
    {- returns coordinates and interpolated known vals for disc. element types }
    {  uses unit global var CurrEl : ElementType. Assumes Linear elements      }
    {  last modified:  11/22/88 6:16 PM                                         }
  var Local1, Local2, Normal, Length, SinPsi, CoSinPsi : float;
      DumbEl : TempEl;

    function N1(X : float) : float;
      {- returns shape function value for linear element }
    begin
      N1 := (Local2-X)/Length;
    end;

    function N2(X : float) : float;
      {- returns shape function value for linear element }
    begin
      N2 := (X-Local1)/Length;
    end;

  begin
  with CurrEl do
    begin
      TElement.A.Coord := GNode^[A.Node];
      TElement.B.Coord := GNode^[B.Node];
      case A.NType of     { assume node type same on each end of element }
        Phi : begin
                TElement.A.KnownVal := A.Phi;
                TElement.B.KnownVal := B.Phi;
              end;
        DPhi,
        Wall: begin
                TElement.A.KnownVal := A.DPhi;
                TElement.B.KnownVal := B.DPhi;
              end;
        Intr: begin
                TElement.A.KnownVal := 0.0;
                TElement.B.KnownVal := 0.0;
              end;
      end;
      TElement.ElType := A.NType;
      DumbEl := TElement;  { you'll see why }
      GetLocalCoords(DumbEl.A.Coord,  { some fixed point, doesn't matter }
                     GNode^[A.Node],
                     GNode^[B.Node],
                     Normal, Local1, Local2, SinPsi, CoSinPsi);
     .Length := abs(Local2 - Local1);
      with DumbEl do
      begin
        if (ElSFType=LDisc) or (ElSfType=Disc) then
        begin
          TElement.A.Coord[X] := A.Coord[X] + (B.Coord[X] - A.Coord[X]) * ElMult1;
          TElement.A.Coord[Y] := A.Coord[Y] + (B.Coord[Y] - A.Coord[Y]) * ElMult1;
          TElement.A.KnownVal := N1(Local1+Length*ElMult1)*A.KnownVal +
                                 N2(Local1+Length*ElMult1)*B.KnownVal;
        end;
        if (ElSfType=TDisc) or (ElSfType=Disc) then
        begin
          TElement.B.Coord[X] := A.Coord[X] + (B.Coord[X] - A.Coord[X]) * ElMult2;
          TElement.B.Coord[Y] := A.Coord[Y] + (B.Coord[Y] - A.Coord[Y]) * ElMult2;
          TElement.B.KnownVal := N1(Local1+Length*ElMult2)*A.KnownVal +
                                 N2(Local1+Length*ElMult2)*B.KnownVal;
        end;
      end; { with }
    end; { with }
end; { of GetElFreedoms }
```

```
procedure MakeElList(Zone : byte);
  {- generates temp list of element coordinates and interpolated known bndy vals }
var TElement : TempEl;
begin
  with ZoneD[Zone] do
  begin
    { make list using TPArray routines }
    TPRArray.MakeA(TempElList, NumElems, 1, Sizeof(TempEl));
    fillChar(TElement,sizeof(TElement),0);
    TPRArray.ClearA(TempElList, TElement, TPRArray.FastInit);
    for Element := 1 to NumElems do
    begin
      CurrENum := GetElementNum(Elements, Element);
      GetElement(CurrENum, CurrEl);
      GetElFreedoms(TElement);
      PutTElement(TempElList,Element,TElement);
    end;
  end;
end; { of MakeElList }

procedure Extrapolate(ElSFType : ElementSFType;
                      var TElement : TempEl);
  {- extrapolates interior values of disc. LINEAR elements to end points }
var N1A, N1B, N2A, N2B : float;
    DumbEl : TempEl;
begin
  case ElSFType of
  Reg : begin
          N1A := 1.0;
          N1B := 0.0;
          N2A := 0.0;
          N2B := 1.0;
        end;
  Disc: begin
          N1A := 1.5;
          N1B := -0.5;
          N2A := -0.5;
          N2B := 1.5;
        end;
  LDisc: begin
          N1A := 4.0/3.0;
          N1B := -1.0/3.0;
          N2A := 0.0;
          N2B := 1.0;
        end;
  TDisc: begin
          N1A := 1.0;
          N1B := 0.0;
          N2A := -1.0/3.0;
          N2B := 4.0/3.0;
        end;
  end; { case }
  with TElement do
  begin
    DumbEl.A.KnownVal := N1A*A.KnownVal + N1B*B.KnownVal;
    DumbEl.B.KnownVal := N2A*A.KnownVal + N2B*B.KnownVal;
  end;
  TElement := DumbEl;
end; { proc Extrapolate }


procedure PlaceSolution;
  {- places system results into proper element locations }
var TEl1, TEl2 : TempEl;
    TWell : SourceNode;
    NextEl : ElementType;
    NextENum : ElementNumber;
    Sign : shortint;

  begin
  for Zone := 1 to Num_Zones do
  with ZoneD[Zone] do
  begin
    PrevENum := GetElementNum(Elements,NumElems);
    GetElement(PrevENum,PrevEl);
    CurrENum := GetElementNum(Elements,1);
    GetElement(CurrENum,CurrEl);
    for Element := 1 to NumElems do
    begin
      Sign := abs(CurrENum) div CurrENum;
      NextENum := GetElementNum(Elements,
                                 wrapEIndex(succ(Element),NumElems));
      GetElement(NextENum,NextEl);

      { get solution values from vector using global DOF's }
      TE11.A.KnownVal := CheckZeroFloat(gvfloat(F,CurrEl.A.DOF,1));
      TE11.B.KnownVal := CheckZeroFloat(gvfloat(F,CurrEl.B.DOF,1));
      Extrapolate(CurrEl.ELSFType, TE11);
      if ((CurrEl.A.NType = Intr) or (CurrEl.A.NType = Wall)) and (Sign > 0) then
      begin
        TE12.A.KnownVal := CheckZeroFloat(gvfloat(F,succ(CurrEl.A.DOF),1));
        TE12.B.KnownVal := CheckZeroFloat(gvfloat(F,succ(CurrEl.B.DOF),1));
        Extrapolate(CurrEl.ELSFType, TE12);
      end; { if - Intr/Wall }

      { keep Phi's consistent across element intersections }
      if (CurrEl.A.NType <> Phi) then
      begin
        if PrevEl.A.NType = Phi then
          TE11.A.KnownVal := PrevEl.B.Phi;
        if NextEl.A.NType = Phi then
          TE11.B.KnownVal := NextEl.A.Phi;
      end;

      { put extrapolated values into place }
      with CurrEl do
      begin
        case A.NType of
        Phi : begin
                 A.DPhi := Sign * TE11.A.KnownVal;
                 B.DPhi := Sign * TE11.B.KnownVal;
               end;
        DPhi,
        Wall: begin
                 A.Phi := TE11.A.KnownVal;
                 B.Phi := TE11.B.KnownVal;
               end;
        Intr: if Sign > 0 then
               begin  { only place values if lowest zone w/ common boundary }
                 A.Phi := TE11.A.KnownVal;
                 B.Phi := TE11.B.KnownVal;
                 A.DPhi := TE12.A.KnownVal;
                 B.DPhi := TE12.B.KnownVal;
               end;
        end; { case }
      end; { with }
      PutElement(CurrENum, CurrEl);
      PrevEl := CurrEl;
      PrevENum := CurrENum;
      CurrEl := NextEl;
      CurrENum := NextENum;
    end; { for }
    for Well := 1 to NumWells do
    begin
      GetWell(Zone,Well,TWell);
      with TWell do
      case SourceType of
        SFlow : Head := gvfloat(F,DOF,1);
        SHead : Flow := gvfloat(F,DOF,1);
      end;
      PutWell(Zone,Well,TWell);
    end; { for }
  end; { with }
end; { proc PlaceSolution }


procedure ClearElList(Zone : byte);
  {- clears temp list of element coordinates and interpolated known bndy vals }
begin
  TPRArray.DisposeA(ZoneD[Zone].TempElList);
  ZoneD[Zone].TempElList := nil;
{$ifdef DEBUG}
```

```
  writeln(Outfile,'===> MaxAvail:   ',Maxavail);
{$endif}
end; { of ClearElList }
end. { of Unit B7PREP <*******************************<< }
```

```
Unit B7Int;
  {---------------------------------------------------------

                      Program GWBEM - Unit B7INT

    Contains analytical integrations used for solution of
    boundary and internal unknowns.  Linear elements.

    Copyright (c) 1989, Mark A. Liebe and Iowa State
    University

    ALL RIGHTS RESERVED

   Last modified : 10/29/88  4:29 PM
   ------------------------------------------------------}

interface

uses B7Def,
     B7Utils,
     TPArr,
     B7Data,
     B7Prep;

Procedure Integrate_Boundary;

procedure Integrate_Interior;
  {-  integrates from each interior node to determine potential and flux values }

{=========================================================================}
implementation

type
  Integral_Vec= Array[1..8] of float;
  Integral_Coefs = array[1..3,1..2] of float;

var
  Integral: Integral_Vec;
  Local1, Local2, Normal, Alpha, Length,
  Sign_Normal, Distance, Conductivity, FundSol : float;
  SourceCoords : CoordinatePair;

{->>>Int_Lin_SF<<<-------------------------------------------------------}
{   Routine to calculate integrals for linear shape functions on the boundary}
{   Last Modified -- 12/08/87.                                            }
{   Note:  Integration in assumed positive in the CLOCKWISE direction.    }
{------------------------------------------------------------------------}
PROCEDURE Int_Lin_SF( Where : NodeType;
                      Normal, E1, E2 : float; { Local coord of bnd. elemnt }
                      var Integral : Integral_Vec);

CONST
   Zero_Tol = 1.0E-5;

VAR
   R_Sqr,
   Ln_R_Sqr,
   Arctn,
   Xi,
   LnR  : array[1..2] of float;
   E2_E1 : float;
   J    : Integer;

   Function INT1(J:Integer): float;      { This integral is used for the    }
   BEGIN                                  { boundary integration.  As such   }
      IF Normal = 0.0 then                { it will have a singularity when }
      BEGIN                               { the source and the field point  }
         IF Xi[J] = 0.0 then INT1 := 0.0  { are the same.                    }
         ELSE INT1 := -1.0 / (2.0 * Xi[J]);
      END
      ELSE
         INT1 := Arctn[J] / Normal;
   END;

   Function INT2(J:Integer): float;      { Same here, see INT1 }
   Begin
```

```
      IF Normal = 0.0 then
      BEGIN
         IF Xi[J] = 0.0 then INT2 := 0.0
         ELSE INT2 := Ln(ABS(Xi[J]));
      END
      ELSE
         INT2 := 0.5 * Ln_R_Sqr[J];
   End;

   Function INT3(J:Integer): float;       { The rest of the integrals are either}
   Begin                                  { not used for boundary integrations, }
      INT3 := Xi[J] - Normal * Arctn[J];  { or contain no non-integrable singu. }
   End;

   Function INT4(J:Integer): float;
   Begin
      IF Normal = 0.0 then
         INT4 := -1.0 / (3.0 * Pow(Xi[J],3.0))
      ELSE
         INT4 := Xi[J]/(2.0 * R_Sqr[J] * Pow(Normal,2.0)) +
                 (Arctn[J] / (2.0 * Pow(Normal,3.0)));
   End;

   Function INT5(J:Integer): float;
   Begin
      IF Normal = 0.0 then
         INT5 := -1.0/(2.0 * Pow(Xi[J],2.0))
      ELSE
         INT5 := - 1.0 / (2.0 * R_Sqr[J]);
   End;

   Function INT6(J:Integer): float;
   Begin
      IF Normal = 0.0 then
         INT6 := - 1.0 / (2.0 * Xi[J])
      ELSE
         INT6 := - (Xi[J] / (2.0 * R_Sqr[J])) + (Arctn[J] / (2.0 * Normal));
   End;

   Function INT7(J:Integer): float;
   Begin
      INT7 := 0.25 * R_Sqr[J] * (Ln_R_Sqr[J] - 1.0);
   End;

   Function INT8(J:Integer): float;
   Begin
      INT8 := 0.5 * (Xi[J] * Ln_R_Sqr[J] - 2.0 * Xi[J] + 2.0 * Normal * Arctn[J]);
   End;

BEGIN
   Fillchar(Integral,Sizeof(Integral),0);
   Xi[1] := E1;
   Xi[2] := E2;
   E2_E1 := E2 - E1;
   FOR J := 1 to 2 do
   BEGIN
      R_Sqr[j] := Normal * Normal + Xi[j] * Xi[j];
      IF Abs(Normal) < Zero_Tol then
         Arctn[j] := 0.0
      ELSE
         Arctn[j] := ArcTan(Xi[j]/Normal);
      IF R_Sqr[j] = 0.0 then
         Ln_R_Sqr[j] := 0.0
      else
         Ln_R_Sqr[j] := Ln(R_Sqr[j]);
   end;
   Case where of
   Boundary : begin
                 Integral[1] := Int1(2) - Int1(1);
                 Integral[2] := Int2(2) - Int2(1);
                 Integral[7] := Int7(2) - Int7(1);
                 Integral[8] := Int8(2) - Int8(1);
              end;

