This is to certify that the

thesis entitled

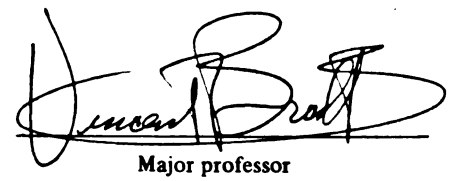
A Condensed Finite Element Analysis
of Microirrigation Hydraulics
Which Incorporates Pipe Components

presented by

Philip Gerrish

has been accepted towards fulfillment
of the requirements for

M.S. degree in Agricultural Engineering



Major professor

Date December 22, 1993

A CONDENSED FINITE ELEMENT ANALYSIS OF MICROIRRIGATION HYDRAULICS WHICH INCORPORATES PIPE COMPONENTS

By

Philip Gerrish

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Agricultural Engineering

1993

ABSTRACT

A CONDENSED FINITE ELEMENT ANALYSIS OF MICROIRRIGATION HYDRAULICS INCLUDING PIPE COMPONENTS

By

Philip Gerrish

The hydraulic design of microirrigation systems is a tedious and time-consuming task. It was the goal of this research to simplify the design of microirrigation systems in order to conserve energy and water. Numerical solutions to microirrigation hydraulics are problematic because of the prohibitive number of network nodes which need to be analyzed. The existing finite element solutions for pipe networks represent pipe components as separate elements, thereby increasing an already formidable number of nodes. In this research, a partial differential equation was developed which incorporates the effect of pipe components at nodes rather than in separate elements. The equation further condenses the network matrices by making use of the virtual node concept in which laterals with evenly spaced emitters are considered single elements with derivative boundary conditions. Results are strongly correlated with existing finite element solutions which show strong correlation with empirical data. Due to the reduced number of nodes, the solution converges rapidly in few iterations. A large network was solved using the condensed finite element analysis developed in this research; where previous methods would require over 12,000 nodes to solve this network, the method developed here required only 80 nodes. Results

correlate strongly to those obtained using the Backstep method.

A development is proposed for a two-dimensional equation describing irrigated sub-plots with evenly spaced laterals as single, two-dimensional elements with derivative boundary conditions. Some preliminary results were obtained and found to be promising.

Approved _____
Major Professor

Approved _____
Department Chairman

ACKNOWLEDGEMENTS

Special thanks to the members of my committee: thanks to Dr. Larry Segerlind and Dr. Raymond Kunze for their comments and involvement with the preparation of this thesis, thanks to Dr. Vincent Bralts and Dr. Walid Shayya who provided the support, encouragement, and intellectual guidance vital to the development of this thesis. And thanks to Dr. Harold W. Belcher for the initial support and guidance in my studies.

Also deserving of thanks and acknowledgement are Jim Schaper, Misael Miranda, Neba Ambe, Alexander White, and Theophilus Okosun for providing an intellectually stimulating environment. My parents deserve special thanks. And off course, I thank my lovely wife, Lucy, for her patience and continual insistance that life should not be too serious.

Table of Contents

List of Tables

List of Figures

Figures in Appendix

I. Introduction

The Preclassic Period, starting around 2500 B.C., marks the beginning of the development of farming communities on this continent. This date corresponds to the rudimentary origins of irrigation practice worldwide. As population densities grew, the development of more sophisticated techniques of water management became necessary. Today, with high-tech irrigation and sophisticated methods of analysis, we struggle more than ever to keep up with the growing number of mouths to feed. While doomists point out the trends in population increase and the seeming impossibility of feeding all these mouths, scientists and engineers collaborate in an attempt to *outwit nature*, pointing out that man's *wit* is in fact ***a part of nature***. Irrigation technology is one of the most dramatic examples of man's intelligence affecting the course of nature.

In affecting nature's *course*, man must be aware of its *limits*. Because the earth's total area of suitable cropland and freshwater resources are limited, there has been a push for more efficient use of existing cropland and water resources--in other words, a preference in the development of *intensive* as opposed to *extensive* agriculture. It has been estimated that by the year 2000, the area of cropland in use will be double the area in use in 1985 (Power, 1986; Holy, 1981), pushing to its limits the world's supply of suitable cropland.

Development of intensive agriculture, however, is not without some costs to the environment. The amount of top-soil decreases more rapidly under intensive

cultivation. Increases in chemical application are often paralleled by an increase in groundwater and runoff contamination. The soil, physically and chemically, is fatigued more rapidly. Any way of reducing these costs to the environment must be investigated thoroughly if we are to look to our future, near and far.

A fine case for the use of environment-friendly intensive agriculture can be made by pointing to the Mayan Civilization--one of the greatest pre-columbian empires of this continent. The case of the Maya is a classic example of the rise and fall of an irrigation society. Their rise is due in great part to the extensive development of sophisticated irrigation systems and practices (Turner and Harrison, 1983). Their fall, although still a great mystery, is thought by many to be related to environmental decay as a result of their highly intensive agriculture (Wiseman, 1989). This history and others like it *must* affect our thinking today, lest we lose "the ability to *understand* recorded historical materials." (Okosun, 1993).

The case for more efficient use of water is made simply by noting that, on a worldwide average, our present efficiency is around 37%. Add to that the fact that 80% of all freshwater resources in use today are being used for irrigation, and the case for water efficiency becomes urgent (Power, 1986).

One requirement for the improvement of water efficiency is the improved control of water application. This is one of the many benefits of microirrigation. Other benefits include zero runoff, reduced labor costs, ease of chemical and fertilizer injection, and higher salinity tolerance due to stable moisture conditions. Microirrigation is highly efficient and environment-friendly; it is therefore a good candidate for future-oriented agriculture.

The design of microirrigation systems is tedious and capital costs are high;

these are the two most prohibiting factors in most cases. While capital costs cannot be changed, it is the aim of this research to facilitate design by improving computer models of the hydraulics involved.

A. Scope and Objectives

Hydraulic networks have traditionally been solved using the backstep, Hardy-Cross, Newton-Raphson, and Linear Theory methods. Bralts and Segerlind (1985) first put linearized flow equations into Finite Element formulation. Wood (1981) and Finkel (1982) point out that pressure losses across pipe components such as elbows, tees and valves may significantly affect pressure heads in a network. These *minor losses* were put into Finite Element formulation by Haghighi et al. (1988).

A drip irrigation system is well designed if the water is applied uniformly throughout. As a measure of uniformity, the *Statistical Uniformity Coefficient* was introduced for microirrigation by Bralts, et al.(1987); it is defined as one minus the coefficient of variation of emitter outputs. The objective of computer models developed for micro irrigation design, therefore, is to accurately predict the output of each emitter and give the uniformity coefficient as an indication of the design's quality. A shortcoming of existing models is the awkwardness with which pipe components are handled; inclusion of pipe components in the network analysis makes solution cumbersome and unstable because new nodes are added to the system.

The overall goal of this research is to conserve water, chemicals and energy used for plant growth through improved hydraulic design of micro irrigation systems. More specifically, the focus of this research will be to develop an improved finite

element formulation of pipe network systems in order to facilitate design. The specific objectives are as follows:

1. Assess the effect of pipe components such as tees, elbows, crosses, expansions, and contractions on the solution of a hydraulic network.

2. Develop a condensed finite element formulation for the incorporation of pipe components which would not increase the number of nodes.

3. Include the virtual node concept in the finite element analysis, thereby further condensing the network to be analyzed.

4. Apply the condensed finite element formulation to the design of microirrigation systems, and compare the results with those of other methods.

II. Review of Theory and Literature

An irrigation system is characteristically a "tree" hydraulic network system (no closed loops) with a *main* as the "trunk", *submains* as primary "branches" and *laterals* as secondary "branches" (see Figure 1). Along the length of each lateral are the *emitters* which are water outlets. The *emitters* are where the water is applied directly to the plant in a drip irrigation system. In a sprinkle irrigation system, the water outlets are *sprinklers*.

A. Hydraulics of Irrigation

1. Equation Governing Emitter Flow

For the purposes of this study, all water outlets will be called *emitters*. The equation describing flow in an emitter is:

$$q_e = kh^x \qquad (1)$$

where q_e = emitter discharge

k = emitter discharge coefficient

h = pressure head

x = emitter discharge exponent

Equation (1), in general, describes orifice flow to the atmosphere (Wu, et al., 1979).

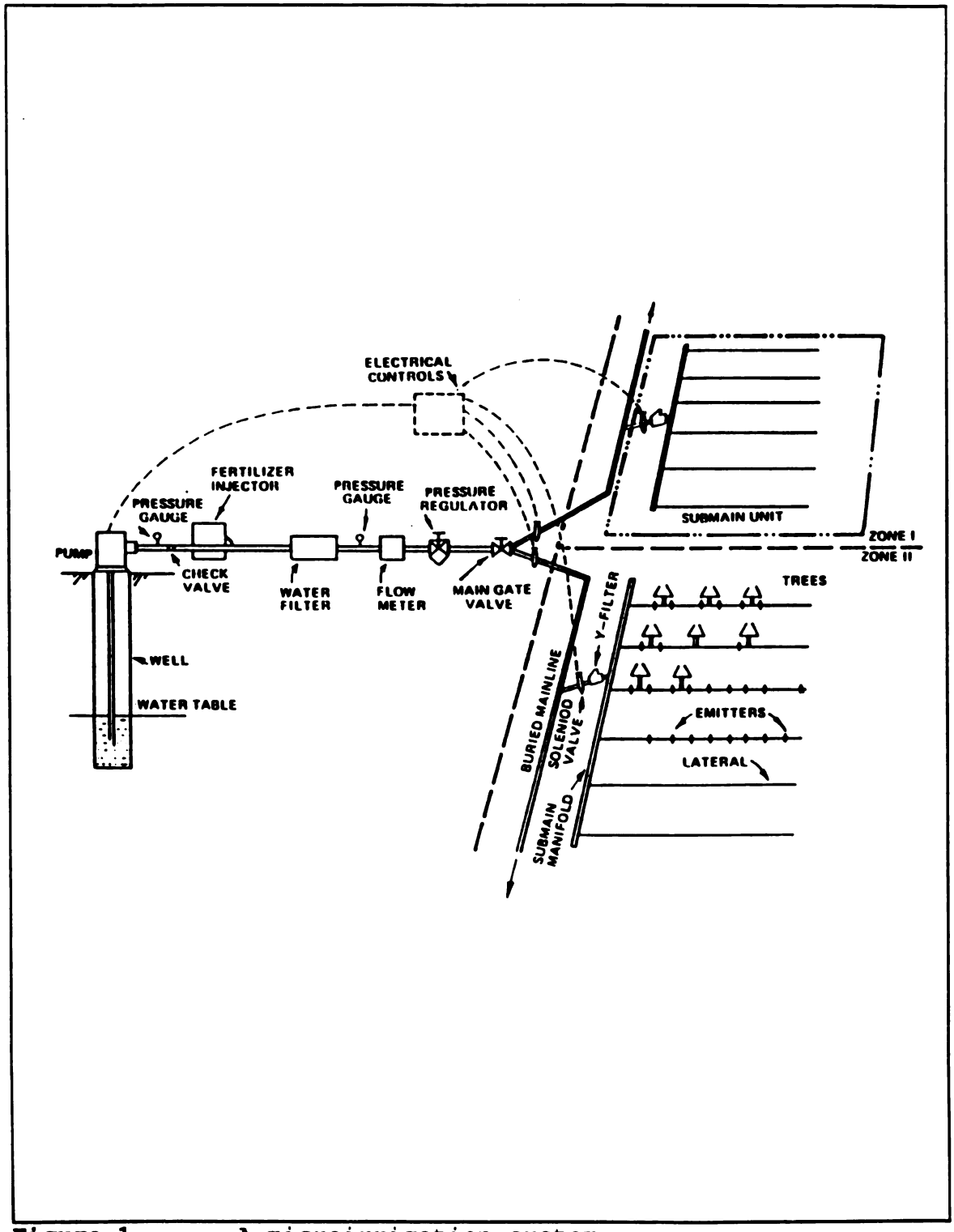


Figure 1 A microirrigation system.

The coefficient and exponent characterize the emitter and are different for different emitters. For a pressure-compensating emitter, for example, the value of x is close to or equal to zero, as flow does not depend on pressure head. The value of x, in other cases, is indicative of the flow regime in the emitter. A value of 0.5 indicates fully turbulent flow, whereas a value of 1 indicates laminar flow.

2. Equations Governing Pipe Flow

A generally accepted form of the differential equations governing fluid flow is the Navier-Stokes equation:

$$\rho \frac{D\boldsymbol{V}}{Dt} = -\nabla p + \rho \boldsymbol{g} + \mu \nabla^2 \boldsymbol{V} \qquad (2)$$

where; ∇ = Laplacian operator,

p = pressure,

ρ = density,

g = acceleration due to gravity,

μ = viscosity,

V = velocity,

The particular solution of the Navier-Stokes equation for pipe flow is the Darcy-Weisbach equation:

$$\Delta H_p = f\frac{L}{D}\frac{V^2}{2g} \tag{3}$$

where ΔH_p = pressure head loss due to pipe friction

f = friction factor

L = length of pipe

D = diameter of pipe

V = velocity of fluid

g = acceleration due to gravity

The friction factor, f, in the above equation is a dimensionless wall shear, $f = \frac{\tau_0}{\frac{1}{8}\rho V^2}$.

It is found to be $f = \frac{64}{Re}$ (where Re = Reynold's number) for laminar flow. This value, however, is not so nicely defined in the transition and turbulent flow regimes. To determine f in these flow regimes, it is common practice to refer to a chart called the Moody diagram. Some empirical equations have been derived and are successful in approximating the friction factor, f.

Another equation which may be used to calculate head loss due to pipe friction is the Blasius equation:

$$h_L = \frac{8(4)^b}{\pi^{(2+b)}}\frac{a}{g}V^{-b}\frac{Q^{2+b}}{D^{5+b}}L \tag{4}$$

where; v = kinematic viscosity

q = flow rate

L = length of pipe

D = pipe diameter

a = constant

b = constant

g = acceleration due to gravity

For PVC pipe, the values of a and b were determined by von Bernuth and Wilson (1989) to be: a = 0.316 and b = -0.25. Hence the equation,

$$h_L = KLv^{0.25}q^{1.75}D^{-4.75} \quad (5)$$

An empirical equation often prefered for its simplicity is the Hazen-Williams equation:

$$\Delta H_p = \frac{k_{sys}L}{C_{HW}^{1.852}D^{4.87}}q^{1.852} \quad (6)$$

where k_{sys} = 4.73 for English units (D and L in feet)

k_{sys} = 10.7 for International System of units

q = flow

C_{HW} = Hazen-Williams roughness coefficient

This equation approximates head-loss over a limited range of Reynolds numbers, a drawback which should be considered especially when working in laminar or

transition flow regimes.

While any of the above pipe flow equations may be used, the algebraic development which follows uses the Hazen-Williams equation. This equation will be used for the sake of comparison with other methods which use the same equation.

3. Equations for Approximating Lateral Flow

Flow in a microirrigation lateral line (a pipe with emitters) can be approximated by constructing a dimensionless energy gradient curve (Wu and Gitlin, 1975). Basic assumptions are (a) flow from all emitters along the lateral is the same, and (b) emitters along the lateral are evenly spaced. Dimensionless head drop ($\Delta H_i/\Delta H$) is plotted against dimensionless length (x/L) and the curve derived is an exponential decay function:

$$R_i = \frac{\Delta H_i}{\Delta H} = 1 - (1 - i)^{m+1} \qquad (7)$$

where, ΔH_i = head drop at point i

ΔH = total head drop in lateral

i = x/L, where x = distance from origin, and L = total length of lateral.

m = 1.852 from Hazen-Williams equation

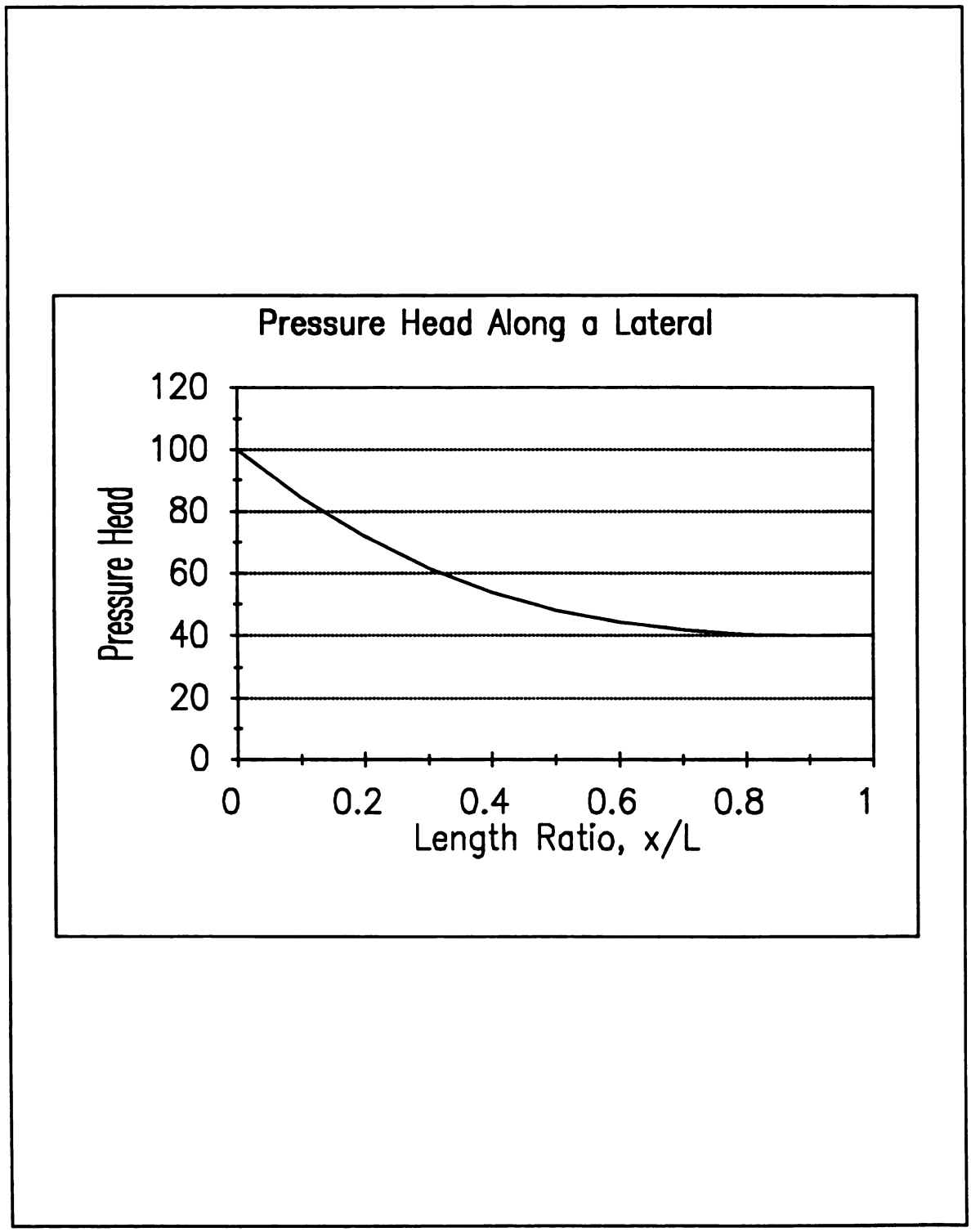


Figure 2. Approximation of pressure head along a lateral.

Head at any point i along the lateral can then be described as:

$$h_i = H_0 - R_i\Delta H \pm R_i'\Delta H' \qquad (8)$$

where, H_0 = head at i = 0

$R_i\Delta H$ = head loss due to pipe friction at point i

$R_i'\Delta H'$ = head loss or gain due to elevation at point i

The curve described by equation (8) is shown graphically in Figure 2. With emitter flow described by $q = kh^x$, and constant emitter flow along the lateral, flow at point i along the lateral is:

$$q_i = k(h_i)^x = k(H_0 - R_i\Delta H \pm R_i'\Delta H')^x \qquad (9)$$

Or, substituting the equality, $q_0 = kH_0^x$ for flow from the first emitter on the lateral gives an equation for flow from the lateral at any point i along the lateral:

$$q_i = q_0[1 - R_i(\Delta H/H_0) \pm R_i'(\Delta H'/H_0)]^x \qquad (10)$$

4. Equation Governing Component Head Loss

An abrupt change in pipe geometry causes turbulence. Where turbulence occurs, energy is lost. Pipe components (tee's, elbows, valves, etc.) present abrupt changes in pipe geometry. Their presence therefore implies loss of energy. A pump, the exception of course, would increase energy. The energy lost[1] at a pipe component

[1]Note here that the terms **energy** and **head** are used interchangeably. This is because in hydraulics, total energy is commonly expressed in potential form. Refer to next section under the heading, *Conservation of Energy*.

is equal to some fraction of the fluid's kinetic energy at that point:

$$\Delta H_c = k_c \frac{V^2}{2g} \tag{11}$$

where ΔH_c = pressure head loss due to pipe component

V = the velocity of the fluid

g = acceleration due to gravity

k_c = a constant peculiar to the component used

The velocity head of water, $\frac{V^2}{2g}$, is the water's kinetic energy expressed as potential energy (head).

B. Analysis of Hydraulic Networks

As with any physical system, both conservation of mass and conservation of energy must be satisfied. These dictate the continuity equations throughout the system.

1. Conservation of Mass

The conservation of mass of a fluid implies that flow in equals flow out:

$$\sum q_{in} = \sum q_{out} \tag{12}$$

When this continuity is met at several points in a system, the result is a system of

simultaneous equations.

2. Conservation of Energy

The conservation of energy of a fluid is expressed by Bernoulli's equation:

$$\frac{V^2}{2g} + h + z + h_f = constant \qquad (13)$$

where V = velocity

g = acceleration due to gravity

h = pressure head

z = elevation

h_f = head loss due to friction

Simply stated, this equation says that the sum of the kinetic energy (in potential form), $\frac{V^2}{2g}$, plus the potential energy, $h+z$, plus the heat energy lost due to friction (in potential form), h_f, is equal to the total energy and must therefore be constant throughout the system.

3. Choice of Unkown

The set of simultaneous equations describing a hydraulic network can be expressed in terms of either hydraulic head (potential energy) or flow as the unknown.

Choosing flow as the unkown has the advantage that many of the equations in the set of simultaneous equations will be linear (Jeppson, 1976). In fact, if no closed

loops exist (i.e. a tree network, as commonly encountered in irrigation systems), all of the equations will be linear. While this is clearly a great advantage, it is countered by the disadvantage that the boundary conditions are expressed in terms of potential (head). A pump, for example, will deliver a certain hydraulic head, the outflow of which depends on the solution of the entire network. Also, the derivative boundary condition described later as the Virtual Node concept is a potential (head) gradient.

The great disadvantage of choosing potential as the unkown is that the resulting equations are non-linear. The attractiveness of this choice, however, is due to (a) the systemmatic facility with which the set of equations is formed, and (b) the ability to apply boundary conditions essential for solution.

Note that the unkown chosen in this research is *potential energy* which, as stated earlier, is the *sum of elevation and pressure head*, **z + h**. This sum is also refered to in this thesis as *hydraulic head*, and is denoted simply by **h** for hydraulic head within an element and **H** for hydraulic head at a node.

4. Notation

The network analysis technique used in this thesis is called the Finite Element Method. The use of Finite Element notation throughout this section will facilitate presentation of the different network analysis techniques and will assure consistency of notation throughout this thesis. The reader, therefore, will be acquainted with Finite Element notation at this point.

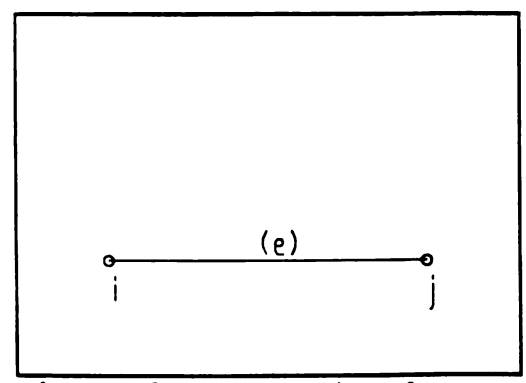


Figure 3. A generic element

Figure 3 shows a generic element--a one-dimensional element as defined by Segerlind (1984). The *element*, (e), lies between two *nodes*, i and j. In the case of hydraulic network systems, an *element* refers to a length of pipe. A *node* must be assigned to every point at which (a) head is known, (b) head is to be calculated, or (c) there is an abrupt change ("discontinuity") in head. For example, to calculate emitter flow, it is necessary to know hydraulic head at that point; so a *node* is assigned to that point.

A variable with a superscript in parenthesis pertains to an *element*, whereas a variable with a subscript pertains to a *node*. The variable, $q^{(7)}$, for example refers to flow through pipe *element* 7, while q_5 refers to flow through emitter *node* 5.

Equations developed here are non-linear and their solution is numerical. Throughout this thesis, the subscript, **n**, denotes the number of the current iteration. Likewise, **n-1** refers to the previous iteration.

C. Some Methods of Network Analysis

1. The Backstep Method

For a string of emitters (a lateral), where one end represents a *source* with known pressure, the Backstep Method can be used to calculate the pressure heads at each of the emitter nodes along the lateral. In Figure 4, for example, node 1 represents a known head while nodes 2 through 8 represent emitters at which head is to be calculated.

To begin the solution, an initial guess is made for head at node 8. Next, the

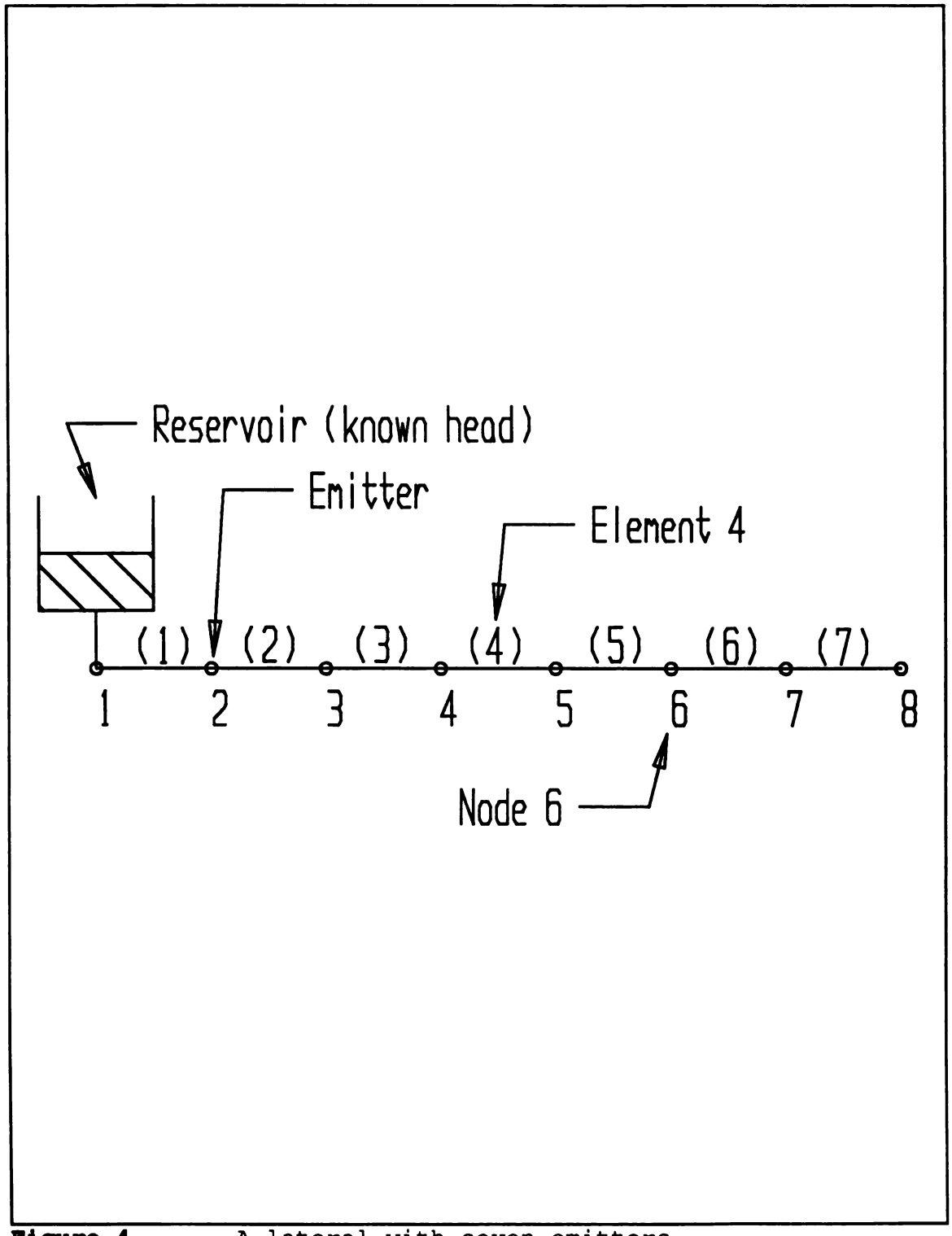


Figure 4. A lateral with seven emitters.

head at node 7 is calculated by satisfying the conservation of mass at node 8. That is, flow *out* of node 8 (emitter flow) is set equal to flow *in* (pipe flow). Pipe flow is calculated by rearranging the Hazen-Williams equation,

$$q^{(7)} = \left(\frac{H_7 - H_8}{k^{(7)}} \right)^{\frac{1}{1.852}} \tag{14}$$

where,

$$k^{(7)} = \frac{k_{sys} L}{C_{HW}^{1.852} D^{4.87}} \tag{15}$$

L = length of pipe *element* 7

C_{HW} = Hazen-Williams roughness coefficient for *element* 7

D = diameter of pipe *element* 7

For example, the equation expressing conservation of mass at node 8 is:

$$q^{(7)} - q_8 = 0 \tag{16}$$

Substituting equations ? and (14) into equation (16) gives:

$$\left(\frac{H_7 - H_8}{k^{(7)}} \right)^{\frac{1}{1.852}} - k(H_8 - z_8)^x = 0 \tag{17}$$

where, H - z = hydraulic head minus elevation = pressure head.

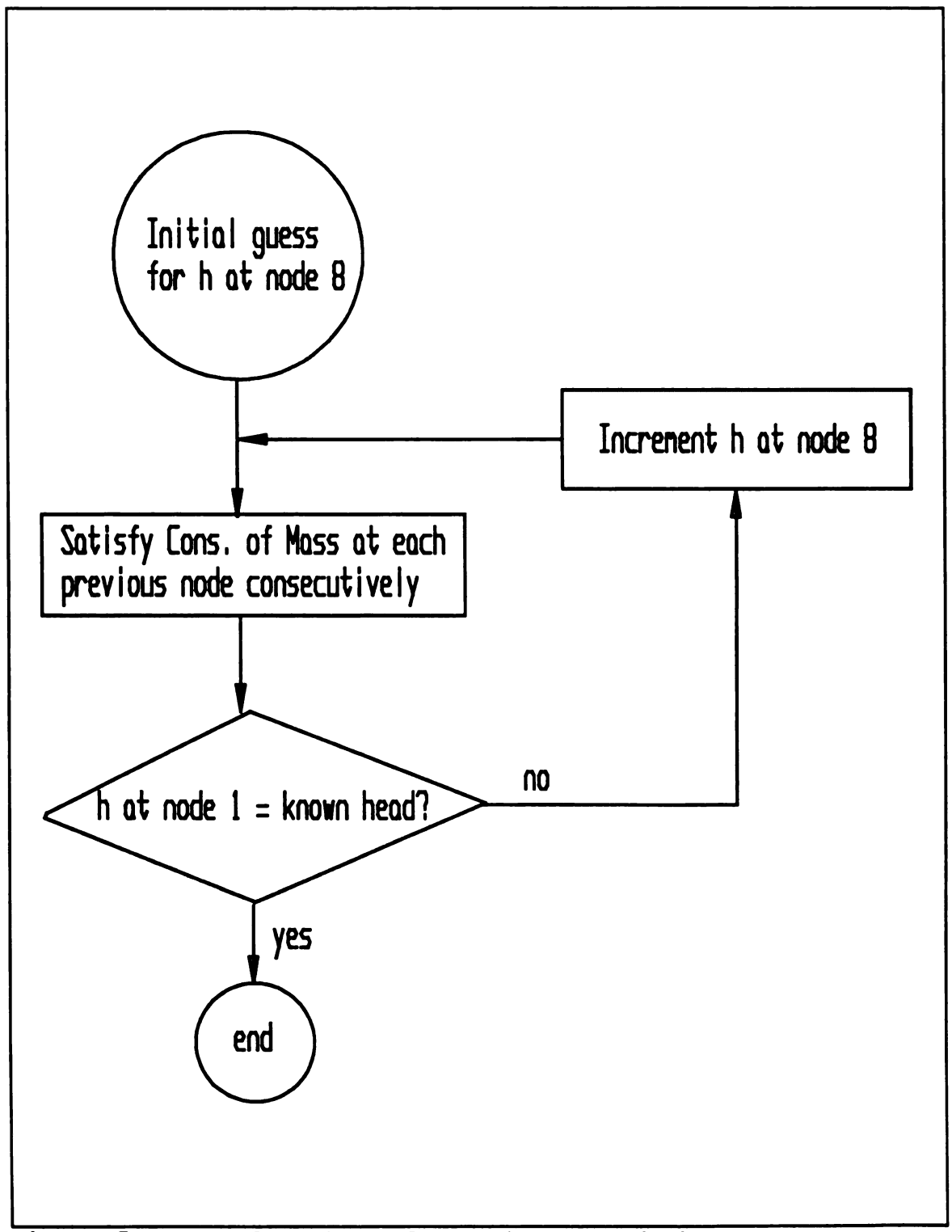


Figure 5. Flow chart for Backstep Method.

With the initial guess for H_8, H_7 is calculated. From the expression for conservation of mass at node 7, H_6 is calculated, and so on until H_1 is calculated. This value for H_1 is then compared with the known value for head at that point. If these values are within a specified error interval, then the procedure is finished; otherwise, the initial guess for h_8 is incremented, and calculations are repeated. The solution is thereby arrived at iteratively as shown in Figure 5.

The advantage of the backstep method is its straight-forward nature and hence its facility of formulation. The great disadvantage it has is the long convergence time required due to the large number of iterations required. Of the three methods presented in this section, this is by far the slowest.