   Interior : begin
                 Integral[1] := Int1(2) - Int1(1);
```

```
                Integral[2] := Int2(2) - Int2(1);
                Integral[3] := Int3(2) - Int3(1);
                Integral[4] := Int4(2) - Int4(1);
                Integral[5] := Int5(2) - Int5(1);
                Integral[6] := Int6(2) - Int6(1);
                Integral[7] := Int7(2) - Int7(1);
                Integral[8] := Int8(2) - Int8(1);
              end;
   end; { case }
end; { of Int_Lin_SF}

{--->>>Integrate_Boundary<<<-------------------------------------------------}
{  Includes code for doubly defined flux at boundary corner                 }
{  Includes code for multiple zones                                         }
{  Last modified:  10/29/86  5:07 PM                                        }
{---------------------------------------------------------------------------}
Procedure Integrate_Boundary;

type ElementEnd = (A,B);

Var  Ke       : Integral_Coefs;
     Row, Node, SourceNum, Field, LastDOF, SourceDOF : GlobalDof;
     SourceENum, FieldENum : ElementNumber;
     SourceEl, FieldEl : ElementType;
     LastElType : BNodeType;
     TSourceEl, TFieldEl, PFieldEl, NFieldEl : TempEl;
     WellBuf : SourceNode;
     LastNode, OKtoCollocate : boolean;
     Zone : byte;



 function AngleAtNode(Zone : byte;
                      var ThisSource : GlobalDOF;
                      var ThisSourceEl : TempEl;
                      Where : ElementEnd) : float;
   {- returns angle at node given source index in local element list }
 var  OtherEl : TempEl;
      OtherENum : GlobalDof;
      TempAlpha : float;
 begin
   with ZoneD[Zone] do
   case where of
     A : begin   { at first node of element }
           OtherENum := WrapWord(NumElems,pred(ThisSource));
           GetTElement(TempElList, OtherENum, OtherEl);
           GetAlpha(OtherEl.A.Coord,
                    ThisSourceEl.A.Coord,
                    ThisSourceEl.B.Coord,
                    TempAlpha);
         end;
     B : begin   { at last node of element }
           OtherENum := WrapWord(NumElems,succ(ThisSource));
           GetTElement(TempElList, OtherENum, OtherEl);
           GetAlpha(ThisSourceEl.A.Coord,
                    ThisSourceEl.B.Coord,
                    OtherEl.B.Coord,
                    TempAlpha);
         end;
   end; { case }
   AngleAtNode := TempAlpha;
 end; { function AngleAtNode }

 procedure GetIntCoefs(var Ke : Integral_Coefs);
 {- returns proper Integral coefficients for addition to system of equations }
 {  implements LINEAR elements for now }
 var Mult1, Mult2 : float;
 begin
   { Check on direction from source point to element }
   Sign_Normal := -(TFieldEl.A.Coord[X] - SourceCoords[X]) *
                   (TFieldEl.B.Coord[Y] - TFieldEl.A.Coord[Y])
                  +(TFieldEl.B.Coord[X] - TFieldEl.A.Coord[X]) *
                   (TFieldEl.A.Coord[Y] - SourceCoords[Y]);
   if FieldEl.ElSFType = Reg then
   begin
   Ke[1,1] := (Local2 * I12 - I11)/Length;
```

```
    Ke[1,2] := (I11 - Local1 * I12)/Length;
    Ke[2,1] := (Local2 * I22 - I21)/Length;
    Ke[2,2] := (I21 - Local1 * I22)/Length;
    end
    else
    begin
      case FieldEl.ElSFType of
        LDisc:begin
                Mult1 := 4.0/(3.0*Length);
                Mult2 := 4.0/3.0 + Mult1 * Local1;
              end;
        TDisc:begin
                Mult1 := 4.0/(3.0*Length);
                Mult2 := 1.0 + Mult1 * Local1;
              end;
        Disc :begin
                Mult1 := 2.0/Length;
                Mult2 := 1.5 + Mult1 * Local1;
              end;
      end; { case }
      Ke[1,1] := Mult2 * I12 - Mult1 * I11;
      Ke[1,2] := Mult1 * I11 - (Mult2-1.0) * I12;
      Ke[2,1] := Mult2 * I22 - Mult1 * I21;
      Ke[2,2] := Mult1 * I21 - (Mult2-1.0) * I22;
    end; { else }
    if SourceDOF = FieldEl.A.DOF then Ke[1,1] := -Alpha; { check for sing. point }
    if SourceDOF = FieldEl.B.DOF then
      if (FieldEl.ElSFType=TDisc) or (FieldEl.ElSFType=Disc) then
        Ke[1,2] := -Alpha;
    if SourceDOF <> FieldEl.A.DOF then Sign(Ke[1,1], Sign_Normal);
    if SourceDOF <> FieldEl.B.DOF then Sign(Ke[1,2], Sign_Normal);

end; { proc GetIntCoefs }

procedure PlaceCoefs(ElSFType : ElementSFType;
                     Prev, This, Next : TempEl;
                     Ke : Integral_Coefs;
                     SourceNum, Field1, Field2 : GlobalDof;
                     TypeofElement : BNodeType;
                     Kx, Ky , ThetaX : float);
{- places integral coefs into system of eq.s based on node type }
var ZoneSign : shortint;
begin
  case TypeofElement of

  Phi : begin
          { first node }
          PVfloat(H, Row, Field1, GVFloat(H, Row, Field1) - (Ke[2,1]));
          PVfloat(F, Row, 1, GVfloat(F, Row, 1) - (Kx*Ke[1,1] * This.A.KnownVal));

          { second node }
          PVfloat(H, Row, Field2, GVFloat(H, Row, Field2) - (Ke[2,2]));
          PVfloat(F, Row, 1, GVfloat(F, Row, 1) - (Kx*Ke[1,2] * This.B.KnownVal));
        end; { Phi }

  DPhi,
  Wall: begin
          { first node }
          if (Prev.ElType <> Phi) or (ElSFType = Disc) or (ElSFType = LDisc) then
            PVfloat(H, Row, Field1, GVFloat(H, Row, Field1) + (Kx*Ke[1,1]))
          else
            PVfloat(F, Row, 1, GVFloat(F, Row, 1) - (Kx*Ke[1,1] * Prev.B.KnownVal));
          PVfloat(F, Row, 1, GVfloat(F, Row, 1) + (Ke[2,1] * This.A.KnownVal));

          { second node }
          if (Next.ElType <> Phi)  or (ElSFType = Disc) or (ElSFType = TDisc) then
            PVfloat(H, Row, Field2, GVFloat(H, Row, Field2) + (Kx*Ke[1,2]))
          else
            PVfloat(F, Row, 1, GVFloat(F, Row, 1) - (Kx*Ke[1,2] * Next.A.KnownVal));
          PVfloat(F, Row, 1, GVfloat(F, Row, 1) + (Ke[2,2] * This.B.KnownVal));
        end;  { DPhi }

  Intr: begin
          { note: double DOFs for each intr nodes, PHI dof ALWAYS 1st }
          if FieldENum < 0 then ZoneSign := -1 else ZoneSign := 1;
          { Phi unknown DOF }
          PVfloat(H, Row, Field1, GVFloat(H, Row, Field1) + (Kx*Ke[1,1]));
          PVfloat(H, Row, Field2, GVFloat(H, Row, Field2) + (Kx*Ke[1,2]));
          { DPhi Unknown DOF }
          PVfloat(H, Row, succ(Field1),
                      ZoneSign * (GVFloat(H, Row, succ(Field1)) - (Ke[2,1])));
          PVfloat(H, Row, succ(Field2),
                      ZoneSign * (GVFloat(H, Row, succ(Field2)) - (Ke[2,2])));
        end; { Intr }

    end; { case }
  end; { proc PlaceCoefs }

procedure GoAroundBoundary(Zone : byte);
  {- performs boundary integration around zone boundary }
var Field : GlobalDOF;
begin
  with ZoneD[Zone] do
  begin
    { set up for first field element}
    GetTElement(TempElList,NumElems,PFieldEl);
    GetTElement(TempElList,1, TFieldEl);
    for Field := 1 to NumElems do
    begin
      FieldENum := GetElementNum(Elements, Field);
      GetElement(FieldENum, FieldEl);
      GetTElement(TempElList,wrapword(NumElems, succ(Field)),NFieldEl);
      GetLocalCoords(SourceCoords,
                     GNode^[FieldEl.A.Node],
                     GNode^[FieldEl.B.Node],
                     Normal, Local1, Local2, SinPsi, CoSinPsi);
      Int_Lin_SF(Boundary,Normal, Local1, Local2, Integral);
      I11 := Normal * Integral[2];
      I12 := Normal * Integral[1];
      I21 := Integral[7];
      I22 := Integral[8];
      Length := Local2 - Local1;

      GetIntCoefs(Ke);
      with FieldEl do
        PlaceCoefs(FieldEl.ElSFType, PFieldEl, TFieldEl, NFieldEl, Ke,
                   Row, A.DOF, B.DOF, A.NType, Kx, Ky, ThetaX);

      { swap temp elements }
      PFieldEl := TFieldEl;
      TFieldEl := NFieldEl;

    end; { Field - Loop }
  end; { with }
end; { of proc GoAroundBoundary }


procedure AddSources(Zone : byte; SourceDOF : GlobalDOF ;
                     WhereSource : NodeType);
  {- collocates from boundary nodes to wells -}
  {- last modified:  11/28/88  7:13 PM       -}
var Field : GlobalDOF;
    TWell : SourceNode;
begin
  with ZoneD[Zone] do
  begin
    if NumWells <= 0 then Exit;
    for Field := 1 to NumWells do
    begin
      GetWell(Zone, Field, TWell);
      Distance := Radius(SourceCoords[x],SourceCoords[y],
                         TWell.Coord[x],TWell.Coord[y]);
      case WhereSource of

      Boundary :
        begin
          FundSol := -ln(Distance);
          case TWell.SourceType of
          SFlow : PvFloat(F,Row,1,
                          GvFloat(F,Row,1) + FundSol * TWell.Flow);

          SHead : PvFloat(H,Row,TWell.Dof, -FundSol);
```

```
            end;
          end; { Boundary }

        Source :
          begin
            if TWell.DOF = SourceDOF then
            begin
              FundSol := -ln(TWell.Radius);
              case TWell.SourceType of
              SFlow :
                begin
                  PVFloat(F,Row,1,GVFloat(F,Row,1) + FundSol * TWell.Flow);
                  PVFloat(H,Row,TWell.DOF,-2*Pi * Conductivity);
                end;

              SHead :
                begin
                  PVFloat(F,Row,1, GVFloat(F,Row,1) +
                                  2*Pi*TWell.Head  * Conductivity );
                  PVFloat(H,Row,TWell.DOF, -FundSol);
                end;
              end; { case }
            end
            else
            begin
              Distance := Radius(SourceCoords[x],SourceCoords[y],
                                 Twell.Coord[x],TWell.Coord[y]);
              FundSol := -ln(Distance);
              case TWell.SourceType of
              SFlow :
                begin
                  PVFloat(F,Row,1,GVFloat(F,Row,1) + FundSol * TWell.Flow);
                  PVFloat(H,Row,TWell.DOF, 0.0);
                end;

              SHead : PVFloat(H,Row,TWell.DOF, -FundSol);
              end; { case }
            end; { else }
          end;  { Interior }
        end; { case }
      end; { field loop }
    end; { with }
  end; { of add sources }

begin { integrate_Boundary }
  for Zone := 1 to Num_Zones do
  with ZoneD[Zone] do
  begin
    MakeElList(Zone);                        { set up temporary element list}
    Row := pred(StartRow);
    SourceNum := 1;                                    { set up for 1st element}
    SourceENum := GetElementNum(Elements, SourceNum);
    Conductivity := Kx;  { for now, just isotropic conductivity }
    GetElement(SourceENum, SourceEl);
    GetTElement(TempElList,SourceNum, TSourceEl);
    LastDOF := succ(SourceEl.B.DOF);                    { an arbitrary DOF}
    LastElType := SourceEl.B.NType;
    LastNode := False;
    while SourceNum <= NumElems do
    begin
      { select source point for collocation }
      if LastNode then                      { check to see if has been colloc}
        case SourceEl.ElSFType of
          TDisc, Disc : begin
                          Alpha := Pi;
                          SourceCoords := TSourceEl.B.Coord;
                          SourceDOF := SourceEl.B.DOF;
                          OKtoCollocate := True;
                          Row := succ(Row);
                        end;
          else OKtoCollocate := False;
        end { case }
      else
      begin
        case SourceEl.ElSFType of
          LDisc, Disc: Alpha := Pi;
```

```
           else Alpha := AngleAtNode(Zone,SourceNum,TSourceEl,A);
         end;
         SourceCoords := TSourceEl.A.Coord;
         SourceDOF := SourceEl.A.DOF;
         OKtoCollocate := True;
         Row := succ(Row);
       end;
       if OKtoCollocate then
       begin
         GoAroundBoundary(Zone);
         AddSources(Zone, SourceDOF, Boundary);
       end;

       if LastNode then
       begin                        { get next element for source collocation}
         SourceNum := succ(SourceNum);
         if SourceNum <= NumElems then
         begin
           SourceENum := GetElementNum(Elements, SourceNum);
           LastDOF := SourceEl.B.DOF;
           LastElType := SourceEl.B.NType;
           GetElement(SourceENum, SourceEl);
           GetTElement(TempElList, SourceNum, TSourceEl);
           LastNode := False;
         end;
       end
       else
       begin
         LastNode := True;
         LastDOF := SourceEl.A.DOF;
       end;
     end; { while - source loop }
     { integrate FROM sources to boundary nodes }
     for SourceNum := 1 to NumWells do
     begin
       Alpha := 2.0 * Pi;
       GetWell(Zone, SourceNum, WellBuf);
       SourceCoords := WellBuf.Coord;
       SourceDOF := WellBuf.DOF;
       Row := succ(Row);
       GoAroundBoundary(Zone);
       AddSources(Zone, SourceDOF, Source);
     end;
     ClearElList(Zone);
  end; { zone loop }
END; { procedure integrate_boundary }