2. The Newton-Raphson Method

This method is based on a truncation of the Taylor series which takes the form,

$$x_n = x_{n-1} - \frac{R(x_{n-1})}{R'(x_{n-1})} \qquad (18)$$

where x = the unknown

R(x) = the residual equation

n = the number of current iteration

This is an iterative solution to the equation R(x) = 0. An equation of this form is known as a *residual* equation. In the example of a string of emitters (Figure 4), a residual equation and its first derivative must be written for each node. The residual equation for any node in a pipe network system is simply, $R(h) = q_{in} - q_{out} = 0$, where h

(head) is now the unknown. Written for all nodes as a system of equations, the matrix formulation takes the following form:

$$\{h_n\} = \{h_{n-1}\} - [D]^{-1}\{R(h_{n-1})\} \tag{19}$$

where [D] = the Jacobian matrix of derivative elements

{R(h)} = the residual vector

{h} = the vector of unknowns (head)

$$[D] = \begin{bmatrix} \frac{\partial R_1}{\partial h_1} & \frac{\partial R_1}{\partial h_2} & \cdots & \frac{\partial R_1}{\partial h_n} \\ \frac{\partial R_2}{\partial h_1} & \frac{\partial R_2}{\partial h_2} & \cdots & \frac{\partial R_2}{\partial h_n} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \frac{\partial R_n}{\partial h_1} & \frac{\partial R_n}{\partial h_2} & \cdots & \frac{\partial R_n}{\partial h_n} \end{bmatrix} \tag{20}$$

Determining the Jacobian matrix makes this method rather cumbersome when seeking a generic formulation for networks. Also, a disadvantage of this method is its instability, especially when the unknown is head. (Greater stability is achieved when flow is the unknown.) Because of its instability, this method depends greatly on the accuracy of the initial guess. Otherwise, this method has the great advantage of quadratic convergence, which means rapid solution.

3. The Linear Theory Method

This method sets up a system of linear equations by "linearizing" the flow equations (Jeppson, 1976). This is achieved by observing that,

$$q^{(e)} = C^{(e)} (H_i - H_j)_n, \text{ where } C^{(e)} = \frac{(H_i - H_j)_{n-1}^{-0.46}}{(k^{(e)})^{0.54}}, \text{ and } k^{(e)} = \frac{k_{sys} L}{C_{HW}^{1.852} D^{4.87}}.$$

The term, $C^{(e)}$, is treated as a constant and its value can be determined because $(H_i - H_j)_{n-1}$ is known. (Note that **n-1** refers to the previous iteration.)

Linearization of the emitter flow equation in terms of hydraulic head is achieved by noting that,

$$q_e = k_e (H - z)^x = \left(k_e \frac{(H-z)^{x-1}}{H}\right) H \qquad (21)$$

so that $q_e = C_e H_n$, where $C_e = k_e \frac{(H-z)^x}{H}\Big|_{n-1}$.

The conservation of mass at node 6, for example, is now expressed in the following form:

$$C^{(5)} (H_5 - H_6) - C^{(6)} (H_6 - H_7) - C_6 H_6 = 0 \qquad (22)$$

Again, notation is important. $C^{(e)}$ refers to the *linearizing constant* for pipe *element* flow. C_e refers to the *linearizing constant* for emitter *node* flow.

This method in general is quite stable and is not highly dependent on the accuracy of the initial guess. The solution converges in relatively few iterations.

4. Finite Element Formulation of Linear Theory Method

One-dimensional Finite Element notation (Segerlind, 1984) is used to number nodes and elements of a hydraulic network system (Bralts et al., 1987). Figure 4 and Figure 6 show examples of this numbering system. A network numbered in this fashion is readily put into matrix form by adding the contribution of each element to the global system of equations.

The global system of equations takes the form,

$$[K]\ \{H\} = \{F\} \tag{23}$$

which can be written in *residual* form as,

$$\{R\} = [K]\ \{H\} - \{F\} = \{0\} \tag{24}$$

where {R} = *global residual vector.*

[K] = *global stiffness matrix.*

{H} = global vector of unknown heads.

{F} = *global force vector.*

The *element stiffness matrix* represents an element's contribution to the global stiffness matrix. The element stiffness matrix, in this case, consists of the *linearizing constant* pertaining to an element:

$$[k^{(e)}] = \begin{bmatrix} C^{(e)} & -C^{(e)} \\ -C^{(e)} & C^{(e)} \end{bmatrix} \tag{25}$$

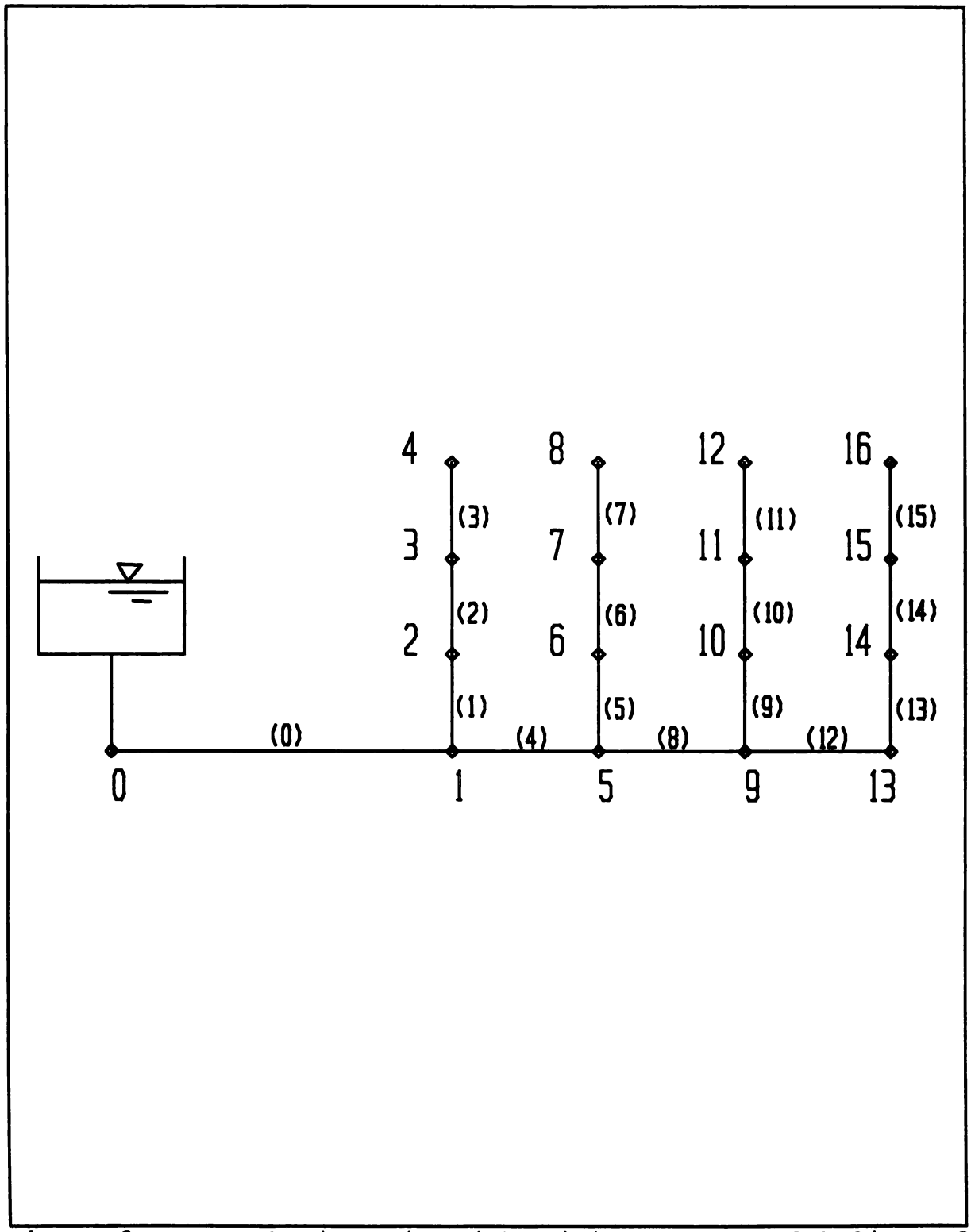


Figure 6 Submain unit with Finite Element labeling of nodes and elements

where; $C^{(e)}$ = linearizing constant,

$k^{(e)}$ = element stiffness matrix.

The *element force vector* is an element's contribution to the global force vector. When minor losses are not accounted for, it equals zero.

The procedure by which element vectors and matrices are added to the global matrices is called the *Direct Stiffness Method* (Segerlind, 1984). The generic form of an element stiffness matrix is,

$$[k^{(e)}] = \begin{bmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2} \end{bmatrix} \tag{26}$$

This matrix is then added to the global stiffness matrix as follows:

$k_{1,1}$ is added to $K_{i,i}$

$k_{1,2}$ is added to $K_{i,j}$

$k_{2,1}$ is added to $K_{j,i}$

$k_{2,2}$ is added to $K_{j,j}$

where, k = value in element matrix

K = value in global matrix

Emitters may be incorporated into the global stiffness matrix by considering them to be separate elements (Bralts et al., 1987).

$$q = C_e (H_1 - H_{atm}) \qquad (27)$$

where H_{atm} is equal to zero (atmosphere) and C_e is the linearizing constant for the emitter flow equation. The result is that, C_e is added to $K_{i,i}$; emitter contributions end up on the diagonal of the global stiffness matrix.

The global stiffness matrix which results is a *banded symmetric matrix.* The iterative solution of the resulting system of equations was written in computer code by Bralts and Segerlind (1985).

An advantage of the Finite Element formulation for the solution of hydraulic network systems is its systematic simplicity. Because of this, it is well suited to the development of a universal network solver. Bralts and Segerlind (1985) list several other advantages.

Accomodating Pipe Components.

The cummulative effect of several pipe components in a hydraulic network is often substantial. While losses due to pipe wall friction often dominate, the "minor losses" due to turbulence in pipe components is seldom negligible. As pointed out by Villemonte (1977), the term "minor losses" is often a misnomer as their effect is frequently "major".

Haghighi et al. (1988 and 1989) proposed that pipe components may be added to the system of equations as separate elements. This is done by adding an element matrix of linearized component coefficients to the global stiffness matrix for each pipe component. The component head loss in a component element is calculated by:

$$\Delta H_c = H_i - H_j = k_c \frac{V^2}{2g} = k_c \frac{8q^2}{g\pi^2 d^4}$$

This can be linearized and rearranged to form,

$$q = C_c(H_i - H_j)$$

where $C_c = \frac{g\pi^2 d^4}{8 k_c q_{n-1}}$

The element matrix for a pipe component is then added to the global stiffness matrix by the Direct Stiffness Method. The component element matrix is similar in form to those for pipe elements:

$$[k_D^{(e)}] = \begin{bmatrix} C_c & -C_c \\ -C_c & C_c \end{bmatrix}$$

The above development works only for certain two-node components such as elbows and valves. A tee, however, is a component with three "fittings"; it is represented by an element with three nodes (Haghighi et al., 1989). This approach has the advantage that the equations are assured global continuity of zeroth order, C^0, meaning that discontinuities do not exist at nodes (see discussion of continuity in Results and Discussion section). The additional nodes added by pipe components, however, mean additional equations, which means increased computer time and memory requirements.

D. The Finite Element Method

One way of approximating the solution to an equation is to multiply its residual form by a weighting function and set the integral of the product equal to zero. This procedure is especially useful in the solution of differential equations. While there are several weighting functions to choose from, the weighting function employed in Galerkin's method is perhaps the most widely used.

Consider the equation,

$$D\frac{\partial^2\phi}{\partial x^2} - G\phi + Q = 0 \qquad (28)$$

Employing the product rule for derivatives together with Greene's theorem, the second-derivative term can be broken down into two first-derivative terms, one of which represents the intra-element *residue* which should go to zero (refer to Segerlind, 1984 or Dhatt and Touzot, 1984). The function is multiplied by its weighting function (or *shape function*) and integrated over an element at each node of the element. A linear element has two nodes, i and j; the integration at node i yields:

$$\int_{x_1}^{x_j} D\frac{\partial N_i}{\partial x}\frac{\partial \phi}{\partial x}dx - \int_{x_1}^{x_j} GN_i\phi dx + \int_{x_1}^{x_j} QN_i dx = 0$$

where N_i = shape function at node i.

Integration at node **j** yields:

$$\int_{x_i}^{x_j} D\frac{\partial N_j}{\partial x}\frac{\partial \phi}{\partial x}dx - \int_{x_i}^{x_j} GN_j\phi dx + \int_{x_i}^{x_j} QN_j dx = 0$$

where N_j = shape function at node **j**

An element's contribution, therefore, to the global system of equations is written in matrix form:

$$\int_{x_i}^{x_j} D\frac{\partial [N]^T}{\partial x}\frac{\partial \phi}{\partial x}dx - \int_{x_i}^{x_j} G[N]^T\phi dx + \int_{x_i}^{x_j} Q[N]^T dx = 0 \qquad (29)$$

where $[N] = [N_i\ N_j]$ = shape function matrix for element (e)

Shape functions are derived so that,

$$\phi^{(e)} = [N]\{\Phi^{(e)}\}$$

Linear shape functions, therefore, satisfy the following:

$$\phi^{(e)} = N_i\Phi_i + N_j\Phi_j$$

where potential, $\phi^{(e)}$, is a straight line connecting nodes **i** and **j**. Hence they take the form:

$$N_i = \frac{X_j - x}{L}$$

$$N_j = \frac{x - X_i}{L}$$

where X_i = x at node i

X_j = x at node j

L = length of element

While this research employs linear shape functions only, the same concepts developed here can be used in conjunction with shape functions of higher-degree polynomials to achieve more accurate results.

Continuity Requirements.

From equation (29), note that a first derivative term is integrated. The first derivative of the function must therefore be defined, requiring continuity of first order, C^1, throughout an element. Also, interelement continuity of zeroth order, C^0, is required, meaning that the value for node **j** of an element must equal the value for node **i** of the next element. In the case of linear elements, the first derivative is undefined at the nodes.

The Virtual Node Technique.

A way to reduce the size of the global stiffness matrix is by merging the effect of a string of several emitters into a single element. This is achieved by considering the effect of these emitters to be *continuous* throughout a single element, thus having the effect of a derivative boundary condition *along* the element. The nodes of such an element are called *virtual nodes* or *virtual emitters* (Kelly, 1989; Bralts et al., 1993). This technique considerably reduces the number of nodes in a system and hence the size of the global stiffness matrix.

The implementation of this technique requires that the flow equations be rearranged to describe head as a residual equation in differential form. This equation is then solved simultaneously for all nodes in the system using the Finite Element Method.

III. Methodology

When applying the Finite Element formulation of the Linear Theory Method to the analysis of a medium-size microirrigation system, a problem which arises is the prohibiting size of the global stiffness matrix. A drip-irrigated plot of one hectare, for example, with emitters spaced one meter apart and laterals spaced 1.5 meters apart would contain 6,600 emitters and would require 6,667 nodes for head calculations. The size of the global stiffness matrix would then be 6,667 x 6,667. Depending on the arrangement, this number could be increased up to two fold due to the presence of pipe components. Such a matrix would prohibit the use of conventional personal computers. Also, when pipe components are taken into account, an elaborate "first guess" procedure is necessary to initialize iterations. In this research, a formulation will be developed to eliminate the extra nodes added by pipe components and thereby eliminate the need to closely approximate nodal values for the solution to converge.

A. Research Approach

The objectives of this research are restated below, each followed by the research approach employed in its development.

1. Assess the cumulative effect of pipe components such as tees, elbows, crosses, expansions and contractions on the solution of a hydraulic

network.

This objective will be addressed by using an existing pipe-network program called ANALYZER (Haghighi et al., 1988, 1992; Mohtar et al., 1991; Shayya et al., 1988) to compare the solution of a hydraulic network not including pipe components to the solution which includes the effect of pipe components.

2. Develop a condensed finite element formulation for the incorporation of pipe components without increasing the number of elements.

This will be accomplished by developing a formulation in which a pipe component is represented at a node rather than as a separate element. Because a node at a junction is needed anyway, no *new* node is added as a result of a component at that junction. The effect of a pipe component will be accounted for in pipe elements *downstream* of that component.

3. Include the virtual node concept in the finite element analysis, thereby further condensing the network to be analyzed.

A string of evenly spaced emitters (a lateral) will be considered a single element with a derivative boundary condition describing out-flow as a continuous function along the length of the element. This way, where previous methods required a node at each emitter along a lateral, this method requires only two nodes (one element) to represent the entire lateral or lateral segment.

4. Apply the condensed Finite Element formulation to the design of microirrigation systems and compare the results with those of other methods.

A pipe network will be analyzed and the results will be compared to those of ANALYZER and KYPIPE (Wood, 1980; Wood and Charles, 1972, 1973; Wood and Rayes, 1981).

B. Effect of Components on Network Solution

The effect of pipe components on the solution of a hydraulic network is often substantial. The following scenario emphasizes the need to include the effect of pipe components if the solution is to be meaningful. Figure 7 shows a hydraulic network; for now, consider only the first diagram in the figure. The solution to this network is achieved using the ANALYZER program. In Figure 8 and Figure 9, two solutions are compared: the solution not including pipe components is compared with the solution which includes the effect of pipe components.

Note that if the effect of pipe components is not included, the head calculated at node 1 is not significantly different than zero, because the error produced by not including pipe components is greater that the head at node 1. It is apparent here that the solution which does not include the effect of pipe components has limited meaning.

Now consider the second diagram in Figure 7. This shows the same network but with pipe components represented by nodes instead of elements. The total number of nodes is reduced from 40 to 27--about two thirds!

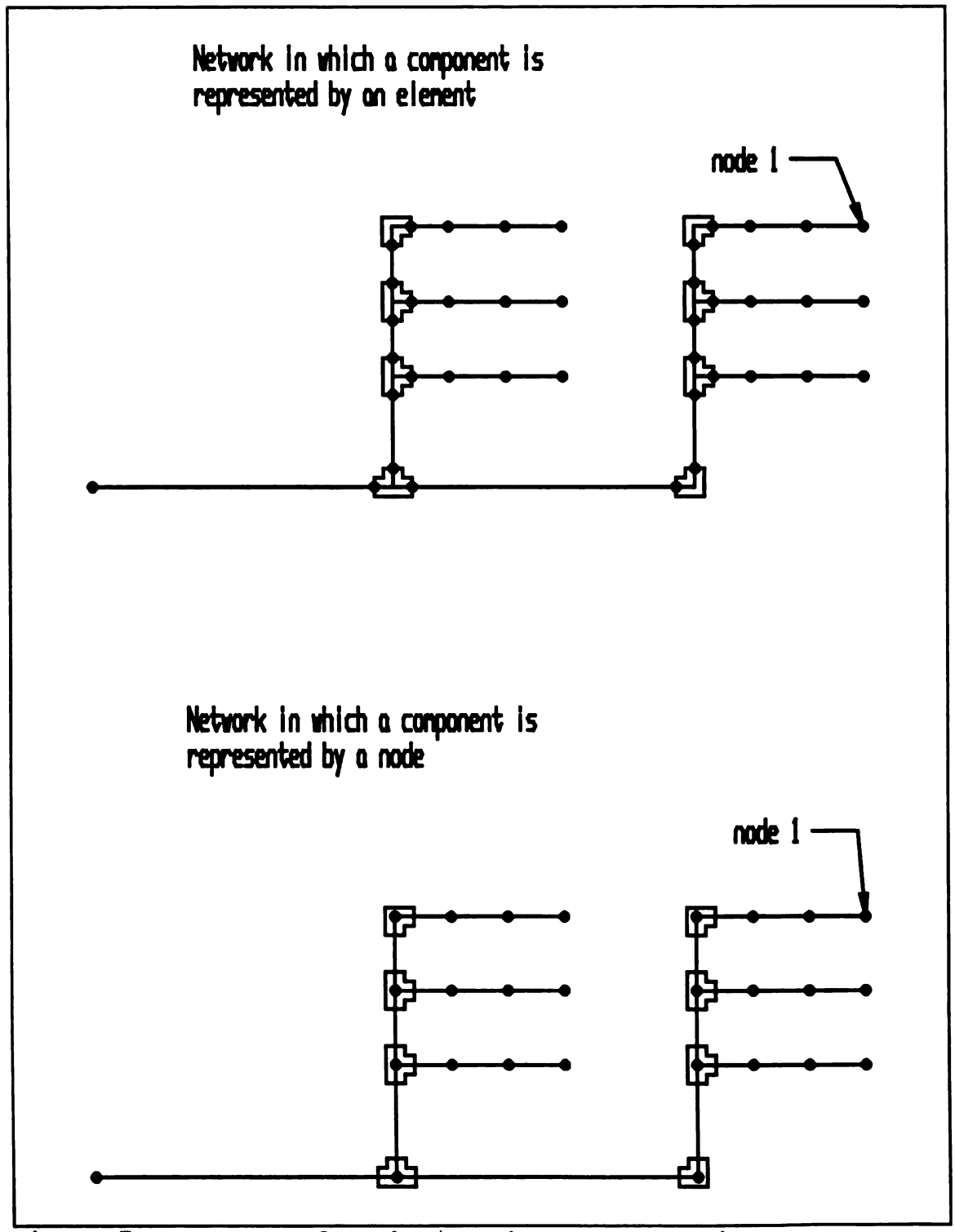


Figure 7 Two ways of analyzing the same network.

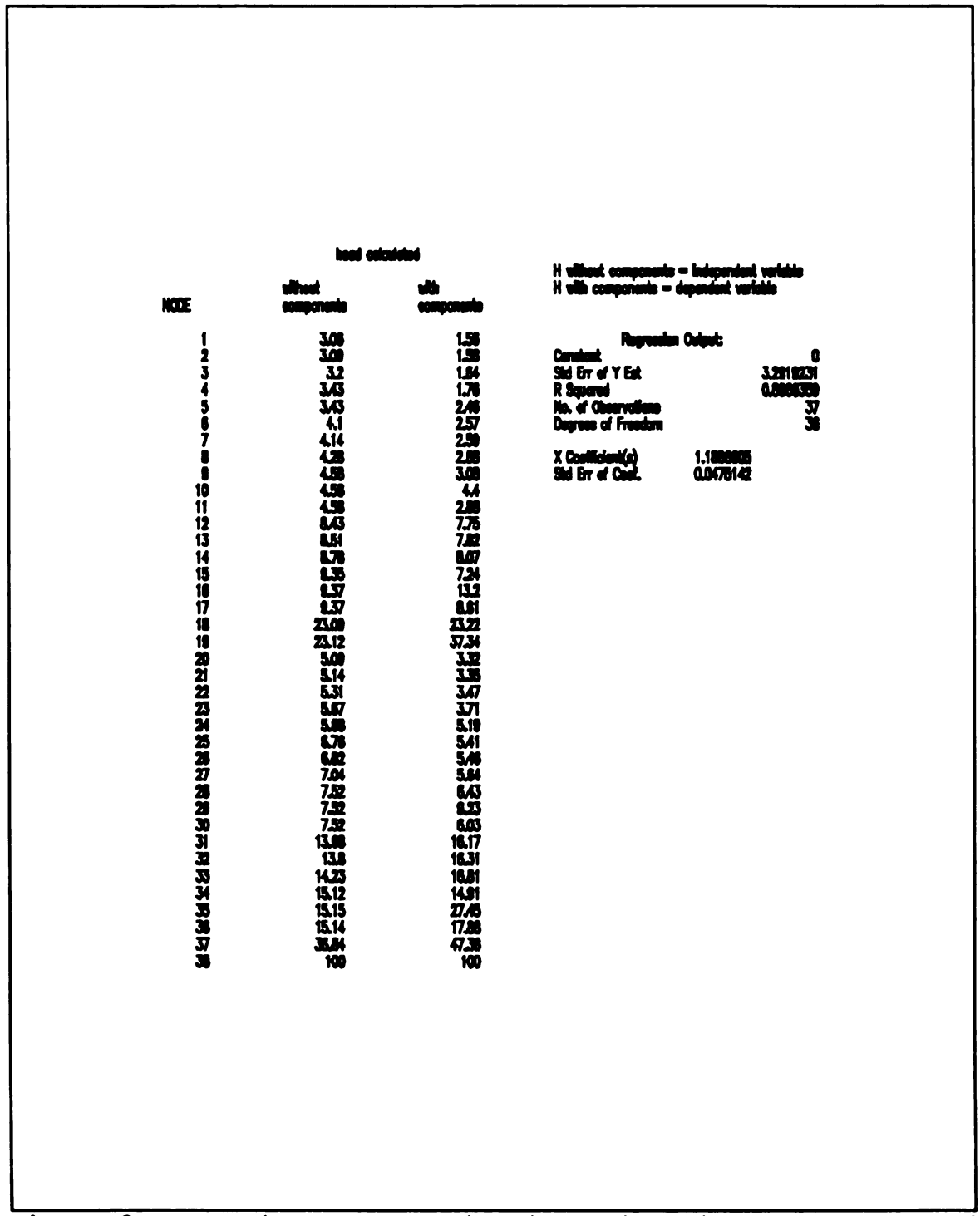


Figure 8 Comparison of solution including pipe components and solution which does not include the effect of pipe components.

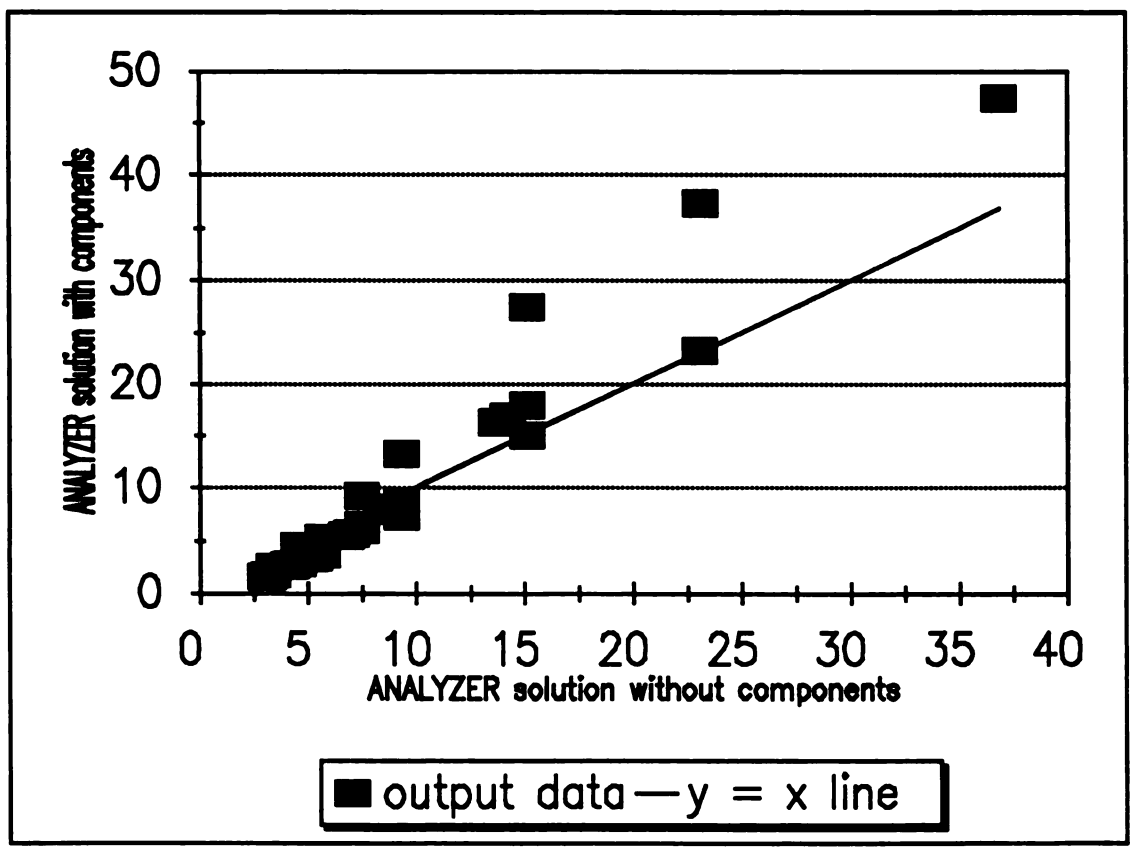


Figure 9 Comparison of network solution with and without pipe components.

The above observations demonstrate the benefits of developing a simple and efficient method for including the effect of pipe components in hydraulic network analysis without increasing the number of nodes.

C. Algebraic Development

The development which follows will result in an algebraic solution for pipe network analysis. Because the residual equations must be linearized, the solution to the resulting system of equations is therefore iterative. This formulation will include the minor losses[2] due to components such as tees, elbows, and crosses. Component head loss[3] will be calculated as a drop in hydraulic head in the elements immediately downstream of a component. An advantage of this formulation is noted in the fact that no new elements are added to the network, thereby adding no new equations to the system.

1. Linearization of Flow Equations including Minor Losses

Node i in Figure 10 represents a *tee* at a pipe junction. Any node such as node i which represents a pipe component will from here on be called a ***component node***. Flow from node i to node j will be called ***positive flow***. If flow is positive, then the

---

[2] Note here that the term *minor losses* refers to the cumulative effect of head loss due to components in an entire network.

[3] Note here that the term ***component head loss*** refers to the hydraulic head loss in a particular pipe element due to the presence of a component attached to the upstream side of the element.

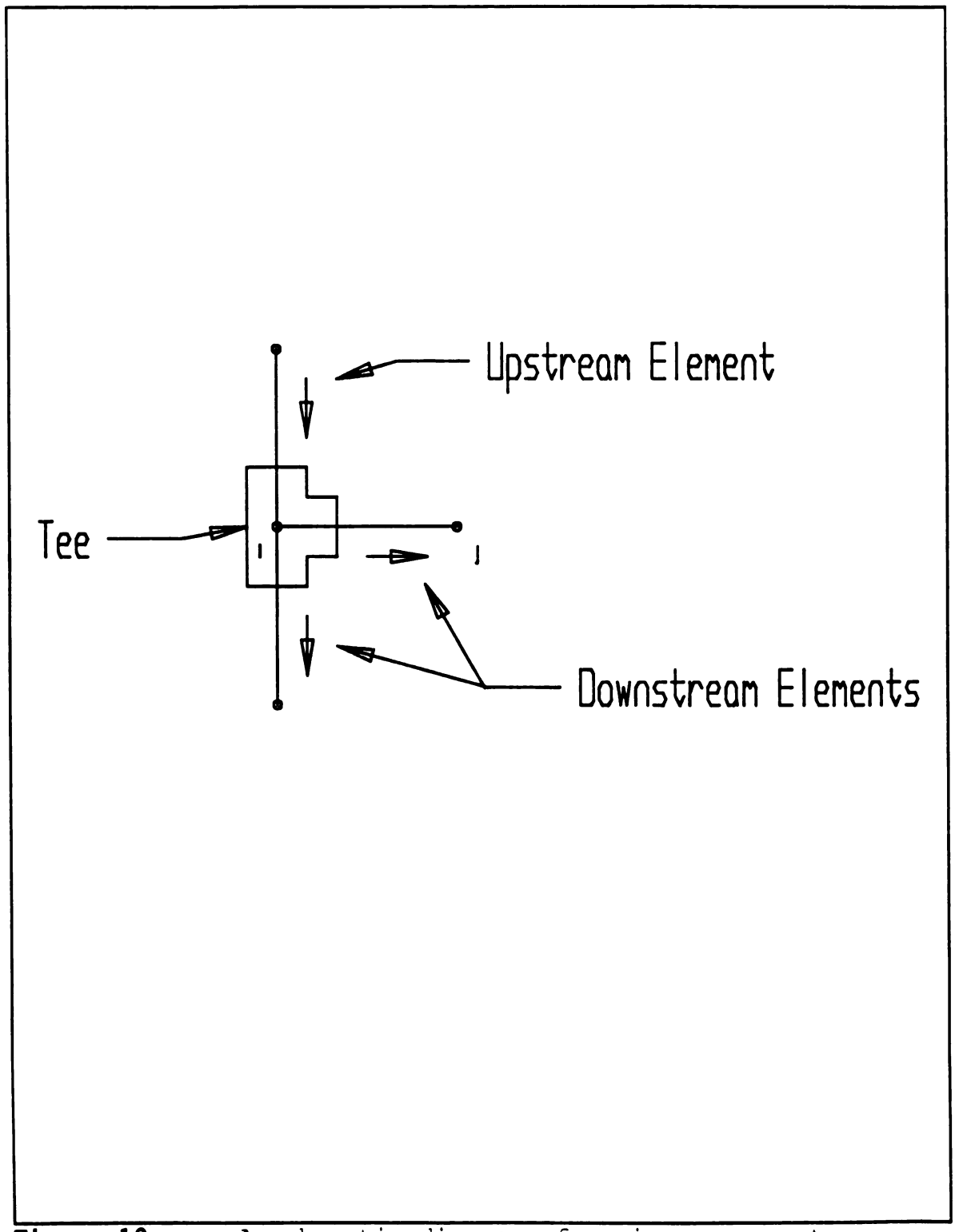


Figure 10. A schematic diagram of a pipe component.

total head difference between i and j is:

$$\Delta H_T = H_i - H_j - \Delta H_c \tag{30}$$

where ΔH_c = head loss due to the pipe component,

ΔH_T = total head loss due to pipe friction

H_i = head at node i

H_j = head at node j

Head loss at a component node is presented in Figure 11 as a sharp drop, or *discontinuity*, in the *energy grade line*; here, the logic leading to equation (30) is visually apparent. The energy grade line is a graphic representation of total energy--kinetic plus potential--and is given in terms of hydraulic head. Thus, the general equation describing the relationship between flow and head difference in an element is:

$$H_i - H_j - \Delta H_c = k^{(e)} Q^{1.852} \tag{31}$$

or, rearranging slightly,

$$H_i - H_j = k^{(e)} Q^{1.852} + \Delta H_c \tag{32}$$

While equations (31) and (32) appear trivially different, they are quite different in terms of global formulation and stability. The two methods shall be discussed separately as Method 1 (equation (31)) and Method 2 (equation (32)).

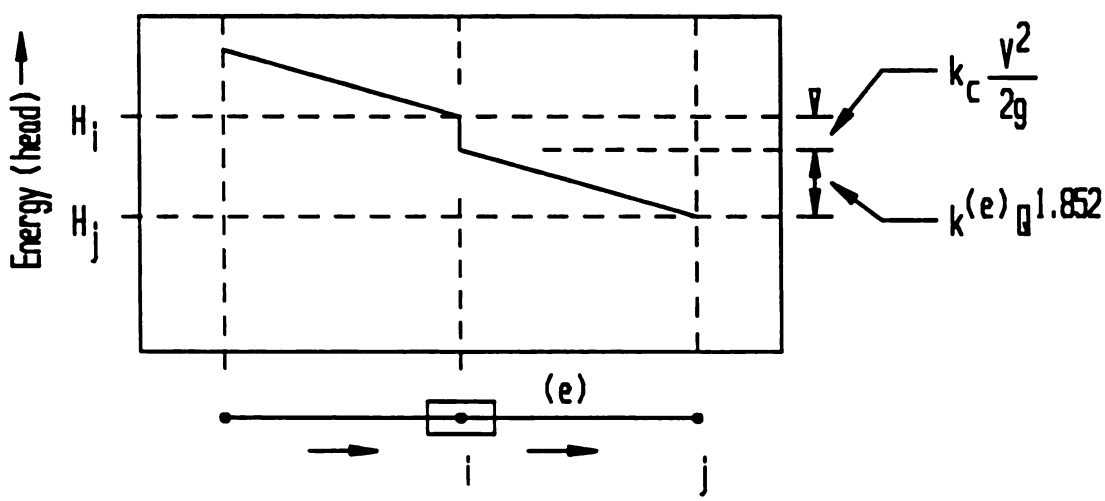


Figure 11. Energy Grade Line of downstream element.

Method 1.

Rearranging equation (31) gives an expression for flow:

$$q^{(e)} = \left(\frac{H_i - H_j - \Delta H_c}{k^{(e)}} \right)^{0.54} \qquad (33)$$

This flow equation can then be linearized by observing that

$$q_n = (H_i - H_j - \Delta H_c)_n C^{(e)} \qquad (34)$$

where $C^{(e)} = (H_i - H_j - \Delta H_c)_{n-1}^{-0.46} (k^{(e)})^{-0.54}$

Method 2.

Equation (32) is rewritten as follows:

$$H_i - H_j = k^{(e)} q^{1.852} + k_c \frac{V^2}{2g}$$

Substituting Q/A for V,

$$H_i - H_j = k^{(e)} q^{1.852} + k_c \frac{8}{g\pi^2 D^4} q^2$$

$$H_i - H_j = (k^{(e)} q^{0.852} + T^{(e)} q)\, q$$

where $k^{(e)} = \frac{4.73L}{C_{HW}^{1.852} D^{4.87}}$

and $$T^{(e)} = k_c \frac{8}{g\pi^2 D^4}$$

Or, in linearized form,

$$q_n = (H_i - H_j)_n C^{(e)} \tag{35}$$

where $$C^{(e)} = \frac{1}{k^{(e)} q_{n-1}^{0.852} + T^{(e)} q_{n-1}}$$

2. Calculation of Component Head Loss.

The component head loss term requires that velocity be known. The velocity head will therefore be based on hydraulic heads calculated in the previous iteration. Note here that at least one iteration must be performed before component head loss is taken into account. This has the advantage of automatically approximating nodal values before including the effects of components. Because continuity is expressed in terms of flow, equation (11) will be rearranged by replacing velocity with flow over area:

$$\Delta H_c = k_c \frac{q^2}{2gA^2} = k_c \frac{8q^2}{g\pi^2 D_i^4} \tag{36}$$

where A is the cross-sectional area of the component

D_i is the inside diameter of the component

The diameter, D_i, in the above equation refers more specifically to the inside

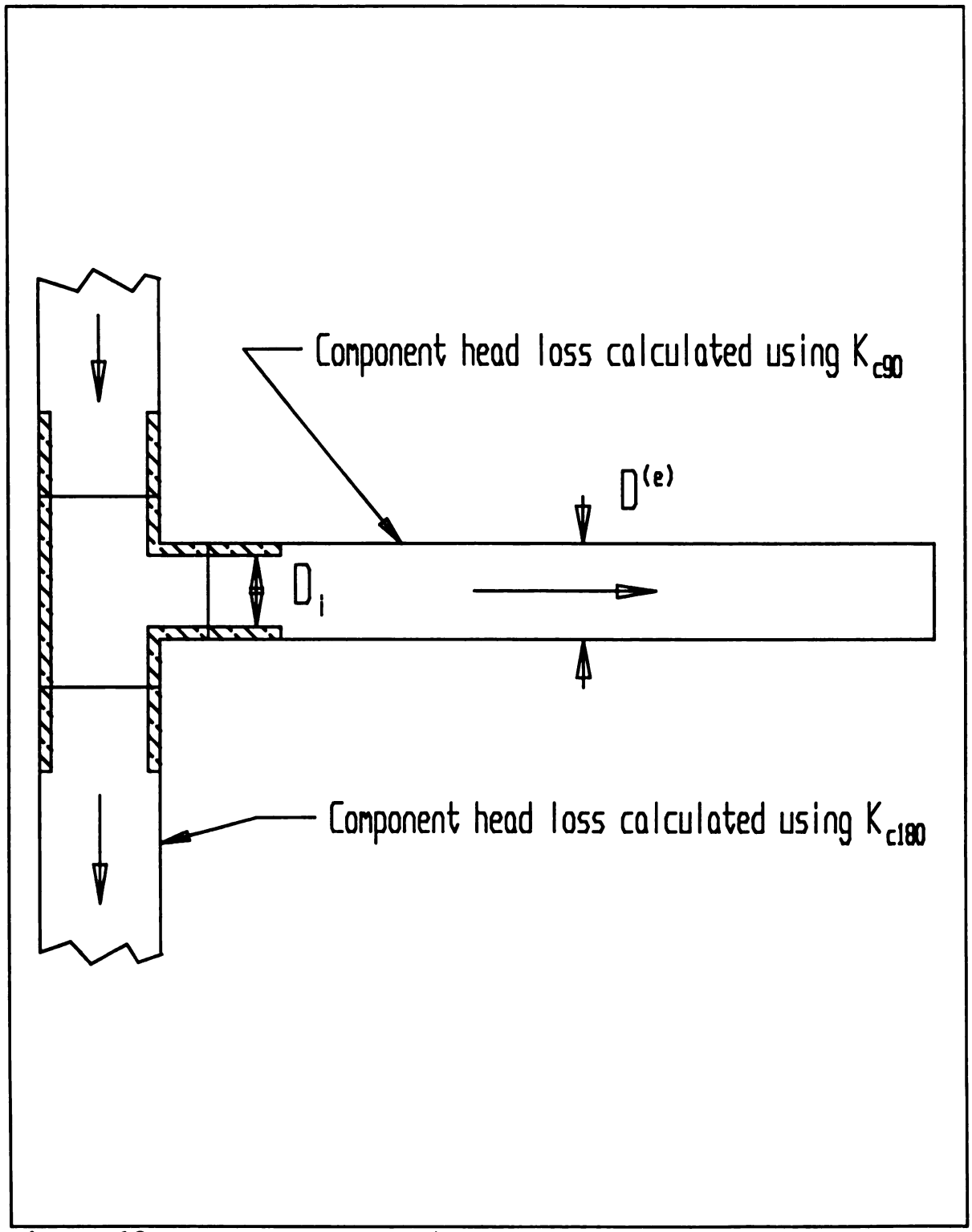


Figure 12. An explanation of references to component diameters.

diameter of the component (see Figure 12) where the pipe element and component are joined. In the case of positive flow, this occurs at node i of the element, hence the subscript. It is important to note that this diameter is specific to the pipe *element* and **not** to the component *node*. The component diameter data is therefore stored as part of the *element* data file for the computer program discussed in a later section.

Method 1.

Substituting Q from equation (33) into equation (36), we get:

$$\Delta H_c = k_{c_1}\frac{8}{g\pi^2 D_1^4}\left[\left(\frac{H_1-H_j-\Delta H_c}{k^{(e)}}\right)^{0.54}\right]^2$$

$$\Delta H_c = k_{c_1}\frac{8}{g\pi^2 D_1^4}\left(\frac{H_1-H_j-\Delta H_c}{k^{(e)}}\right)^{1.08} \qquad (37)$$

This equation must be solved numerically for ΔH_c, but a first approximation can be made by noting that the exponent is approximately equal to one ($1.08 \cong 1$). Setting the exponent equal to one and solving for ΔH_c yields:

$$\Delta H_c = \frac{T_1 (H_1-H_j)}{(k^{(e)}+T)} \qquad (38)$$

where $T_1 = k_{c_1}\dfrac{8}{g\pi^2 D_1^4}$

Equation (38) can then be used as a first approximation to solve for ΔH_c in equation (37). The numerical method employed here is the Newton-Raphson method, chosen for its rapid convergence.

Method 2.

While component head loss is not incorporated as a separate head loss *term*, it is accounted for by the $T^{(e)}q$ term. Again, the resulting equation must be solved numerically for flow:

$$q_{n-1} = \frac{(H_i - H_j)_{n-1}}{k^{(e)} q_{n-1}^{0.852} + T^{(e)} q_{n-1}} \qquad (39)$$

A first approximation is made by setting $q^{0.852} = q^1$. q can then be solved for:

$$q = \sqrt{\frac{H_i - H_j}{k^{(e)} + T^{(e)}}}$$

This first approximation is then used as a seed value for the Newton-Raphson numerical solution of equation (39).

3. Algebraic Incorporation of Component Head Loss

The global system of equations is made up of several element contributions. An element is affected by a component only if it is located immediately downstream of a component node. For every node which represents a component, therefore, the computer algorithm must first locate all elements touching the node and then determine which of these elements are downstream of the node.

Elements downstream of a component node.

The elements found to be immediately downstream of a component node will from here on be called simply *downstream elements* (see Figure 10). These elements are treated differently than other elements. First, for each downstream element, the component head loss, ΔH_c, is calculated as indicated in equation (11). Component head loss is based on the heads calculated as a result of the previous iteration and is treated as a *known* in the current iteration. This head loss, ΔH_c, is incorporated in the linearized coefficients ($C^{(e)}$s) for downstream elements as seen in equation (7).

Method 1.

Component head loss contributes to both the linearized coefficients as in equation (34) and to the right-hand side of the equations as follows. The equation describing conservation of mass at node i of a downstream element has the constant, $C^{(e)} T_1$, added to the right-hand side. Added to the right side at node j is $-C^{(e)} T_1$. In matrix formulation, this is expressed as a contribution to the forcing vector:

$$\{f^{(e)}\} = \begin{Bmatrix} C^{(e)} T_1 \\ -C^{(e)} T_1 \end{Bmatrix} \qquad (40)$$

The above equation applies only to the condition of positive flow as previously defined. When flow is negative (i.e. when flow is from node j to node i) and node j represents a component, the element contribution is then:

$$\{f^{(e)}\} = \begin{Bmatrix} -C^{(e)} T_j \\ C^{(e)} T_j \end{Bmatrix} \qquad (41)$$

Method 2.

Component head loss is incorporated only in the linearized coefficient, $C^{(e)}$, as defined in equation (35).

Selection of Component Coefficient, k_c.

A computer program called ALGNET was written based on the preceding algebraic development. The algorithm used for selecting a component's loss coefficient is described here.

For a component with only two fittings, e.g. a coupling or an elbow, only one coefficient is needed, whereas a component with more than two fittings, e.g. a tee or a cross, requires at least two coefficients for the calculation of minor losses. The nodal data file for the computer program, ALGNET, contains two coefficient values for each component node. The first value, k_{c180}, is used if an element exists *upstream* of the component node which lies in a straight line (180 degrees) with the element under scrutiny. The second value, k_{c90}, is the default value. If no upstream element exists at 180 degrees, then the coefficient, k_c, is assigned the value of k_{c90}. In Figure 13, for example, the pipe component coefficient chosen for downstream element (2) is k_{c180} because an upstream element exists, element (1), which lies between 160° and 200° from element (2). The coefficient chosen for element (3) is k_{c90} because no upstream

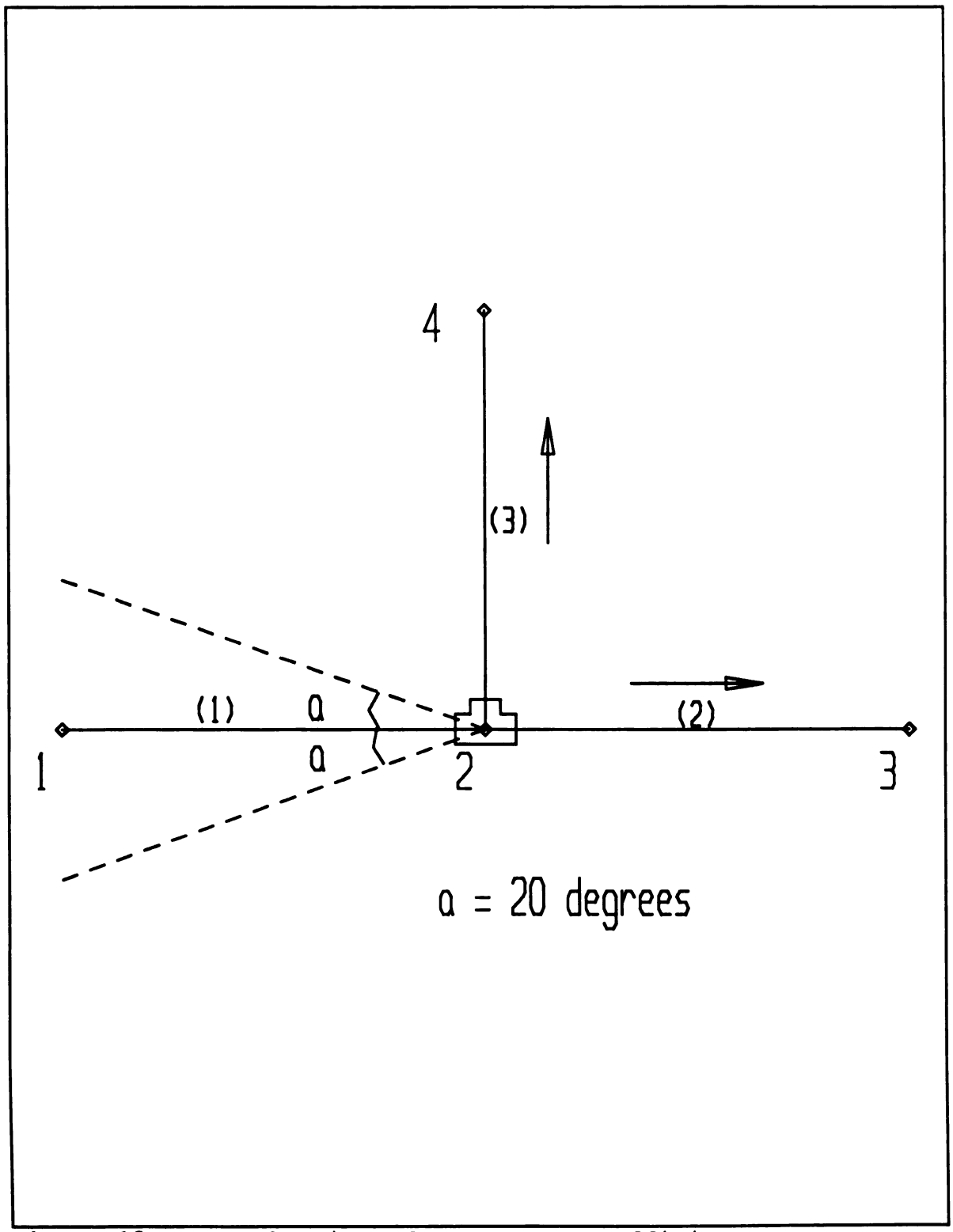


Figure 13. Selection of component coefficient.

element exists between 160° and 200° from element (3). See also Figure 12.

D. Development of Partial Differential Equation

Consider the pipe element depicted in Figure 14. Conservation of mass dictates that flow at **x** equals flow at **x+dx** plus the out-flow from the section, **dx**. As discussed in the literature review, a *virtual node* approach requires that the out-flow be considered a continuous function of **x**. The flow lost per length along a section of pipe can be formulated as a constant flow gradient:

$$\frac{\partial q_e}{\partial x} = \frac{n_e k(\bar{h}-\bar{z})^x}{L} \qquad (42)$$

where n_e = number of emitters on lateral

k = emitter constant (same for all emitters on lateral)

h = average head along lateral (see Appendix C for calculation)

z = average elevation along lateral

L = length of lateral

Conservation of mass for the pipe element in Figure 14 requires that:

$$q_x = q_{x+dx} + \frac{\partial q_e}{\partial x} dx \qquad (43)$$

Equations relating head loss to flow can be given the general form,

$$\Delta h = aq^n x$$

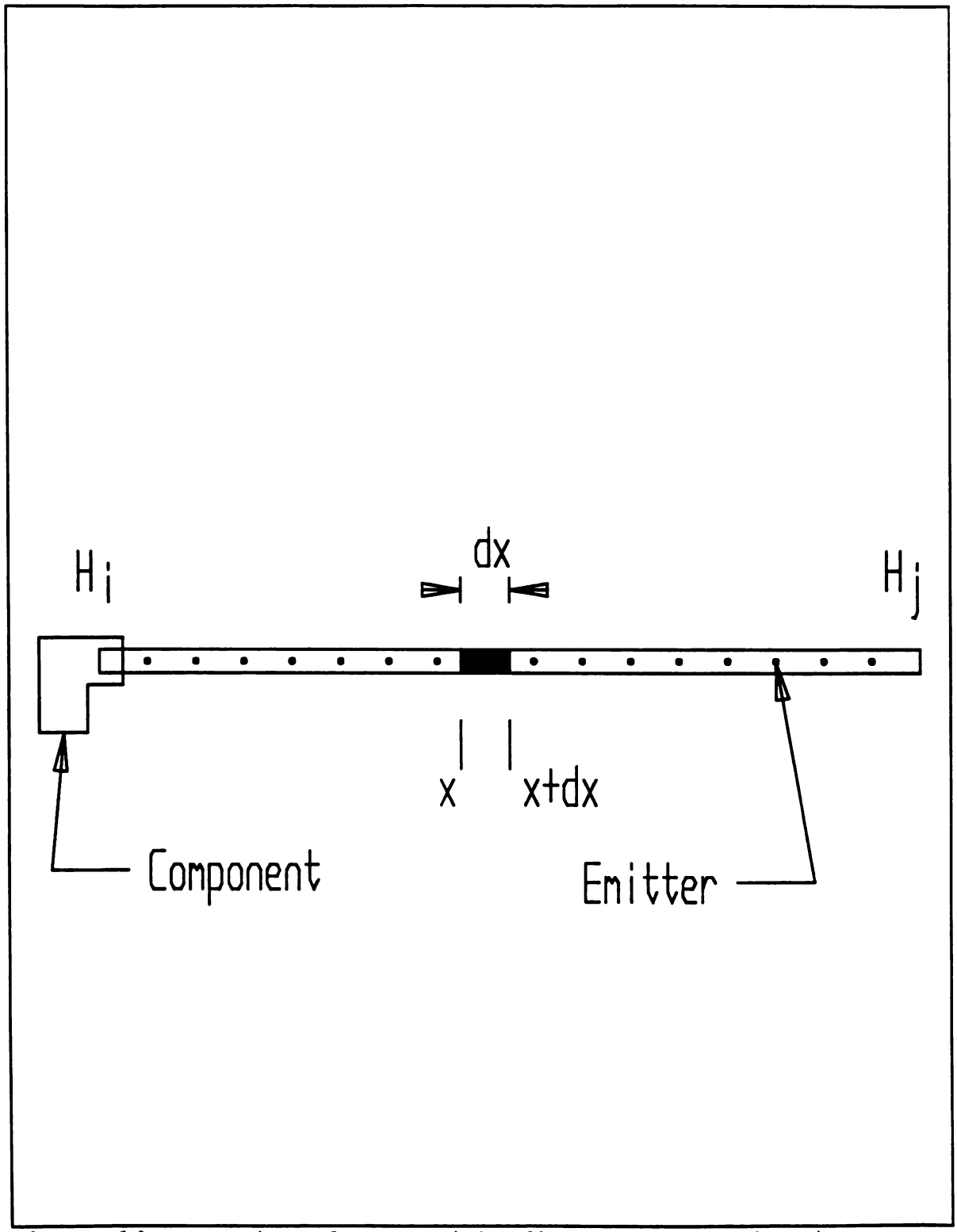


Figure 14. Pipe element with elbow and several emitters.

where, Δh = head loss due to pipe friction

x = length of pipe

m = exponent: 1.852 for Hazen-Williams or 2 for Darcy-Weisbach.

a = coefficient: $\frac{k_{sys}}{C_{HW}^{m} D^{4.87}}$ for Hazen-Williams or $f\frac{16}{\pi^2 D^5}$ for Darcy-Weisbach (see lit. review)

Over a distance, dx, this equation takes the form,

$$\frac{\partial h}{\partial x} dx = a q^m dx$$

Solving for flow,

$$q = \left(\frac{1}{a}\frac{\partial h}{\partial x}\right)^{\frac{1}{m}}$$

and linearizing the partial derivative,

$$q = D_x \frac{\partial h}{\partial x}$$

where,

$$D_x = \frac{1}{a^{\frac{1}{m}}}\left(\frac{dh}{dx}\right)^{\left(\frac{1}{m}-1\right)}$$

Substituting into equation (43) gives,

$$D_x \frac{\partial h}{\partial x}\bigg|_x = D_x \frac{\partial h}{\partial x}\bigg|_{x+dx} + \frac{\partial q_e}{\partial x} dx \qquad (45)$$

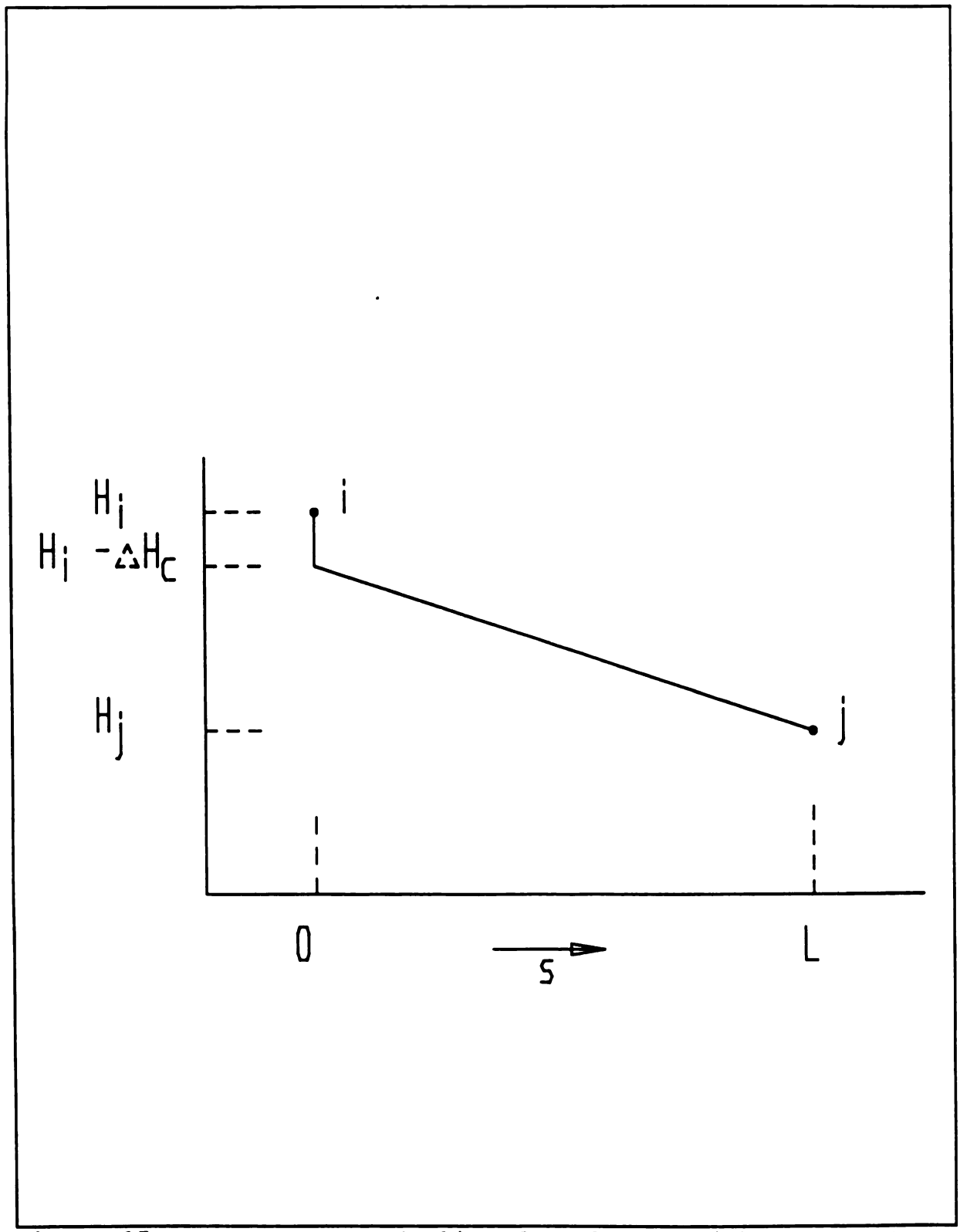


Figure 15. Energy grade line for element with component.

where,

$$D_x \frac{\partial h}{\partial x}\bigg|_{x+dx} = D_x \frac{\partial h}{\partial x}\bigg|_{x} + D_x \frac{\partial^2 h}{\partial x^2} dx \qquad (46)$$

Substituting equation (46) into equation (45) gives,

$$D_x \frac{\partial^2 h}{\partial x^2} + \frac{\partial q_e}{\partial x} = 0 \qquad (47)$$

where $\frac{\partial q_e}{\partial x}$ can be treated as a constant gradient, Q:

$$Q = \frac{\partial q_e}{\partial x} = \frac{n_e k (\bar{h} - \bar{z})^x}{L} \qquad (48)$$

or as a linearized function of h, Gh:

$$G\,h = \frac{\partial q_e}{\partial x} = \left(\frac{n_e k}{L} \frac{(\bar{h} - \bar{z})^x}{\bar{h}} \right) h \qquad (49)$$

Results of both approaches are compared in the Results and Discussion section. Equation (47) is the governing equation for lateral flow which is was the equation presented by Bralts et. al (1993). This equation, however, does not account for the presence of pipe components.

1. Incorporation of pipe components

The fundamental problem of pipe components is the energy discontinuity which they present. This becomes especially problematic when a component is treated as a part of its downstream element. As a result, the energy grade line (Figure 11) contains a discontinuity *within* an element as shown in Figure 15[4]. Intra-element discontinuities are often prohibitive because their derivatives are undefined; however, this one can be dealt with because it occurs at a node. Again, two approaches shall be considered.

Method 1.

The first approach accommodates the discontinuity in the head equation, using a linear shape function to weight the residuals and employing Galerkin's method. Essentially, the function shown in Figure 15 is a linear function with the following boundary conditions:

$$h = H_i - \Delta H_c \quad at \quad s = 0$$

$$h = H_j \quad at \quad s = L$$

The equation for head is then derived as a linear function of **s**:

$$h = (H_i - \Delta H_c) + \left(\frac{H_j - H_i + \Delta H_c}{L}\right) s$$

[4]Note here that the coordinate system has been changed from the global system, x, to the local system, s. Because scale remains the same, derivatives are not changed.

The linear shape functions are:

$$N_i = \left(1 - \frac{s}{L}\right)$$

$$N_j = \frac{s}{L}$$

Integration at node i for the second-partial-derivative term (or *D-term*) then yields:

$$\int_0^L D \frac{dN_i}{ds} \frac{dh}{ds} ds = \frac{D}{L}(H_i - H_j - \Delta H_c) \tag{50}$$

and at node j:

$$\int_0^L D \frac{dN_j}{ds} \frac{dh}{ds} ds = \frac{D}{L}(-H_i + H_j + \Delta H_c) \tag{51}$$

Equation (50) and equation (51) combine to take on the matrix form,

$$\int_0^L D \frac{d[N]^T}{ds} \frac{dh}{ds} ds = \frac{D}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} H_i \\ H_j \end{Bmatrix} - \frac{D}{L}\Delta H_c \begin{Bmatrix} 1 \\ -1 \end{Bmatrix} \tag{52}$$

or, in matrix notation,

$$\int_0^L D \frac{\partial [N]^T}{\partial s} \frac{\partial h}{\partial s} ds = [k_D^{(e)}]\{H^{(e)}\} - \{f_D^{(e)}\}\Delta H_c^{(e)} \tag{53}$$

Likewise, integration of the G-term yields:

$$\int_0^L G[N]^T h \, ds = \frac{GL}{6}\begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}\begin{Bmatrix}H_1\\ H_J\end{Bmatrix} - \frac{GL}{6}\Delta H_c\begin{Bmatrix}2\\ 1\end{Bmatrix} \tag{54}$$

$$= [k_G^{(e)}]\,\{H^{(e)}\} - \{f_G^{(e)}\}\,\Delta H_c^{(e)}$$

And integration of the Q-term yields:

$$\int_0^L Q[N]^T ds = \frac{QL}{2}\begin{Bmatrix}1\\ 1\end{Bmatrix} = \{f_Q^{(e)}\} \tag{55}$$

If emitter flow is considered a linearized function of head (in other words, if emitter flow is expressed in the *G-term*), the following residual equation results for element, (e):

$$\{R^{(e)}\} = ([k_D^{(e)}] + [k_G^{(e)}])\,\{H^{(e)}\} - \{f_D^{(e)}\}\,\Delta H_c^{(e)} \tag{56}$$

Otherwise, if emitter flow is treated as a constant (in other words, if emitter flow is expressed in the *Q-term*), the residual equation is:

$$\{R^{(e)}\} = [k_D^{(e)}]\,\{H^{(e)}\} - \{f_D^{(e)}\}\,\Delta H_c^{(e)} - \{f_Q^{(e)}\} \tag{57}$$

Method 2.

The second approach accommodates the energy discontinuity again by adding

pipe head loss, ΔH_p, to component head loss, ΔH_c:

$$\Delta H_T^{(\bullet)} = \Delta H_p^{(\bullet)} + \Delta H_c^{(\bullet)}$$

Total head loss over length, L, (Figure 14) takes the form:

$$\frac{\partial h}{\partial x} L = aQ^m L + TQ^2 \qquad (58)$$

where $T = \dfrac{8k_c}{g\pi^2 d^4}$ as in the algebraic development.

L = length of element

Equation (58) can be solved for linearized flow as follows:

$$\frac{\partial h}{\partial x} L = \left(aQ_{n-1}^{m-1} L + TQ_{n-1}\right) Q_n$$

$$Q_n = \left[\frac{L}{aQ_{n-1}^{m-1} L + TQ_{n-1}}\right] \frac{\partial h}{\partial x}$$

Similar logic to that depicted in equations (45) through (47) results in the following general equation:

$$D_x \frac{\partial^2 h}{\partial x^2} + G\,h - Q = 0 \qquad (59)$$

where,

$$D_x = \left[\frac{L}{aQ^{m-1} L + TQ}\right]_{n-1}$$

Equation (59) is the general form of a second-order partial differential equation. The coefficients, D_x, G, and Q are listed in Table I for both methods.

Table I: List of coefficients for four different approaches.

	D_x	G	Q
Method 1: $\frac{\partial q_e}{\partial x} = Q$	$\left.\frac{1}{a^{\frac{1}{m}}}\left(\frac{dh}{dx}\right)^{(\frac{1}{m}-1)}\right\|_{n-1}$	0	$\left.\frac{n_e k(\bar{h}-\bar{z})^x}{L}\right\|_{n-1}$
Method 1: $\frac{\partial q_e}{\partial x} = G\,h$	$\left.\frac{1}{a^{\frac{1}{m}}}\left(\frac{dh}{dx}\right)^{(\frac{1}{m}-1)}\right\|_{n-1}$	$\left.\frac{n_e k}{L}\frac{(\bar{h}-\bar{z})^x}{\bar{h}}\right\|_{n-1}$	0
Method 2: $\frac{\partial q_e}{\partial x} = Q$	$\left[\frac{L}{aq^{m-1}L + Tq}\right]_{n-1}$	0	$\left.\frac{n_e k(\bar{h}-\bar{z})^x}{L}\right\|_{n-1}$
Method 2: $\frac{\partial q_e}{\partial x} = G\,h$	$\left[\frac{L}{aq^{m-1}L + Tq}\right]_{n-1}$	$\left.\frac{n_e k}{L}\frac{(\bar{h}-\bar{z})^x}{\bar{h}}\right\|_{n-1}$	0

It is worth noting that the results of the algebraic development are in agreement with the equations developed in this section.

Each method's contribution to the global stiffness matrix and forcing vector is listed in table II.

Table II: Element contributions to global system.

	Stiffness Matrix	Forcing Vector
Method 1: $\frac{\partial q_e}{\partial x} = Q$	$[k_D^{(e)}]$	$\{f_Q^{(e)}\} + \{f_D^{(e)}\}\Delta H_c^{(e)}$
Method 1: $\frac{\partial q_e}{\partial x} = G\,h$	$[k_D^{(e)}] + [k_G^{(e)}]$	$\{f_D^{(e)}\}\Delta H_c^{(e)} + \{f_G^{(e)}\}\Delta H_c^{(e)}$
Method 2: $\frac{\partial q_e}{\partial x} = Q$	$[k_D^{(e)}]$	$\{f_Q^{(e)}\}$
Method 2: $\frac{\partial q_e}{\partial x} = G\,h$	$[k_D^{(e)}] + [k_G^{(e)}]$	0

The vectors and matrices are listed below:

$$[k_D^{(e)}] = \frac{D}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$[k_G^{(e)}] = \frac{GL}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$\{f_Q^{(e)}\} = \frac{QL}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

$$\{f_D^{(e)}\} = \frac{D}{L}\begin{Bmatrix} 1 \\ -1 \end{Bmatrix}$$

$$\{f_G^{(e)}\} = \frac{GL}{6}\begin{Bmatrix} 2 \\ 1 \end{Bmatrix}$$

IV. Results and Discussion

A. Evaluation of Solution without Virtual Nodes

The effect of pipe components in a hydraulic network system is incorporated in the ALGNET and DIFNET computer programs (see Appendix A for code). ALGNET1 encodes method 1 as described in the Algebraic Development section; ALGNET2 encodes method 2. The results of both ALGNET programs were compared with the ANALYZER and KYPIPE programs. ANALYZER makes use of the Finite Element Method and incorporates pipe components as separate elements. KYPIPE uses the Linear Theory method and includes pipe components by adding their effect to the pipe elements. Both ANALYZER and KYPIPE programs have been empirically tested and found to be accurate. Several different hydraulic networks were analyzed by these three programs, and the results were found to be very strongly correlated in all cases.

As an example, a hydraulic network system is shown in Figure 16 and Figure 17. Figure 16 depicts how the network would be labelled for analysis by the ANALYZER program, whereas Figure 17 shows the same network labelled for analysis by the ALGNET program. Note here the reduction in the number of nodes to be analyzed, from 12 to 6. A full explanation of how ALGNET data files are set up for this same network is given in Appendix B.