{--->>>Integrate_Interior<<<------------------------------------------------}
{  Last modified 12/02/88  12:53 PM                                         }
{--------------------------------------------------------------------------}
procedure Integrate_Interior;
  {-  integrates from each interior node to determine potential and flux values }

var DX, DY     : float;
    ING        : array[1..3,1..4] of float;
    Phi_Ke,
    DPhiX_Ke,
    DPhiY_Ke   : Integral_Coefs;
    IntNodeBuf : InteriorNode;
    Node : GlobalDOF;

  Procedure GetIntCoef(Var Ke : Integral_Coefs);
  begin
     Ke[1,1] :=(-ING[1,1] +           Local2 *   ING[1,2]             )/
                 (Length);

     Ke[1,2] :=( ING[1,1] -           Local1 *   ING[1,2]             )/
                 (Length);

     Ke[2,1] :=(-ING[2,1] + ING[2,3] + Local2 * ( ING[2,2] - ING[2,4]))/
                 (Length);

     Ke[2,2] :=( ING[2,1] - ING[2,3] + Local1 * (-ING[2,2] + ING[2,4]))/
                 (Length);

     Ke[3,1] :=(-ING[3,1] + ING[3,3] + Local2 * ( ING[3,2] - ING[3,4]))/
```

```
              (Length);

   Ke[3,2] :=( ING[3,1] - ING[3,3] + Local1 * (-ING[3,2] + ING[3,4]))/
              (Length);
end;

procedure GoAroundBoundary(Zone : byte);
  {- performs boundary integration around zone boundary }
var Field : GlobalDOF;
    FieldEl : ElementType;
    FieldENum : ElementNumber;
    Sign : shortint;
begin
  with ZoneD[Zone] do
  begin
    { set up for first field element}
    for Field := 1 to NumElems do
    begin
      FieldENum := GetElementNum(Elements, Field);
      Sign := abs(FieldENum) div FieldENum;
      GetElement(FieldENum, FieldEl);
      with FieldEl do  { reverse sign on boundary fluxes if secondary zone }
      begin
        A.DPhi := Sign * A.DPhi;
        B.DPhi := Sign * B.DPhi;
      end;
      GetLocalCoords(SourceCoords,
                     GNode^[FieldEl.A.Node],
                     GNode^[FieldEl.B.Node],
                     Normal, Local1, Local2, SinPsi, CoSinPsi);
      Int_Lin_SF(Interior,Normal, Local1, Local2, Integral);
      Length := Local2 - Local1;

      { Calculate integral components for Phi }
      FillChar(ING,SizeOf(ING),0);
      ING[1,1] := Normal * Integral[2];
      ING[1,2] := Normal * Integral[1];
      ING[2,1] := Integral[7];
      ING[2,2] := Integral[8];
      GetIntCoef(Phi_Ke);   { Note : there is no 3rd term in Phi eq }

      { Calculate integral components for DPhi - X direction }
      FillChar(ING,SizeOf(ING),0);
      ING[1,1] := SinPsi * Integral[2];
      ING[1,2] := SinPsi * Integral[1];
      ING[2,1] := 2.0 * Normal * CoSinPsi * Integral[6];
      ING[2,2] := 2.0 * Normal * CoSinPsi * Integral[5];
      ING[2,3] := 2.0 * Normal * Normal * SinPsi * Integral[5];
      ING[2,4] := 2.0 * Normal * Normal * SinPsi * Integral[4];
      ING[3,1] := CoSinPsi * Integral[3];
      ING[3,2] := CoSinPsi * Integral[2];
      ING[3,3] := Normal * SinPsi * Integral[2];
      ING[3,4] := Normal * SinPsi * Integral[1];
      GetIntCoef(DPhiX_Ke);

      { Calculate integral components for DPhi - Y direction }
      FillChar(ING,SizeOf(ING),0);
      ING[1,1] := - CoSinPsi * Integral[2];
      ING[1,2] := - CoSinPsi * Integral[1];
      ING[2,1] := 2.0 * Normal * SinPsi * Integral[6];
      ING[2,2] := 2.0 * Normal * SinPsi * Integral[5];
      ING[2,3] := - 2.0 * Normal * Normal * CoSinPsi * Integral[5];
      ING[2,4] := - 2.0 * Normal * Normal * CoSinPsi * Integral[4];
      ING[3,1] := SinPsi * Integral[3];
      ING[3,2] := SinPsi * Integral[2];
      ING[3,3] := - Normal * CoSinPsi * Integral[2];
      ING[3,4] := - Normal * CoSinPsi * Integral[1];
      GetIntCoef(DPhiY_Ke);

      IntNodeBuf.Phi := IntNodeBuf.Phi +
          Conductivity*(Phi_Ke[1,1] * FieldEl.A.Phi +
                        Phi_Ke[1,2] * FieldEl.B.Phi) -
                       (Phi_Ke[2,1] * FieldEl.A.Dphi +
                        Phi_Ke[2,2] * FieldEl.B.DPhi);

      IntNodeBuf.DPhiX := IntNodeBuf.DPhiX + Conductivity *
```

```
           ((DPhiX_Ke[1,1] + DPhiX_Ke[2,1]) * FieldEl.A.Phi +
            (DPhiX_Ke[1,2] + DPhiX_Ke[2,2]) * FieldEl.B.Phi) +
            (DPhiX_Ke[3,1] * FieldEl.A.Dphi +
             DPhiX_Ke[3,2] * FieldEl.B.DPhi);

        IntNodeBuf.DPhiY := IntNodeBuf.DPhiY + Conductivity *
           ((DPhiY_Ke[1,1] + DPhiY_Ke[2,1]) * FieldEl.A.Phi +
            (DPhiY_Ke[1,2] + DPhiY_Ke[2,2]) * FieldEl.B.Phi) +
            (DPhiY_Ke[3,1] * FieldEl.A.Dphi +
             DPhiY_Ke[3,2] * FieldEl.B.DPhi);

      end; { Field - Loop }
    end; { with }
  end; { of proc GoAroundBoundery }

  procedure AddSources(Zone : byte);
    {- adds source information to interior nodes -}
    {- last modified:  12/04/88  7:58 AM       -}
  var Field : GlobalDOF;
      DX, DY : float;
      TWell : SourceNode;
  begin
    with ZoneD[Zone] do
    begin
      if NumWells < 1 then Exit;
      for Field := 1 to NumWells do
      begin
        GetWell(Zone, Field, TWell);
        Distance := Radius(SourceCoords[x],SourceCoords[y],
                           TWell.Coord[x],TWell.Coord[y]);
        DX := (TWell.Coord[X]-SourceCoords[X]);
        DY := (TWell.Coord[Y]-SourceCoords[Y]);
        IntNodeBuf.Phi := IntNodeBuf.Phi +{-} (ln(Distance) * TWell.Flow{* Conductivity});
        IntNodeBuf.DPhiX := IntNodeBuf.DPhiX -
                           (DX*TWell.Flow)/({Conductivity*}Distance*Distance);
        IntNodeBuf.DPhiY := IntNodeBuf.DPhiY -
                           (DY*TWell.Flow)/({Conductivity*}Distance*Distance);
      end; { field loop }
    end; { with }
  end; { of add sources }

begin { integrate_Interior }
  for Zone := 1 to Num_Zones do
  with ZoneD[Zone] do
  begin
    for Node := 1 to NumInt do
    begin
      Alpha := 2.0 * Pi;
      Conductivity := Kx;  { for now, just isotropic conductivity }
      GetIntNode(Zone, Node, IntNodeBuf);
      SourceCoords := IntNodeBuf.Coord;
      GoAroundBoundary(Zone);
      AddSources(Zone);
      with IntNodeBuf do
      begin
        Phi := Phi / (Conductivity * Alpha);
        DPhiX := -DPhiX{ * Conductivity} / Alpha;
        DPhiY := -DPhiY{ * Conductivity} / Alpha;
      end;
      PutIntNode(Zone, Node, IntNodeBuf);
    end;
  end; { zone loop }
end; { procedure integrate_interior }

end. { unit B7int <******************************<< }
```

```
unit B7File;
  {-----------------------------------------------------

                    Program GWBEM - Unit B7FILE

    Contains file I/O routines. Linear elements.

    Copyright (c) 1989, Mark A. Liebe and Iowa State
    University

    ALL RIGHTS RESERVED

    TPRArray, TPVArray, and TPString units copywrite (c)
    1987 by TurboPower Software.  Part of Turbo Profes-
    sional Programmer's Toolbox V4.0.  For information,
    contact:

        TurboPower Software
        3109 Scotts Valley Drive, Suite 122
        Scotts Valley, CA  95086
        (408) 438-8608

   Last modified : 12/12/88  11:15 AM
   -----------------------------------------------------}

interface

uses DOS,
     B7def,
     TPVArray,
     TPRArray,
     TPArr,
     TPString,
     B7Data;

Type File_Op = (Rd, Wrt);

procedure GET_DATA;
  {- loads data from text file }

procedure OPEN_TEXT_FILE(var File_to_Open : text;
                             Name_File : File_name;
                             Flag : File_Op);
  {- prepares a text file for reading or writing }

procedure WriteFloatTPV(Mat : TPVarray.TPArray; Size :word; Length, Decimal : byte;
                        Header : Titlestring);
  {- output formatted listing of TP virtual array to outfile }

procedure Dump_System(Suffix : File_Ext);
  {- outputs system element definitions }

procedure DumpH_F(Var H, F : TPVArray.TPArray;
                  RowSize, ColSize :word; Length, Decimal : byte;
                      Header : Titlestring);
  { dumps matrix out in row order in a linear list }

procedure WriteBoundarySolution;
  {- formatted output for LINEAR element solution }

procedure WriteSourceSolution;
  {- formatted output for source solution }

procedure WriteInteriorSolution;
  {- formatted output for interior node solution }

procedure writeGridFile;
  {- prints out solution grid files for contouring }

procedure DumpSolVec;
  {- dumps solution vector to outfile }

{==================================================================}
implementation

var Hour, Min, Sec, Sec100,
      Year, Month, Day, DayofWeek : Word;
      Node, Element, Nodeindex, Blindex: word;
      ElementNum : ElementNumber;
      Zone : byte;
      CurrEl : ElementType;
      CurrElNum : ElementNumber;
      CurrINode : InteriorNode;
      CurrWell : Sourcenode;

const Zero : byte = 0;
      RealWidth = 12;
      Width = 11;
      Places = 4;

procedure GET_DATA;
  {- loads data from text file }

VAR Dumbtype : byte;
    CurrWell : SourceNode;
    CurrINode : InteriorNode;
    Distance : Float;   {<<<}

   function SelectBType(Dumb : byte):BNodeType;
     {- returns enumerated value of byte dumb }
   begin
     case Dumb of
       0  : SelectBType := Phi;
       1  : SelectBType := DPhi;
       2  : SelectBType := Intr;
       3  : SelectBType := Wall;
     end;
   end;  { of SelectBType }

   function SelectSType(Dumb : byte):SNodeType;
     {- returns enumerated value of byte dumb }
   begin
     case Dumb of
       0  : SelectSType := SFlow;
       1  : SelectSType := SHead;
     end;
   end;  { of SelectSType }


 BEGIN
     READ(Infile,TITLE);
     WRITEln(Outfile,TITLE);
     writeln(Outfile);
     writeln(OutFile,'  Run date:  ',Month:2,'/',Day:2,'/',Year:4,
                     '  Run time:  ',Hour:2,':',Min:2);
     writeln(OutFile);

     { read in global node coordinates }
     Readln(InFile,Num_BNodes);
     writeln(Outfile,'***> Global Node Information <***');
     Writeln(Outfile,'Number Global Nodes :  ',Num_BNodes);
     writeln(Outfile);
     writeln(OutFile,'---> Global Node Definition <---');
     writeln(Outfile);
     writeln(OutFile,'Node         X-Coor          Y-Coor');
     writeln(OutFile,'----         ------          ------');
     writeln(Outfile);
     for Node := 1 to Num_BNodes do
       Readln(InFile,NodeIndex,GNode^[NodeIndex][X],GNode^[NodeIndex][Y]);
     for Node := 1 to Num_BNodes do
       Writeln(Outfile,Node:4,'    ',GNode^[Node][X]:RealWidth,'    ',GNode^[Node][Y]:RealWidth);