The hydraulic head values calculated by ALGNET are compared to the values

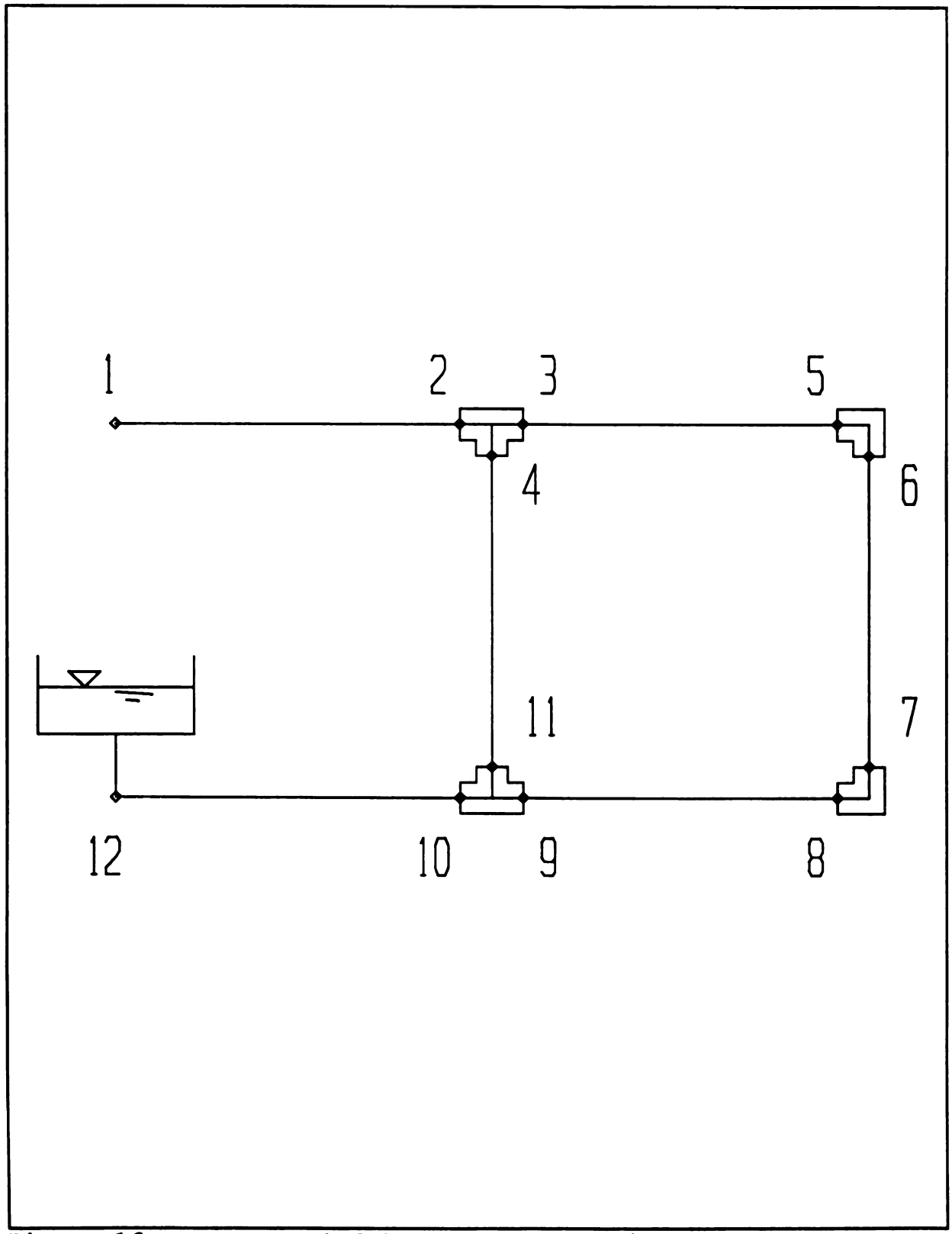


Figure 16 Network labeled for analysis by ANALYZER

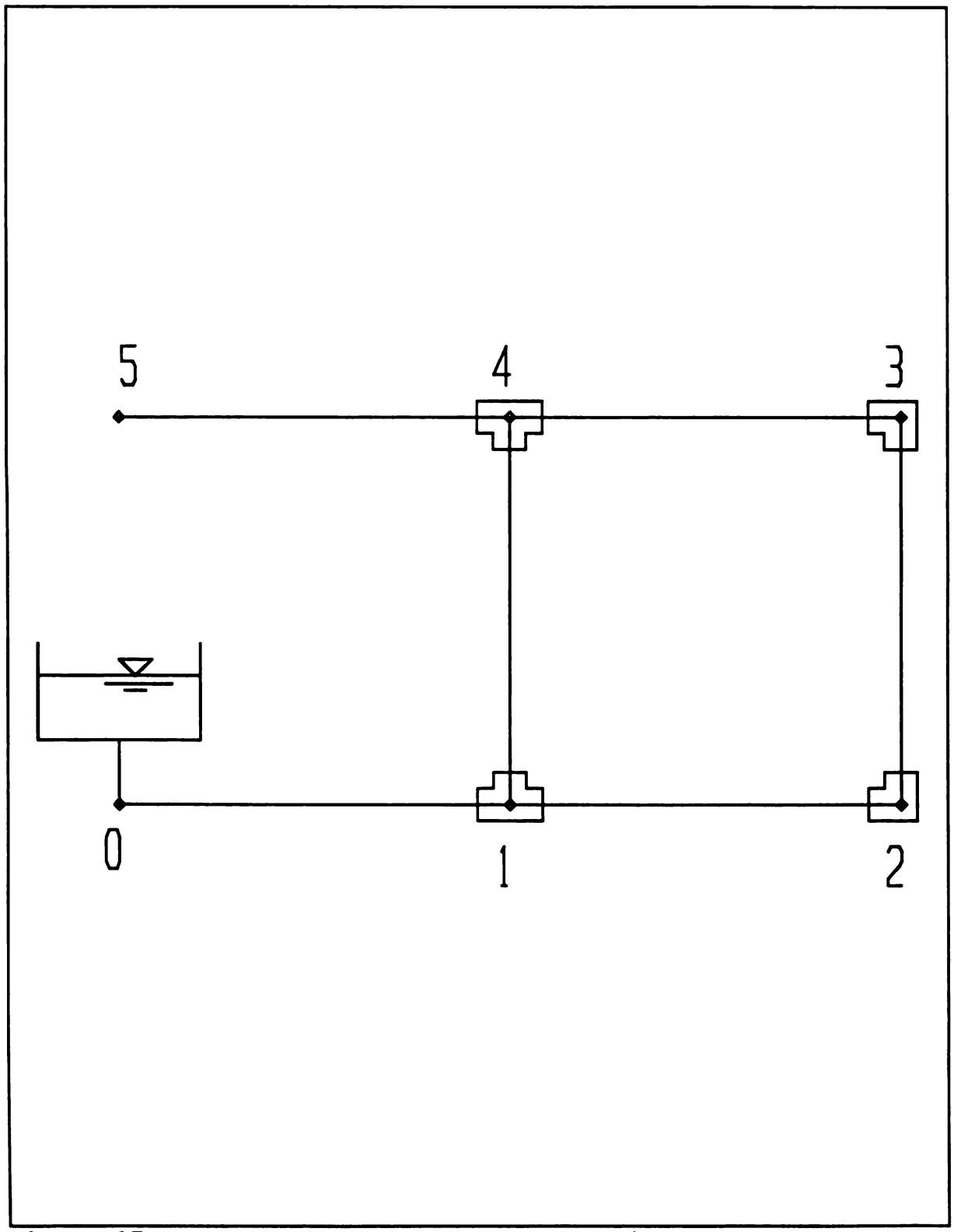


Figure 17 Network labeled for analysis by ALGNET

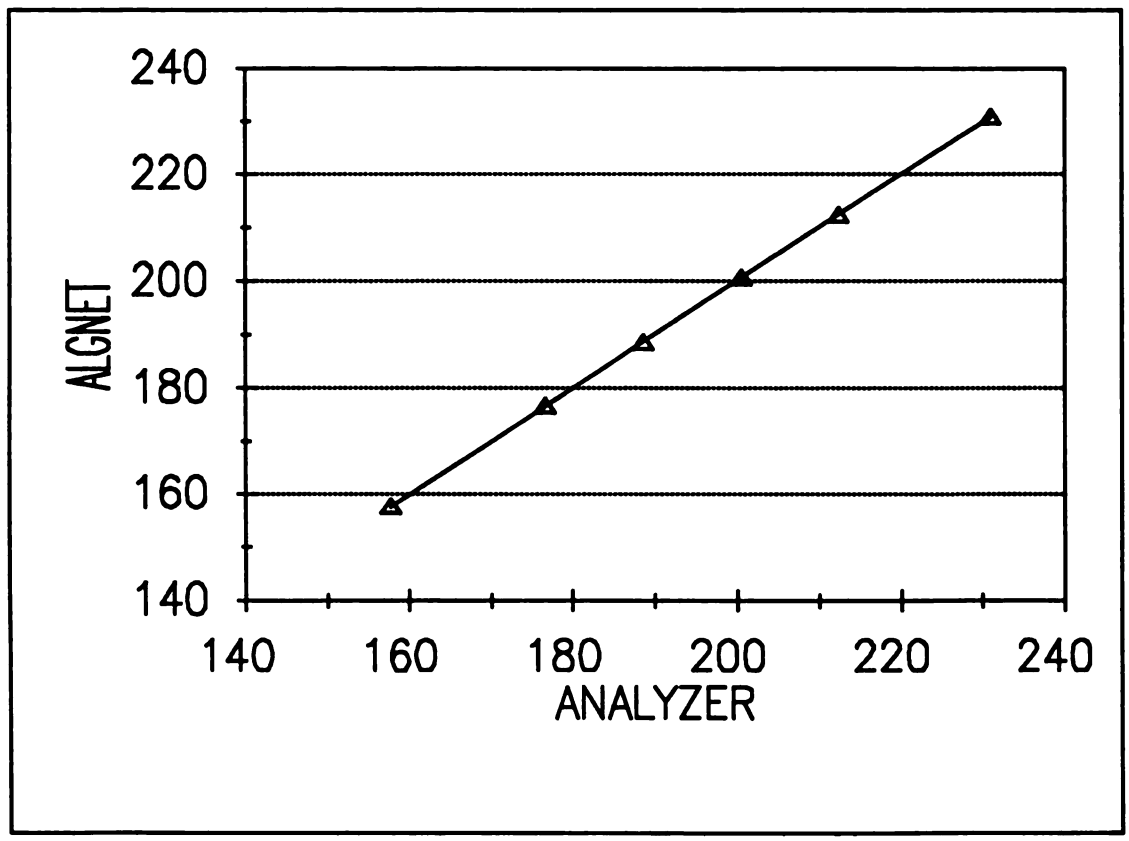


Figure 18 Regression plotted.

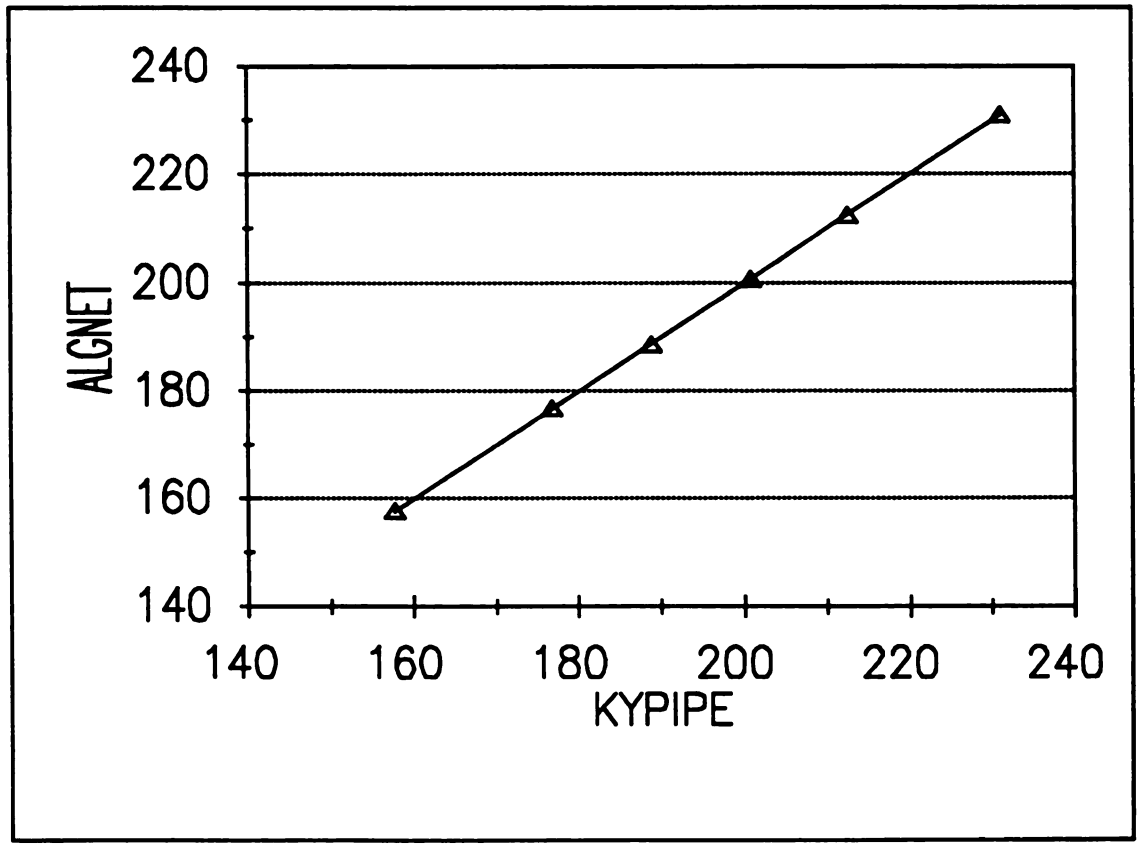


Figure 19 Regression plotted.

head calculations

ANALYZE nodes	ALGNET nodes	DIFNET1 ALGNET1	DIFNET2 ALGNET2	ANALYZE	KYPIPE
12	0	231.00	231.00	231.00	231.00
10	1	212.51	212.51	212.34	212.50
8	2	200.81	200.81	200.57	200.82
6	3	188.80	188.80	188.60	188.83
4	4	176.79	176.79	176.64	176.84
1	5	157.73	157.73	157.74	157.74

ALGNET1 = dependent variable
ANALYZER = independent variable

Regression Output:

Constant	0
Std Err of Y Est	0.105599
R Squared	0.999984
No. of Observations	6
Degrees of Freedom	5
X Coefficient(s)	1.000632
Std Err of Coef.	0.00022

ALGNET2 = dependent variable
ANALYZER = independent variable

Regression Output:

Constant	0
Std Err of Y Est	0.105625
R Squared	0.999984
No. of Observations	6
Degrees of Freedom	5
X Coefficient(s)	1.000637
Std Err of Coef.	0.00022

ALGNET1 = dependent variable
KYPIPE = independent variable

Regression Output:

Constant	0
Std Err of Y Est	0.023208
R Squared	0.999999
No. of Observations	6
Degrees of Freedom	5
X Coefficient(s)	0.999925
Std Err of Coef.	4.83E-05

ALGNET2 = dependent variable
KYPIPE = independent variable

Regression Output:

Constant	0
Std Err of Y Est	0.02227
R Squared	0.999999
No. of Observations	6
Degrees of Freedom	5
X Coefficient(s)	0.99993
Std Err of Coef.	4.64E-05

Figure 20 The strong correlation between methods is demonstrated here.

calculated by ANALYZER and KYPIPE in Figure 20. Comparison of the two methods is somewhat complicated by the fact that ANALYZER takes pipe components as separate elements while ALGNET does not. The effect of a component in ALGNET is incorporated into the downstream element; as a result, the ANALYZER nodes chosen for comparison with ALGNET should be nodes situated "upstream" of components.

The strong correlation (Figure 20) between a method which accommodates pipe components as separate elements and a method which accommodates pipe components as energy discontinuities within elements indicates that the error introduced as a result of the discontinuities is negligible for practical purposes.

B. Evaluation of Solutions with Virtual Nodes

The computer code, DIFNET1, incorporates the virtual node concept using Method 1 as defined in the Theoretical Development; DIFNET2 incorporates this concept using Method 2. The network shown in Figure 21, for example, is reduced to the network shown in Figure 22 by incorporation of the virtual node concept. DIFNET takes each lateral to be a single linear element, thus greatly reducing the number of nodes--in this case, from 21 nodes to 9 nodes. A complete explanation of how DIFNET data files are set up for the same network is found in Appendix B.

1. Error introduced by virtual node concept

While the number of nodes in a network is greatly decreased by using virtual nodes, error is slightly increased. Results, however, seem still quite acceptable. Figure 23 shows a strong correlation between DIFNET and ALGNET. If greater

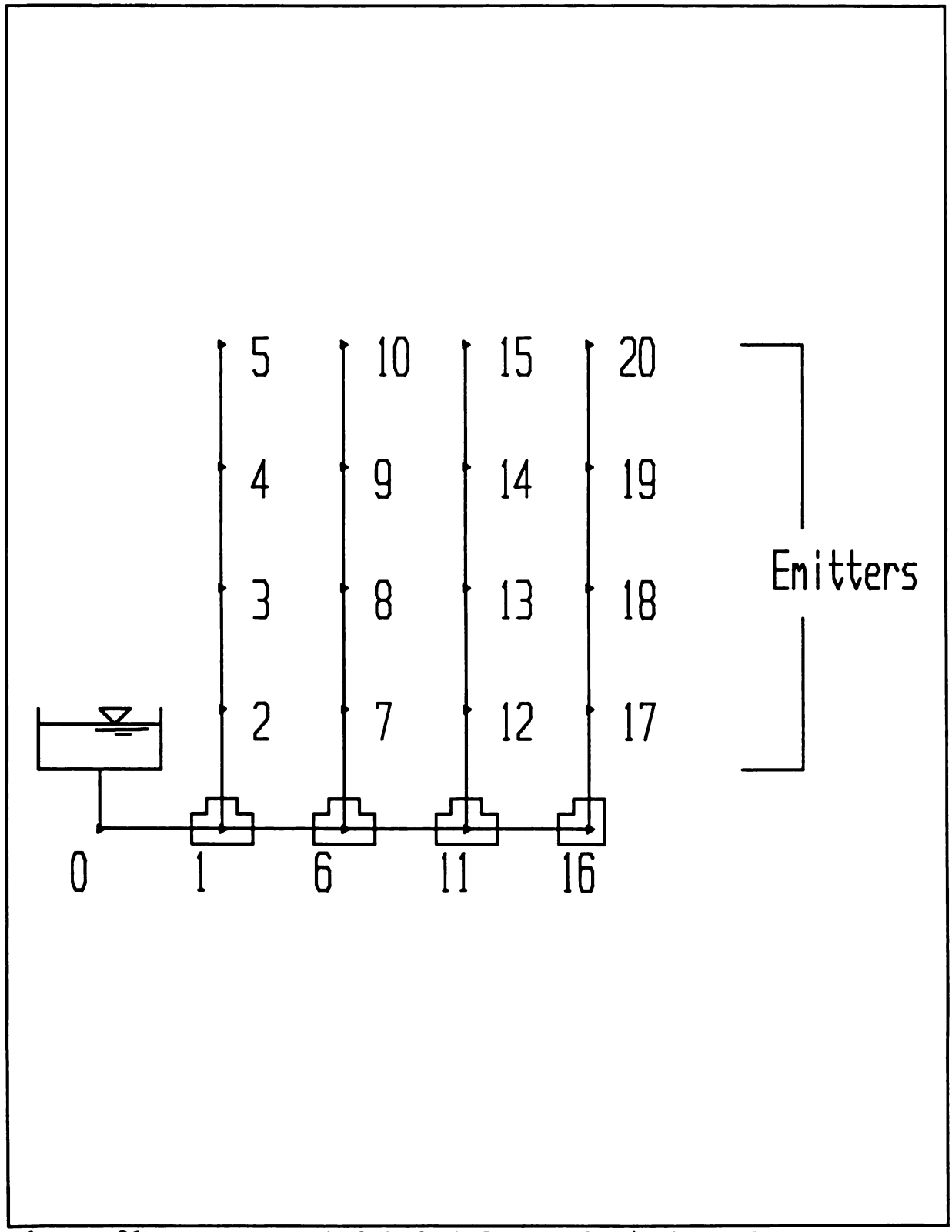


Figure 21. Network labeled for analysis by ALGNET.

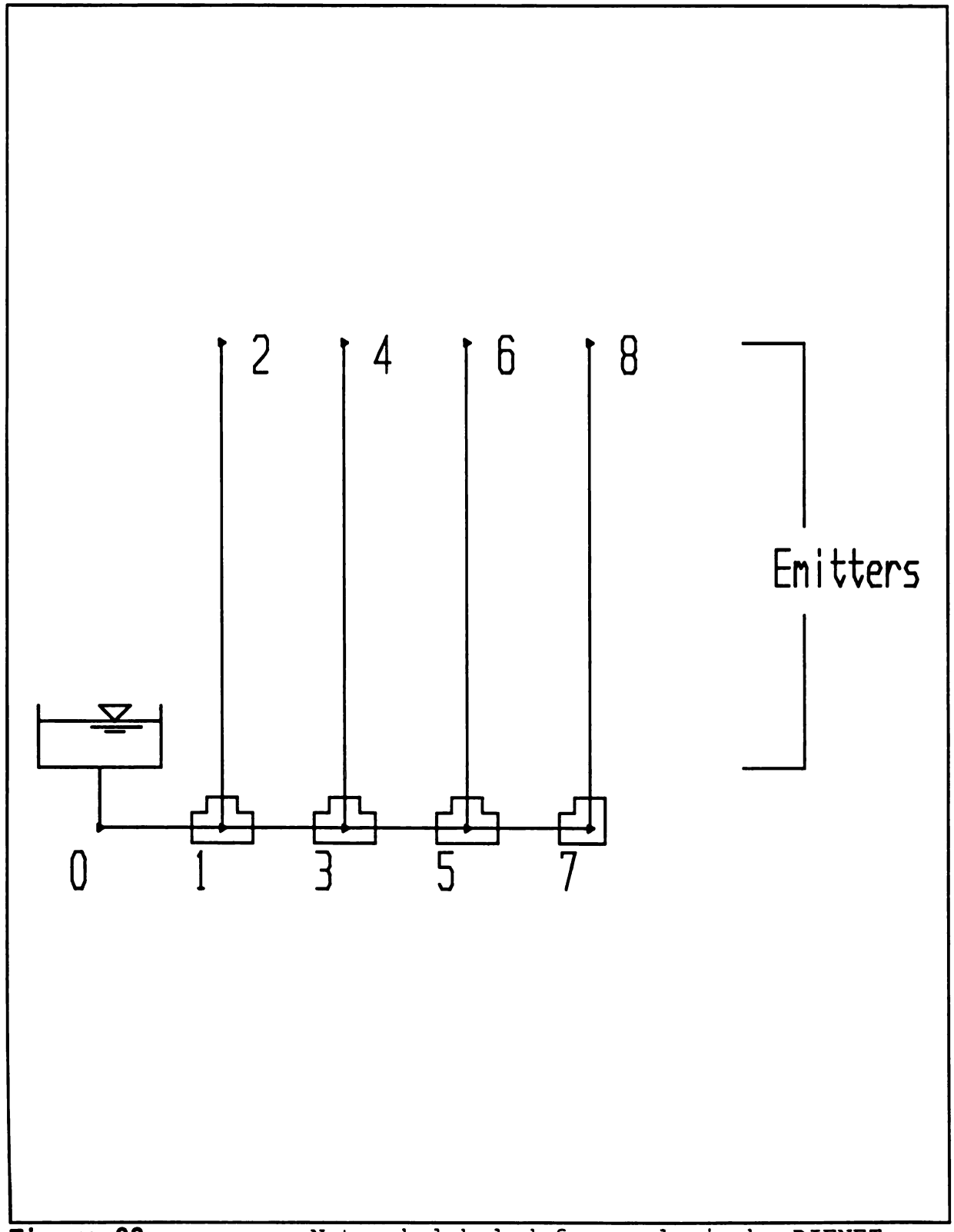


Figure 22. Network labeled for analysis by DIFNET.

node	ALGNET	DIFNET2	residuals
0	100.00	100.00	0.00
1	73.11	71.92	-1.20
2	54.10	60.27	6.17
3	37.56	35.51	-2.05
4	27.64	29.61.	1.98
5	24.12	21.99	-2.13
6	17.68	18.27	0.59
7	20.87	18.75	-2.12
8	15.28	15.56	0.28

average of residuals = 0.168767
standard deviation of residuals = 2.512838

Regression Output:

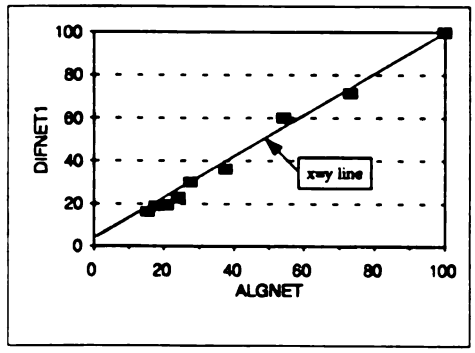
Constant	0
Std Err of Y Est	2.649425
R Squared	0.991964
No. of Observations	9
Degrees of Freedom	8
X Coefficient(s)	1.0065
Std Err of Coef.	0.017858

node	ALGNET	DIFNET1	residuals
0	100.00	100.00	0.00
1	73.11	71.52	-1.59
2	54.10	59.99	5.89
3	37.56	36.04	-1.52
4	27.64	30.08	2.44
5	24.12	22.65	-1.47
6	17.68	18.84	1.16
7	20.87	19.40	-1.47
8	15.28	16.12	0.84

average of residuals = 0.476142
standard deviation of residuals = 2.352907

Regression Output:

Constant	0
Std Err of Y Est	2.509788
R Squared	0.992619
No. of Observations	9
Degrees of Freedom	8
X Coefficient(s)	1.008183
Std Err of Coef.	0.016917



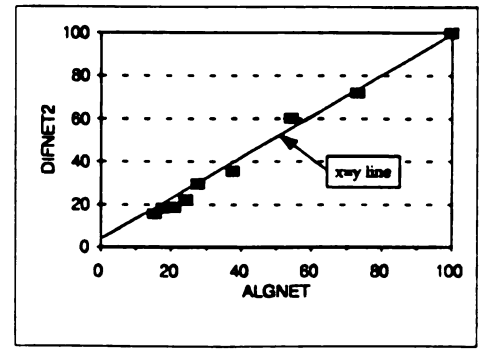


Figure 23 Assessment of Virtual Node technique by comparing DIFNET against ALGNET. Note that results are slightly different for DIFNET1 and DIFNET2.

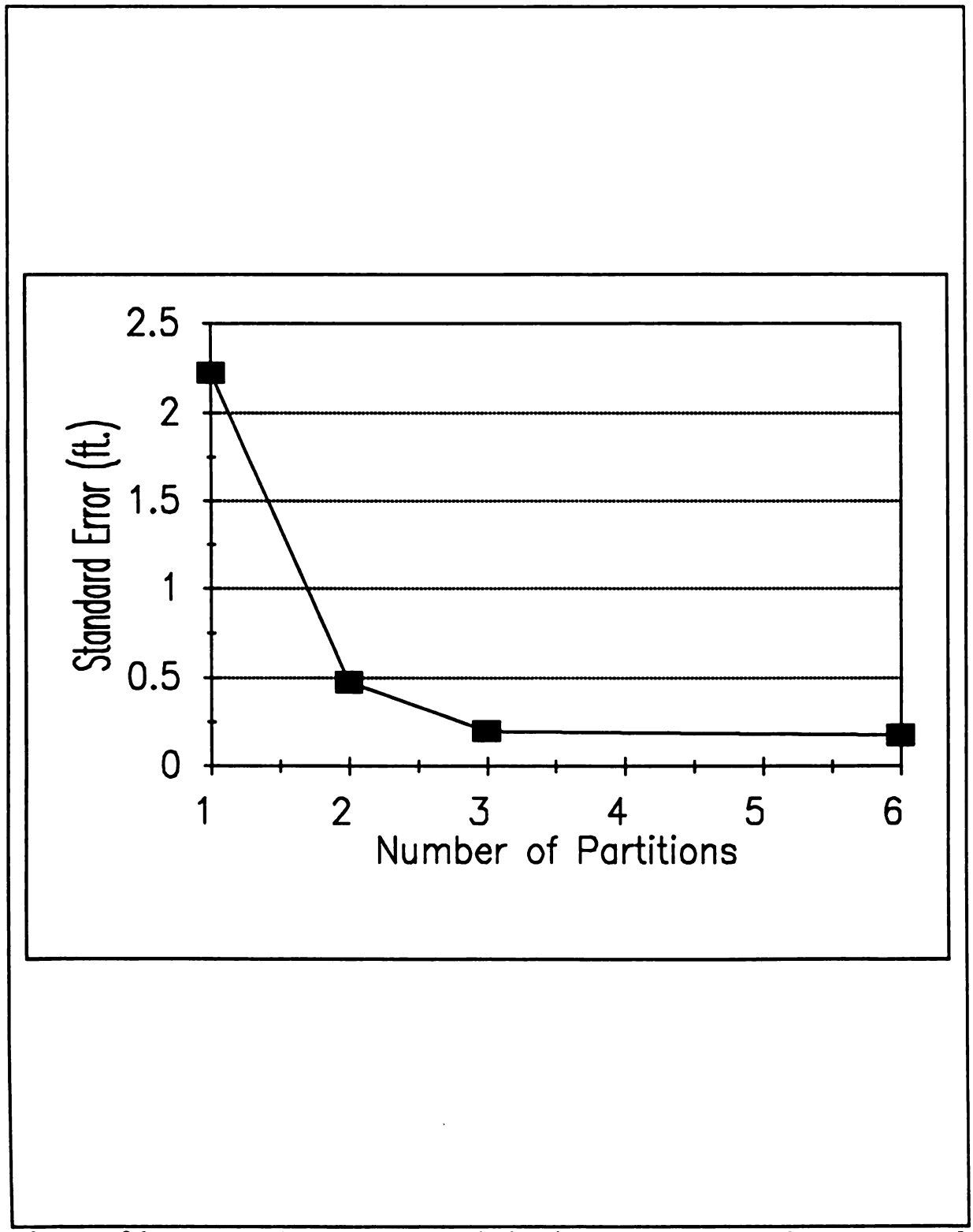


Figure 24. Effect of partitioning the laterals on total error.

accuracy is desired, error can be reduced by subdividing the laterals into two or more linear elements. Figure 24 shows the effect which partitioning the laterals has on the total error. Note that the error in this graph seems to approach an asymptote which is not zero. The fact that total error does not approach zero with an increasing number of partitions is largely due to the error inherent in approximating a discrete function (emitters on a lateral) with a continuous function (derivative boundary condition)--refered to in this thesis as *continuization error*. Another strategy for error reduction, perhaps a little closer to the true nature of flow in pipe networks, is the use of quadratic elements in place of linear elements. This strategy was used by Kelly (1989) and Bralts et al. (1993).

C. Reducing a large network to a small, manageable size

What follows is a demonstration of the benefits of the concepts developed in this thesis. Figure 25 shows the system to be analyzed, a medium sized submain unit consisting of 20 laterals and 600 emitters per lateral for a total of 12,000 emitters in an area of 1.37 acres. A one-percent slope goes downhill from the submain. Previous methods would require over 12,000 nodes for analysis of this network. The methods developed in this research require only 80 nodes for analysis, about 0.7 percent of 12,000. (This is achieved by partitioning each lateral into three virtual elements as seen in Figure 25.) A great reduction in computer time and memory requirements is evident as a result of this drastic reduction of nodes. Figure 26 compares the solution to the Backstep solution on the last lateral. Values between virtual nodes are calculated by interpolation. Correlation between the two solutions was calculated

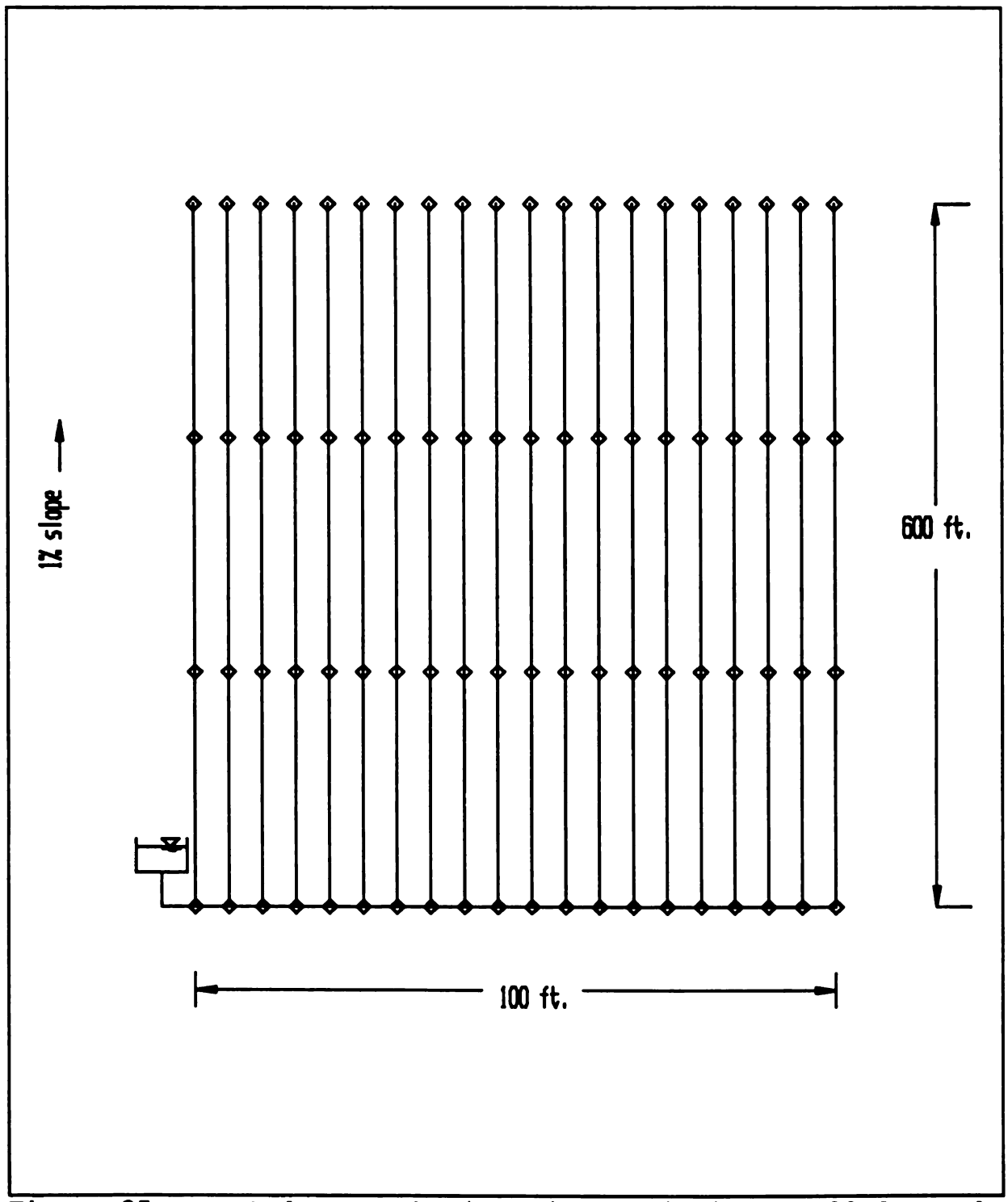


Figure 25. A large submain unit consisting of 20 laterals spaced 5 ft. apart and 600 emitters per lateral spaced 1 ft. apart. Previous methods would require 12,060 nodes for analysis. Here, only 80 nodes are used.

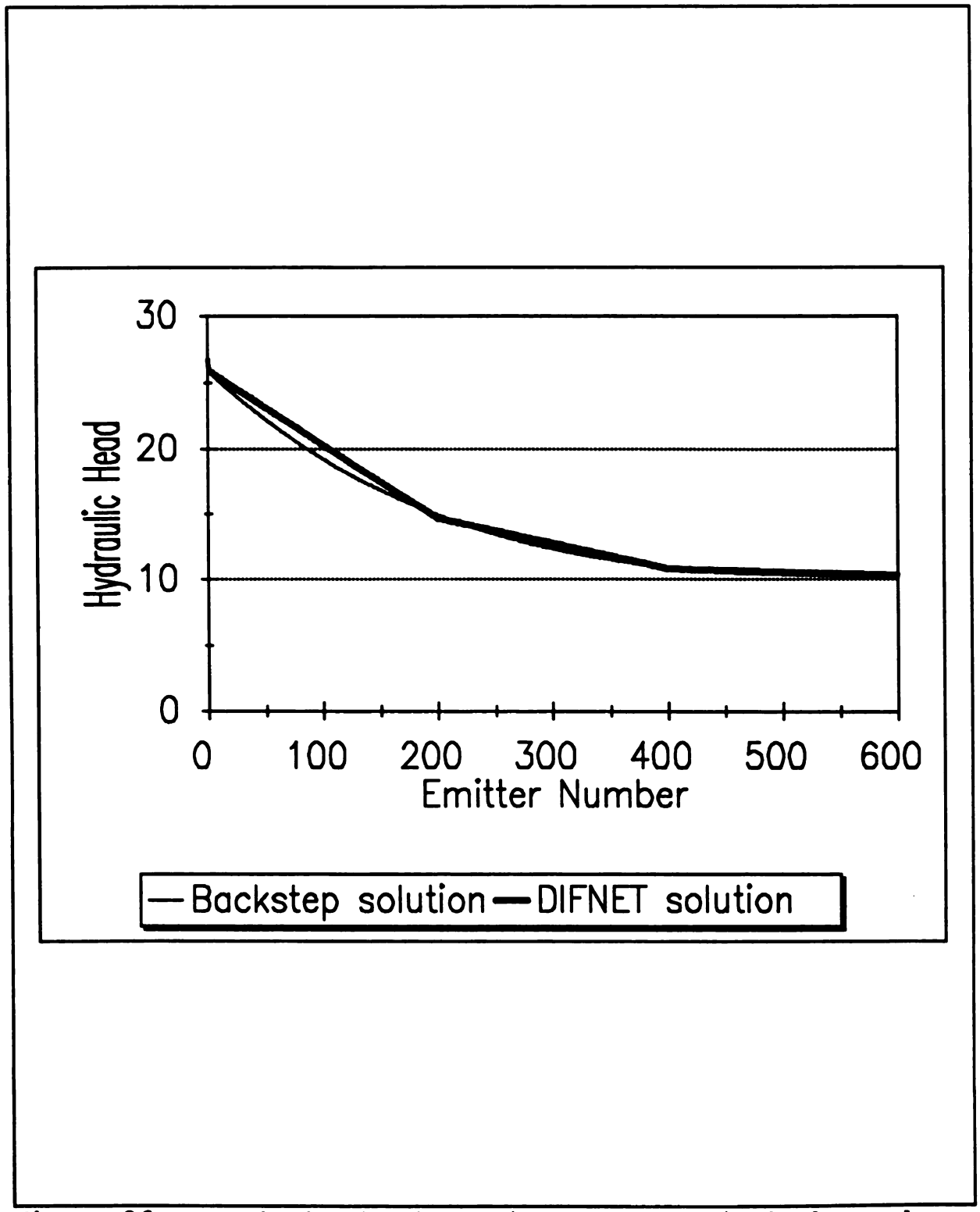


Figure 26. The head calculations along a single lateral are compared; The Backstep solution and DIFNET solution are plotted together.

using all 600 data points; R-squared = 0.998.

D. Stability

1. Methods 1 and 2

Stability is a problem with the solution derived by Method 1 (see Appendix D). This is most likely due to the fact that the energy discontinuities at component nodes are incorporated into the forcing vector. The system of equations is more sensitive to values in the forcing vector than to values in the stiffness matrix. Although stability is a problem with Method 1, the solution has eventually converged in every case tested. Method 2 proves to be the more practical of the two methods.

2. Collective emitter flow in virtual node concept

When the flow of several emitters is incorporated into a single element using the virtual node concept, their treatment as a constant, Q, in the differential equation caused instability. Again, this is because, as a constant, their effect ends up all in the forcing vector. If treated as a function of head Gh, however, instability ceases to be a problem because their effect is incorporated into the stiffness matrix. The instability caused by treating collective emitter flow as a constant, Q, was great enough to cause divergence in some cases.

3. The Newton Raphson Method

The computer program, NEWTON, was written to explore the possibilities of using the Newton-Raphson method so as to achieve faster convergence. This proved to be impractical, however, due to instability (see Appendix D).

E. Continuity Requirements

As seen in Figure 15, the intra-element function is discontinuous at node **i**. The discontinuity, however, does not prohibit solution for the following reasons. The discontinuity exists at a node. The derivative at that node is therefore *undefined*. The derivative of a linear element *without* this discontinuity, however, is *discontinuous* at both nodes **i** and **j**. The only difference, therefore, lies in the size of the jump; the jump in both cases is finitely defined as the difference in slope between the two adjacent elements. In the case of the discontinuity, however, the jump goes to negative infinity and back in the process. Still, the final result in both cases is a finite jump.

Continuity of first order, C^1, is conserved throughout an element because ΔH_e is treated as a constant and therefore does not appear in the derivative. Also, continuity of zeroth order, C^0, is conserved because head at node **j** of one element is equal to head at node **i** of the next element. So in practice, the continuity requirements for solution of a second-order partial differential equation by the Finite Element Method are met.

F. Discussion

For most practical purposes, the accuracy of the DIFNET programs should be sufficient. Where this level of accuracy is not sufficient, the laterals can be broken down into two or more segments, increasing the number of segments to increase accuracy; or, another way to increase accuracy would be to implement higher order shape functions.

G. Ideas for further investigation

The equations developed in this thesis describe flow through a pipe element in one dimension. The virtual node concept converts a series of discrete flow losses (a series of emitters) to a continuous derivative boundary condition, thereby reducing a lateral with several emitter nodes to one element.

Perhaps the virtual node concept could also be applied in two dimensions where a derivative boundary condition is applied to a *surface*. Or for that matter, why not include the third dimension to incorporate infiltration into the soil, making the model complete.

1. Some ideas for a two dimensional development

Consider, for example, a rectangular field with a submain and several laterals as shown in Figure 27. The thicker lines represent pipes whereas the thin lines represent the boundaries of the rectangular field. The larger dots represent the nodes

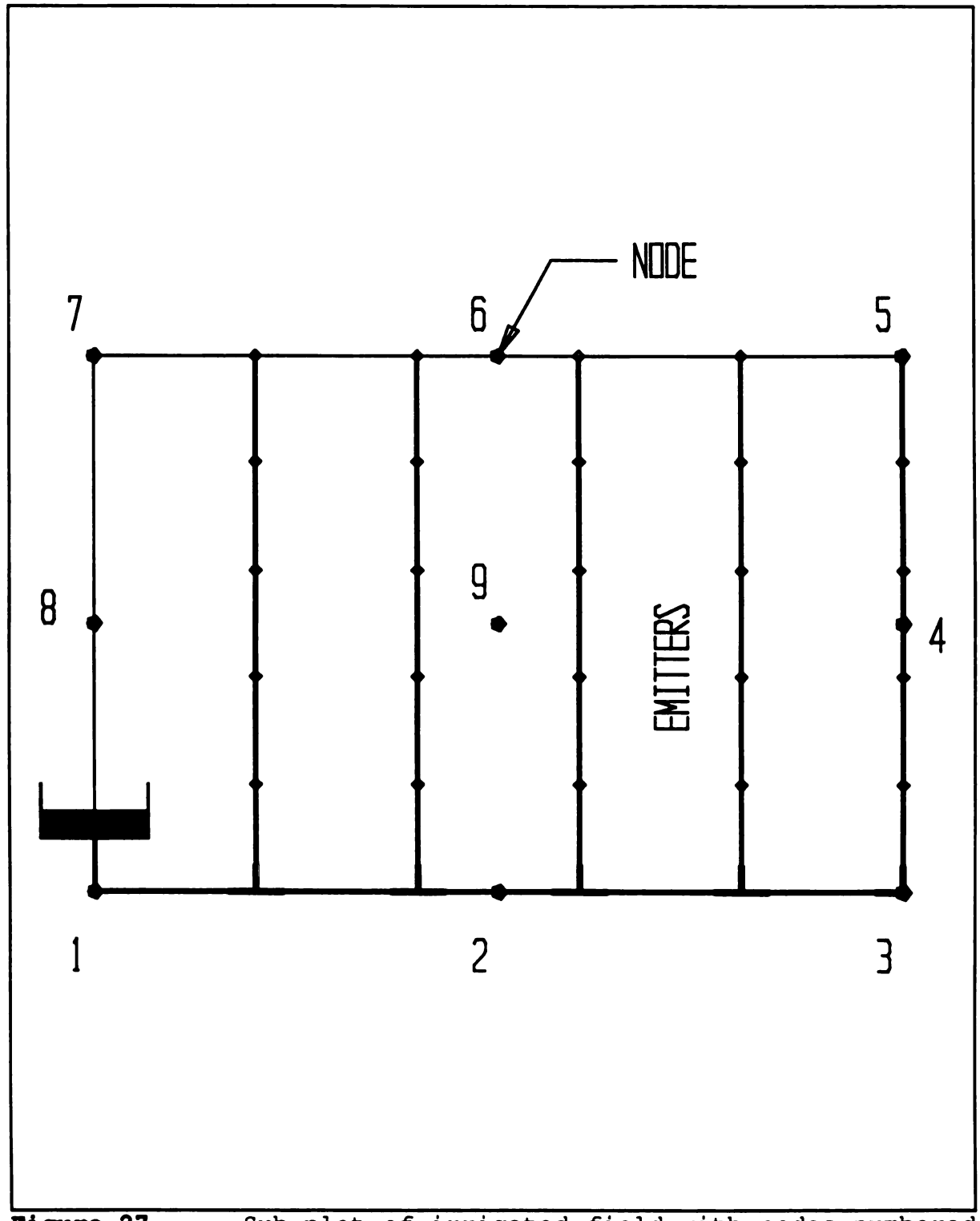


Figure 27 Sub-plot of irrigated field with nodes numbered for two-dimensional analysis using rectangular Lagrangian element.

of a two-dimensional rectangular *Lagrangian* element. The Lagrangian element has nine nodes; its polynomial is second degree. The shape functions for a rectangular Lagrangian element are listed in table IV. These shape functions are given in the *Natural Coordinate System* (see Segerlind, 1984).

Table III: Shape functions for nine-node Lagrangian element.

Node	Shape Function
1	$\frac{\xi\eta}{4}(\xi-1)(\eta-1)$
2	$-\frac{\eta}{2}(\eta-1)(\xi^2-1)$
3	$\frac{\xi\eta}{4}(\xi+1)(\eta-1)$
4	$-\frac{\xi}{2}(\xi+1)(\eta^2-1)$
5	$\frac{\xi\eta}{4}(\xi+1)(\eta+1)$
6	$-\frac{\eta}{2}(\eta+1)(\xi^2-1)$
7	$\frac{\xi\eta}{4}(\xi-1)(\eta+1)$
8	$-\frac{\xi}{2}(\xi-1)(\eta^2-1)$
9	$(\xi^2-1)(\eta^2-1)$

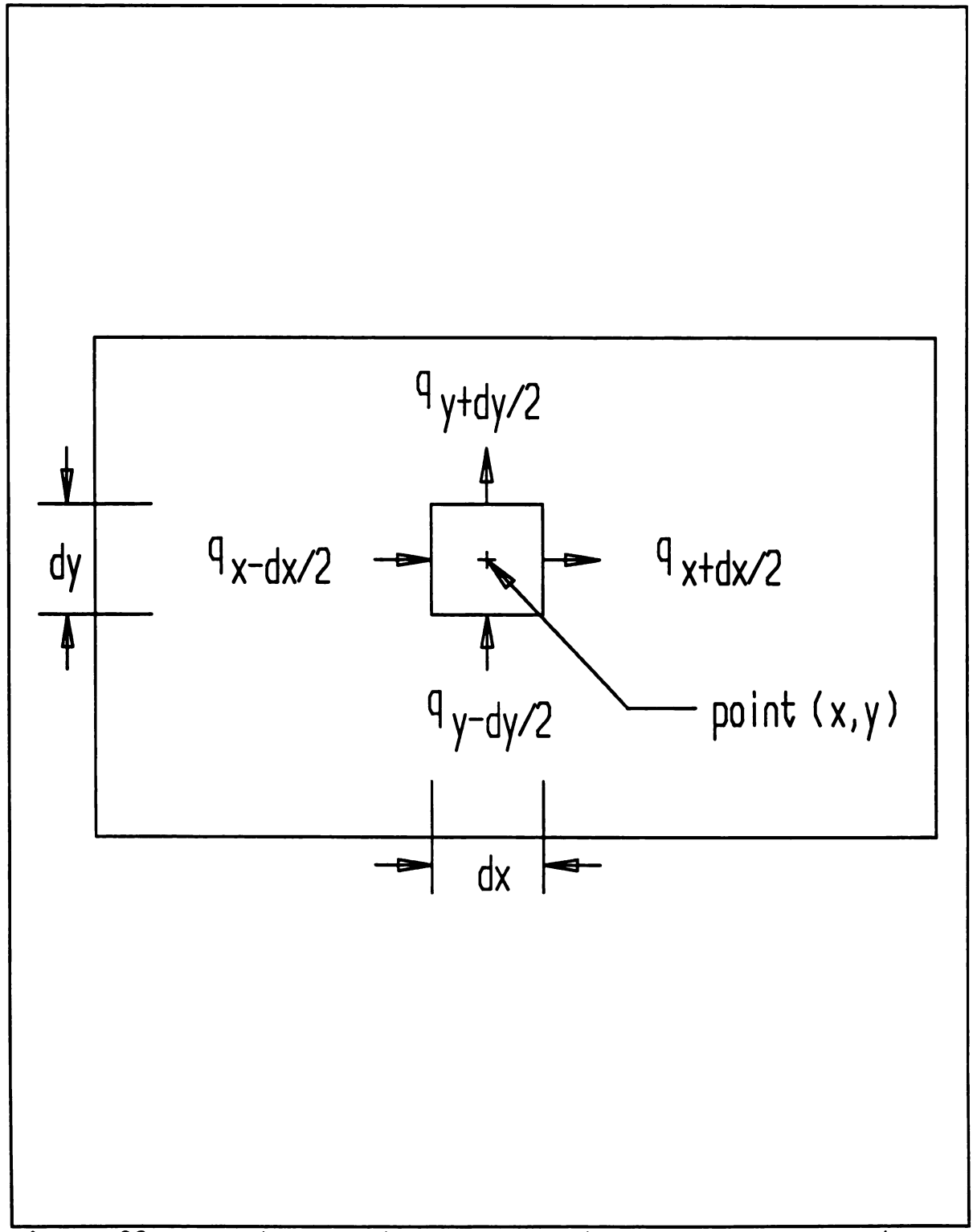


Figure 28. Differential conservation of mass as continuous function in two dimensions.

It may be possible to approximate a hydraulic head *topology* of the surface shown in Figure 27 by considering its discrete function to be continuous.

As an extension to this research, a partial differential equation was developed to describe in two dimensions this topology created by an irrigation network. This was achieved by formulating conservation of mass as a continuous function in two dimensions. Initial observations are presented in Figure 28. Conservation of mass then dictates that flow into the control "volume" equal flow out:

$$q_{x-\frac{dx}{2}} + q_{y-\frac{dy}{2}} - q_{x+\frac{dx}{2}} - q_{y+\frac{dy}{2}} - \frac{\partial q}{\partial x}dx - \frac{\partial q}{\partial y}dy = 0$$

Or, shifting the point (x,y) to the lower left corner of the control "volume",

$$q_x + q_y - q_{x+dx} - q_{y+dy} - \frac{\partial q}{\partial x}dx - \frac{\partial q}{\partial y}dy = 0 \qquad (60)$$

where,

$$q_x = D_x \frac{\partial h}{\partial x}\Big|_x \qquad (61)$$

and as before,

$$q_{x+dx} = D_x \frac{\partial h}{\partial x}\Big|_x + D_x \frac{\partial^2 h}{\partial x^2}dx \qquad (62)$$

The same applies in the y-direction. Substituting equations (61) and (62) into equation (60) gives the desired partial differential equation:

$$D_x \frac{\partial^2 h}{\partial x^2} dx + D_y \frac{\partial^2 h}{\partial y^2} dy - \frac{\partial q}{\partial x} dx - \frac{\partial q}{\partial y} dy = 0 \qquad (63)$$

where $\frac{\partial q}{\partial x} dx + \frac{\partial q}{\partial y} dy$ can be considered either a constant, Q, or a function of head, Gh:

$$\frac{\partial q}{\partial x} dx + \frac{\partial q}{\partial y} dy = Q = \frac{n_e n_l k h_g^x}{LW} \qquad (64)$$

or,

$$\frac{\partial q}{\partial x} dx + \frac{\partial q}{\partial y} dy = G h = \left(\frac{n_e n_l k h_g^{x-1}}{LW} \right) h \qquad (65)$$

where n_e = number of emitters per lateral

n_l = number of laterals

L = length of element

W = width of element

The general form of equation (63) is:

$$D_x \frac{\partial^2 h}{\partial x^2} + D_y \frac{\partial^2 h}{\partial y^2} - G h - Q = 0 \qquad (66)$$

In this form, the operators D_x and D_y resemble conductivity of heat transfer problems. Here they represent the conductivity of pipe flow in a two-dimensional grid. Their

determination is not straight-forward and is likely the key to solving this problem. Some observations are (1) the conductivity, D_y, looks like it should be the sum of the lateral conductivities divided by the length of the element[5]:

$$D_y = \frac{n_1}{dx^2} \frac{1}{a_1^{\frac{1}{m}}} \left(\frac{dh}{dy}\right)^{\left(\frac{1}{m}-1\right)} \qquad (67)$$

and (2) the conductivity, D_x, depends on the position in y. The second observation becomes obvious with a glance at Figure 29. Water at point (x,y) to get to point (x+dx,y) must first go back to the main through a lateral (distance, y), then through a piece of the main (distance, dx), and then back through a lateral (distance, y). The conductivity of this route is then calculated by adding *resistances*. The total resistance is equal to the sum of the resistances in each segment:

$$r_x = R_y + r_{x0} + R_y$$

where r_x = resistivity[6] in x-direction at any point (x,y)

r_{x0} = resistivity in x-direction at any point (x,0) (in other words, resistivity of main)

R_y = resistance in y-direction (over length y)

Or, in terms of conductivity,

---

[5]Note that one over **dx** is squared. The extra **dx** comes from the division of **dxdy** in equation (63). Likewise, the equation for D_x has one over **dy** in it.

[6]Resistivity is the inverse of conductivity; resistance is resistivity times length. Here, resistivity in the x-direction is equal to the resistivity of the main plus the *resistance* in the y-direction (a function of y).

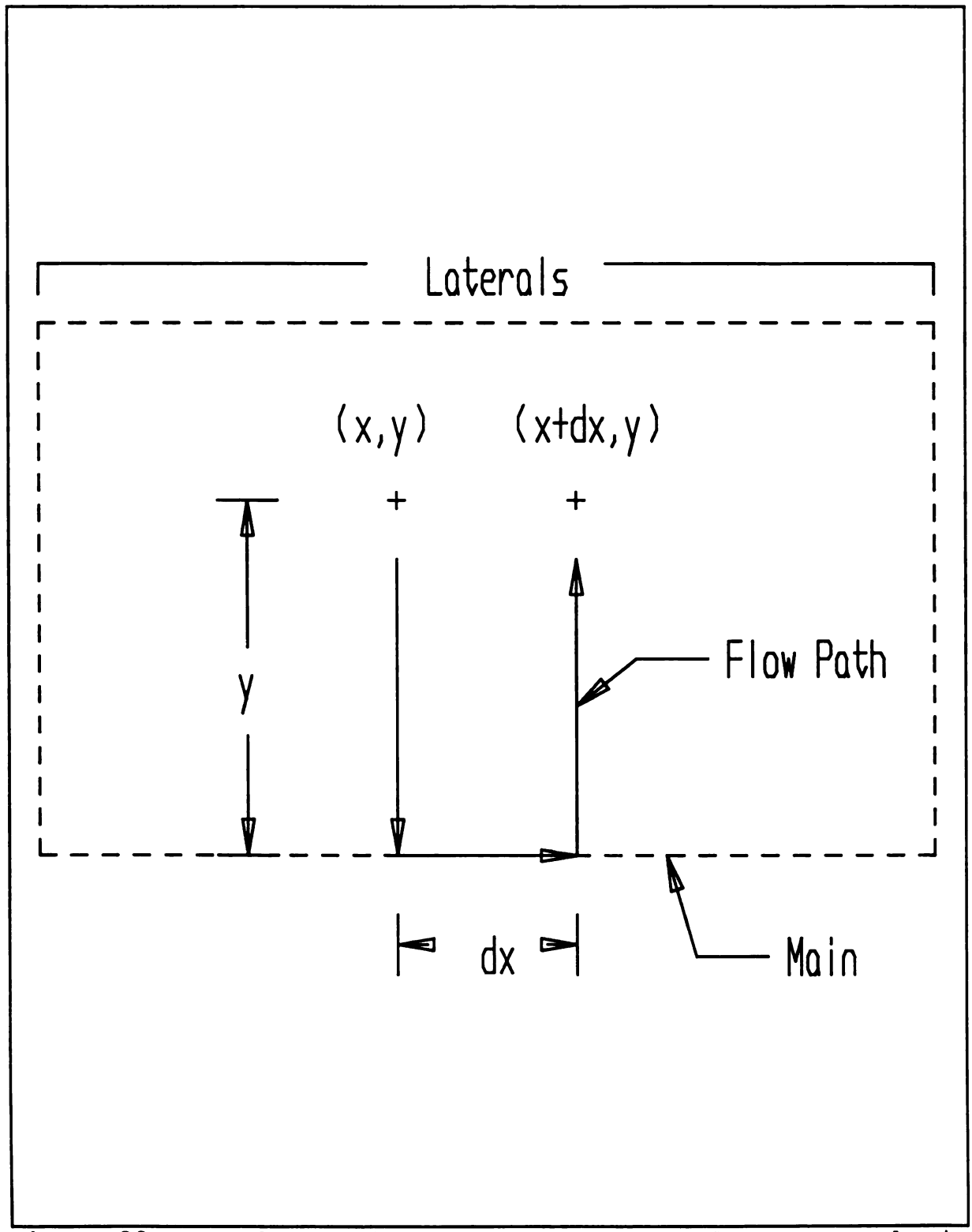


Figure 29 Flow path for water to get from **x** to **dx** in irrigated sub-plot.

$$\frac{1}{D_x} = \frac{1}{D_{x0}} + \frac{2y}{D_y}$$

Note that the conductivity, D_x, is a function of y. It is solved for as:

$$D_x = \left[\frac{1}{\frac{1}{D_{x0}} + \frac{2y}{D_y}}\right]\frac{1}{dy} \qquad (68)$$

where,

$$D_{x0} = \frac{1}{a_{x0}^{\frac{1}{m}}}\left(\frac{\partial h}{\partial x}\right)^{\left(\frac{1}{m}-1\right)}$$

where, a_{x0} = constant, a, for main

and D_y is as previously defined.

Integration of equation (66) times the shape functions is facilitated by recalling that $\phi = [N]\{\Phi\}$. So the integration,

$$\int_{x_1}^{x_2} \frac{\partial [N]^T}{\partial x}\frac{\partial \phi}{\partial x}dx$$

becomes,

$$\int_{x_1}^{x_2} \frac{\partial [N]^T}{\partial x}\frac{\partial [N]}{\partial x}dx\{\Phi^{(e)}\}$$

The integration of equation (66) looks like:

$$\int_{-1}^{+1} D_x \frac{\partial [N]^T}{\partial \xi} \frac{\partial [N]}{\partial \xi} dA \{\Phi^{(e)}\} + \int_{-1}^{+1} D_y \frac{\partial [N]^T}{\partial \eta} \frac{\partial [N]}{\partial \eta} dA \{\Phi^{(e)}\} - \int_{-1}^{+1} G [N]^T [N] dA \{\Phi^{(e)}\} - \int_{-1}^{+1} Q [N]^T dA = 0 \quad (69)$$

The fact that D_x is a function of y makes integration messy. The resulting equations, while quite messy indeed, were found to have four basic forms. This made computer coding of the 9 x 9 stiffness matrix feasible. The results of these integrations are encoded in the computer program LAGRANGE, found in Appendix A. A typical output of this program is plotted in Figure 30.

While results are known to be "in the ball park" of the correct values, time did not allow for a thorough development and evaluation of these ideas. Hence, the preliminary results achieved at this point are inconclusive. If continuization error is significant, perhaps similitude modeling techniques could be used to correct for this error.

2. Adding the third dimension

A topology of hydraulic head is easily converted to a topology of flow. Described in terms of flow, the two-dimensional analysis could be expanded to include the third dimension, modeling infiltration as well as distribution. Such a model would be based on Darcy's equation for flow through porous media:

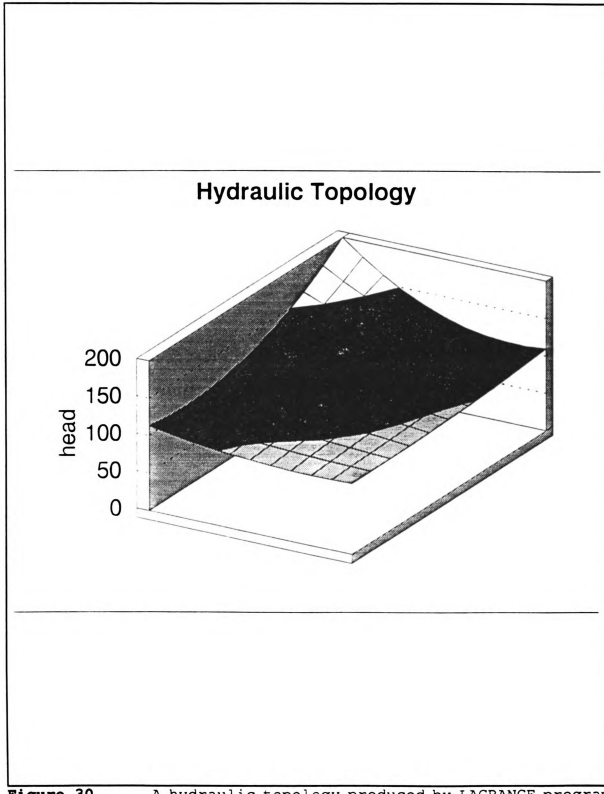


Figure 30 A hydraulic topology produced by LAGRANGE program

$$\mathbf{q} = -K \nabla H$$

where, $\mathbf{q}$ = flow

K = hydraulic conductivity

$$\Delta H = \frac{\partial H}{\partial x}\hat{i} + \frac{\partial H}{\partial y}\hat{j} + \frac{\partial H}{\partial z}\hat{k}$$

The hydraulic topology created by the irrigation system would then be applied as a boundary condition on the soil surface. The concepts used in the development of the FINDIT computer program (Kunze and Shayya, 1990) would be essential to the development of such a model. A complete three dimensional model would have many obvious benefits in the design and evaluation of an irrigation system, not to mention applications to environmental studies. Figure 31 shows the geometry of a possible three-dimensional finite element grid for modeling distribution and infiltration.

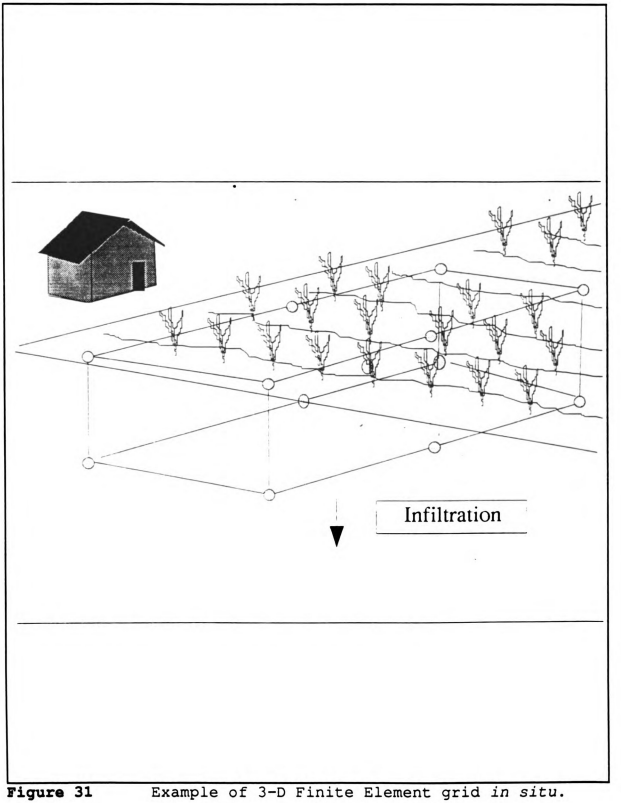


Figure 31 Example of 3-D Finite Element grid *in situ*.

V. Conclusions and Recommendations

Conclusions of this research are listed below:

1. The cumulative effect of pipe components in a hydraulic network is significant. Neglecting the effect of pipe components may introduce error large enough to misguide the design of a microirrigation system. Existing pipe-network programs which include components were found to significantly increase the number of nodes necessary for analysis.

2. A partial differential equation was developed which incorporates pipe components at nodes rather than as separate elements. Galerkin's weighted residual method was employed in the Finite Element solution of this equation. This solution was found to agree with an algebraic development of the Linear Theory method.

3. The virtual node concept was successfully incorporated in the partial differential equation to further reduce the number of nodes required for analysis.

4. The results of these developments were compared with those of existing pipe-network analysis programs, namely ANALYZER and KYPIPE. Correlation among all methods was found to be very strong. As expected, some error was

introduced by the virtual node concept. While this error was small (certainly negligible), it was found to be further reduced by partitioning the laterals (the "virtual elements") into two or more segments.

Some recommendations for further investigation follow:

1. The *Ideas for Further Investigation* section in Chapter IV outlines a possible development of a two-dimensional flow equation which describes a microirrigation system as having a continuous topology of hydraulic head. This development is supplemented by an alternative approach outlined in Appendix E. A thorough evaluation of these ideas was not performed as a part of this research and would be a logical next step. The reader is encouraged to explore and improve upon the possibilities opened up by these developments.

2. Also included in the *Ideas for Further Investigation* section is a suggestion that one might apply the two-dimensional topology of hydraulic head as a boundary condition to a three-dimensional porous-media flow equation in order to model flow of irrigated water through soil.

3. The benefits of the equations derived in this thesis will only be realized when they are incorporated into a presentable, user-friendly computer program.

Appendix

Appendix A: Computer Code

The computer language used is Quick Basic version 4.5.