     { read in global element definitions }
     readln(InFile,Num_Boundaries);
     writeln(Outfile);
     writeln(Outfile,'***> Global Element Information <***');
     writeln(Outfile);
     Writeln(Outfile,'Number Global Elements :  ',Num_Boundaries);
     writeln(Outfile);
     writeln(OutFile,'                    ----> Global Element Definition <----');
     writeln(Outfile);
     writeln(Outfile,'            First node              Second node');
writeln(OutFile,'Elem| Node Type  Known Value   | Node Type  Known Value ');
writeln(OutFile,'----| ---- ----  -----------   | ---- ----  ----------- ');
for Element := 1 to Num_Boundaries do
begin
  read(InFile,ElIndex);
  with GElem^[ElIndex] do
  begin
    { take care of A node stuff }
    read(InFile,A.Node,DumbType);
    A.NType := SelectBType(DumbType);
    case A.NType of    { write out node A info }
      Phi :   read(InFile,A.Phi);

      dPhi:   read(InFile,A.DPhi);

      else  begin
              read(InFile,A.DPhi);
              A.DPhi := 0.0;  { since this is a fake value, cancel here }
            end;
    end; { case }

    { take care of B node stuff }
    Read(InFile,B.Node,DumbType);
    B.NType := SelectBType(DumbType);
    case B.NType of   { write out node B info }
      Phi :   readln(InFile,B.Phi);

      dPhi:   readln(InFile,B.DPhi);

      else  begin
              read(InFile,B.DPhi);
              B.DPhi := 0.0;  { since this is a fake value, cancel here }
            end;
    end; { case }
  end; { with }
end; { for }
for Element := 1 to Num_Boundaries do
with GElem^[Element] do
begin
  case A.NType of     { write out node A info }
    Phi :  write(OutFile,Element:4,'|',
                 A.Node:4,'  ',
                 BTypeStr[A.NType],'  ',
                 A.Phi:RealWidth,'   |');
    dPhi:  write(OutFile,Element:4,'|',
                 A.Node:4,'  ',
                 BTypeStr[A.NType],'  ',
                 A.DPhi:RealWidth,'   |');
    else   write(OutFile,Element:4,'|',
                 A.Node:4,'  ',
                 BTypeStr[A.NType],'  ',
                 Center('***',RealWidth),'   |');
  end; { case }

  case B.NType of   { write out node B info }
    Phi :   writeln(Outfile,B.Node:4,'  ',
                    BTypeStr[B.NType],'  ',
                    B.Phi:RealWidth);
    dPhi:   writeln(Outfile,B.Node:4,'  ',
                    BTypeStr[B.NType],'  ',
                    B.DPhi:RealWidth);
    else    writeln(Outfile,B.Node:4,'  ',
                    BTypeStr[B.NType],'  ',
                    Center('***',RealWidth));
  end; { case }
end;
{ read in zone information }
readln(InFile,Num_Zones);
writeln(Outfile);
writeln(Outfile,'***> Zone  Information <***');
writeln(Outfile);
Writeln(Outfile,'Number Zones:  ',Num_Zones);
writeln(Outfile);
if Num_Zones > Max_Zones then
begin
  writeln(^G'Sorry, too many zones!!');
  halt(1);
 end;
for Zone := 1 to Num_Zones do
with ZoneD[Zone] do
begin
  writeln(OutFile,'---> Zone ',Zone:2,' date <---');
  READln(InFIle,NumElems, NumWells, NumInt, Kx, Ky, ThetaX);
  writeln(Outfile,'   Number Boundary Elements: ',NumElems);
  writeln(Outfile,'   Number Wells:             ',NumWells);
  writeln(Outfile,'   Number Interior Points:   ',NumInt);
  writeln(Outfile,'   Kx:                       ',Kx:RealWidth);
  writeln(Outfile,'   Ky:                       ',Ky:RealWidth);
  writeln(Outfile,'   Theta-x:                  ',ThetaX:RealWidth);
  writeln(Outfile);

  { make dynamic arrays for this zone }
  if NumElems > 0 then
  begin
    TPRArray.MakeA(Elements, NumElems, 1, Sizeof(ElementNumber));
    ElementNum := 0;
    TPRArray.ClearA(Elements, ElementNum,Fastinit);
  end;
  if NumWells > 0 then
  begin
    TPRArray.MakeA(Wells, NumWells, 1, SizeOf(SourceNode));
    FillChar(CurrWell,Sizeof(CurrWell),0);
    TPRArray.ClearA(Wells, CurrWell,Fastinit);
  end;
  if NumInt > 0 then
  begin
    TPRArray.MakeA(IntNodes, NumInt, 1, SizeOf(InteriorNode));
    FillChar(CurrINode,Sizeof(CurrINode),0);
    TPRArray.ClearA(IntNodes, CurrINode,Fastinit);
  end;

  {read in boundary element list for this zone }
  writeln(OutFile,'   Global element list -- Zone ',Zone:3);
  writeln(OutFile);
  writeln(OutFile,'   Local      Global  ');
  writeln(OutFile,'   -----      ------  ');
  if NumElems > 0 then
  for Element := 0 to pred(NumElems) do
  begin
    Read(InFile,ElementNum);
    TPRArray.SetA(Elements,Element,0,ElementNum);
    if Eoln(InFile) or (IOResult <> 0) then readln(Infile);
    writeln(Outfile,'    ',succ(Element):4,'         ',ElementNum:4);
  end;

  {read in well information for this zone }
  writeln(OutFile);
  writeln(OutFile,'   Well definition -- Zone ',Zone:3);
  writeln(OutFile);
  writeln(OutFile,'Node    X Coor        Y Coor        Radius       Type   Spec. Value');
  writeln(OutFile,'----    ------        ------        ------       ----   -----------');
  if NumWells > 0 then
  for Node := 1 to NumWells do
  with CurrWell do
  begin
    Read(InFile,NodeIndex,Coord[x], Coord[y], Radius, DumbType);
    SourceType := SelectSType(DumbType);
    case SourceType of
      SFlow : readln(Infile, Flow);
      SHead : readln(Infile, Head);
    end;
    PutWell(Zone, NodeIndex, CurrWell);
  end;
  for Node := 1 to NumWells do
  begin
    GetWell(Zone, Node, CurrWell);
    with CurrWell do
    case SourceType of
      SFlow :  writeln(Outfile,Node:4,'    ',
                       Coord[x]:RealWidth,'  ',
                       Coord[y]:RealWidth,'  ',
                       Radius:RealWidth,'  ',
                             STypeStr[SourceType],'  ',
                             Flow:RealWidth);
            SHead :  writeln(Outfile,Node:4,'   ',
                             Coord[x]:RealWidth,'  ',
                             Coord[y]:RealWidth,' ',
                             Radius:RealWidth,'  ',
                             STypeStr[SourceType],'  ',
                             Head:RealWidth);
           end;
         end;

         {read in interior node data for this zone }
         writeln(OutFile);
         writeln(OutFile,'   Interior node definition -- Zone ',Zone:3);
         writeln(OutFile);
         writeln(OutFile,'Node    X Coor         Y Coor');
         writeln(OutFile,'----    ------         ------');
         if NumInt > 0 then
         for Node := 1 to NumInt do
         with CurrINode do
         begin
           Readln(InFile,Coord[x], Coord[y]);
           PutIntNode(Zone,Node,CurrINode);
           writeln(Outfile,succ(Node):4,'   ',
                           Coord[x]:RealWidth,'   ',
                           Coord[y]:RealWidth);
         end;
       end; { of zone loop }
END; { procedure get_DATA }

{--->>>Open_Text_File<<<-----------------------------------------------------}
{   procedure to open text files for either reading of writing.              }
{----------------------------------------------------------------------------}
procedure OPEN_TEXT_FILE(var File_to_Open : text;
                             Name_File : File_name;
                             Flag : File_Op);

   begin
      Assign(File_To_Open,Name_File);
      Case Flag of
         Rd : begin
                  {$I-}
                  Reset(File_To_Open);
                  {$I+}
                  if IOresult <> 0 then
                  begin
                     writeln(^G);
                     writeln('File does not exist ');
                     halt;  {Stop program}
                  end;
              end; { of Read Case }
         Wrt: Rewrite(File_To_Open);
      end; { of case }
   end; { of OPEN_TEXT_FILE }


procedure WriteFloatTPV(Mat: TPVArray.TPArray; Size :word; Length, Decimal : byte;
                        Header : Titlestring);
  {- output formatted listing of TP virtual array to outfile }

Const Pagewidth = 80;

Var
   Min, Max, Mn1, Row, Col, Numberval : word;
   Blank, Padstr1, Padstr2   : string[10];

BEGIN
   Blank := '          ';
   Numberval := (Pagewidth - 5) div (length + 1) - 1;
   Max := 0;
   Writeln(Outfile);
   Writeln(Outfile,Header);
   Writeln(Outfile);
   Repeat
     Min := Max + 1;
     Max := Min + Numberval;
```

```
      MN1 := Size - Min;
      If (Mn1 < Numberval + 1) then Max := Size;
      Write(Outfile, 'R/C');
      PadStr1 := copy(Blank,1,Length div 2 + 1);
      Padstr2 := copy(Blank,1,Length div 2 - 2);
      For Col := Min to Max do Write(Outfile,Padstr1,Col:3,Padstr2);
      Writeln(Outfile);
      For Row := 1 to Size do
      Begin
         Write(Outfile,Row:3,'  ');
         For Col := Min To Max do Write(Outfile,
                                 gvfloat(Mat,Row,Col):Length:Decimal,' ');
         Writeln(Outfile);
      End;
      Writeln(Outfile);
   Until Max >= Size;
   Writeln(Outfile,'--------------------------');
End; { procedure write matrix }

procedure Dump_System(Suffix : File_Ext);
  {- outputs system element definitions }
var Dumpfile : text;
    DumpFileName : File_Name;

begin
  DumpFileName := ForceExtension(OutFileName,Suffix);
  Open_Text_File(DumpFile,DumpfileName,Wrt);
  for Zone := 1 to Num_Zones do
  with ZoneD[Zone] do
  begin
    writeln(DumpFile,'Zone : ',Zone:3,' data dump ++++++');
    writeln(DumpFile);
    writeln(DumpFile,'StartRow : ',StartRow);
    writeln(Dumpfile);
    for Element := 1 to NumElems do
    begin
      CurrElNum := GetElementNum(Elements,Element);
      GetElement(CurrElNum,CurrEl);
      with CurrEl do
      begin
        writeln(DumpFile,'Zonal El #: ',Element:5,'  ','Global El #: ',CurrElNum:5);
        writeln(DumpFile,'Element SF Type: ',ElSFTypeStr[ElSFType]);
        writeln(DumpFile,'Anode ',A.Node:5,'   ',
                         'ADOF ',A.DOF :5,'  |',
                         'Bnode ',B.Node:5,'   ',
                         'BDOF ',B.DOF :5)
      end;
    end;
  end;
  close(DumpFile);
end;

procedure DumpH_F(Var H, F : TPVArray.TPArray;
                   RowSize, ColSize :word; Length, Decimal : byte;
                       Header : Titlestring);
  { dumps matrix out in row order in a linear list }
Var
  Mn1, Row, Col, Numberval : byte;
  Min, Max  : float;
  GrdFile : text;
  GrdFileName : File_Name;

BEGIN
  GrdFileName := ForceExtension(OutFileName,'MAT');
  Assign(GrdFile,GrdFileName);
  Rewrite(GrdFile);
  Min := 0.0;
  Max := 0.0;
  { find max-min of mat }
  for Row := 1 to RowSize do
  begin
    if Min > gvfloat(F,Row,1) then Min := gvfloat(F,Row,1);
    if Max < gvfloat(F,Row,1) then Max := gvfloat(F,Row,1);
    for col := 1 to ColSize do
    begin
      if Min > gvfloat(H,Row,Col) then Min := gvfloat(H,Row,Col);
```

```
        if Max < gvfloat(H,Row,Col) then Max := gvfloat(H,Row,Col);
      end;
    end;
    Writeln(GrdFile,'DSAA');
    Writeln(GrdFile,ColSize + 1,'  ',RowSize);
    Writeln(GrdFile,1,'  ',ColSize+1);
    Writeln(GrdFile,1,'  ',RowSize);
    Writeln(GrdFile,Min:12,'  ',Max:12);
    for Row := 1 to RowSize do
    begin
       For Col := 1 to ColSize do
          Write(GrdFile,gvfloat(H,Row,Col):Length:Decimal,' ');
       Writeln(GrdFile,gvfloat(F,Row,1):Length:Decimal);
    end;
    Close(GrdFile);
End; { procedure write matrix }

procedure WriteBoundarySolution;
  {- formatted output for LINEAR element solution }

Const Numberval = 2;

Var
  Min, Max, Mn1, J : byte;
  Sign : shortint;
  Conductivity : float;

BEGIN
  Writeln(Outfile);
  Writeln(Outfile,'>>> Boundary values ');
  Writeln(Outfile);
  For Zone := 1 to Num_Zones do
  with ZoneD[Zone] do
  begin
    Conductivity := Kx; {isotropic for now}
    Max := 0;
    writeln(Outfile,'Zone : ',Zone:3);
    Repeat
       Min := Max + 1;
       Max := Min + Numberval;
       MN1 := NumElems - Min;
       If (Mn1 < Numberval + 1) then Max := NumElems;
       Write(Outfile, 'Zone Elem');
       For J := Min to Max do Write(Outfile,Center(Long2Str(J),Width*2));
       Writeln(Outfile);
       write(OutFile, 'Global El');
       For J := Min to Max do
       begin
         CurrElNum := GetElementNum(Elements, J);
         write(OutFile,Center(Long2Str(CurrElNum),Width*2));
       end;
       writeln(OutFile);
       write(OutFile,'Elem. Node ');
       for J := Min to Max do write(OutFile,CenterCh('A','-',Width),
                                            CenterCh('B','-',Width));
       writeln(OutFile);
       Write(Outfile,'X-coor     ');
       For J := Min To Max do
       begin
         CurrElNum := GetElementNum(Elements, J);
         GetElement(CurrElNum, CurrEl);
         Write(Outfile,GNode^[CurrEl.A.Node][X]:Width:Places,
                       GNode^[CurrEl.B.Node][X]:Width:Places);
       end;
       Writeln(Outfile);
       Write(OutFile,'Y-coor     ');
       For J := Min To Max do
       begin
         CurrElNum := GetElementNum(Elements, J);
         GetElement(CurrElNum, CurrEl);
         Write(Outfile,GNode^[CurrEl.A.Node][Y]:Width:Places,
                      GNode^[CurrEl.B.Node][Y]:Width:Places);
       end;
       Writeln(Outfile);
       Write(Outfile,'N Flux in  ');
       For J := Min To Max do
```

```
        begin
          CurrElNum := GetElementNum(Elements, J);
          Sign := abs(CurrElNum) div CurrElNum;
          GetElement(CurrElNum, CurrEl);
          Write(Outfile,-Sign * CurrEl.A.DPhi/Conductivity:Width:Places,
                        -Sign * CurrEl.B.DPhi/Conductivity:Width:Places);
        end;
        Writeln(Outfile);
        Write(Outfile,'N Flow in  ');
        For J := Min To Max do
        begin
          CurrElNum := GetElementNum(Elements, J);
          Sign := abs(CurrElNum) div CurrElNum;
          GetElement(CurrElNum, CurrEl);
          Write(Outfile,Sign * CurrEl.A.DPhi:Width:Places,
                        Sign * CurrEl.B.DPhi:Width:Places);
        end;
        Writeln(Outfile);
        Write(Outfile,'Potential  ');
        For J := Min To Max do
        begin
          CurrElNum := GetElementNum(Elements, J);
          GetElement(CurrElNum, CurrEl);
          Write(Outfile,CurrEl.A.Phi:Width:Places,CurrEl.B.Phi:Width:Places);
        end;
        Writeln(Outfile);
        writeln(OutFile);
      Until Max >= NumElems;
    end; { with/for }
End; { procedure write boundary solution }

procedure WriteSourceSolution;
  {- formatted output for source solution }

Const Numberval = 5;

Var
  Min, Max, Mn1, J : byte;


BEGIN
  Writeln(Outfile);
  Writeln(Outfile,'>>> Source/sink values ');
  Writeln(Outfile);
  For Zone := 1 to Num_Zones do
  with ZoneD[Zone] do
  begin
    Max := 0;
    writeln(Outfile,'Zone : ',Zone:3);
    if NumWells > 0 then
    Repeat
      Min := Max + 1;
      Max := Min + Numberval;
      MN1 := NumWells - Min;
      If (Mn1 < Numberval + 1) then Max := NumWells;
      Write(Outfile, 'Well      ');
      For J := Min to Max do Write(Outfile,Center(Long2Str(J),Width));
      Writeln(Outfile);
      Write(Outfile,'X-coor     ');
      For J := Min To Max do
      begin
        GetWell(Zone,J,CurrWell);
        Write(Outfile,CurrWell.Coord[X]:Width:Places);
      end;
      Writeln(Outfile);
      Write(OutFile,'Y-coor     ');
      For J := Min To Max do
      begin
        GetWell(Zone,J,CurrWell);
        Write(Outfile,CurrWell.Coord[Y]:Width:Places);
      end;
      Writeln(Outfile);
      Write(Outfile,'Potential  ');
      For J := Min To Max do
      begin
        GetWell(Zone,J,CurrWell);
        Write(Outfile,CurrWell.Head:Width:Places);
      end;
      Writeln(Outfile);
      Write(Outfile,'Flow (in-) ');
      For J := Min To Max do
      begin
        GetWell(Zone,J,CurrWell);
        Write(Outfile,CurrWell.Flow:Width:Places);
      end;
      Writeln(Outfile);
      writeln(OutFile);
    Until Max >= NumWells;
  end; { with/for }
End; { procedure write source solutions }

procedure WriteInteriorSolution;
  {- formatted output for interior node solution }

Const Numberval = 5;

var
  Min, Max, Mn1, J : byte;

begin
  Writeln(Outfile);
  Writeln(Outfile,'>>> Interior node values ');
  Writeln(Outfile);
  For Zone := 1 to Num_Zones do
  with ZoneD[Zone] do
  begin
    Max := 0;
    writeln(Outfile,'Zone : ',Zone:3);
    if NumInt > 0 then
    Repeat
      Min := Max + 1;
      Max := Min + Numberval;
      MN1 := NumInt - Min;
      If (Mn1 < Numberval + 1) then Max := NumInt;
      Write(Outfile, 'Int. Node');
      For J := Min to Max do Write(Outfile,Center(Long2Str(J),Width));
      Writeln(Outfile);
      Write(Outfile,'X-coor     ');
      For J := Min To Max do
      begin
        GetIntNode(Zone,J,CurrINode);
        Write(Outfile,CurrINode.Coord[X]:Width:Places);
      end;
      Writeln(Outfile);
      Write(OutFile,'Y-coor     ');
      For J := Min To Max do
      begin
        GetIntNode(Zone,J,CurrINode);
        Write(Outfile,CurrINode.Coord[Y]:Width:Places);
      end;
      Writeln(Outfile);
      Write(Outfile,'Potential  ');
      For J := Min To Max do
      begin
        GetIntNode(Zone,J,CurrINode);
        Write(Outfile,CurrINode.Phi:Width:Places);
      end;
      Writeln(Outfile);
      Write(Outfile,'Flow X-dir ');
      For J := Min To Max do
      begin
        GetIntNode(Zone,J,CurrINode);
        Write(Outfile,CurrINode.DPhiX:Width:Places);
      end;
      Writeln(Outfile);
      Write(Outfile,'Flow Y-dir ');
      For J := Min To Max do
      begin
        GetIntNode(Zone,J,CurrINode);
        Write(Outfile,CurrINode.DPhiY:Width:Places);
      end;
      Writeln(Outfile);
      writeln(OutFile);
    Until Max >= NumInt;
  end; { with/for }
End; { procedure write interior Solutions }

procedure writeGridFile;
  {- prints out solution grid files for contouring }
var
   Phifile,
   DPXfile,
   DPYfile : text;
   PhiFileName,
   DPXFileName,
   DPYFileName: File_Name;

begin
  { write out phi solution }
  PHIFileName := ForceExtension(OutFileName,'PHI');
  Open_Text_File(PHIFile, PHIFileName,Wrt);
  for Zone := 1 to Num_Zones do
  with ZoneD[Zone] do
  begin

    { write out leading node for each LINEAR element - for now }
    for Element := 1 to NumElems do
    begin
      CurrElNum := GetElementNum(Elements, Element);
      if CurrElNum > 0 then
      begin
        GetElement(CurrElNum, CurrEl);
        with CurrEl.A do  { write only leading nodes for now-quadratic nodes will need other code }
        begin
          if not NodeF^[Node] then
          begin
            Writeln(PhiFile,GNode^[Node][X]:Width:Places,' ',
                            GNode^[Node][Y]:Width:Places,' ',
                            Phi:Width:Places);
            NodeF^[Node] := True;
          end;
        end;
      end;
    end;
    { write out interior phis }
    for Node := 1 to NumInt do
    begin
      GetIntNode(Zone,Node,CurrINode);
      with CurrINode do
        Writeln(PhiFile,Coord[X]:Width:Places,' ',
                        Coord[Y]:Width:Places,' ',
                        Phi:Width:Places);
    end;
    { write out well heads }
    for Node := 1 to NumWells do
    begin
      GetWell(Zone,Node,CurrWell);
      with CurrWell do
        Writeln(PhiFile,Coord[X]:Width:Places,' ',
                        Coord[Y]:Width:Places,' ',
                        Head:Width:Places);
    end;
  end;
  Close(PHIFile);

  DPXFileName := ForceExtension(OutFileName,'DPX');
  DPYFileName := ForceExtension(OutFileName,'DPY');
  Open_Text_File(DPXFile, DPXFileName,Wrt);
  Open_Text_File(DPYFile, DPYFileName,Wrt);
  for Zone := 1 to Num_Zones do
  with ZoneD[Zone] do
    for Node := 1 to NumInt do
    begin
      GetIntNode(Zone,Node,CurrINode);
      with CurrINode do
      begin
        Writeln(DPXFile,Coord[X]:Width:Places,' ',
                        Coord[Y]:Width:Places,' ',
                          DPhiX:Width:Places);
        Writeln(DPYFile,Coord[X]:Width:Places,' ',
                          Coord[Y]:Width:Places,' ',
                          DPhiY:Width:Places);
      end;
    end;
  Close(DPXFile);
  Close(DPYFile);
end; { of WritegridSol }

procedure DumpSolVec;
  {- dumps solution vector to outfile }
var DOF : globaldof;
begin
  writeln(OutFile);
  writeln(OutFile,'===> Solution vector dump ');
  for DOF := 1 to DOFCount do
    writeln(OutFile,DOF:4,'  ',gvfloat(F,DOF, 1):12:4);
  writeln(OutFile);
end;

begin
  GetDate(Year, Month, Day, DayofWeek);
  GetTime(Hour, Min, Sec, Sec100);
end. { of Unit B7FILE <******************************<< }
```

```
Unit B7Solver;
  {-----------------------------------------------------

                  Program GWBEM - Unit B7SOLVER

     Contains code for solving system equations using
     virtual arrays.

     Copyright (c) 1989, Mark A. Lisbe and Iowa State
     University

     ALL RIGHTS RESERVED

     TPRArray and TPVArray units copywrite (c) 1987 by
     TurboPower Software.  Part of Turbo Professional
     Programmer's Toolbox V4.0.  For information, contact:

        TurboPower Software
        3109 Scotts Valley Drive, Suite 122
        Scotts Valley, CA  95066
        (408) 438-8608

   Last modified : 10/30/88  11:37 AM
   -----------------------------------------------------}

interface

uses TPRArray,
     TPVArray,
     TPArr;

function Solver(Size : word;
                var A,B : Pointer; { system matrix/ known vector }
                var Cond_Num : float) : boolean;
  {- main routine for system solver.  Returns false if system singular }

{======================================================================}
implementation

uses B7Utils;

type str80 = string[80];

const ZeroB : byte = 0;
      ZeroF : float = 0.0;
      MaxFloat : float = 1.7e308;

var IPVT : Pointer;

{--->SOLVE<-----------------------------------------------------------}
{       Routine for the solution of [A] (x) = (B) system of equations   }
{       Note: all matrices are ZERO based, meaning first index is always 0. }
{       As such, this routine will look somewhat awkward.  Access to all   }
{       matrices is forced through function g?float and procedure p?float,  }
{       since these are dynamic matrices.                                   }
{       A is a virtual array, while all other vectors are RAM arrays.       }
{       Last modified:  August 29, 1988 9:34 AM                             }
{--------------------------------------------------------------------}
procedure SOLVE( Size : word;            { system size }
                 var A,B : Pointer);   { A matrix/ B vector     }
var
   I, J, K, KB, KP1, KM1, M : integer;
   T : float;

begin
   if ( Size <> 1 ) then
   begin
      for K := 1 to pred(Size) do
      begin
         KP1 := succ(K);
         M := grword(IPVT,K,1);
         T := gvfloat(B,M,1);
         pvfloat(B,M,1,gvfloat(B,K,1));
         pvfloat(B,K,1,T);
         for I := KP1 to Size do pvfloat(B,I,1,gvfloat(B,I,1) + gvfloat(A,I,K) * T);
      end; { K - loop }
      for KB := 1 to pred(Size) do
      begin
         KM1 := Size - KB;
         K := succ(KM1);
         pvfloat(B,K,1, gvfloat(B,K,1) / gvfloat(A,K,K));
         T := -gvfloat(B,K,1);
         for I := 1 to KM1 do pvfloat(B,I,1, gvfloat(B,I,1) + gvfloat(A,I,K) * T);
      end; { KB - loop }
   end; { Size - if }
   pvfloat(B,1,1, gvfloat(B,1,1) / gvfloat(A,1,1));
end; { SOLVE procedure }