ALGNET1

```
'
'* * * Program to solve for pipe network flow * * *
'
DECLARE SUB BCcalc (Matrix!(), RHSMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
DECLARE SUB Newton (T!, Hi!, Lo!, x!, k!)
DECLARE SUB ConstSort2 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
DECLARE SUB ConstSort1 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
DECLARE SUB CalcLength (x!(), y!(), z!(), ii%(), jj%(), ElLength!, i%)
DECLARE SUB Stats (NumArray!(), count%, Mean!, Median!, StanDev!, Min!, Max!)

DECLARE SUB MatSave (MatrixA!(), MatrixSize%)
DECLARE SUB AddSquare (Hi%, Lo%, value!, Matrix!(), NumNode%)
DECLARE SUB MatShow (MatrixA!(), MatrixSize%)
DECLARE SUB MatAdd (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
DECLARE SUB AddDiagonal (position%, value!, Matrix!(), NumNode%)
DECLARE FUNCTION Determ! (MatrixA!(), MatrixSize%)
DECLARE SUB DoDet (Term!, M%, k%, MatrixA!(), Done%(), ValDeterm!, MatrixSize%)

DECLARE SUB GetReply (First$, Last$, Reply$)
DECLARE SUB Key2Arr (Num2Array!(), RowCount%)
DECLARE SUB MatInv (MatrixA!(), MatrixB!(), MatrixSize%, OK AS INTEGER)
DECLARE SUB MatMult (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
DECLARE SUB Show2Arr (Num2Array!(), RowCount%, N%, M%, NumColumns%)
DECLARE SUB WaitKey ()
'
CLS
PRINT "Program to calculate heads of hydraulic network system"
PRINT
INPUT "Enter name of element data file:  ", element$
INPUT "Enter name of nodal coordinate data file:  ", node$
INPUT "Enter name of boundary conditions file (<ENTER> if none):  ", bc$
IF bc$ <> "" THEN
        INPUT "Enter number of boundary conditions:  ", NumBC%
        DIM knownval!(NumBC%), BCnode%(NumBC%)
END IF
INPUT "Enter value for initial head, H(0) (m):  ", H0!

'* * * start timer * * *
StartTime! = TIMER
'* * * * * * * * * * * *
'
'* * * read data files * * *
'
OPEN element$ FOR INPUT AS #1
    INPUT #1, NumElem%
    i% = NumElem%
    DIM elem%(i%), ii%(i%), jj%(i%), dia!(i%), HW%(i%), idia!(i%), jdia!(i%)
        FOR i% = 0 TO NumElem%
                INPUT #1, elem%(i%), ii%(i%), jj%(i%), dia!(i%), HW%(i%), idia!(i%), jdia!(i%)
        NEXT i%
CLOSE (1)
OPEN node$ FOR INPUT AS #1
    INPUT #1, NumNode%
    i% = NumNode%
    DIM node%(i%), x!(i%), y!(i%), z!(i%), ctype%(i%), Cq180!(i%), Cq90!(i%)
        FOR i% = 0 TO NumNode%
                INPUT #1, node%(i%), x!(i%), y!(i%), z!(i%), ctype%(i%), Cq180!(i%), Cq90!(i%)
        NEXT i%
CLOSE (1)
IF bc$ <> "" THEN
        OPEN bc$ FOR INPUT AS #1
                FOR i% = 1 TO NumBC%
                        INPUT #1, BCnode%(i%), knownval!(i%)
                NEXT i%
        CLOSE (1)
END IF
'
'* * * dimension arrays * * *
'
DIM k!(NumElem% + 1), ke!(NumNode%), kstar!(NumElem% + 1), H!(NumNode%)
DIM Matrix!(NumNode%, NumNode%), RHSMatrix!(NumNode%, NumNode%)
DIM InvertedMatrix!(NumNode%, NumNode%), AnswerMatrix!(NumNode%, NumNode%)
DIM NumArray!(NumNode%)
```

```
DIM Q!(NumElem% + 1)
DIM flow%(NumElem%), CompTerm!(NumElem%)
DIM OK AS INTEGER
'
counter% = 1
'
'* * * constants * * *
'
H!(0) = H0!
g! = 32.2                          'acc. due to gravity ft/s^2
pi! = 3.1415926540
NumIter% = 2                       'Number of iterations before including effect of components
HWconst! = 4.73
'
'* * * calculate constant, k * * *
'
FOR i% = 0 TO NumElem%
        CALL CalcLength(x!(), y!(), z!(), ii%(), jj%(), ElLength!, i%)
        k!(i%) = HWconst! * ElLength! / (HW%(i%) ^ 1.852 * dia!(i%) ^ 4.87)
NEXT i%
'
'* * * initialize head values * * *
'
FOR i% = 1 TO NumNode%
        H!(i%) = H!(0) - i%
NEXT i%
'
LOCATE 10, 30: COLOR 31, 0: PRINT "-- Converging --": COLOR 7, 0
'
'* * * Start iterative procedure * * *
'
Again:
'
'* * * First few iterations do not include effect of components * * *
'
IF counter% <= NumIter% THEN
        FOR i% = 0 TO NumElem%
                IF H!(ii%(i%)) - H!(jj%(i%)) = 0 THEN
                        kstar!(i%) = 0
                ELSE
                        kstar!(i%) = (ABS(H!(ii%(i%)) - H!(jj%(i%)))) ^ -(1 - 1 / 1.852) * k!(i%) ^ -(1 / 1.852)
                END IF
        NEXT i%
        GOTO skip1
END IF
'
'* * * reset values * * *
'
FOR i% = 0 TO NumElem%
        flow%(i%) = 0
        CompTerm!(i%) = 0
NEXT i%
'
'* * * Calculate new values for component head-loss terms * * *
'
FOR i% = 0 TO NumNode%
        IF ctype%(i%) = 2 THEN
                FOR j% = 0 TO NumElem%
                        IF ii%(j%) = i% AND H!(ii%(j%)) - H!(jj%(j%)) > 0 THEN
                                flow%(j%) = 1    ' 1 - positive flow
                                CALL ConstSort1(ii%(), jj%(), x!(), y!(), j%, Cq90!(), Cq180!(), Cq!, NumElem%)
                                T! = Cq! * 8 / (g! * pi! ^ 2 * idia!(j%) ^ 4)
                                CompTerm!(j%) = T! * (H!(ii%(j%)) - H!(jj%(j%))) / (k!(j%) + T!)
                                CALL Newton(T!, H!(ii%(j%)), H!(jj%(j%)), CompTerm!(j%), k!(j%))
                                'PRINT "Left:   "; CompTerm!(j%)
                                'PRINT "Right:  "; T! * ((H!(ii%(j%)) - H!(jj%(j%)) - CompTerm!(j%)) / k!(i%)) ^ 1.08
                        ELSEIF jj%(j%) = i% AND H!(jj%(j%)) - H!(ii%(j%)) > 0 THEN
                                flow%(j%) = -1   ' -1 - negative flow
                                CALL ConstSort2(ii%(), jj%(), x!(), y!(), j%, Cq90!(), Cq180!(), Cq!, NumElem%)
                                T! = Cq! * 8 / (g! * pi! ^ 2 * jdia!(j%) ^ 4)
                                CompTerm!(j%) = (T! * (H!(jj%(j%)) - H!(ii%(j%))) / k!(j%)) / (1 + T! / k!(j%))
                                CALL Newton(T!, H!(jj%(j%)), H!(ii%(j%)), CompTerm!(j%), k!(j%))
                        END IF
                NEXT j%
        END IF
NEXT i%
'
'* * * calculate new kstar's * * *
'
FOR i% = 0 TO NumElem%
        IF H!(ii%(i%)) - H!(jj%(i%)) = 0 THEN
                kstar!(i%) = 0
        ELSEIF flow%(i%) = 1 THEN
                kstar!(i%) = (H!(ii%(i%)) - H!(jj%(i%)) - CompTerm!(i%)) ^ -(1 - 1 / 1.852) * k!(i%) ^ -(1 / 1.852)
        ELSEIF flow%(i%) = -1 THEN
                kstar!(i%) = (H!(jj%(i%)) - H!(ii%(i%)) - CompTerm!(i%)) ^ -(1 - 1 / 1.852) * k!(i%) ^ -(1 / 1.852)
        ELSE
                kstar!(i%) = (ABS(H!(ii%(i%)) - H!(jj%(i%)))) ^ -(1 - 1 / 1.852) * k!(i%) ^ -(1 / 1.852)
        END IF
NEXT i%
'
skip1:
'
'* * * Calculate new values for emitter constants * * *
'
FOR i% = 1 TO NumNode%
        IF ctype%(i%) = 1 THEN
                k! = Cq180!(i%)
                x! = Cq90!(i%)
                ke!(i%) = k! * (H!(i%) - z!(i%)) ^ x! / H!(i%)
        END IF
NEXT i%
'
'* * * Set Up Global Stiffness Matrix * * *
'
```

```
FOR i% = 1 TO NumNode%
        FOR j% = 1 TO NumNode%
                Matrix!(i%, j%) = 0
        NEXT j%
NEXT i%
'****************************************
'*         Add element matrices         *
'****************************************
'
FOR i% = 1 TO NumElem%
        value! = kstar!(i%)
        Hi% = jj%(i%)
        Lo% = ii%(i%)
        CALL AddSquare(Hi%, Lo%, value!, Matrix!(), NumNode%)
NEXT i%
'
'************************************************
'* Add emitter constants to diagonal of matrix *
'************************************************
'
FOR i% = 1 TO NumNode%
        IF ctype%(i%) = 1 THEN
                value! = -1 * ke!(i%)
                CALL AddDiagonal(i%, value!, Matrix!(), NumNode%)
        END IF
NEXT i%
'
'*** Add kmain!(0) to position (1,1) ***
'
value! = -1 * kstar!(0)
CALL AddDiagonal(1, value!, Matrix!(), NumNode%)
'
'* * * set up forcing vector as matrix * * *
'
FOR j% = 1 TO NumNode%
   FOR k% = 1 TO NumNode%
      RHSMatrix!(j%, k%) = 0!
   NEXT k%
NEXT j%
'
'* * * include boundary conditions * * *
'
IF bc$ <> "" THEN
        CALL BCcalc(Matrix!(), RHSMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
END IF
'
'* * * first time thru, don't include effect of components * * *
'
IF counter% < NumIter% THEN GOTO skip2
'
'* * * effect of components in forcing vector * * *
'
FOR j% = 0 TO NumElem%
        IF flow%(j%) = 1 THEN
                RHSMatrix!(ii%(j%), 1) = RHSMatrix!(ii%(j%), 1) - CompTerm!(j%) * kstar!(j%)
                RHSMatrix!(jj%(j%), 1) = RHSMatrix!(jj%(j%), 1) + CompTerm!(j%) * kstar!(j%)
        ELSEIF flow%(j%) = -1 THEN
                RHSMatrix!(jj%(j%), 1) = RHSMatrix!(jj%(j%), 1) - CompTerm!(j%) * kstar!(j%)
                RHSMatrix!(ii%(j%), 1) = RHSMatrix!(ii%(j%), 1) + CompTerm!(j%) * kstar!(j%)
        END IF
NEXT j%
'
skip2:
'
'* * * add initial head Boundary Condition to forcing vector * * *
'
RHSMatrix!(1, 1) = RHSMatrix!(1, 1) - kstar!(0) * H!(0)
'
'*******************************
'* solve simultaneous equations *
'*******************************
'
CALL MatInv(Matrix!(), InvertedMatrix!(), NumNode%, OK)
IF NOT OK THEN
   BEEP
   PRINT "Bad input, no solution is possible."
   END
END IF
CALL MatMult(InvertedMatrix!(), RHSMatrix!(), AnswerMatrix!(), NumNode%)
'* * * * * * *
cnt% = 0
FOR i% = 1 TO NumNode%
        IF ABS(AnswerMatrix!(i%, 1) - H!(i%)) > .01 THEN
                GOTO OtraVez
        END IF
NEXT i%
'* * * stop timer * * *
TotalTime! = TIMER - StartTime!
'* * * * * * * * * * * *
FOR i% = 1 TO NumNode%
        H!(i%) = AnswerMatrix!(i%, 1)
        NumArray!(i%) = H!(i%)
NEXT i%
'
'* * * calculate coefficient of uniformity * * *
'
CALL Stats(NumArray!(), NumNode%, Mean!, Median!, StanDev!, Min!, Max!)
U! = 100 * (1 - StanDev! / Mean!)
'
'* * * print results * * *
'
CLS : LOCATE 8, 1
i% = 0
PRINT "Head at Node ("; i%; "): "; H!(i%)
FOR i% = 1 TO NumNode%
```

```
        PRINT "Head at Node ("; i%; "): "; H!(i%)
NEXT i%
PRINT
PRINT "Coefficient of Uniformity = "; U!; "%"
PRINT "Converged after "; counter%; " iterations."
PRINT "Total time of convergence = "; TotalTime!; " seconds"
PRINT
'
'* * * save data in DIFNET format for comparison                * * *
'* * * (save only those points which correspond to DIFNET output * * *
'

biggesty! = 0
FOR i% = 0 TO NumNode%
    IF y!(i%) > biggesty! THEN
        biggesty! = y!(i%)
    END IF
NEXT i%

outfile$ = "a:an1_" + MID$(node$, 7, 1) + ".out"
OPEN outfile$ FOR OUTPUT AS #1
FOR i% = 0 TO NumNode%
    IF y!(i%) = 0 THEN
        PRINT #1, H!(i%)
    ELSEIF y!(i%) = biggesty! THEN
        PRINT #1, H!(i%)
    END IF
NEXT i%
CLOSE (1)
'
'* * * save all data points ?? * * *
'
INPUT "Save results ? (y/n) ", ans$
IF UCASE$(ans$) = "Y" THEN
INPUT "Enter name of output file:  ", filename$
OPEN filename$ FOR OUTPUT AS #1
i% = 0
PRINT #1, "Head at Node ("; i%; "): "; H!(i%)
PRINT #1,
FOR i% = 1 TO NumNode%
        PRINT #1, "Head at Node ("; i%; "): "; H!(i%)
NEXT i%
PRINT #1,
PRINT #1, "Coefficient of Uniformity = "; U!; "%"
PRINT #1, "Converged after "; counter%; " iterations."
PRINT #1, "Total time of convergence = "; TotalTime!; " seconds"
END IF
END

OtraVez:
FOR i% = 1 TO NumNode%
        H!(i%) = AnswerMatrix!(i%, 1)
NEXT i%
LOCATE 15, 20: PRINT "Number of Iterations:  "; counter%
counter% = counter% + 1
GOTO Again

SUB AddDiagonal (position%, value!, Matrix!(), NumNode%)
Matrix!(position%, position%) = Matrix!(position%, position%) + value!
END SUB

SUB AddSquare (Hi%, Lo%, value!, Matrix!(), NumNode%)
'
'* * * adds square element matrix to global stiffness matrix * * *
'
Matrix!(Hi%, Lo%) = Matrix!(Hi%, Lo%) + value!
Matrix!(Lo%, Hi%) = Matrix!(Lo%, Hi%) + value!
Matrix!(Lo%, Lo%) = Matrix!(Lo%, Lo%) - value!
Matrix!(Hi%, Hi%) = Matrix!(Hi%, Hi%) - value!
END SUB

SUB BCcalc (Matrix!(), RHSMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
'
'* * * include known values in matrices * * *
'
FOR i% = 1 TO NumBC%
        FOR j% = 1 TO NumNode%
                IF j% <> BCnode%(i%) THEN
                        RHSMatrix!(j%, 1) = RHSMatrix!(j%, 1) - Matrix!(j%, BCnode%(i%)) * knownval!(i%)
                        Matrix!(j%, BCnode%(i%)) = 0
                        Matrix!(BCnode%(i%), j%) = 0
                ELSE
                        RHSMatrix!(j%, 1) = Matrix!(j%, BCnode%(i%)) * knownval!(i%)
                END IF
        NEXT j%
NEXT i%
END SUB

SUB CalcLength (x!(), y!(), z!(), ii%(), jj%(), ElLength!, i%)
'
'* * * calculate length of element * * *
'
        xtemp! = x!(jj%(i%)) - x!(ii%(i%))
        ytemp! = y!(jj%(i%)) - y!(ii%(i%))
        ztemp! = z!(jj%(i%)) - z!(ii%(i%))
        ElLength! = (xtemp! ^ 2 + ytemp! ^ 2 + ztemp! ^ 2) ^ .5
END SUB

SUB ConstSort1 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
'
'* * * sort out which constant applies * * *
'
FOR j% = 0 TO NumElem%
        IF ii%(i%) = jj%(j%) THEN
                IF (x!(ii%(j%)) = x!(jj%(i%))) OR (y!(ii%(j%)) = y!(jj%(i%))) THEN
```

```
                        Cq! = Cq180!(ii%(i%))
                        EXIT SUB        ' 180 - degree constants have preference
                ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
                        Cq! = Cq90!(ii%(i%))
                ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) /
(x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
                        Cq! = Cq180!(ii%(i%))
                ELSE
                        Cq! = Cq90!(ii%(i%))
                END IF
        ELSEIF ii%(i%) = ii%(j%) AND jj%(i%) <> jj%(j%) THEN
                IF (x!(jj%(j%)) = x!(jj%(i%))) OR (y!(jj%(j%)) = y!(jj%(i%))) THEN
                        Cq! = Cq180!(ii%(i%))
                        EXIT SUB        ' 180 - degree constants have preference
                ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
                        Cq! = Cq90!(ii%(i%))
                ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) /
(x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
                        Cq! = Cq180!(ii%(i%))
                ELSE
                        Cq! = Cq90!(ii%(i%))
                END IF
        END IF
NEXT j%
END SUB

SUB ConstSort2 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
'
'* * * sort out which constant applies * * *
'
FOR j% = 0 TO NumElem%
        IF jj%(i%) = ii%(j%) THEN
                IF x!(ii%(i%)) = x!(jj%(j%)) OR y!(ii%(i%)) = y!(jj%(j%)) THEN
                        Cq! = Cq180!(jj%(i%))
                        EXIT SUB
                ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
                        Cq! = Cq90!(jj%(i%))
                ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) /
(x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
                        Cq! = Cq180!(jj%(i%))
                ELSE
                        Cq! = Cq90!(jj%(i%))
                END IF
        ELSEIF jj%(i%) = jj%(j%) AND ii%(i%) <> ii%(j%) THEN
                IF x!(ii%(i%)) = x!(ii%(j%)) OR y!(ii%(i%)) = y!(ii%(j%)) THEN
                        Cq! = Cq180!(jj%(i%))
                        EXIT SUB
                ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
                        Cq! = Cq90!(jj%(i%))
                ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) /
(x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
                        Cq! = Cq180!(jj%(i%))
                ELSE
                        Cq! = Cq90!(jj%(i%))
                END IF
        END IF
NEXT j%

END SUB

FUNCTION Determ! (MatrixA(), MatrixSize%)
'
'* * * evaluate determinant of matrix * * *
'
   CONST False = 0
   CONST True = NOT False
   DIM Done%(MatrixSize%)
   FOR j% = 1 TO MatrixSize%
      Done%(j%) = False
   NEXT j%
   ValDeterm! = 0!
   CALL DoDet(1!, 0, 1, MatrixA!(), Done%(), ValDeterm!, MatrixSize%)
   Determ! = ValDeterm!
END FUNCTION

SUB DoDet (Term!, M%, k%, MatrixA!(), Done%(), ValDeterm!, MatrixSize%)
'
'* * * evaluate determinant of matrix * * *
'
   CONST False = 0
   CONST True = NOT False
   IF k% > MatrixSize% THEN
      Sign% = -1
      IF (M% MOD 2) = 0 THEN Sign% = 1
      ValDeterm! = ValDeterm! + Sign% * Term!
   ELSE
      IF Term! <> 0! THEN
         N% = 0
         FOR j% = MatrixSize% TO 1 STEP -1
            IF Done%(j%) THEN
               N% = N% + 1
            ELSE
               Done%(j%) = True
               A1! = Term! * MatrixA!(k%, j%)
               A2% = M% + N%
               A3% = k% + 1
               A4% = MatrixSize%
               CALL DoDet(A1!, A2%, A3%, MatrixA!(), Done%(), ValDeterm!, A4%)
               Done%(j%) = False
            END IF
         NEXT j%
      END IF
   END IF
END SUB
```

```
SUB GetReply (First$, Last$, Reply$)
   Lo$ = LEFT$(First$, 1)
   Hi$ = LEFT$(Last$, 1)
   IF Lo$ > Hi$ THEN SWAP Lo$, Hi$
   PRINT USING "Enter reply from & to &"; Lo$; Hi$
   DO
      Reply$ = INKEY$
   LOOP UNTIL (Reply$ >= Lo$) AND (Reply$ <= Hi$)
END SUB

SUB Key2Arr (Num2Array!(), RowCount%)
   CONST EndNumber! = 10101                                    'Mod. #1
   RowSize% = UBOUND(Num2Array!, 1)
   ColSize% = UBOUND(Num2Array!, 2)
   ' PRINT "--- Enter data for 2-dimensional array. ---"       'Mod. #2
   ' PRINT "--- Maximum array size ="; RowSize%; " rows. --"   'Mod. #2
   ' PRINT "--- There are "; ColSize%; " columns per row. ---" 'Mod. #2
   ' PRINT "--- Enter "; EndNumber!; " to end data entry. ---" 'Mod. #2
   IF RowCount% >= RowSize% THEN
      PRINT CHR$(7)
      PRINT "--- RowCount% too large. ---"
      EXIT SUB
   END IF
   DO WHILE RowCount% < RowSize%
      RowCount% = RowCount% + 1
      PRINT "<<< Row number"; RowCount%; ">>>"
      IF RowCount% = RowSize% THEN PRINT "*** Last row ***"
      ColCount% = 0
      DO WHILE ColCount% < ColSize%
         ColCount% = ColCount% + 1
         PRINT "   Entry ("; RowCount%; ","; ColCount%; "):";
         INPUT " ", Entry!
         IF Entry! = EndNumber! THEN
            IF ColCount% = 1 THEN
               EXIT DO
            ELSE
               PRINT CHR$(7);
               PRINT "*** Cannot end now.  Please reenter number. ***"
               ColCount% = ColCount% - 1
            END IF
         ELSE
            Num2Array!(RowCount%, ColCount%) = Entry!
         END IF
      LOOP
      IF Entry! = EndNumber! THEN
         RowCount% = RowCount% - 1
         EXIT DO
      END IF
   LOOP
   PRINT "---"; RowCount%; "rows entered. ---"
   PRINT "--- Data entry complete ---"
END SUB

SUB MatAdd (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
'
'* * * matrix addition subroutine * * *
'
   FOR i% = 1 TO MatrixSize%
      FOR j% = 1 TO MatrixSize%
         MatrixC!(i%, j%) = MatrixA!(i%, j%) + MatrixB!(i%, j%)
      NEXT j%
   NEXT i%
END SUB

SUB MatInv (MatrixA!(), MatrixB!(), MatrixSize%, OK AS INTEGER)
'
'* * * matrix inversion subroutine * * *
'
   CONST ErrorBound! = .0000000010                             'Mod. #1
   CONST False = 0
   CONST True = NOT False
   DIM MatrixC!(MatrixSize%, MatrixSize%)

   FOR i% = 1 TO MatrixSize%
      FOR j% = 1 TO MatrixSize%
         MatrixC!(i%, j%) = MatrixA!(i%, j%)
         IF i% = j% THEN
            MatrixB!(i%, j%) = 1!
         ELSE
            MatrixB!(i%, j%) = 0!
         END IF
      NEXT j%
   NEXT i%
   FOR j% = 1 TO MatrixSize%
      i% = j%
      WHILE ABS(MatrixC!(i%, j%)) < ErrorBound!
         IF i% = MatrixSize% THEN
            OK = False
            EXIT SUB
         END IF
         i% = i% + 1
      WEND
      FOR k% = 1 TO MatrixSize%
         SWAP MatrixC!(i%, k%), MatrixC!(j%, k%)
         SWAP MatrixB!(i%, k%), MatrixB!(j%, k%)
      NEXT k%
      Factor! = 1! / MatrixC!(j%, j%)
      FOR k% = 1 TO MatrixSize%
         MatrixC!(j%, k%) = Factor! * MatrixC!(j%, k%)
         MatrixB!(j%, k%) = Factor! * MatrixB!(j%, k%)
      NEXT k%
      FOR M% = 1 TO MatrixSize%
         IF M% <> j% THEN
            Factor! = -MatrixC!(M%, j%)
            FOR k% = 1 TO MatrixSize%
```

```
            MatrixC!(M%, k%) = MatrixC!(M%, k%) + Factor! * MatrixC!(j%, k%)
            MatrixB!(M%, k%) = MatrixB!(M%, k%) + Factor! * MatrixB!(j%, k%)
         NEXT k%
      END IF
   NEXT M%
NEXT j%
OK = True
END SUB

SUB MatMult (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
'
'* * * matrix multiplication subroutine * * *
'
   FOR i% = 1 TO MatrixSize%
      FOR j% = 1 TO MatrixSize%
         TempSum! = 0!
         FOR k% = 1 TO MatrixSize%
            TempSum! = TempSum! + MatrixA!(i%, k%) * MatrixB!(k%, j%)
         NEXT k%
         MatrixC!(i%, j%) = TempSum!
      NEXT j%
   NEXT i%
END SUB

SUB MatSave (MatrixA!(), MatrixSize%)
OPEN "a:matrix.dat" FOR OUTPUT AS #1
   MatFormat$ = " ##.#^^^^"                                    'Mod. #1
   FOR i% = 1 TO MatrixSize%
      FOR j% = 1 TO MatrixSize%
         PRINT #1, USING MatFormat$; MatrixA!(i%, j%);
      NEXT j%
      PRINT #1,
   NEXT i%
   PRINT #1,
CLOSE (1)
END SUB

SUB MatShow (MatrixA!(), MatrixSize%)
   MatFormat$ = " ##.#^^^^"                                    'Mod. #1
   FOR i% = 1 TO MatrixSize%
      FOR j% = 1 TO MatrixSize%
         PRINT USING MatFormat$; MatrixA!(i%, j%);
      NEXT j%
      PRINT
   NEXT i%
   PRINT
END SUB

SUB Newton (T!, Hi!, Lo!, x!, k!)
'
'* * * solve for deltah (x! here) using Newton-Raphson method * * *
'
DO
        F! = T! * ((Hi! - Lo! - x!) / k!) ^ (2 / 1.852) - x!
        dF! = 1.08 * T! * ((Hi! - Lo! - x!) / k!) ^ (2 / 1.852 - 1) * (-1 / k!) - 1
        newx! = x! - F! / dF!
        x! = newx!
LOOP UNTIL F! / dF! < .01 * x!
END SUB

SUB Show2Arr (Num2Array!(), RowCount%, N%, M%, NumColumns%)
   NumRows% = 20                                                'Mod. #1
   IF (RowCount% < 1) OR (N% < 0) OR (M% < 0) OR (NumColumns% < 1) THEN
      BEEP
      PRINT "--- Parameter error in Show2Arr ---"
      EXIT SUB
   END IF
   ColSize% = UBOUND(Num2Array!, 2)
   LastElement% = RowCount% * ColSize%
   PageSize% = NumRows% * NumColumns%
   ElementCount% = 0
   NumOnPage% = 0
   IndexFormat$ = "(&,&)"
   IF N% = 0 THEN
      NumFormat$ = " ##." + STRING$(M%, "#") + "^^^^"
   ELSE
      NumFormat$ = STRING$(N%, "#") + "." + STRING$(M%, "#")
   END IF
   IF N% = 0 THEN
      ColWidth% = 27
   ELSE
      ColWidth% = N% + M% + 10                                  'Mod. #2
   END IF
   CLS
   FOR j% = 1 TO RowCount%
      StrJ$ = RIGHT$(STR$(j%), LEN(STR$(j%)) - 1)
      FOR k% = 1 TO ColSize%
         StrK$ = RIGHT$(STR$(k%), LEN(STR$(k%)) - 1)
         RowLoc% = (NumOnPage% \ NumColumns%) + 1
         ColLoc% = (NumOnPage% MOD NumColumns%) * ColWidth% + 1
         LOCATE RowLoc%, ColLoc%
         PRINT USING IndexFormat$; StrJ$; StrK$;
         PRINT USING NumFormat$; Num2Array!(j%, k%)             'Mod. #2
         ElementCount% = ElementCount% + 1
         NumOnPage% = ElementCount% MOD PageSize%
         IF (NumOnPage% = 0) OR (ElementCount% = LastElement%) THEN
            PRINT "-- Press a key to continue --"
            DO
            LOOP UNTIL INKEY$ <> ""
            IF ElementCount% <> LastElement% THEN
               CLS
            ELSE
               PRINT
            END IF
         END IF
```

```
      NEXT k%
   NEXT j%
END SUB

SUB Stats (NumArray!(), count%, Mean!, Median!, StanDev!, Min!, Max!)
   IF count% < 1 THEN EXIT SUB
   FOR j% = 2 TO count%
      Temp! = NumArray!(j%)
      k% = j% - 1
      DO WHILE ((Temp! < NumArray!(k%)) AND (k% > 0))
         NumArray!(k% + 1) = NumArray!(k%)
         k% = k% - 1
      LOOP
      NumArray!(k% + 1) = Temp!
   NEXT j%
   FOR j% = 1 TO count%
      ValueSum! = ValueSum! + NumArray!(j%)
      SquareSum! = SquareSum! + NumArray!(j%) ^ 2
   NEXT j%
   Min! = NumArray!(1)
   Max! = NumArray!(count%)
   IF ((count% + 1) \ 2) = count% \ 2 THEN
      Mid% = count% \ 2
      Median! = (NumArray!(Mid%) + NumArray!(Mid% + 1)) / 2!
   ELSE
      Median! = NumArray!((count% + 1) \ 2)
   END IF
   Mean! = ValueSum! / count%
   IF count% = 1 THEN
      StanDev! = 0!
   ELSE
      StanDev! = SQR((SquareSum! - count% * Mean! * Mean!) / (count% - 1))
   END IF
END SUB

SUB WaitKey
   PRINT                                                    'Mod. #1
   PRINT "--- PRESS ANY KEY TO CONTINUE ---"
   DO
   LOOP WHILE INKEY$ = ""
END SUB
```