{--->DECOMP<-----------------------------------------------------------------}
{           Decomposes the matrix A and returns the condition number         }
{           Adapted from:                                                    }
{           Forsythe G. E., M. A. Malcolm and C.E. Moler                     }
{           Computer Methods for Mathematical Computations                   }
{           Prentice-Hall, 1977.                                             }
{           Note: To calculate det of A, simply multiply the returned diag.  }
{           values together and then multiply product by +1 if even # or row }
{           interchanges, and by -1 if odd # of row interchanges.  # of int- }
{           erchanges is returned in last element of IPVT (i.e. IPVT[Size]  }
{           Last modified August 29, 1988 9:34 AM                            }
{----------------------------------------------------------------------------}
procedure DECOMP (Size : word;
                  var A : Pointer;
                  var Condition : float);
var I, K, J, M, KP1, KM1, KB : word;
    T, Anorm, Ynorm, Znorm, EK : float;
    Work : Pointer; { dynamic RAM array -- Turbo Professional v4.0}

  procedure SOLVEDR( Size : word;
                     var A,B : Pointer);
    {- special version of SOLVE routine to handle Virtual A mat & RAM B mat }
  var
     I, J, K, KB, KP1, KM1, M : integer;
     T : float;

  begin
     if ( Size <> 1 ) then
     begin
        for K := 1 to pred(Size) do
        begin
           KP1 := succ(K);
           M := grword(IPVT,K,1);
           T := grfloat(B,M,1);
           prfloat(B,M,1,grfloat(B,K,1));
           prfloat(B,K,1,T);
           for I := KP1 to Size do prfloat(B,I,1,grfloat(B,I,1) + gvfloat(A,I,K) * T);
        end; { K - loop }
        for KB := 1 to pred(Size) do
        begin
           KM1 := Size - KB;
           K := succ(KM1);
           prfloat(B,K,1, grfloat(B,K,1) / gvfloat(A,K,K));
           T := -grfloat(B,K,1);
           for I := 1 to KM1 do prfloat(B,I,1, grfloat(B,I,1) + gvfloat(A,I,K) * T);
        end; { KB - loop }
     end; { Size - if }
     prfloat(B,1,1, grfloat(B,1,1) / gvfloat(A,1,1));
  end; { SOLVE procedure }

begin
   { make the Work vector }
   TPRArray.MakeA(Work, Size, 1, sizeof(Float));
   TPRArray.ClearA(Work,ZeroB,TPRArray.FastInit);
   CONDITION := MaxFloat;       { Set init value of condition - reset later }
   prword(IPVT,Size,1,1);       { This is used to in determinant calculation }
   if (Size <> 1) then
   begin
      Anorm := 0.0;
      for J := 1 to Size do
      begin
         T := 0.0;
         for I := 1 to Size do T := T + abs(gvfloat(A,I,J));
         if T > Anorm then Anorm := T;
        end;
        for K := 1 to pred(Size) do
        begin
           KP1 := succ(K);
           M := K;
           for I := KP1 to Size do
             if abs(gvfloat(A,I,K)) > abs(gvfloat(A,M,K)) then M := I;
           prword(IPVT,K,1,M);

           { IPVT(Size) contains the # of row interchanges done. }
           if ( M <> K ) then prword(IPVT,Size,1,succ(grword(IPVT,Size,1)));
           T := gvfloat(A,M,K);
           pvfloat(A,M,K, gvfloat(A,K,K));
           pvfloat(A,K,K,T);
           if (T <> 0.0) then
           begin
              for I := KP1 to Size do pvfloat(A,I,K, -gvfloat(A,I,K)/T);
              for J := KP1 to Size do
              begin
                 T := gvfloat(A,M,J);
                 pvfloat(A,M,J,gvfloat(A,K,J));
                 pvfloat(A,K,J,T);
                 if (T <> 0.0) then
                 for I := KP1 to Size do pvfloat(A,I,J,gvfloat(A,I,J) + gvfloat(A,I,K) * T);
              end; { J loop }
           end; { T - if }
        end; { K loop }
        for K := 1 to Size do
        begin
           T := 0.0;
           if ( K <> 1 ) then
           begin
              KM1 := pred(K);
              for I := 1 to KM1 do T := T + gvfloat(A,I,K) * grfloat(Work,i,1);
           end; { K - if }
           EK := 1.0;
           if ( T < 0.0 ) then EK := -1.0;
           if ( gvfloat(A,K,K) = 0.0) then
           begin
             TPRArray.DisposeA(Work);
             exit; { to main block }
           end;
           prfloat(WORK,K,1, -(EK + T)/gvfloat(A,K,K));
        end; { K - loop }
        for KB := 1 to pred(Size) do
        begin
           K := Size - KB;
           T := 0.0;
           KP1 := succ(K);
           for I := KP1 to Size do T := T + gvfloat(A,I,K) * grfloat(Work,k,1);
           prfloat(WORK,K,1,T);
           M := grword(IPVT,K,1);
           if (M <> K)  then
           begin
              T := grfloat(WORK,M,1);
              prfloat(WORK,M,1,grfloat(WORK,K,1));
              prfloat(WORK,K,1,T);
           end; { M - if }
        end; { KB - loop }
        YNORM := 0.0;
        for I := 1 to Size do Ynorm := Ynorm + abs(grfloat(WORK,I,1));
        SOLVEDR(Size, A, WORK);
        Znorm := 0.0;
        for I := 1 to Size do Znorm := Znorm + abs(grfloat(WORK,I,1));
        CONDITION := Anorm * Znorm / Ynorm;
        if (CONDITION < 1.0 ) then CONDITION := 1.0;
    end { Size - if }
    else if gvfloat(A,1,1) <> 0.0 then CONDITION := 1.0;
    TPRArray.DisposeA(Work);
end; { DECOMP procedure }


function Solver(Size : word;
                var A,B : Pointer; { system matrix/ known vector }
                var Cond_Num : float) : boolean;
  {- main routine for system solver.  Returns false if system singular }
```

```
begin
   Solver := False;
   TPRArray.MakeA(IPVT,Size,1, sizeof(word));    { use RAM array for pivot vector }
   TPRArray.ClearA(IPVT, ZeroB, TPRArray.FastInit);
   StartTimer('Decomposing system matrix');
   Decomp(Size, A, Cond_Num);
   StopTimer('decompose system matrix');
   If (Cond_Num + 1.0) = Cond_Num then exit; { singular system w/in precision of machine }
   StartTimer('Solving system equations');
   Solve(Size,A,B);
   StopTimer('solve system of equations');
   TPRArray.DisposeA(IPVT);
   Solver := True;
End; { procedure solve }

end. { of Unit B7Solver <*****************************<< }
```

```
Unit B7Error;

  {-----------------------------------------------------

                 Program GWBEM - Unit B7Error

    Contains error trapping code for GWBEM.

    Copyright (c) 1989, Mark A. Liebe and Iowa State
    University

    ALL RIGHTS RESERVED


   Last modified : 10/30/88
   -----------------------------------------------------}

interface

Uses B7DEF;

type IOErrset = set of byte;
     ErrorStr = string[40];

procedure ShowError(ErrorMes : ErrorStr; Stop : boolean);
  {- puts up error window and stops program if stop true }

function IO_Bad(IORes : word; OKSet : IOErrSet) : boolean;
  {- checks IOResult, if in OK error set then returns false, else displays }
  {  message and returns true }

procedure ErrorMem;
  {- last ditch bail out }

{===================================================================}
{$F+}
implementation

uses TPString,
     TPCrt,
     TPWindow,
     TPVArray;

const ErrorAttr : byte = $4E;
      Escape = #27;

var
  CH : Char;
  ErAttr : byte;
  savedExitProc : Pointer; { old exitProc Pointer }

procedure ErrorMem;
begin
  Window(1,1,80,25);
  NormVideo;
  Clrscr;
  Writeln('Insufficient Memory');
  Halt(1);
end;

function ErrorText(ErrorNum : word) : errorStr;
  {- returns error message associated with error messages}
begin
  case ErrorNum of
      2: ErrorText := 'File not found';
      3: ErrorText := 'Path not found';
      4: ErrorText := 'Too many open files';
      5: ErrorText := 'File access denied';
      6: ErrorText := 'Invalid file handle';
     12: ErrorText := 'Invalid file access code';
     15: ErrorText := 'Invalid drive number';
     16: ErrorText := 'Cannot remove current directory';
     17: ErrorText := 'Cannot rename across drives';
    100: ErrorText := 'Disk read error';
    101: ErrorText := 'Disk write error';
    102: ErrorText := 'File not assigned';
```

```
        103: ErrorText := 'File not open';
        104: ErrorText := 'File not open for input';
        105: ErrorText := 'File not open for output';
        106: ErrorText := 'Invalid numeric format';
        150: ErrorText := 'Disk is write-protected';
        151: ErrorText := 'Unknown unit';
        152: ErrorText := 'Drive not ready';
        153: ErrorText := 'Unknown command';
        154: ErrorText := 'CRC error in data';
        155: ErrorText := 'Bad drive request structure length';
        156: ErrorText := 'Disk seek error';
        157: ErrorText := 'Unknown media type';
        158: ErrorText := 'Sector not found';
        159: ErrorText := 'Printer out of paper';
        160: ErrorText := 'Device write fault';
        161: ErrorText := 'Device read fault';
        162: ErrorText := 'Hardware failure';
        200: ErrorText := 'Division by zero';
        201: ErrorText := 'Range check error';
        202: ErrorText := 'Stack overflow error';
        203: ErrorText := 'Heap overflow error';
        204: ErrorText := 'Invalid pointer operation';
        205: ErrorText := 'Floating point overflow';
        206: ErrorText := 'Floating point underflow';
        207: ErrorText := 'Invalid floating point operation';
        else
        ErrorText := 'Unknown Error # '+ Long2Str(ExitCode)
    end;
 end; { func ErrorText }

 procedure WriteIOError( IORes : word);
   {- puts up error window and writes IO error message }
 var
   IOText : ErrorStr;
   IOWindow : WindowPtr;
 begin
     IoText := ErrorText(IORes);
     FrameChars := '┌└┐┘─│';
     if not MakeWindow(IOWindow,19,14,61,21,True,True,False,ErAttr,ErAttr,ErAttr,'')
       then ErrorMem;
     if Not DisplayWindow(IOWindow) THEN ErrorMem;
     FastWriteWindow(Center('-- IO ERROR --',40),1,1,ErAttr);
     FastWriteWindow(Center(IOText,40),3,1,ErAttr);
     FastWriteWindow(Center('Press ESCAPE',40),5,1,ErAttr);
     repeat until Readkey = Escape;
     IOWindow := EraseTopWindow;
     DisposeWindow(IOWindow);
 end;  { of WriteIOError }

 function IO_Bad(IORes : word; OKSet : IOErrSet) : boolean;
   {- checks IOResult, if in OK error set then returns false, else displays }
   {  message and returns true }
 begin
   IO_Bad := True;
   OKSet := OKSet + [0];
   if byte(IORes) in OKSet then
   begin
     IO_Bad := False;
     Exit
   end
   else WriteIOError(IoRes);
 end;  { of IO_Bad }

 procedure Cleanup;
   {- frees arrays, closes files, etc }
 begin
   if H <> nil then DisposeA(H,True);
   if F <> nil then DisposeA(F,True);
   {$I-}
   Close(InFile);
   Close(OutFile);
   {$I+}
 end; { proc Cleanup }

 procedure ShowError(ErrorMes : ErrorStr; Stop : boolean);
```

```
  {- puts up error window and stops program if stop true }
 var ErrorMesWindow : WindowPtr;
 begin
   FrameChars := '┌└┐┘─│';
   if not MakeWindow(ErrorMesWindow,19,14,61,21,True,True,False,ErAttr,ErAttr,ErAttr,'')
     then ErrorMem;
   if Not DisplayWindow(ErrorMesWindow) THEN ErrorMem;
   FastWriteWindow(Center(ErrorMes,40),3,1,ErAttr);
   FastWriteWindow(Center('Press ESCAPE',40),5,1,ErAttr);
   repeat until Readkey = Escape;
   DisposeWindow(ErrorMesWindow);
   NormalCursor;
   if Stop then
   begin
     CleanUp;
     Halt;
   end;
 end; { proc ShowError }

 PROCEDURE ExitSolver;
   {- custom error handler }
 var ExitText : ErrorStr;
     ErrorWindow : WindowPtr;
 begin
   IF ErrorAddr <> Nil THEN
   begin
     ExitText := ErrorText(ExitCode);
     FrameChars := '┌└┐┘─│';
     if not MakeWindow(ErrorWindow,19,14,61,21,True,True,False,ErAttr,ErAttr,ErAttr,'')
       then ErrorMem;
     if Not DisplayWindow(ErrorWindow) THEN ErrorMem;
     FastWriteWindow(Center(ExitText,40),3,1,ErAttr);
     FastWriteWindow(Center('Press ESCAPE',40),5,1,ErAttr);
     repeat until Readkey = Escape;
     DisposeWindow(ErrorWindow);
     NormalCursor;
     ErrorAddr := Nil;
   end;
   Cleanup;
   ExitProc := SavedExitProc;
   halt;
 end;

 begin
   savedExitProc := exitProc;          { install current exit proc for this unit }
   ExitProc := @ExitSolver;
   MapColors := True;
   ErAttr := MapColor(ErrorAttr);
 end. { of Unit B7Error <******************************<< }
```

```
Unit TPArr;

  {---------------------------------------------------------

                    Program GWBEM - Unit TPARR

     Contains code for accessing virtual arrays.  Used this
     technique in case data structure changed so as to avoid
     extensive changes to solution code.

     Copyright (c) 1989, Mark A. Lieba and Iowa State
     University

     ALL RIGHTS RESERVED

     TPRArray and TPVArray units copywrite (c) 1987 by
     TurboPower Software.  Part of Turbo Professional
     Programmer's Toolbox V4.0.  For information, contact:

         TurboPower Software
         3109 Scotts Valley Drive, Suite 122
         Scotts Valley, CA  95066
         (408) 438-8608

   Last modified :  9/21/88   1:29 PM
   ---------------------------------------------------------}

interface

uses TPRArray,
     TPVarray;

type float = double;

procedure Printeger(var Arr : Pointer; Row, Col : integer; val : integer);
  {- puts integer value in RAM array }

function Grinteger(var Arr : Pointer; Row, Col : integer) : integer;
  {- gets integer value from RAM array }

procedure Prword(var Arr : Pointer; Row, Col : word; val : word);
  {- puts word value in RAM array }

function Grword(var Arr : Pointer; Row, Col : word) : word;
  {- gets word value from RAM array }

procedure Prfloat(var Arr : Pointer; Row, Col : word; val : float);
  {- puts float value in RAM array }

function Grfloat(var Arr : Pointer; Row, Col : word) : float;
  {- gets float value from RAM array }

procedure Pvword(var Arr : Pointer; Row, Col : word; val : word);
  {- puts word value in VIRTUAL array }

function Gvword(var Arr : Pointer; Row, Col : word) : word;
  {- gets word value from VIRTUAL array }

procedure Pvfloat(var Arr : Pointer; Row, Col : word; val : float);
  {- puts float value in VIRTUAL array }

function Gvfloat(var Arr : Pointer; Row, Col : word) : float;
  {- gets float value from VIRTUAL array }

{======================================================================}
implementation

procedure Printeger(var Arr : Pointer; Row, Col : integer; val : integer);
  {- puts integer value in RAM array }
begin
  TPRArray.SetA(Arr, pred(Row), pred(Col), val);
end;

function Grinteger(var Arr : Pointer; Row, Col : integer) : integer;
  {- gets integer value from RAM array }
var Temp : integer;
```

```
begin
  TPRArray.RetA(Arr, pred(Row), pred(Col), Temp); Grinteger := Temp;
end;

procedure Prword(var Arr : Pointer; Row, Col : word; val : word);
  {- puts word value in RAM array }
begin
  TPRArray.SetA(Arr, pred(Row), pred(Col), val);
end;

function Grword(var Arr : Pointer; Row, Col : word) : word;
  {- gets word value from RAM array }
var Temp : word;
begin
  TPRArray.RetA(Arr, pred(Row), pred(Col), Temp); Grword := Temp;
end;

procedure Prfloat(var Arr : Pointer; Row, Col : word; val : float);
  {- puts float value in RAM array }
begin
  TPRArray.SetA(Arr, pred(Row), pred(Col), val);
end;

function Grfloat(var Arr : Pointer; Row, Col : word) : float;
  {- gets float value from RAM array }
var Temp : float;
begin
  TPRArray.RetA(Arr, pred(Row), pred(Col), Temp); Grfloat := Temp;
end;

procedure Pvword(var Arr : Pointer; Row, Col : word; val : word);
  {- puts word value in VIRTUAL array }
begin
  TPVArray.SetA(Arr, pred(Row), pred(Col), val);
end;

function Gvword(var Arr : Pointer; Row, Col : word) : word;
  {- gets word value from VIRTUAL array }
var Temp : word;
begin
  TPVArray.RetA(Arr, pred(Row), pred(Col), Temp); Gvword := Temp;
end;

procedure Pvfloat(var Arr : Pointer; Row, Col : word; val : float);
  {- puts float value in VIRTUAL array }
begin
  TPVArray.SetA(Arr, pred(Row), pred(Col), val);
end;

function Gvfloat(var Arr : Pointer; Row, Col : word) : float;
  {- gets float value from VIRTUAL array }
var Temp : float;
begin
  TPVArray.RetA(Arr, pred(Row), pred(Col), Temp); Gvfloat := Temp;
end;

end. { of Unit TPArr <*****************************<< }
```

APPENDIX B: SAMPLE INPUT AND MAIN OUTPUT FILES FOR MODEL GWBEM

```
USGS problem C2;one zone ( K=2);one well @ center;no potential at boundary
    28  ; Number global boundary nodes
1          0.00      0.00    ; Global node definition
2          0.00      2.00
3          0.00      4.00
4          0.00      6.00
5          0.00      8.00
6          2.00      8.00
7          4.00      8.00
8          6.00      8.00
9          8.00      8.00
10        10.00      8.00
11        12.00      8.00
12        14.00      8.00
13        16.00      8.00
14        18.00      8.00
15        20.00      8.00
16        20.00      6.00
17        20.00      4.00
18        20.00      2.00
19        20.00      0.00
20        18.00      0.00
21        16.00      0.00
22        14.00      0.00
23        12.00      0.00
24        10.00      0.00
25         8.00      0.00
26         6.00      0.00
27         4.00      0.00
28         2.00      0.00
28      ; Number of boundary elements
 1     1 0       0.00            2        0        0.0
 2     2 0       0.00            3        0        0.0
 3       3       0        0.00           4        0        0.0
 4       4       0        0.00           5        0        0.0
 5       5       1        0.0            6        1        0.0
 6       6       1        0.00           7        1        0.0
 7       7       1        0.00           8        1        0.00
 8       8       1        0.00           9        1        0.00
 9       9       1        0.00          10        1        0.00
10      10       1        0.00          11        1        0.00
11      11       1        0.00          12        1        0.00
12      12       1        0.00          13        1        0.00
13      13       1        0.00          14        1        0.00
14      14       1        0.00          15        1        0.00
15      15       0        0.00          16        0        0.00
16      16       0        0.00          17        0        0.00
17      17       0        0.00          18        0        0.00
18      18       0        0.00          19        0        0.00
19      19       1        0.00          20        1        0.00
20      20       1        0.00          21        1        0.00
21      21       1        0.00          22        1        0.00
22      22       1        0.00          23        1        0.00
23      23       1        0.00          24        1        0.00
24      24       1        0.00          25        1        0.00
25      25       1        0.00          26        1        0.00
26      26       1        0.00          27        1        0.00
27      27       1        0.00          28        1        0.00
28      28       1        0.00           1        1        0.00
  1     ; Number of zones
28 1 60 2.0 2.0 0.0    ; Numelems, numwells, numinterior, Kx, Ky, ThetaX
         1        2        3        4        5        6        7        8
         9       10       11       12       13       14       15       16
        17
        18
        19
        20
        21
        22
        23
        24
        25
        26
        27
        28
1 10.0 4.0 0.05 0 100.0 ; Well number, x-coor, y-coor, radius, type, known val
         1.50      1.50  ; Interior node definitions
         1.50      2.50
 1.50    3.50
 1.50    4.50
 1.50    5.50
'1.50    6.50
 3.50    1.50
 3.50    2.50
 3.50    3.50
 3.50    4.50
 3.50    5.50
 3.50    6.50
 5.50    1.50
 5.50    2.50
 5.50    3.50
 5.50    4.50
 5.50    5.50
 5.50    6.50
 7.50    1.50
 7.50    2.50
 7.50    3.50
 7.50    4.50
 7.50    5.50
 7.50    6.50
 9.50    1.50
 9.50    2.50
 9.50    3.50
 9.50    4.50
 9.50    5.50
 9.50    6.50
11.50    1.50
11.50    2.50
11.50    3.50
11.50    4.50
11.50    5.50
11.50    6.50
13.50    1.50
13.50    2.50
13.50    3.50
13.50    4.50
13.50    5.50
13.50    6.50
15.50    1.50
15.50    2.50
15.50    3.50
15.50    4.50
15.50    5.50
15.50    6.50
17.50    1.50
17.50    2.50
17.50    3.50
17.50    4.50
17.50    5.50
17.50    6.50
19.50    1.50
19.50    2.50
19.50    3.50
19.50    4.50
19.50    5.50
19.50    6.50
```

```
USGS problem C2;one zone ( K=2);one well @ center;no potential at boundary

  Run date:    3/03/1989  Run time:   10:01

***> Global Node Information <***
Number Global Nodes :  28

---> Global Node Definition <---

Node           X-Coor           Y-Coor
----           ------           ------

   1      0.000E+0000      0.000E+0000
   2      0.000E+0000      2.000E+0000
   3      0.000E+0000      4.000E+0000
   4      0.000E+0000      6.000E+0000
   5      0.000E+0000      8.000E+0000
   6      2.000E+0000      8.000E+0000
   7      4.000E+0000      8.000E+0000
   8      6.000E+0000      8.000E+0000
   9      8.000E+0000      8.000E+0000
  10      1.000E+0001      8.000E+0000
  11      1.200E+0001      8.000E+0000
  12      1.400E+0001      8.000E+0000
  13      1.600E+0001      8.000E+0000
  14      1.800E+0001      8.000E+0000
  15      2.000E+0001      8.000E+0000
  16      2.000E+0001      6.000E+0000
  17      2.000E+0001      4.000E+0000
  18      2.000E+0001      2.000E+0000
  19      2.000E+0001      0.000E+0000
  20      1.800E+0001      0.000E+0000
  21      1.600E+0001      0.000E+0000
  22      1.400E+0001      0.000E+0000
  23      1.200E+0001      0.000E+0000
  24      1.000E+0001      0.000E+0000
  25      8.000E+0000      0.000E+0000
  26      6.000E+0000      0.000E+0000
  27      4.000E+0000      0.000E+0000
  28      2.000E+0000      0.000E+0000

***> Global Element Information <***

Number Global Elements :  28

                   ----> Global Element Definition <----

              First node                     Second node
Elem| Node Type  Known Value       | Node Type  Known Value
----  ---- ----  -----------         ---- ----  -----------
   1|    1 Phi   0.000E+0000       |    2 Phi   0.000E+0000
   2|    2 Phi   0.000E+0000       |    3 Phi   0.000E+0000
   3|    3 Phi   0.000E+0000       |    4 Phi   0.000E+0000
   4|    4 Phi   0.000E+0000       |    5 Phi   0.000E+0000
   5|    5 DPhi  0.000E+0000       |    6 DPhi  0.000E+0000
   6|    6 DPhi  0.000E+0000       |    7 DPhi  0.000E+0000
   7|    7 DPhi  0.000E+0000       |    8 DPhi  0.000E+0000
   8|    8 DPhi  0.000E+0000       |    9 DPhi  0.000E+0000
   9|    9 DPhi  0.000E+0000       |   10 DPhi  0.000E+0000
  10|   10 DPhi  0.000E+0000       |   11 DPhi  0.000E+0000
  11|   11 DPhi  0.000E+0000       |   12 DPhi  0.000E+0000
  12|   12 DPhi  0.000E+0000       |   13 DPhi  0.000E+0000
  13|   13 DPhi  0.000E+0000       |   14 DPhi  0.000E+0000
  14|   14 DPhi  0.000E+0000       |   15 DPhi  0.000E+0000
  15|   15 Phi   0.000E+0000       |   16 Phi   0.000E+0000
  16|   16 Phi   0.000E+0000       |   17 Phi   0.000E+0000
  17|   17 Phi   0.000E+0000       |   18 Phi   0.000E+0000
  18|   18 Phi   0.000E+0000       |   19 Phi   0.000E+0000
  19|   19 DPhi  0.000E+0000       |   20 DPhi  0.000E+0000
  20|   20 DPhi  0.000E+0000       |   21 DPhi  0.000E+0000
  21|   21 DPhi  0.000E+0000       |   22 DPhi  0.000E+0000
  22|   22 DPhi  0.000E+0000       |   23 DPhi  0.000E+0000
  23|   23 DPhi  0.000E+0000       |   24 DPhi  0.000E+0000
  24|   24 DPhi  0.000E+0000       |   25 DPhi  0.000E+0000
  25|   25 DPhi  0.000E+0000       |   26 DPhi  0.000E+0000
  26|   26 DPhi  0.000E+0000       |   27 DPhi  0.000E+0000
  27|   27 DPhi  0.000E+0000       |   28 DPhi  0.000E+0000
  28|   28 DPhi  0.000E+0000       |    1 DPhi  0.000E+0000
```

```
***> Zone  Information <***

Number Zones:  1

---> Zone  1 date <---
    Number Boundary Elements: 28
    Number Wells:             1
    Number Interior Points:   60
    Kx:                          2.000E+0000
    Ky:                          2.000E+0000
    Theta-x:                     0.000E+0000

    Global element list -- Zone   1

    Local      Global
    -----      ------
        1          1
        2          2
        3          3
        4          4
        5          5
        6          6
        7          7
        8          8
        9          9
       10         10
       11         11
       12         12
       13         13
       14         14
       15         15
       16         16
       17         17
       18         18
       19         19
       20         20
       21         21
       22         22
       23         23
       24         24
       25         25
       26         26
       27         27
       28         28

   Well definition -- Zone   1

 Node    X Coor        Y Coor        Radius        Type   Spec. Value
 ----    ------        ------        ------        ----   -----------
    1    1.000E+0001   4.000E+0000   5.000E-0002   Flow   1.000E+0002