ALGNET2

```
'
'* * * Program to solve for flow in pipe networks * * *
'
DECLARE SUB BCcalc (Matrix!(), RHSMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
DECLARE SUB Newton (T!, Hi!, Lo!, x!, k!)
DECLARE SUB ConstSort2 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
DECLARE SUB ConstSort1 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
DECLARE SUB CalcLength (x!(), y!(), z!(), ii%(), jj%(), ElLength!, i%)
DECLARE SUB Stats (NumArray!(), count%, Mean!, Median!, StanDev!, Min!, Max!)

DECLARE SUB MatSave (MatrixA!(), MatrixSize%)
DECLARE SUB AddSquare (Hi!%, Lo%, value!, Matrix!(), NumNode%)
DECLARE SUB MatShow (MatrixA!(), MatrixSize%)
DECLARE SUB MatAdd (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
DECLARE SUB AddDiagonal (position%, value!, Matrix!(), NumNode%)
DECLARE FUNCTION Determ! (MatrixA!(), MatrixSize%)
DECLARE SUB DoDet (Term!, M%, k%, MatrixA!(), Done%(), ValDeterm!, MatrixSize%)

DECLARE SUB GetReply (First$, Last$, Reply$)
DECLARE SUB Key2Arr (Num2Array!(), RowCount%)
DECLARE SUB MatInv (MatrixA!(), MatrixB!(), MatrixSize%, OK AS INTEGER)
DECLARE SUB MatMult (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
DECLARE SUB Show2Arr (Num2Array!(), RowCount%, N%, M%, NumColumns%)
DECLARE SUB WaitKey ()
'
CLS
PRINT "Program to calculate heads of hydraulic network system"
PRINT
INPUT "Enter name of element data file:  ", element$
INPUT "Enter name of nodal coordinate data file:  ", node$
INPUT "Enter name of boundary conditions file (<ENTER> if none):  ", bc$
IF bc$ <> "" THEN
      INPUT "Enter number of boundary conditions:  ", NumBC%
      DIM knownval!(NumBC%), BCnode%(NumBC%)
END IF
INPUT "Enter value for initial head, H(0) (m):  ", Hi!

'* * * start timer * * *
StartTime! = TIMER
'* * * * * * * * * * * *
'
'* * * read data files * * *
'
OPEN element$ FOR INPUT AS #1
   INPUT #1, NumElem%
   i% = NumElem%
   DIM elem%(i%), ii%(i%), jj%(i%), dia!(i%), HW%(i%), idia!(i%), jdia!(i%)
      FOR i% = 0 TO NumElem%
            INPUT #1, elem%(i%), ii%(i%), jj%(i%), dia!(i%), HW%(i%), idia!(i%), jdia!(i%)
      NEXT i%
CLOSE (1)
OPEN node$ FOR INPUT AS #1
   INPUT #1, NumNode%
   i% = NumNode%
   DIM node%(i%), x!(i%), y!(i%), z!(i%), ctype%(i%), Cq180!(i%), Cq90!(i%)
      FOR i% = 0 TO NumNode%
            INPUT #1, node%(i%), x!(i%), y!(i%), z!(i%), ctype%(i%), Cq180!(i%), Cq90!(i%)
      NEXT i%
```

```
CLOSE (1)
IF bc$ <> "" THEN
    OPEN bc$ FOR INPUT AS #1
        FOR i% = 1 TO NumBC%
            INPUT #1, BCnode%(i%), knownval!(i%)
        NEXT i%
    CLOSE (1)
END IF
'
'* * * dimension arrays * * *
'
DIM k!(NumElem% + 1), ke!(NumNode%), kstar!(NumElem% + 1), H!(NumNode%)
DIM Matrix!(NumNode%, NumNode%), RHSMatrix!(NumNode%, NumNode%)
DIM InvertedMatrix!(NumNode%, NumNode%), AnswerMatrix!(NumNode%, NumNode%)
DIM NumArray!(NumNode%)
DIM Q!(NumElem% + 1), T!(NumElem% + 1)
DIM flow%(NumElem%), CompTerm!(NumElem%)
DIM OK AS INTEGER
'
counter% = 1
'
'* * * constants * * *
'
H!(0) = H0!
g! = 32.2                    'acc. due to gravity m/s^2
pi! = 3.1415926548
NumIter% = 5                 'Number of iterations before including effect of components
NumIter% = NumIter% + 1
HWconst! = 4.73
'
'* * * * * * * * * * * * calculate constant, k * * *
FOR i% = 0 TO NumElem%
    CALL CalcLength(x!0, y!0, z!0, ii%0, jj%0, ElLength!, i%)
    k!(i%) = HWconst! * ElLength! / (HW%(i%) ^ 1.852 * dia!(i%) ^ 4.87)
NEXT i%
'
'* * * initialize head values * * *
'
FOR i% = 1 TO NumNode%
    H!(i%) = H!(0) - i%
NEXT i%
'
LOCATE 10, 30: COLOR 31, 0: PRINT "-- Converging --": COLOR 7, 0
'
'* * * Start iterative procedure * * *
'
Again:
'
'* * * First few iterations do not include effect of components * * *
'
IF counter% < NumIter% THEN
    FOR i% = 0 TO NumElem%
        IF H!(ii%(i%)) - H!(jj%(i%)) = 0 THEN
            kstar!(i%) = 0
        ELSE
            kstar!(i%) = (ABS(H!(ii%(i%)) - H!(jj%(i%)))) ^ -(1 - 1 / 1.852) * k!(i%) ^ -(1 / 1.852)
        END IF
    NEXT i%
    GOTO skip1
END IF
'
'* * * reset values * * *
'
FOR i% = 0 TO NumElem%
    flow%(i%) = 0
NEXT i%
'
'* * * Calculate new values for component head-loss terms * * *
'
FOR i% = 0 TO NumNode%
    IF ctype%(i%) = 2 THEN
        FOR j% = 0 TO NumElem%
            IF ii%(j%) = i% AND H!(ii%(j%)) - H!(jj%(j%)) > 0 THEN
                flow%(j%) = 1   ' 1 - positive flow
                CALL ConstSort1(ii%0, jj%0, x!0, y!0, j%, Cq90!0, Cq180!0, Cq!, NumElem%)
                T!(j%) = Cq! * 8 / (g! * pi! ^ 2 * dia!(j%) ^ 4)
                Q!(j%) = ((H!(ii%(j%)) - H!(jj%(j%))) / (k!(j%) + T!(j%))) ^ .5
                IF Q!(j%) > 0 THEN
```

```
                    CALL Newton(T!(j%), H!(ii%(j%)), H!(jj%(j%)), Q!(j%), k!(j%))
                END IF
            ELSEIF jj%(j%) = i% AND H!(jj%(j%)) - H!(ii%(j%)) > 0 THEN
                flow%(j%) = -1  ' -1 - negative flow
                CALL ConstSort2(ii%0, jj%0, xi0, yi0, j%, Cq90!0, Cq180!0, Cq!, NumElem%)
                T!(j%) = Cq! * 8 / (g! * pi! ^ 2 * jdia!(j%) ^ 4)
                Q!(j%) = ((H!(jj%(j%)) - H!(ii%(j%))) / (k!(j%) + T!(j%))) ^ .5
                IF Q!(j%) > 0 THEN
                    CALL Newton(T!(j%), H!(jj%(j%)), H!(ii%(j%)), Q!(j%), k!(j%))
                END IF
            END IF
        NEXT j%
    END IF
NEXT i%
'
'*** calculate new kstar's ***
'
FOR i% = 0 TO NumElem%
    IF H!(ii%(i%)) - H!(jj%(i%)) = 0 THEN
        kstar!(i%) = 0
    ELSEIF flow%(i%) = 1 THEN
        kstar!(i%) = 1 / (k!(i%) * Q!(i%) ^ .852 + T!(i%) * Q!(i%))
    ELSEIF flow%(i%) = -1 THEN
        kstar!(i%) = 1 / (k!(i%) * Q!(i%) ^ .852 + T!(i%) * Q!(i%))
    ELSE
        kstar!(i%) = (ABS(H!(ii%(i%)) - H!(jj%(i%)))) ^ -(1 - 1 / 1.852) * k!(i%) ^ -(1 / 1.852)
    END IF
NEXT i%
'
skip1:
'
'*** Calculate new values for emitter constants ***
'
FOR i% = 1 TO NumNode%
    IF ctype%(i%) = 1 THEN
        k! = Cq180!(i%)
        x! = Cq90!(i%)
        ke!(i%) = k! * (H!(i%) - z!(i%)) ^ x! / H!(i%)
    END IF
NEXT i%
'
'*** Set Up Global Stiffness Matrix ***
'
FOR i% = 1 TO NumNode%
    FOR j% = 1 TO NumNode%
        Matrix!(i%, j%) = 0
    NEXT j%
NEXT i%
'*****************************************
'*      Add element matrices             *
'*****************************************
'
FOR i% = 1 TO NumElem%
    value! = kstar!(i%)
    Hi% = jj%(i%)
    Lo% = ii%(i%)
    CALL AddSquare(Hi%, Lo%, value!, Matrix!(), NumNode%)
NEXT i%
'
'*************************************************
'* Add emitter constants to diagonal of matrix   *
'*************************************************
'
FOR i% = 1 TO NumNode%
    IF ctype%(i%) = 1 THEN
        value! = -1 * ke!(i%)
        CALL AddDiagonal(i%, value!, Matrix!(), NumNode%)
    END IF
NEXT i%
'
'*** Add kmain!(0) to position (1,1) ***
'
value! = -1 * kstar!(0)
CALL AddDiagonal(1, value!, Matrix!(), NumNode%)
'
'*** set up forcing vector as matrix ***
'
FOR j% = 1 TO NumNode%
```

```
  FOR k% = 1 TO NumNode%
    RHSMatrix!(j%, k%) = 0!
  NEXT k%
NEXT j%
'
'* * * include boundary conditions * * *
'
IF bc$ <> "" THEN
      CALL BCcalc(Matrix!(), RHSMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
END IF
'
'* * * add initial head Boundary Condition to forcing vector * * *
'
RHSMatrix!(1, 1) = RHSMatrix!(1, 1) - kstar!(0) * HI(0)
'
'*******************************
'* solve simultaneous equations *
'*******************************
'
CALL MatInv(Matrix!(), InvertedMatrix!(), NumNode%, OK)
IF NOT OK THEN
  BEEP
  PRINT "Bad input, no solution is possible."
  END
END IF
CALL MatMult(InvertedMatrix!(), RHSMatrix!(), AnswerMatrix!(), NumNode%)
'* * * * * * *
cnt% = 0
FOR i% = 1 TO NumNode%
      IF ABS(AnswerMatrix!(i%, 1) - HI(i%)) > .01 THEN
            GOTO OtraVez
      END IF
NEXT i%
'* * * stop timer * * *
TotalTime! = TIMER - StartTime!
'* * * * * * * * * * * *
FOR i% = 1 TO NumNode%
      HI(i%) = AnswerMatrix!(i%, 1)
      NumArray!(i%) = HI(i%)
NEXT i%
'
'* * * calculate coefficient of uniformity * * *
'
CALL Stats(NumArray!(), NumNode%, Mean!, Median!, StanDev!, Min!, Max!)
UI = 100 * (1 - StanDev! / Mean!)
'
'* * * print results * * *
'
CLS : LOCATE 8, 1
i% = 0
PRINT "Head at Node ("; i%; "): "; HI(i%)
FOR i% = 1 TO NumNode%
      PRINT "Head at Node ("; i%; "): "; HI(i%)
NEXT i%
PRINT
PRINT "Coefficient of Uniformity = "; UI; "%"
PRINT "Converged after "; counter%; " iterations."
PRINT "Total time of convergence = "; TotalTime!; " seconds"
PRINT
'
'* * * save data in DIFNET format for comparison * * *
'* * * (save only points corresponding to DIFNET output) * * *
'
biggesty! = 0
FOR i% = 0 TO NumNode%
   IF y!(i%) > biggesty! THEN
      biggesty! = y!(i%)
   END IF
NEXT i%

outfile$ = "a:an2_" + MID$(node$, 7, 1) + ".out"
OPEN outfile$ FOR OUTPUT AS #1
FOR i% = 0 TO NumNode%
   IF y!(i%) = 0 THEN
      PRINT #1, HI(i%)
   ELSEIF y!(i%) = biggesty! THEN
      PRINT #1, HI(i%)
   END IF
```

```
NEXT i%
CLOSE (1)
'
'* * * save all data points ?? * * *
'
INPUT "Save results ? (y/n) ", ans$
IF UCASE$(ans$) = "Y" THEN
INPUT "Enter name for output file:  "; filename$
OPEN filename$ FOR OUTPUT AS #1
i% = 0
PRINT #1, "Head at Node ("; i%; "): "; Hl(i%)
PRINT #1,
FOR i% = 1 TO NumNode%
    PRINT #1, "Head at Node ("; i%; "): "; Hl(i%)
NEXT i%
PRINT #1,
PRINT #1, "Coefficient of Uniformity = "; Ul; "%"
PRINT #1, "Converged after "; counter%; " iterations."
PRINT #1, "Total time of convergence = "; TotalTime!; " seconds"
CLOSE (1)
END IF
END

OtraVez:
FOR i% = 1 TO NumNode%
    Hl(i%) = AnswerMatrix!(i%, 1)
NEXT i%
LOCATE 15, 20: PRINT "Number of Iterations:  "; counter%
counter% = counter% + 1
GOTO Again

SUB AddDiagonal (position%, value!, Matrix!(), NumNode%)
Matrix!(position%, position%) = Matrix!(position%, position%) + value!
END SUB

SUB AddSquare (Hi%, Lo%, value!, Matrix!(), NumNode%)
Matrix!(Hi%, Lo%) = Matrix!(Hi%, Lo%) + value!
Matrix!(Lo%, Hi%) = Matrix!(Lo%, Hi%) + value!
Matrix!(Lo%, Lo%) = Matrix!(Lo%, Lo%) - value!
Matrix!(Hi%, Hi%) = Matrix!(Hi%, Hi%) - value!
END SUB

SUB BCcalc (Matrix!(), RHSMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
FOR i% = 1 TO NumBC%
    FOR j% = 1 TO NumNode%
        IF j% <> BCnode%(i%) THEN
            RHSMatrix!(j%, 1) = RHSMatrix!(j%, 1) - Matrix!(j%, BCnode%(i%)) * knownval!(i%)
            Matrix!(j%, BCnode%(i%)) = 0
            Matrix!(BCnode%(i%), j%) = 0
        ELSE
            RHSMatrix!(j%, 1) = Matrix!(j%, BCnode%(i%)) * knownval!(i%)
        END IF
    NEXT j%
NEXT i%
END SUB

SUB CalcLength (x!(), y!(), z!(), ii%(), jj%(), ElLength!(), i%)
    xtemp! = x!(jj%(i%)) - x!(ii%(i%))
    ytemp! = y!(jj%(i%)) - y!(ii%(i%))
    ztemp! = z!(jj%(i%)) - z!(ii%(i%))
    ElLength! = (xtemp! ^ 2 + ytemp! ^ 2 + ztemp! ^ 2) ^ .5
END SUB

SUB ConstSort1 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
FOR j% = 0 TO NumElem%
    IF ii%(i%) = jj%(j%) THEN
        IF (x!(ii%(j%)) = x!(jj%(i%))) OR (y!(ii%(j%)) = y!(jj%(i%))) THEN
            Cq! = Cq180!(ii%(i%))
            EXIT SUB        ' 180 - degree constants have preference
'       ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
'           Cq! = Cq90!(ii%(i%))
'       ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) / (x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
'           Cq! = Cq180!(ii%(i%))
        ELSE
            Cq! = Cq90!(ii%(i%))
        END IF
    ELSEIF ii%(i%) = ii%(j%) AND jj%(i%) <> jj%(j%) THEN
        IF (x!(jj%(j%)) = x!(jj%(i%))) OR (y!(jj%(j%)) = y!(jj%(i%))) THEN
```

```
                Cq! = Cq180!(ii%(i%))
                EXIT SUB        ' 180 - degree constants have preference
'           ELSEIF xl(ii%(j%)) = xl(jj%(j%)) OR xl(ii%(i%)) = xl(jj%(i%)) THEN
'                Cq! = Cq90!(ii%(i%))
'           ELSEIF ABS((yl(ii%(j%)) - yl(jj%(j%))) / (xl(ii%(j%)) - xl(jj%(j%))) - (yl(ii%(i%)) - yl(jj%(i%))) / (xl(ii%(i%)) - xl(jj%(i%)))) < .5 THEN
'                Cq! = Cq180!(ii%(i%))
            ELSE
                Cq! = Cq90!(ii%(i%))
            END IF
        END IF
    NEXT j%
    END SUB

    SUB ConstSort2 (ii%(), jj%(), xl(), yl(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
    FOR j% = 0 TO NumElem%
        IF jj%(i%) = ii%(j%) THEN
            IF xl(ii%(i%)) = xl(jj%(j%)) OR yl(ii%(i%)) = yl(jj%(j%)) THEN
                Cq! = Cq180!(jj%(i%))
                EXIT SUB
'           ELSEIF xl(ii%(j%)) = xl(jj%(j%)) OR xl(ii%(i%)) = xl(jj%(i%)) THEN
'                Cq! = Cq90!(jj%(i%))
'           ELSEIF ABS((yl(ii%(j%)) - yl(jj%(j%))) / (xl(ii%(j%)) - xl(jj%(j%))) - (yl(ii%(i%)) - yl(jj%(i%))) / (xl(ii%(i%)) - xl(jj%(i%)))) < .5 THEN
'                Cq! = Cq180!(jj%(i%))
            ELSE
                Cq! = Cq90!(jj%(i%))
            END IF
        ELSEIF jj%(i%) = jj%(j%) AND ii%(i%) <> ii%(j%) THEN
            IF xl(ii%(i%)) = xl(ii%(j%)) OR yl(ii%(i%)) = yl(ii%(j%)) THEN
                Cq! = Cq180!(jj%(i%))
                EXIT SUB
'           ELSEIF xl(ii%(j%)) = xl(jj%(j%)) OR xl(ii%(i%)) = xl(jj%(i%)) THEN
'                Cq! = Cq90!(jj%(i%))
'           ELSEIF ABS((yl(ii%(j%)) - yl(jj%(j%))) / (xl(ii%(j%)) - xl(jj%(j%))) - (yl(ii%(i%)) - yl(jj%(i%))) / (xl(ii%(i%)) - xl(jj%(i%)))) < .5 THEN
'                Cq! = Cq180!(jj%(i%))
            ELSE
                Cq! = Cq90!(jj%(i%))
            END IF
        END IF
    NEXT j%

    END SUB

    FUNCTION Determ! (MatrixA(), MatrixSize%)
      CONST False = 0
      CONST True = NOT False
      DIM Done%(MatrixSize%)
      FOR j% = 1 TO MatrixSize%
        Done%(j%) = False
      NEXT j%
      ValDeterm! = 0!
      CALL DoDet(1!, 0, 1, MatrixA!(), Done%(), ValDeterm!, MatrixSize%)
      Determ! = ValDeterm!
    END FUNCTION

    SUB DoDet (Term!, M%, k%, MatrixA!(), Done%(), ValDeterm!, MatrixSize%)
      CONST False = 0
      CONST True = NOT False
      IF k% > MatrixSize% THEN
        Sign% = -1
        IF (M% MOD 2) = 0 THEN Sign% = 1
        ValDeterm! = ValDeterm! + Sign% * Term!
      ELSE
        IF Term! <> 0! THEN
          N% = 0
          FOR j% = MatrixSize% TO 1 STEP -1
            IF Done%(j%) THEN
              N% = N% + 1
            ELSE
              Done%(j%) = True
              A1! = Term! * MatrixA!(k%, j%)
              A2% = M% + N%
              A3% = k% + 1
              A4% = MatrixSize%
              CALL DoDet(A1!, A2%, A3%, MatrixA!(), Done%(), ValDeterm!, A4%)
              Done%(j%) = False
            END IF
          NEXT j%
        END IF
```

```
  END IF
END SUB

SUB GetReply (First$, Last$, Reply$)
  Lo$ = LEFT$(First$, 1)
  Hi$ = LEFT$(Last$, 1)
  IF Lo$ > Hi$ THEN SWAP Lo$, Hi$
  PRINT USING "Enter reply from & to &"; Lo$; Hi$
  DO
    Reply$ = INKEY$
  LOOP UNTIL (Reply$ >= Lo$) AND (Reply$ <= Hi$)
END SUB

SUB Key2Arr (Num2Array!(), RowCount%)
  CONST EndNumber! = 10101                              'Mod. #1
  RowSize% = UBOUND(Num2Array!, 1)
  ColSize% = UBOUND(Num2Array!, 2)
  ' PRINT "-- Enter data for 2-dimensional array. --"      'Mod. #2
  ' PRINT "-- Maximum array size ="; RowSize%; " rows. --"   'Mod. #2
  ' PRINT "-- There are "; ColSize%; " columns per row. --"  'Mod. #2
  ' PRINT "-- Enter "; EndNumber!; " to end data entry. --"  'Mod. #2
  IF RowCount% >= RowSize% THEN
    PRINT CHR$(7)
    PRINT "-- RowCount% too large. --"
    EXIT SUB
  END IF
  DO WHILE RowCount% < RowSize%
    RowCount% = RowCount% + 1
    PRINT "<<< Row number"; RowCount%; ">>>"
    IF RowCount% = RowSize% THEN PRINT "*** Last row ***"
    ColCount% = 0
    DO WHILE ColCount% < ColSize%
      ColCount% = ColCount% + 1
      PRINT "   Entry ("; RowCount%; ","; ColCount%; "):";
      INPUT " ", Entry!
      IF Entry! = EndNumber! THEN
        IF ColCount% = 1 THEN
          EXIT DO
        ELSE
          PRINT CHR$(7);
          PRINT "*** Cannot end now.  Please reenter number. ***"
          ColCount% = ColCount% - 1
        END IF
      ELSE
        Num2Array!(RowCount%, ColCount%) = Entry!
      END IF
    LOOP
    IF Entry! = EndNumber! THEN
      RowCount% = RowCount% - 1
      EXIT DO
    END IF
  LOOP
  PRINT "--"; RowCount%; "rows entered. --"
  PRINT "-- Data entry complete --"
END SUB

SUB MatAdd (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
  FOR i% = 1 TO MatrixSize%
    FOR j% = 1 TO MatrixSize%
      MatrixC!(i%, j%) = MatrixA!(i%, j%) + MatrixB!(i%, j%)
    NEXT j%
  NEXT i%
END SUB

SUB Matlav (MatrixA!(), MatrixB!(), MatrixSize%, OK AS INTEGER)
  CONST ErrorBound! = .000000001#                       'Mod. #1
  CONST False = 0
  CONST True = NOT False
  DIM MatrixC!(MatrixSize%, MatrixSize%)

  FOR i% = 1 TO MatrixSize%
    FOR j% = 1 TO MatrixSize%
      MatrixC!(i%, j%) = MatrixA!(i%, j%)
      IF i% = j% THEN
        MatrixB!(i%, j%) = 1!
      ELSE
        MatrixB!(i%, j%) = 0!
      END IF
```

```
    NEXT j%
  NEXT i%
  FOR j% = 1 TO MatrixSize%
    i% = j%
    WHILE ABS(MatrixC!(i%, j%)) < ErrorBound!
      IF i% = MatrixSize% THEN
        OK = False
        EXIT SUB
      END IF
      i% = i% + 1
    WEND
    FOR k% = 1 TO MatrixSize%
      SWAP MatrixC!(i%, k%), MatrixC!(j%, k%)
      SWAP MatrixB!(i%, k%), MatrixB!(j%, k%)
    NEXT k%
    Factor! = 1! / MatrixC!(j%, j%)
    FOR k% = 1 TO MatrixSize%
      MatrixC!(j%, k%) = Factor! * MatrixC!(j%, k%)
      MatrixB!(j%, k%) = Factor! * MatrixB!(j%, k%)
    NEXT k%
    FOR M% = 1 TO MatrixSize%
      IF M% <> j% THEN
        Factor! = -MatrixC!(M%, j%)
        FOR k% = 1 TO MatrixSize%
          MatrixC!(M%, k%) = MatrixC!(M%, k%) + Factor! * MatrixC!(j%, k%)
          MatrixB!(M%, k%) = MatrixB!(M%, k%) + Factor! * MatrixB!(j%, k%)
        NEXT k%
      END IF
    NEXT M%
  NEXT j%
  OK = True
END SUB

SUB MatMult (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
  FOR i% = 1 TO MatrixSize%
    FOR j% = 1 TO MatrixSize%
      TempSum! = 0!
      FOR k% = 1 TO MatrixSize%
        TempSum! = TempSum! + MatrixA!(i%, k%) * MatrixB!(k%, j%)
      NEXT k%
      MatrixC!(i%, j%) = TempSum!
    NEXT j%
  NEXT i%
END SUB

SUB MatSave (MatrixA!(), MatrixSize%)
OPEN "a:matrix.dat" FOR OUTPUT AS #1
  MatFormat$ = " ##.#^^^^"                    'Mod. #1
  FOR i% = 1 TO MatrixSize%
    FOR j% = 1 TO MatrixSize%
      PRINT #1, USING MatFormat$; MatrixA!(i%, j%);
    NEXT j%
    PRINT #1,
  NEXT i%
  PRINT #1,
CLOSE (1)
END SUB

SUB MatShow (MatrixA!(), MatrixSize%)
  MatFormat$ = " ##.#^^^^"                    'Mod. #1
  FOR i% = 1 TO MatrixSize%
    FOR j% = 1 TO MatrixSize%
      PRINT USING MatFormat$; MatrixA!(i%, j%);
    NEXT j%
    PRINT
  NEXT i%
  PRINT
END SUB

SUB Newton (Tl, Hl!, Lo!, xl, kl)
DO
      Fl = (Hl! - Lo!) / (kl * xl ^ .852 + Tl * xl) - xl
      dFl = -1 * (Hl! - Lo!) * (.852 * kl * xl ^ -.148 + Tl) / (kl * xl ^ .852 + Tl * xl) ^ 2 - 1
      newxl = xl - Fl / dFl
      xl = newxl
LOOP UNTIL Fl / dFl < .01 * xl
END SUB
```

```
SUB Show2Arr (Num2Array!(), RowCount%, N%, M%, NumColumns%)
  NumRows% = 20                                          'Mod. #1
  IF (RowCount% < 1) OR (N% < 0) OR (M% < 0) OR (NumColumns% < 1) THEN
    BEEP
    PRINT "— Parameter error in Show2Arr —"
    EXIT SUB
  END IF
  ColSize% = UBOUND(Num2Array!, 2)
  LastElement% = RowCount% * ColSize%
  PageSize% = NumRows% * NumColumns%
  ElementCount% = 0
  NumOnPage% = 0
  IndexFormat$ = "(&,&)"
  IF N% = 0 THEN
    NumFormat$ = " #0." + STRING$(M%, "#") + "^^^^"
  ELSE
    NumFormat$ = STRING$(N%, "#") + "." + STRING$(M%, "#")
  END IF
  IF N% = 0 THEN
    ColWidth% = 27
  ELSE
    ColWidth% = N% + M% + 10                             'Mod. #2
  END IF
  CLS
  FOR j% = 1 TO RowCount%
    Strj$ = RIGHT$(STR$(j%), LEN(STR$(j%)) - 1)
    FOR k% = 1 TO ColSize%
      StrK$ = RIGHT$(STR$(k%), LEN(STR$(k%)) - 1)
      RowLoc% = (NumOnPage% \ NumColumns%) + 1
      ColLoc% = (NumOnPage% MOD NumColumns%) * ColWidth% + 1
      LOCATE RowLoc%, ColLoc%
      PRINT USING IndexFormat$; Strj$; StrK$;
      PRINT USING NumFormat$; Num2Array!(j%, k%)         'Mod. #2
      ElementCount% = ElementCount% + 1
      NumOnPage% = ElementCount% MOD PageSize%
      IF (NumOnPage% = 0) OR (ElementCount% = LastElement%) THEN
        PRINT "— Press a key to continue —"
        DO
        LOOP UNTIL INKEY$ <> ""
        IF ElementCount% <> LastElement% THEN
          CLS
        ELSE
          PRINT
        END IF
      END IF
    NEXT k%
  NEXT j%
END SUB

SUB Stats (NumArray!(), count%, Mean!, Median!, StanDev!, Min!, Max!)
  IF count% < 1 THEN EXIT SUB
  FOR j% = 2 TO count%
    Temp! = NumArray!(j%)
    k% = j% - 1
    DO WHILE ((Temp! < NumArray!(k%)) AND (k% > 0))
      NumArray!(k% + 1) = NumArray!(k%)
      k% = k% - 1
    LOOP
    NumArray!(k% + 1) = Temp!
  NEXT j%
  FOR j% = 1 TO count%
    ValueSum! = ValueSum! + NumArray!(j%)
    SquareSum! = SquareSum! + NumArray!(j%) ^ 2
  NEXT j%
  Min! = NumArray!(1)
  Max! = NumArray!(count%)
  IF ((count% + 1) \ 2) = count% \ 2 THEN
    Mid% = count% \ 2
    Median! = (NumArray!(Mid%) + NumArray!(Mid% + 1)) / 2!
  ELSE
    Median! = NumArray!((count% + 1) \ 2)
  END IF
  Mean! = ValueSum! / count%
  IF count% = 1 THEN
    StanDev! = 0!
  ELSE
    StanDev! = SQR((SquareSum! - count% * Mean! * Mean!) / (count% - 1))
  END IF
```

```
END SUB

SUB WaitKey
  PRINT                                    'Mod. #1
  PRINT "— PRESS ANY KEY TO CONTINUE —"
  DO
  LOOP WHILE INKEY$ = ""
END SUB
```

DIFNET1

```
'
' * * * Irrigation Network program implementing virtual node technique * * *
'
DECLARE SUB MatShow (MatrixA!(), MatrixSize%)
DECLARE SUB WaitKey ()
DECLARE SUB BCcalc (Matrix!(), RHSMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
DECLARE SUB MatAdd (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
DECLARE SUB MatInv (MatrixA!(), MatrixB!(), MatrixSize%, OK AS INTEGER)
DECLARE SUB MatMult (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
DECLARE SUB Newton (T!, a!, dh!, dx!, m!, deltah!)
DECLARE SUB ConstSort2 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
DECLARE SUB ConstSort1 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
DECLARE SUB CalcLength (x!(), y!(), z!(), ii%(), jj%(), ElLength!, i%)
'
COMMON SHARED m!, g!, pi!
COMMON SHARED H!()
'
' * * * enter data * * *
'
CLS
INPUT "Enter name of element data file: ", elemfile$
INPUT "Enter name of nodal data file: ", nodefile$
PRINT "Enter emitter parameters: "
INPUT "   k = "; ke!
INPUT "   x = "; xe!
INPUT "Enter initial head, H(0), (ft.): ", H0!
OPEN elemfile$ FOR INPUT AS #1
OPEN nodefile$ FOR INPUT AS #2
INPUT #1, NumElem%
DIM elem%(NumElem%), ii%(NumElem%), jj%(NumElem%), dia!(NumElem%), HW%(NumElem%), idia!(NumElem%), jdia!(NumElem%),
NumEmit%(NumElem%)
FOR i% = 0 TO NumElem%
   INPUT #1, elem%(i%), ii%(i%), jj%(i%), dia!(i%), HW%(i%), idia!(i%), jdia!(i%), NumEmit%(i%)
NEXT i%
INPUT #2, NumNode%
DIM node%(NumNode%), x!(NumNode%), y!(NumNode%), z!(NumNode%), ctype%(NumNode%), Cq180!(NumNode%), Cq90!(NumNode%)
FOR i% = 0 TO NumNode%
   INPUT #2, node%(i%), x!(i%), y!(i%), z!(i%), ctype%(i%), Cq180!(i%), Cq90!(i%)
NEXT i%
CLOSE (1)
CLOSE (2)
'
counter% = 1
'
DIM H!(NumNode%), D!(NumElem%), MM!(NumElem%)
DIM DMatrix!(NumNode%, NumNode%), MMatrix!(NumNode%, NumNode%), KMatrix!(NumNode%, NumNode%)
DIM KInv!(NumNode%, NumNode%), FMatrix!(NumNode%, NumNode%)
DIM ans!(NumNode%, NumNode%)
DIM a!(NumElem%), dx!(NumElem%), dh!(NumElem%)
DIM flow%(NumElem%), Q!(NumElem%), deltah!(NumElem%)
'
' * * * constants * * *
'
H!(0) = H0!
g! = 32.2
pi! = 3.1415926548
HWconst! = 4.73
m! = 1.852
```

```
'
'* * * calculate element length, dx, and constant, a * * *
'
FOR i% = 0 TO NumElem%
    CALL CalcLength(x!(), y!(), z!(), ii%(), jj%(), ElLength!, i%)
    dx!(i%) = ElLength!
    a!(i%) = HWconst! / (HW%(i%) ^ m! * dia!(i%) ^ 4.87)  '1.166
NEXT i%
'
'* * * initialize head values * * *
'
FOR i% = 1 TO NumNode%
    H!(i%) = H!(0) - i%
NEXT i%
'
'* * * start iterative procedure * * *
'
Nextiteration:
'
'* * * reset all matrices * * *
'
FOR i% = 0 TO NumNode%
  FOR j% = 0 TO NumNode%
    DMatrix!(i%, j%) = 0
    MMatrix!(i%, j%) = 0
    KMatrix!(i%, j%) = 0
    FMatrix!(i%, j%) = 0
  NEXT j%
NEXT i%
'
'* * * calculate dh * * *
'
FOR i% = 0 TO NumElem%
  dh!(i%) = H!(ii%(i%)) - H!(jj%(i%))
NEXT i%
'
'* * * Calculate deltah * * *
'
FOR j% = 0 TO NumElem%
  IF ctype%(ii%(j%)) = 2 AND dh!(j%) > 0 THEN
    flow%(j%) = 1   ' 1 - positive flow
    CALL ConstSort1(ii%(), jj%(), x!(), y!(), j%, Cq90!(), Cq180!(), Cq!, NumElem%)
    T! = Cq! * 8 / (gl * pi! ^ 2 * idia!(j%) ^ 4)
    CALL Newton(T!, a!(j%), dh!(j%), dx!(j%), m!, deltah!(j%))
  ELSEIF ctype%(jj%(j%)) = 2 AND dh!(j%) < 0 THEN
    flow%(j%) = -1  ' -1 - negative flow
    CALL ConstSort2(ii%(), jj%(), x!(), y!(), j%, Cq90!(), Cq180!(), Cq!, NumElem%)
    T! = Cq! * 8 / (gl * pi! ^ 2 * jdia!(j%) ^ 4)
    CALL Newton(T!, a!(j%), ABS(dh!(j%)), dx!(j%), m!, deltah!(j%))
  ELSE
    deltah!(j%) = 0
  END IF
NEXT j%
'
'* * * Calculate D's * * *
'
FOR i% = 0 TO NumElem%
  D!(i%) = 1 / a!(i%) ^ (1 / m!) * ABS((dh!(i%) - deltah!(i%)) / dx!(i%)) ^ (1 / m! - 1)
NEXT i%
'
'* * * Calculate M's and Q's * * *
'
FOR i% = 0 TO NumElem%
  IF NumEmit%(i%) > 0 THEN
    Have! = (1 / (m! + 1)) * (H!(ii%(i%)) - deltah!(i%)) + (1 - 1 / (m! + 1)) * H!(jj%(i%))
        Zave! = (z!(ii%(i%)) + z!(jj%(i%))) / 2
    MM!(i%) = NumEmit%(i%) * ke! * (Have! - Zave!) ^ xe! / Have! / dx!(i%)
    Q!(i%) = NumEmit%(i%) * ke! * (Have! - Zave!) ^ xe! / dx!(i%)
  ELSE
    MM!(i%) = 0
    Q!(i%) = 0
  END IF
NEXT i%
'
'* * * Construct Global Stiffness Matrix * * *
'

'
```

```
'* * * Add element contributions to D-Matrix * * *
'
FOR i% = 0 TO NumElem%
   DMatrix!(ii%(i%), ii%(i%)) = DMatrix!(ii%(i%), ii%(i%)) + D!(i%) / dx!(i%)
   DMatrix!(jj%(i%), jj%(i%)) = DMatrix!(jj%(i%), jj%(i%)) + D!(i%) / dx!(i%)
   DMatrix!(ii%(i%), jj%(i%)) = DMatrix!(ii%(i%), jj%(i%)) - D!(i%) / dx!(i%)
   DMatrix!(jj%(i%), ii%(i%)) = DMatrix!(jj%(i%), ii%(i%)) - D!(i%) / dx!(i%)
NEXT i%
'
'* * * Add element contributions to M-Matrix * * *
'
FOR i% = 0 TO NumElem%
   MMatrix!(ii%(i%), ii%(i%)) = MMatrix!(ii%(i%), ii%(i%)) + MM!(i%) * dx!(i%) / 3
   MMatrix!(jj%(i%), jj%(i%)) = MMatrix!(jj%(i%), jj%(i%)) + MM!(i%) * dx!(i%) / 3
   MMatrix!(ii%(i%), jj%(i%)) = MMatrix!(ii%(i%), jj%(i%)) + MM!(i%) * dx!(i%) / 6
   MMatrix!(jj%(i%), ii%(i%)) = MMatrix!(jj%(i%), ii%(i%)) + MM!(i%) * dx!(i%) / 6
NEXT i%
'
'* * * add discontinuity's contribution to force vector * * *
'
FOR i% = 0 TO NumElem%
   FMatrix!(ii%(i%), 0) = FMatrix!(ii%(i%), 0) + D!(i%) * deltah!(i%) / dx!(i%)
   FMatrix!(jj%(i%), 0) = FMatrix!(jj%(i%), 0) - D!(i%) * deltah!(i%) / dx!(i%)
NEXT i%
'
'* * * add discontinuity's contribution to M-Matrix * * *
'
FOR i% = 0 TO NumElem%
   FMatrix!(ii%(i%), 0) = FMatrix!(ii%(i%), 0) + MM!(i%) * dx!(i%) * deltah!(i%) / 3
   FMatrix!(jj%(i%), 0) = FMatrix!(jj%(i%), 0) + MM!(i%) * dx!(i%) * deltah!(i%) / 6
NEXT i%
'
'* * * Solve for {H} * * *
'
CALL MatAdd(DMatrix!(), MMatrix!(), KMatrix!(), NumNode%)
'
'* * * add boundary condition (known value at node 1) * * *
'
NumBC% = 1
BCnode%(1) = 0
knownval!(1) = H!(0)
CALL BCcalc(KMatrix!(), FMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
'
'* * *
'
CALL MatInv(KMatrix!(), KInv!(), NumNode%, OK%)
IF NOT OK% THEN
   BEEP
   PRINT "No solution possible."
   END
END IF
CALL MatMult(KInv!(), FMatrix!(), ans!(), NumNode%)
'
'* * * display results * * *
'
CLS
FOR i% = 0 TO NumNode%
   PRINT i%, ans!(i%, 0)
NEXT i%
'
'* * * check against previous iteration * * *
'
FOR i% = 1 TO NumNode%
   IF ABS(H!(i%) - ans!(i%, 0)) > .01 THEN
   FOR j% = 1 TO NumNode%
      H!(j%) = ans!(j%, 0)
   NEXT j%
   GOTO NextIteration
   END IF
NEXT i%
PRINT "done"
'
'* * * save results * * *
'
'INPUT "Save results (y/n)"; resp$
'IF UCASE$(resp$) <> "N" THEN
'INPUT "Enter name of output file: ", outfile$
outfile$ = "a:din1_" + MID$(nodefile$, 8, 1) + ".out"
```

```
OPEN outfile$ FOR OUTPUT AS #3
FOR i% = 0 TO NumNode%
   PRINT #3, ans!(i%, 0)
NEXT i%
CLOSE (3)
'END IF
END

SUB BCcalc (Matrix!(), RHSMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
'
'* * * subroutine to include known values * * *
'
FOR i% = 1 TO NumBC%
      FOR j% = 0 TO NumNode%
            IF j% <> BCnode%(i%) THEN
                  RHSMatrix!(j%, 0) = RHSMatrix!(j%, 0) - Matrix!(j%, BCnode%(i%)) * knownval!(i%)
                  Matrix!(j%, BCnode%(i%)) = 0
                  Matrix!(BCnode%(i%), j%) = 0
            ELSE
                  RHSMatrix!(j%, 0) = Matrix!(j%, BCnode%(i%)) * knownval!(i%)
            END IF
      NEXT j%
NEXT i%
END SUB

SUB CalcLength (x!(), y!(), z!(), ii%(), jj%(), ELLength!, i%)
'
'* * * subroutine to calculate length of element * * *
'
      xtemp! = x!(jj%(i%)) - x!(ii%(i%))
      ytemp! = y!(jj%(i%)) - y!(ii%(i%))
      ztemp! = z!(jj%(i%)) - z!(ii%(i%))
      ELLength! = (xtemp! ^ 2 + ytemp! ^ 2 + ztemp! ^ 2) ^ .5
END SUB

SUB ConstSort1 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
'
'* * * subroutine to sort which constant applies * * *
'
FOR j% = 0 TO NumElem%
      IF ii%(i%) = jj%(j%) THEN
            IF (x!(ii%(j%)) = x!(jj%(i%))) OR (y!(ii%(j%)) = y!(jj%(i%))) THEN
                  Cq! = Cq180!(ii%(i%))
                  EXIT SUB       ' 180 - degree constants have preference
            ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
                  Cq! = Cq90!(ii%(i%))
            ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) / (x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
                  Cq! = Cq180!(ii%(i%))
            ELSE
                  Cq! = Cq90!(ii%(i%))
            END IF
      ELSEIF ii%(i%) = ii%(j%) AND jj%(i%) <> jj%(j%) THEN
            IF (x!(jj%(j%)) = x!(jj%(i%))) OR (y!(jj%(j%)) = y!(jj%(i%))) THEN
                  Cq! = Cq180!(ii%(i%))
                  EXIT SUB       ' 180 - degree constants have preference
'           ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
'                 Cq! = Cq90!(ii%(i%))
'           ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) / (x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
'                 Cq! = Cq180!(ii%(i%))
            ELSE
                  Cq! = Cq90!(ii%(i%))
            END IF
      END IF
NEXT j%
END SUB

SUB ConstSort2 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
'
'* * * subroutine to sort which constant applies * * *
'
FOR j% = 0 TO NumElem%
      IF jj%(i%) = ii%(j%) THEN
            IF x!(ii%(i%)) = x!(jj%(j%)) OR y!(ii%(i%)) = y!(jj%(j%)) THEN
                  Cq! = Cq180!(jj%(i%))
                  EXIT SUB
            ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
                  Cq! = Cq90!(jj%(i%))
            ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) / (x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
```

```
            Cq! = Cq180!(jj%(i%))
          ELSE
            Cq! = Cq90!(j%(i%))
          END IF
      ELSEIF jj%(i%) = jj%(j%) AND ii%(i%) <> ii%(j%) THEN
          IF x!(ii%(i%)) = x!(ii%(j%)) OR y!(ii%(i%)) = y!(ii%(j%)) THEN
            Cq! = Cq180!(jj%(i%))
            EXIT SUB
'         ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
'           Cq! = Cq90!(j%(i%))
'         ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) / (x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
'           Cq! = Cq180!(jj%(i%))
          ELSE
            Cq! = Cq90!(j%(i%))
          END IF
      END IF
NEXT j%

END SUB

SUB MatAdd (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
'
'*** matrix addition subroutine ***
'
  FOR i% = 0 TO MatrixSize%
    FOR j% = 0 TO MatrixSize%
      MatrixC!(i%, j%) = MatrixA!(i%, j%) + MatrixB!(i%, j%)
    NEXT j%
  NEXT i%
END SUB

SUB MatInv (MatrixA!(), MatrixB!(), MatrixSize%, OK AS INTEGER)
'
'*** matrix inversion subroutine ***
'
  CONST ErrorBound! = .000000001#                    'Mod. #1
  CONST False = 0
  CONST True = NOT False
  DIM MatrixC!(MatrixSize%, MatrixSize%)

  FOR i% = 0 TO MatrixSize%
    FOR j% = 0 TO MatrixSize%
      MatrixC!(i%, j%) = MatrixA!(i%, j%)
      IF i% = j% THEN
        MatrixB!(i%, j%) = 1!
      ELSE
        MatrixB!(i%, j%) = 0!
      END IF
    NEXT j%
  NEXT i%
  FOR j% = 0 TO MatrixSize%
    i% = j%
    WHILE ABS(MatrixC!(i%, j%)) < ErrorBound!
      IF i% = MatrixSize% THEN
        OK = False
        EXIT SUB
      END IF
      i% = i% + 1
    WEND
    FOR k% = 0 TO MatrixSize%
      SWAP MatrixC!(i%, k%), MatrixC!(j%, k%)
      SWAP MatrixB!(i%, k%), MatrixB!(j%, k%)
    NEXT k%
    Factor! = 1! / MatrixC!(j%, j%)
    FOR k% = 0 TO MatrixSize%
      MatrixC!(j%, k%) = Factor! * MatrixC!(j%, k%)
      MatrixB!(j%, k%) = Factor! * MatrixB!(j%, k%)
    NEXT k%
    FOR m% = 0 TO MatrixSize%
      IF m% <> j% THEN
        Factor! = -MatrixC!(m%, j%)
        FOR k% = 0 TO MatrixSize%
          MatrixC!(m%, k%) = MatrixC!(m%, k%) + Factor! * MatrixC!(j%, k%)
          MatrixB!(m%, k%) = MatrixB!(m%, k%) + Factor! * MatrixB!(j%, k%)
        NEXT k%
      END IF
    NEXT m%
  NEXT j%
```

```
  OK = True
END SUB

SUB MatMult (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
'
'* * * matrix multiplication subroutine * * *
'
  FOR i% = 0 TO MatrixSize%
    FOR j% = 0 TO MatrixSize%
      TempSum! = 0!
      FOR k% = 0 TO MatrixSize%
        TempSum! = TempSum! + MatrixA!(i%, k%) * MatrixB!(k%, j%)
      NEXT k%
      MatrixC!(i%, j%) = TempSum!
    NEXT j%
  NEXT i%
END SUB

SUB MatShow (MatrixA!(), MatrixSize%)
'
'* * * subroutine to display matrix * * *
'
  MatFormat$ = "##^^^^  "                          'Mod. #1
  FOR i% = 0 TO 8 ' MatrixSize%
    FOR j% = 0 TO 8 ' MatrixSize%
      PRINT USING MatFormat$; MatrixA!(i%, j%);
    NEXT j%
    PRINT
  NEXT i%
  PRINT
END SUB

SUB Newton (T!, a!, dh!, dx!, m!, deltah!)
'
'* * * subroutine to calculate deltah using Newton-Raphson method * * *
'
deltah! = .001
DO
      F! = T! * (1 / a! * (dh! - deltah!) / dx!) ^ (2 / m!) - deltah!
      dF! = -2 * T! / m! / a! / dx! * (1 / a! * (dh! - deltah!) / dx!) ^ (2 / m! - 1) - 1
      newx! = deltah! - F! / dF!
      deltah! = newx!
LOOP UNTIL F! / dF! < .01 * deltah!
END SUB

SUB WaitKey
  PRINT                                            'Mod. #1
  PRINT "— PRESS ANY KEY TO CONTINUE —"
  DO
  LOOP WHILE INKEY$ = ""
END SUB
```