   Interior node definition -- Zone   1

 Node    X Coor        Y Coor
 ----    ------        ------
    2    1.500E+0000   1.500E+0000
    3    1.500E+0000   2.500E+0000
    4    1.500E+0000   3.500E+0000
    5    1.500E+0000   4.500E+0000
    6    1.500E+0000   5.500E+0000
    7    1.500E+0000   6.500E+0000
    8    3.500E+0000   1.500E+0000
    9    3.500E+0000   2.500E+0000
   10    3.500E+0000   3.500E+0000
   11    3.500E+0000   4.500E+0000
   12    3.500E+0000   5.500E+0000
   13    3.500E+0000   6.500E+0000
   14    5.500E+0000   1.500E+0000
   15    5.500E+0000   2.500E+0000
   16    5.500E+0000   3.500E+0000
   17    5.500E+0000   4.500E+0000
   18    5.500E+0000   5.500E+0000
   19    5.500E+0000   6.500E+0000
   20    7.500E+0000   1.500E+0000
   21    7.500E+0000   2.500E+0000
   22    7.500E+0000   3.500E+0000
   23    7.500E+0000   4.500E+0000
 24    7.500E+0000    5.500E+0000
 25    7.500E+0000    6.500E+0000
 26    9.500E+0000    1.500E+0000
 27    9.500E+0000    2.500E+0000
 28    9.500E+0000    3.500E+0000
 29    9.500E+0000    4.500E+0000
 30    9.500E+0000    5.500E+0000
 31    9.500E+0000    6.500E+0000
 32    1.150E+0001    1.500E+0000
 33    1.150E+0001    2.500E+0000
 34    1.150E+0001    3.500E+0000
 35    1.150E+0001    4.500E+0000
 36    1.150E+0001    5.500E+0000
 37    1.150E+0001    6.500E+0000
 38    1.350E+0001    1.500E+0000
 39    1.350E+0001    2.500E+0000
 40    1.350E+0001    3.500E+0000
 41    1.350E+0001    4.500E+0000
 42    1.350E+0001    5.500E+0000
 43    1.350E+0001    6.500E+0000
 44    1.550E+0001    1.500E+0000
 45    1.550E+0001    2.500E+0000
 46    1.550E+0001    3.500E+0000
 47    1.550E+0001    4.500E+0000
 48    1.550E+0001    5.500E+0000
 49    1.550E+0001    6.500E+0000
 50    1.750E+0001    1.500E+0000
 51    1.750E+0001    2.500E+0000
 52    1.750E+0001    3.500E+0000
 53    1.750E+0001    4.500E+0000
 54    1.750E+0001    5.500E+0000
 55    1.750E+0001    6.500E+0000
 56    1.950E+0001    1.500E+0000
 57    1.950E+0001    2.500E+0000
 58    1.950E+0001    3.500E+0000
 59    1.950E+0001    4.500E+0000
 60    1.950E+0001    5.500E+0000
 61    1.950E+0001    6.500E+0000

===>   0.8600 seconds to get data from input file<===

===>   0.3300 seconds to prepare system for integration<===

===>   7.5200 seconds to integrate boundary equations<===

===>   7.6900 seconds to decompose system matrix<===

===>   0.8200 seconds to solve system of equations<===

***>>> CONDITION NUMBER :  7.81677284212825E+0001

===> Solution vector dump
   1        6.3032
   2        6.2388
   3        6.2562
   4        6.2388
   5        6.3032
   6       -6.2390
   7      -12.4241
   8      -18.3979
   9      -23.4727
  10      -25.7063
  11      -23.4727
  12      -18.3979
  13      -12.4241
  14       -6.2390
  15        6.3032
  16        6.2388
  17        6.2562
  18        6.2388
  19        6.3032
  20       -6.2390
  21      -12.4241
  22      -18.3979
```

```
   23      -23.4727
   24      -25.7063
   25      -23.4727
   26      -18.3979
   27      -12.4241
   28       -6.2390
   29      -56.8914


>>> Boundary values

Zone :   1

Zone Elem        1                   2                   3
Global El        1                   2                   3
Elem. Node -----A----------B----------A----------B----------A----------B-----
X-coor        0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
Y-coor        0.0000     2.0000     2.0000     4.0000     4.0000     6.0000
N Flux in    -3.1516    -3.1194    -3.1194    -3.1281    -3.1281    -3.1194
N Flow in     6.3032     6.2388     6.2388     6.2562     6.2562     6.2388
Potential     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000

Zone Elem        4                   5                   6
Global El        4                   5                   6
Elem. Node -----A----------B----------A----------B----------A----------B-----
X-coor        0.0000     0.0000     0.0000     2.0000     2.0000     4.0000
Y-coor        6.0000     8.0000     8.0000     8.0000     8.0000     8.0000
N Flux in    -3.1194    -3.1516    -0.0000    -0.0000    -0.0000    -0.0000
N Flow in     6.2388     6.3032     0.0000     0.0000     0.0000     0.0000
Potential     0.0000     0.0000     0.0000    -6.2390    -6.2390   -12.4241

Zone Elem        7                   8                   9
Global El        7                   8                   9
Elem. Node -----A----------B----------A----------B----------A----------B-----
X-coor        4.0000     6.0000     6.0000     8.0000     8.0000    10.0000
Y-coor        8.0000     8.0000     8.0000     8.0000     8.0000     8.0000
N Flux in    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000
N Flow in     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
Potential   -12.4241   -18.3979   -18.3979   -23.4727   -23.4727   -25.7063

Zone Elem       10                  11                  12
Global El       10                  11                  12
Elem. Node -----A----------B----------A----------B----------A----------B-----
X-coor       10.0000    12.0000    12.0000    14.0000    14.0000    16.0000
Y-coor        8.0000     8.0000     8.0000     8.0000     8.0000     8.0000
N Flux in    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000
N Flow in     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
Potential   -25.7063   -23.4727   -23.4727   -18.3979   -18.3979   -12.4241

Zone Elem       13                  14                  15
Global El       13                  14                  15
Elem. Node -----A----------B----------A----------B----------A----------B-----
X-coor       16.0000    18.0000    18.0000    20.0000    20.0000    20.0000
Y-coor        8.0000     8.0000     8.0000     8.0000     8.0000     6.0000
N Flux in    -0.0000    -0.0000    -0.0000    -0.0000    -3.1516    -3.1194
N Flow in     0.0000     0.0000     0.0000     0.0000     6.3032     6.2388
Potential   -12.4241    -6.2390    -6.2390     0.0000     0.0000     0.0000

Zone Elem       16                  17                  18
Global El       16                  17                  18
Elem. Node -----A----------B----------A----------B----------A----------B-----
X-coor       20.0000    20.0000    20.0000    20.0000    20.0000    20.0000
Y-coor        6.0000     4.0000     4.0000     2.0000     2.0000     0.0000
N Flux in    -3.1194    -3.1281    -3.1281    -3.1194    -3.1194    -3.1516
N Flow in     6.2388     6.2562     6.2562     6.2388     6.2388     6.3032
Potential     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000

Zone Elem       19                  20                  21
Global El       19                  20                  21
Elem. Node -----A----------B----------A----------B----------A----------B-----
X-coor       20.0000    18.0000    18.0000    16.0000    16.0000    14.0000
Y-coor        0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
N Flux in    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000
N Flow in     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
Potential     0.0000    -6.2390    -6.2390   -12.4241   -12.4241   -18.3979

Zone Elem       22                  23                  24
Global El       22                  23                  24
```

```
Elem. Node -----A----------B----------A----------B----------A----------B-----
X-coor       14.0000    12.0000    12.0000    10.0000    10.0000     8.0000
Y-coor        0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
N Flux in    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000
N Flow in     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
Potential   -18.3979   -23.4727   -23.4727   -25.7083   -25.7083   -23.4727

Zone Elem         25                    26                    27
Global El         25                    26                    27
Elem. Node -----A----------B----------A----------B----------A----------B-----
X-coor        8.0000     6.0000     6.0000     4.0000     4.0000     2.0000
Y-coor        0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
N Flux in    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000
N Flow in     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
Potential   -23.4727   -18.3979   -18.3979   -12.4241   -12.4241    -6.2390

Zone Elem         28
Global El         28
Elem. Node -----A----------B-----
X-coor        2.0000     0.0000
Y-coor        0.0000     0.0000
N Flux in    -0.0000    -0.0000
N Flow in     0.0000     0.0000
Potential    -6.2390     0.0000

>>> Source/sink values

Zone :    1
Well           1
X-coor         10.0000
Y-coor          4.0000
Potential     -56.8914
Flow (in-)    100.0000

===>  31.4800 seconds to integrate for interior node solutions<===

>>> Interior node values

Zone :    1
Int. Node     1          2          3          4          5          6
X-coor        1.5000     1.5000     1.5000     1.5000     1.5000     1.5000
Y-coor        1.5000     2.5000     3.5000     4.5000     5.5000     6.5000
Potential    -4.6767    -4.6809    -4.6845    -4.6845    -4.6809    -4.6767
Flow X-dir    6.2331     6.2417     6.2491     6.2491     6.2417     6.2331
Flow Y-dir    0.0053     0.0093     0.0042    -0.0042    -0.0093    -0.0053

Int. Node     7          8          9          10         11         12
X-coor        3.5000     3.5000     3.5000     3.5000     3.5000     3.5000
Y-coor        1.5000     2.5000     3.5000     4.5000     5.5000     6.5000
Potential   -10.8954   -10.9265   -10.9501   -10.9501   -10.9265   -10.8954
Flow X-dir    6.1976     6.2543     6.2957     6.2957     6.2543     6.1976
Flow Y-dir    0.0544     0.0620     0.0267    -0.0267    -0.0620    -0.0544

Int. Node     13         14         15         16         17         18
X-coor        5.5000     5.5000     5.5000     5.5000     5.5000     5.5000
Y-coor        1.5000     2.5000     3.5000     4.5000     5.5000     6.5000
Potential   -17.0411   -17.2138   -17.3425   -17.3425   -17.2138   -17.0411
Flow X-dir    6.0618     6.3446     6.5601     6.5601     6.3446     6.0618
Flow Y-dir    0.3146     0.3383     0.1458    -0.1458    -0.3383    -0.3146

Int. Node     19         20         21         22         23         24
X-coor        7.5000     7.5000     7.5000     7.5000     7.5000     7.5000
Y-coor        1.5000     2.5000     3.5000     4.5000     5.5000     6.5000
Potential   -22.8560   -23.7024   -24.4303   -24.4303   -23.7024   -22.8560
Flow X-dir    5.3821     6.6761     8.0249     8.0249     6.6761     5.3821
Flow Y-dir    1.4473     1.7856     0.8862    -0.8862    -1.7856    -1.4473

Int. Node     25         26         27         28         29         30
X-coor        9.5000     9.5000     9.5000     9.5000     9.5000     9.5000
Y-coor        1.5000     2.5000     3.5000     4.5000     5.5000     6.5000
Potential   -26.8359   -29.6085   -35.8098   -35.8098   -29.6085   -26.8359
Flow X-dir    1.7148     3.6102    16.3180    16.3180     3.6102     1.7148
Flow Y-dir    4.0052     8.3302    15.5168   -15.5168    -8.3302    -4.0052

Int. Node     31         32         33         34         35         36
```

```
X-coor        11.5000     11.5000     11.5000     11.5000     11.5000     11.5000
Y-coor         1.5000      2.5000      3.5000      4.5000      5.5000      6.5000
Potential    -25.2934    -27.0461    -29.0074    -29.0074    -27.0461    -25.2934
Flow X-dir    -4.2012     -6.5550    -10.7336    -10.7336     -6.5550     -4.2012
Flow Y-dir     2.7133      4.1584      2.8065     -2.8065     -4.1584     -2.7133

Int. Node    37          38          39          40          41          42
X-coor        13.5000     13.5000     13.5000     13.5000     13.5000     13.5000
Y-coor         1.5000      2.5000      3.5000      4.5000      5.5000      6.5000
Potential    -20.0277    -20.4154    -20.7150    -20.7150    -20.4154    -20.0277
Flow X-dir    -5.8585     -6.4757     -6.9903     -6.9903     -6.4757     -5.8585
Flow Y-dir     0.6966      0.7735      0.3458     -0.3458     -0.7735     -0.6966

Int. Node    43          44          45          46          47          48
X-coor        15.5000     15.5000     15.5000     15.5000     15.5000     15.5000
Y-coor         1.5000      2.5000      3.5000      4.5000      5.5000      6.5000
Potential    -13.9843    -14.0594    -14.1151    -14.1151    -14.0594    -13.9843
Flow X-dir    -6.1532     -6.2813     -6.3752     -6.3752     -6.2813     -6.1532
Flow Y-dir     0.1363      0.1469      0.0627     -0.0627     -0.1469     -0.1363

Int. Node    49          50          51          52          53          54
X-coor        17.5000     17.5000     17.5000     17.5000     17.5000     17.5000
Y-coor         1.5000      2.5000      3.5000      4.5000      5.5000      6.5000
Potential     -7.7904     -7.8023     -7.8119     -7.8119     -7.8023     -7.7904
Flow X-dir    -6.2203     -6.2445     -6.2624     -6.2624     -6.2445     -6.2203
Flow Y-dir     0.0192      0.0246      0.0110     -0.0110     -0.0246     -0.0192

Int. Node    55          56          57          58          59          60
X-coor        19.5000     19.5000     19.5000     19.5000     19.5000     19.5000
Y-coor         1.5000      2.5000      3.5000      4.5000      5.5000      6.5000
Potential     -1.5582     -1.5604     -1.5613     -1.5613     -1.5604     -1.5582
Flow X-dir    -6.2402     -6.2400     -6.2445     -6.2445     -6.2400     -6.2402
Flow Y-dir     0.0042      0.0033      0.0007     -0.0007     -0.0033     -0.0042
```