DIFNET2

```
'
' * * * Irrigation Network program implementing virtual node technique * * *
'
DECLARE SUB MatShow (MatrixA!(), MatrixSize%)
DECLARE SUB WaitKey ()
DECLARE SUB BCcalc (Matrix!(), RHSMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
DECLARE SUB MatAdd (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
DECLARE SUB MatInv (MatrixA!(), MatrixB!(), MatrixSize%, OK AS INTEGER)
DECLARE SUB MatMult (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
DECLARE SUB Newton (T!, a!, dh!, dx!, m!, deltah!)
DECLARE SUB ConstSort2 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
DECLARE SUB ConstSort1 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
DECLARE SUB CalcLength (x!(), y!(), z!(), ii%(), jj%(), ElLength!, i%)
'
COMMON SHARED m!, g!, pi!
COMMON SHARED H!()
'
' * * * enter data * * *
'
CLS
INPUT "Enter name of element data file:  ", elemfile$
INPUT "Enter name of nodal data file:  ", nodefile$
PRINT "Enter emitter parameters: "
INPUT "   k = "; ke!
INPUT "   x = "; xe!
INPUT "Enter initial head, H(0), (ft.):  ", H0!
OPEN elemfile$ FOR INPUT AS #1
OPEN nodefile$ FOR INPUT AS #2
INPUT #1, NumElem%
DIM elem%(NumElem%), ii%(NumElem%), jj%(NumElem%), dia!(NumElem%), HW%(NumElem%), idia!(NumElem%), jdia!(NumElem%),
NumEmit%(NumElem%)
FOR i% = 0 TO NumElem%
   INPUT #1, elem%(i%), ii%(i%), jj%(i%), dia!(i%), HW%(i%), idia!(i%), jdia!(i%), NumEmit%(i%)
NEXT i%
INPUT #2, NumNode%
DIM node%(NumNode%), x!(NumNode%), y!(NumNode%), z!(NumNode%), ctype%(NumNode%), Cq180!(NumNode%), Cq90!(NumNode%)
FOR i% = 0 TO NumNode%
   INPUT #2, node%(i%), x!(i%), y!(i%), z!(i%), ctype%(i%), Cq180!(i%), Cq90!(i%)
NEXT i%
CLOSE (1)
CLOSE (2)
'
counter% = 1
'
DIM H!(NumNode%), D!(NumElem%), MM!(NumElem%)
DIM DMatrix!(NumNode%, NumNode%), MMatrix!(NumNode%, NumNode%), KMatrix!(NumNode%, NumNode%)
DIM KInv!(NumNode%, NumNode%), FMatrix!(NumNode%, NumNode%)
DIM ans!(NumNode%, NumNode%)
DIM a!(NumElem%), dx!(NumElem%), dh!(NumElem%)
DIM flow%(NumElem%), Q!(NumElem%), deltah!(NumElem%)
DIM qe!(NumElem%), T!(NumElem%)
'
' * * * constants * * *
'
H!(0) = H0!
g! = 32.2
pi! = 3.1415926548
```

```
HWconst! = 4.73
mi = 1.852
'
'* * * calculate element length, dx, and constant, a * * *
'
FOR i% = 0 TO NumElem%
    CALL CalcLength(xi(), yi(), zi(), ii%(), jj%(), ElLength!, i%)
    dx!(i%) = ElLength!
    a!(i%) = HWconst! / (HW%(i%) ^ mi * dia!(i%) ^ 4.87)  '1.166
NEXT i%
'
'* * * initialize head values * * *
'
FOR i% = 1 TO NumNode%
    Hi(i%) = Hi(0) - i%
NEXT i%
'
'* * * start iterative procedure * * *
'
NextIteration:
'
'* * * reset all matrices * * *
'
FOR i% = 0 TO NumNode%
  FOR j% = 0 TO NumNode%
    DMatrix!(i%, j%) = 0
    MMatrix!(i%, j%) = 0
    KMatrix!(i%, j%) = 0
    FMatrix!(i%, j%) = 0
  NEXT j%
NEXT i%
'
'* * * calculate dh * * *
'
FOR i% = 0 TO NumElem%
  dh!(i%) = Hi(ii%(i%)) - Hi(jj%(i%))
NEXT i%
'
'* * * Calculate deltah * * *
'
FOR j% = 0 TO NumElem%
  IF ctype%(ii%(j%)) = 2 AND dh!(j%) > 0 THEN
    flow%(j%) = 1   ' 1 - positive flow
    CALL ConstSort1(ii%(), jj%(), xi(), yi(), j%, Cq90!(), Cq180!(), Cq!, NumElem%)
    Ti(j%) = Cq! * 8 / (g! * pi! ^ 2 * idia!(j%) ^ 4)
    CALL Newton(Ti(j%), a!(j%), dh!(j%), dx!(j%), mi, qe!(j%))
  ELSEIF ctype%(jj%(j%)) = 2 AND dh!(j%) < 0 THEN
    flow%(j%) = -1  ' -1 - negative flow
    CALL ConstSort2(ii%(), jj%(), xi(), yi(), j%, Cq90!(), Cq180!(), Cq!, NumElem%)
    Ti(j%) = Cq! * 8 / (g! * pi! ^ 2 * jdia!(j%) ^ 4)
    CALL Newton(Ti(j%), a!(j%), dh!(j%), dx!(j%), mi, qe!(j%))
  ELSE
    qe!(j%) = (dh!(j%) / a!(j%) / dx!(j%)) ^ (1 / mi)
    Ti(j%) = 0
  END IF
NEXT j%
'
'* * * Calculate D's * * *
'
FOR i% = 0 TO NumElem%
  Di(i%) = dx!(i%) / (a!(i%) * qe!(i%) ^ (mi - 1) * dx!(i%) + Ti(i%) * qe!(i%))
NEXT i%
'
'* * * Calculate M's * * *
'
FOR i% = 0 TO NumElem%
  IF NumEmit%(i%) > 0 THEN
    Have! = Hi(ii%(i%))  ' (Hi(ii%(i%)) + 3 * Hi(jj%(i%))) / 4
    MM!(i%) = NumEmit%(i%) * ke! * (Have! - Zave!) ^ xe! / Have! / dx!(i%)
    Q!(i%) = NumEmit%(i%) * ke! * (Have! - Zave!) ^ xe! / dx!(i%)
  ELSE
    MM!(i%) = 0
    Q!(i%) = 0
  END IF
NEXT i%
'
'* * * Construct Global Stiffness Matrix * * *
'
```

```
'
'* * * Add element contributions to D-Matrix * * *
'
FOR i% = 0 TO NumElem%
   DMatrix!(ii%(i%), ii%(i%)) = DMatrix!(ii%(i%), ii%(i%)) + D!(i%) / dx!(i%)
   DMatrix!(jj%(i%), jj%(i%)) = DMatrix!(jj%(i%), jj%(i%)) + D!(i%) / dx!(i%)
   DMatrix!(ii%(i%), jj%(i%)) = DMatrix!(ii%(i%), jj%(i%)) - D!(i%) / dx!(i%)
   DMatrix!(jj%(i%), ii%(i%)) = DMatrix!(jj%(i%), ii%(i%)) - D!(i%) / dx!(i%)
NEXT i%
'
'* * * Add element contributions to M-Matrix * * *
'
FOR i% = 0 TO NumElem%
   MMatrix!(ii%(i%), ii%(i%)) = MMatrix!(ii%(i%), ii%(i%)) + MM!(i%) * dx!(i%) / 3
   MMatrix!(jj%(i%), jj%(i%)) = MMatrix!(jj%(i%), jj%(i%)) + MM!(i%) * dx!(i%) / 3
   MMatrix!(ii%(i%), jj%(i%)) = MMatrix!(ii%(i%), jj%(i%)) + MM!(i%) * dx!(i%) / 6
   MMatrix!(jj%(i%), ii%(i%)) = MMatrix!(jj%(i%), ii%(i%)) + MM!(i%) * dx!(i%) / 6
NEXT i%
'
'* * * Solve for {H} * * *
'
CALL MatAdd(DMatrix!(), MMatrix!(), KMatrix!(), NumNode%)
'
'* * * include boundary condition: known value at node 0 * * *
'
NumBC% = 1
BCnode%(1) = 0
knownval!(1) = H!(0)
CALL BCcalc(KMatrix!(), FMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
'
'* * *
'
CALL MatInv(KMatrix!(), KInv!(), NumNode%, OK%)
IF NOT OK% THEN
   BEEP
   PRINT "No solution possible."
   END
END IF
CALL MatMult(KInv!(), FMatrix!(), ans!(), NumNode%)
'
'* * * display results * * *
'
CLS
FOR i% = 0 TO NumNode%
   PRINT ans!(i%, 0)
NEXT i%
'
'* * * check against previous iteration * * *
'
FOR i% = 1 TO NumNode%
   IF ABS(H!(i%) - ans!(i%, 0)) > .01 THEN
   FOR j% = 1 TO NumNode%
      H!(j%) = ans!(j%, 0)
   NEXT j%
   GOTO NextIteration
   END IF
NEXT i%
PRINT "done"
'
'* * * save results * * *
'
outfile$ = "axin2_" + MID$(nodefile$, 8, 1) + ".out"
OPEN outfile$ FOR OUTPUT AS #3
FOR i% = 0 TO NumNode%
   PRINT #3, ans!(i%, 0)
NEXT i%
CLOSE (3)
END

SUB BCcalc (Matrix!(), RHSMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
'
'* * * include known values in matrices * * *
'
FOR i% = 1 TO NumBC%
      FOR j% = 0 TO NumNode%
            IF j% <> BCnode%(i%) THEN
                  RHSMatrix!(j%, 0) = RHSMatrix!(j%, 0) - Matrix!(j%, BCnode%(i%)) * knownval!(i%)
                  Matrix!(j%, BCnode%(i%)) = 0
```

```
            Matrix!(BCnode%(i%), j%) = 0
        ELSE
            RHSMatrix!(j%, 0) = Matrix!(j%, BCnode%(i%)) * knownval!(i%)
        END IF
    NEXT j%
NEXT i%
END SUB

SUB CalcLength (x!(), y!(), z!(), ii%(), jj%(), ElLength!, i%)
'
'*** subroutine to calculate length of element ***
'
    xtemp! = x!(jj%(i%)) - x!(ii%(i%))
    ytemp! = y!(jj%(i%)) - y!(ii%(i%))
    ztemp! = z!(jj%(i%)) - z!(ii%(i%))
    ElLength! = (xtemp! ^ 2 + ytemp! ^ 2 + ztemp! ^ 2) ^ .5
END SUB

SUB ConstSort1 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
'
'*** sort out which constant applies ***
'
FOR j% = 0 TO NumElem%
    IF ii%(i%) = jj%(j%) THEN
        IF (x!(ii%(j%)) = x!(jj%(i%))) OR (y!(ii%(j%)) = y!(jj%(i%))) THEN
            Cq! = Cq180!(ii%(i%))
            EXIT SUB        ' 180 - degree constants have preference
        ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
            Cq! = Cq90!(ii%(i%))
        ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) / (x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
            Cq! = Cq180!(ii%(i%))
        ELSE
            Cq! = Cq90!(ii%(i%))
        END IF
    ELSEIF ii%(i%) = ii%(j%) AND jj%(i%) <> jj%(j%) THEN
        IF (x!(jj%(j%)) = x!(jj%(i%))) OR (y!(jj%(j%)) = y!(jj%(i%))) THEN
            Cq! = Cq180!(ii%(i%))
            EXIT SUB        ' 180 - degree constants have preference
        ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
            Cq! = Cq90!(ii%(i%))
        ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) / (x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
            Cq! = Cq180!(ii%(i%))
        ELSE
            Cq! = Cq90!(ii%(i%))
        END IF
    END IF
NEXT j%
END SUB

SUB ConstSort2 (ii%(), jj%(), x!(), y!(), i%, Cq90!(), Cq180!(), Cq!, NumElem%)
'
'*** sort out which constant applies ***
'
FOR j% = 0 TO NumElem%
    IF jj%(i%) = ii%(j%) THEN
        IF x!(ii%(i%)) = x!(jj%(j%)) OR y!(ii%(i%)) = y!(jj%(j%)) THEN
            Cq! = Cq180!(jj%(i%))
            EXIT SUB
        ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
            Cq! = Cq90!(jj%(i%))
        ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) / (x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
            Cq! = Cq180!(jj%(i%))
        ELSE
            Cq! = Cq90!(jj%(i%))
        END IF
    ELSEIF jj%(i%) = jj%(j%) AND ii%(i%) <> ii%(j%) THEN
        IF x!(ii%(i%)) = x!(ii%(j%)) OR y!(ii%(i%)) = y!(ii%(j%)) THEN
            Cq! = Cq180!(jj%(i%))
            EXIT SUB
        ELSEIF x!(ii%(j%)) = x!(jj%(j%)) OR x!(ii%(i%)) = x!(jj%(i%)) THEN
            Cq! = Cq90!(jj%(i%))
        ELSEIF ABS((y!(ii%(j%)) - y!(jj%(j%))) / (x!(ii%(j%)) - x!(jj%(j%))) - (y!(ii%(i%)) - y!(jj%(i%))) / (x!(ii%(i%)) - x!(jj%(i%)))) < .5 THEN
            Cq! = Cq180!(jj%(i%))
        ELSE
            Cq! = Cq90!(jj%(i%))
        END IF
    END IF
NEXT j%
```

```
END SUB

SUB MatAdd (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
'
'* * * matrix addition subroutine * * *
'
  FOR i% = 0 TO MatrixSize%
    FOR j% = 0 TO MatrixSize%
      MatrixC!(i%, j%) = MatrixA!(i%, j%) + MatrixB!(i%, j%)
    NEXT j%
  NEXT i%
END SUB

SUB MatInv (MatrixA!(), MatrixB!(), MatrixSize%, OK AS INTEGER)
'
'* * * matrix inversion subroutine * * *
'
  CONST ErrorBound! = .000000001#                    'Mod. #1
  CONST False = 0
  CONST True = NOT False
  DIM MatrixC!(MatrixSize%, MatrixSize%)

  FOR i% = 0 TO MatrixSize%
    FOR j% = 0 TO MatrixSize%
      MatrixC!(i%, j%) = MatrixA!(i%, j%)
      IF i% = j% THEN
        MatrixB!(i%, j%) = 1!
      ELSE
        MatrixB!(i%, j%) = 0!
      END IF
    NEXT j%
  NEXT i%
  FOR j% = 0 TO MatrixSize%
    i% = j%
    WHILE ABS(MatrixC!(i%, j%)) < ErrorBound!
      IF i% = MatrixSize% THEN
        OK = False
        EXIT SUB
      END IF
      i% = i% + 1
    WEND
    FOR k% = 0 TO MatrixSize%
      SWAP MatrixC!(i%, k%), MatrixC!(j%, k%)
      SWAP MatrixB!(i%, k%), MatrixB!(j%, k%)
    NEXT k%
    Factor! = 1! / MatrixC!(j%, j%)
    FOR k% = 0 TO MatrixSize%
      MatrixC!(j%, k%) = Factor! * MatrixC!(j%, k%)
      MatrixB!(j%, k%) = Factor! * MatrixB!(j%, k%)
    NEXT k%
    FOR m% = 0 TO MatrixSize%
      IF m% <> j% THEN
        Factor! = -MatrixC!(m%, j%)
        FOR k% = 0 TO MatrixSize%
          MatrixC!(m%, k%) = MatrixC!(m%, k%) + Factor! * MatrixC!(j%, k%)
          MatrixB!(m%, k%) = MatrixB!(m%, k%) + Factor! * MatrixB!(j%, k%)
        NEXT k%
      END IF
    NEXT m%
  NEXT j%
  OK = True
END SUB

SUB MatMult (MatrixA!(), MatrixB!(), MatrixC!(), MatrixSize%)
'
'* * * matrix multiplication subroutine * * *
'
  FOR i% = 0 TO MatrixSize%
    FOR j% = 0 TO MatrixSize%
      TempSum! = 0!
      FOR k% = 0 TO MatrixSize%
        TempSum! = TempSum! + MatrixA!(i%, k%) * MatrixB!(k%, j%)
      NEXT k%
      MatrixC!(i%, j%) = TempSum!
    NEXT j%
  NEXT i%
END SUB
```

```
SUB MatShow (MatrixA!(), MatrixSize%)
'
'* * * display matrix * * *
'
  MatFormat$ = "##^^^^  "                          'Mod. #1
  FOR i% = 0 TO 8 ' MatrixSize%
    FOR j% = 0 TO 8 ' MatrixSize%
      PRINT USING MatFormat$; MatrixA!(i%, j%);
    NEXT j%
    PRINT
  NEXT i%
  PRINT
END SUB

SUB Newton (T!, a!, dh!, dx!, m!, qe!)
'
'* * * solve for qe using Newton-Raphson method * * *
'
qe! = .00001
DO
      F! = a! * qe! ^ m! * dx! + T! * qe! ^ 2 - dh!
      dF! = m! * a! * dx! * qe! ^ (m! - 1) + 2 * T! * qe!
      newx! = qe! - F! / dF!
      qe! = newx!
LOOP UNTIL ABS(F! / dF!) < .01 * ABS(qe!)
END SUB

SUB WaitKey
  PRINT                                            'Mod. #1
  PRINT "— PRESS ANY KEY TO CONTINUE —"
  DO
  LOOP WHILE INKEY$ = ""
END SUB
```

LAGRANGE

```
'
'*** program for construction of 2-D hydraulic topography using Lagrangian element
'

DECLARE SUB MatXConst (Matrix!(), NewMatrix!(), kl, MatSize%)
DECLARE SUB MatTrans (Matrixa(), Matrixb(), MatrixSize%)
DECLARE SUB MatSub (Matrixa(), Matrixb(), MatrixC!(), MatrixSize%)
DECLARE SUB MatMult (Matrixa(), Matrixb(), MatrixC!(), MatrixSize%)
DECLARE SUB MatInv (Matrixa(), Matrixb(), MatrixSize%, OK AS INTEGER)
DECLARE SUB MatAdd (Matrixa(), Matrixb(), MatrixC!(), MatrixSize%)
DECLARE SUB Fvector (a, b, F!())
DECLARE SUB Ky (a, b, KDy!())
DECLARE SUB Kx (a, b, KDx!(), Dxl, Dyl)
DECLARE SUB Kg (a, b, G!())
DECLARE SUB MatShow (Matrixa(), MatrixSize%)
DECLARE SUB BCcalc (Matrix!(), RHSMatrix!(), NumBC%, BCnode%(), knownval!(), NumNode%)
'
MatSize% = 9
'
DIM G!(MatSize%, MatSize%), NewG!(MatSize%, MatSize%)
DIM KDx!(MatSize%, MatSize%), NewKDx!(MatSize%, MatSize%)
DIM KDy!(MatSize%, MatSize%), NewKDy!(MatSize%, MatSize%)
DIM F!(MatSize%, MatSize%), NewF!(MatSize%, MatSize%)
DIM KD!(MatSize%, MatSize%), KDinv!(MatSize%, MatSize%), ans!(MatSize%, MatSize%)
'
'*** key in data ***
'
CLS
INPUT "Enter length of element: ", Ll
INPUT "Enter width of element: ", Wl
'GOTO skip
INPUT "Enter number of laterals: ", NumLat%
INPUT "Enter number of emitters per lateral: ", NumEmit%
PRINT "Enter emitter parameters: "
INPUT "k = "; kl
INPUT "x = "; xl
INPUT "Enter diameter of main: ", MainDia
INPUT "Enter H.W. coefficient for main: ", HWmain%
INPUT "Enter diameter of laterals: ", LatDia
INPUT "Enter H.W. coefficient for laterals: ", HWlat%
INPUT "Enter initial head in feet: ", Hl(1)
'
'*** constants ***
pil = 3.14159
HWconstl = 3.027
'
Amainl = MainDia ^ 2 * pil / 4
Alatl = LatDia ^ 2 * pil / 4
'
FOR i% = 2 TO MatSize%
   Hl(i%) = Hl(1) - i%
NEXT i%
'
axl = HWconstl / (HWmain% ^ 1.852 * MainDia ^ 1.166)
ayl = HWconstl / (HWlat% ^ 1.852 * LatDia ^ 1.166)
'
'*** Calculate Lagrangian element matrices ***
'
skip:
b = Ll / 2
a = Wl / 2
CALL Kg(a, b, G!())
CALL Ky(a, b, KDy!())
CALL Fvector(a, b, F!())
'
'*** iterative procedure starts here ***
'
```

```
NextIteration:
'
Dxl = 1 / axl ^ .54 * (ABS(Hl(1) - Hl(3)) / Ll) ^ -.46
Dyl = 1 / ayl ^ .54 * (ABS(Hl(1) + Hl(3) - Hl(5) - Hl(7)) / 2 / Wl) ^ -.46
IF Hl(9) > 0 THEN
   Gconstl = NumLat% * NumEmit% * kl * Hl(9) ^ (xl - 1) / Ll / Wl
   Qel = -1 * NumLat% * NumEmit% * kl * Hl(9) ^ xl / Ll / Wl
   Qzl = -1 * Gconstl
ELSE
   Gconstl = 0
   Qel = 0
   Qzl = 0
END IF
'
'*** Calculate new values for KDx matrix ***
'
CALL Kx(a, b, KDxl(), Dxl, Dyl)
'
'*** Calculate new values for other matrices ***
'
Kcl = Gconstl * a * b / 225
CALL MatXConst(Gl(), NewGl(), Kcl, MatSize%)
Kcl = 1 / (2 * a)
CALL MatXConst(KDxl(), NewKDxl(), Kcl, MatSize%)
Kcl = Dyl * b / (90 * a) / (2 * b)
CALL MatXConst(KDyl(), NewKDyl(), Kcl, MatSize%)
Kcl = 0   'Qel * a * b / 9                    * * * Kcl = 0 when using G(h)
CALL MatXConst(Fl(), NewFl(), Kcl, MatSize%)   '* * * (Q = 0)
'
'*** Solve Matrix Eqn. ***
'
CALL MatAdd(NewKDxl(), NewKDyl(), KDl(), MatSize%)
CALL MatAdd(KDl(), NewGl(), KDl(), MatSize%)
'
'* * * add boundary condition (known value at node 1) * * *
'
NumBC% = 1
BCnode%(1) = 1
knownval!(1) = Hl(1)
CALL BCcalc(KDl(), NewFl(), NumBC%, BCnode%(), knownval!(), MatSize%)
'
'* * *
'
CALL MatInv(KDl(), KDinvl(), MatSize%, OK%)
IF NOT OK% THEN
   BEEP
   PRINT "Bad input. No solution is possible"
   END
END IF
CALL MatMult(KDinvl(), NewFl(), ansl(), MatSize%)
CLS
PRINT "Head Calculations: "
PRINT
FOR i% = 1 TO MatSize%
   PRINT ansl(i%, 1)
NEXT i%
WHILE INKEY$ = "": WEND
FOR i% = 1 TO MatSize%
   IF ABS(ansl(i%, 1) - H(i%)) > .01 THEN
      GOTO TryAgain
   END IF
NEXT i%
PRINT "Done"
END

TryAgain:
FOR i% = 1 TO MatSize%
   H(i%) = ansl(i%, 1)
NEXT i%
GOTO NextIteration

SUB BCcalc (Matrixl(), RHSMatrixl(), NumBC%, BCnode%(), knownval!(), NumNode%)
'
'* * * this subroutine evaluates matrices to include known values * * *
'
FOR i% = 1 TO NumBC%
      FOR j% = 1 TO NumNode%
            IF j% <> BCnode%(i%) THEN
```

```
                RHSMatrix!(j%, 1) = RHSMatrix!(j%, 1) - Matrix!(j%, BCnode%(i%)) * knownval!(i%)
                Matrix!(j%, BCnode%(i%)) = 0
                Matrix!(BCnode%(i%), j%) = 0
            ELSE
                RHSMatrix!(j%, 1) = Matrix!(j%, BCnode%(i%)) * knownval!(i%)
            END IF
        NEXT j%
    NEXT i%
    END SUB

    SUB Fvector (a, b, F!())
    '
    '* * * this subroutine sets constants in forcing vector * * *
    '
    F!(1, 1) = 1
    F!(2, 1) = 4
    F!(3, 1) = 1
    F!(4, 1) = 4
    F!(5, 1) = 1
    F!(6, 1) = 4
    F!(7, 1) = 1
    F!(8, 1) = 4
    F!(9, 1) = 16
    FOR i% = 1 TO 9
      FOR j% = 2 TO 9
        F!(i%, j%) = 0!
      NEXT j%
    NEXT i%

    END SUB

    SUB Kg (a, b, G!())
    '
    '* * * this subroutine sets constants in G-matrix * * *
    '
    G!(1, 1) = 16
    G!(1, 2) = 8
    G!(1, 3) = -4
    G!(1, 4) = -2
    G!(1, 5) = 1
    G!(1, 6) = -2
    G!(1, 7) = -4
    G!(1, 8) = 8
    G!(1, 9) = 4
    G!(2, 2) = 64
    G!(2, 3) = 8
    G!(2, 4) = 4
    G!(2, 5) = -2
    G!(2, 6) = -16
    G!(2, 7) = -2
    G!(2, 8) = 4
    G!(2, 9) = 32
    G!(3, 3) = 16
    G!(3, 4) = 8
    G!(3, 5) = -4
    G!(3, 6) = -2
    G!(3, 7) = 1
    G!(3, 8) = -2
    G!(3, 9) = 4
    G!(4, 4) = 64
    G!(4, 5) = 8
    G!(4, 6) = 4
    G!(4, 7) = -2
    G!(4, 8) = -16
    G!(4, 9) = 32
    G!(5, 5) = 16
    G!(5, 6) = 8
    G!(5, 7) = -4
    G!(5, 8) = -2
    G!(5, 9) = 4
    G!(6, 6) = 64
    G!(6, 7) = 8
    G!(6, 8) = 4
    G!(6, 9) = 32
    G!(7, 7) = 16
    G!(7, 8) = 8
    G!(7, 9) = 4
    G!(8, 8) = 64
```

```
G!(8, 9) = 32
G!(9, 9) = 256
'
'*** copy values to bottom half of matrix ***
'
FOR i% = 1 TO 9
   FOR j% = i% TO 9
      G!(j%, i%) = G!(i%, j%)
   NEXT j%
NEXT i%

END SUB

SUB Kx (a, b, KDx!(), Dx!, Dy!)
'
'* * * this subroutine evaluates Dx-matrix * * *
'
DIM ln#(8)
'
'*** calculate c1 and c2 for Dx(effective) ***
'
c1 = 1 / Dx!
c2 = 2 / Dy!
'
'*** The equations were found to come in four basic forms. ***
'*** These forms are calculated below.                     ***
'
ln#(1) = 6 * a * c1 ^ 3 * c2 + 30 * a ^ 2 * c1 ^ 2 * c2 ^ 2 + 50 * a ^ 3 * c1 * c2 ^ 3
ln#(2) = 30 * a ^ 4 * c2 ^ 4 + 3 * c1 ^ 4 * LOG(c1) + 18 * a * c1 ^ 3 * c2 * LOG(c1)
ln#(3) = 39 * a ^ 2 * c1 ^ 2 * c2 ^ 2 * LOG(c1) + 36 * a ^ 3 * c1 * c2 ^ 3 * LOG(c1)
ln#(4) = 12 * a ^ 4 * c2 ^ 4 * LOG(c1) - 3 * c1 ^ 4 * LOG(c1 + 2 * a * c2)
ln#(5) = -18 * a * c1 ^ 3 * c2 * LOG(c1 + 2 * a * c2)
ln#(6) = -39 * a ^ 2 * c1 ^ 2 * c2 ^ 2 * LOG(c1 + 2 * a * c2)
ln#(7) = -36 * a ^ 3 * c1 * c2 ^ 3 * LOG(c1 + 2 * a * c2)
ln#(8) = -12 * a ^ 4 * c2 ^ 4 * LOG(c1 + 2 * a * c2)
Aform# = 0#
FOR i% = 1 TO 8
   Aform# = Aform# + ln#(i%)
NEXT i%
ln#(1) = 6 * a * c1 ^ 2 * c2 + 12 * a ^ 2 * c1 * c2 ^ 2 + 2 * a ^ 3 * c2 ^ 3
ln#(2) = 3 * c1 ^ 3 * LOG(c1) + 9 * a * c1 ^ 2 * c2 * LOG(c1)
ln#(3) = 6 * a ^ 2 * c1 * c2 ^ 2 * LOG(c1) - 3 * c1 ^ 3 * LOG(c1 + 2 * a * c2)
ln#(4) = -9 * a * c1 ^ 2 * c2 * LOG(c1 + 2 * a * c2)
ln#(5) = -6 * a ^ 2 * c1 * c2 ^ 2 * LOG(c1 + 2 * a * c2)
Bform# = 0#
FOR i% = 1 TO 5
   Bform# = Bform# + ln#(i%)
NEXT i%
ln#(1) = 6 * a * c1 ^ 3 * c2 + 18 * a ^ 2 * c1 ^ 2 * c2 ^ 2 + 8 * a ^ 3 * c1 * c2 ^ 3
ln#(2) = -4 * a ^ 4 * c2 ^ 4 + 3 * c1 ^ 4 * LOG(c1) + 12 * a * c1 ^ 3 * c2 * LOG(c1)
ln#(3) = 12 * a ^ 2 * c1 ^ 2 * c2 ^ 2 * LOG(c1) - 3 * c1 ^ 4 * LOG(c1 + 2 * a * c2)
ln#(4) = -12 * a * c1 ^ 3 * c2 * LOG(c1 + 2 * a * c2)
ln#(5) = -12 * a ^ 2 * c1 ^ 2 * c2 ^ 2 * LOG(c1 + 2 * a * c2)
Cform# = 0#
FOR i% = 1 TO 5
   Cform# = Cform# + ln#(i%)
NEXT i%
ln#(1) = 6 * a * c1 ^ 3 * c2 + 6 * a ^ 2 * c1 ^ 2 * c2 ^ 2 + 2 * a ^ 3 * c1 * c2 ^ 3 - 2 * a ^ 4 * c2 ^ 4
ln#(2) = 3 * c1 ^ 4 * LOG(c1) + 6 * a * c1 ^ 3 * c2 * LOG(c1)
ln#(3) = 3 * a ^ 2 * c1 ^ 2 * c2 ^ 2 * LOG(c1) - 3 * c1 ^ 4 * LOG(c1 + 2 * a * c2)
ln#(4) = -6 * a * c1 ^ 3 * c2 * LOG(c1 + 2 * a * c2)
ln#(5) = -3 * a ^ 2 * c1 ^ 2 * c2 ^ 2 * LOG(c1 + 2 * a * c2)
Dform# = 0#
FOR i% = 1 TO 5
   Dform# = Dform# + ln#(i%)
NEXT i%
'
'*** Values are now calculated for the upper triangle matrix ***
'
KDx!(1, 1) = -7 * Aform# / (72 * a ^ 4 * b * c2 ^ 5)
KDx!(1, 2) = Aform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(1, 3) = -1 * Aform# / (72 * a ^ 4 * b * c2 ^ 5)
KDx!(1, 4) = (c1 + 2 * a * c2) * Bform# / (36 * a ^ 4 * b * c2 ^ 5)
KDx!(1, 5) = -(c1 + a * c2) * Bform# / (72 * a ^ 4 * b * c2 ^ 5)
KDx!(1, 6) = (c1 + a * c2) * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(1, 7) = -7 * (c1 + a * c2) * Bform# / (72 * a ^ 4 * b * c2 ^ 5)
KDx!(1, 8) = 7 * (c1 + 2 * a * c2) * Bform# / (36 * a ^ 4 * b * c2 ^ 5)
KDx!(1, 9) = -2 * (c1 + 2 * a * c2) * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
```

```
KDx!(2, 2) = -2 * Aform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(2, 3) = Aform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(2, 4) = -2 * (c1 + 2 * a * c2) * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(2, 5) = (c1 + a * c2) * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(2, 6) = -2 * (c1 + a * c2) * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(2, 7) = (c1 + a * c2) * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(2, 8) = -2 * (c1 + 2 * a * c2) * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(2, 9) = 2 * (c1 + 2 * a * c2) * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(3, 3) = -7 * Aform# / (72 * a ^ 4 * b * c2 ^ 5)
KDx!(3, 4) = 7 * (c1 + 2 * a * c2) * Bform# / (36 * a ^ 4 * b * c2 ^ 5)
KDx!(3, 5) = -7 * (c1 + a * c2) * Bform# / (72 * a ^ 4 * b * c2 ^ 5)
KDx!(3, 6) = (c1 + a * c2) * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(3, 7) = -1 * (c1 + a * c2) * Bform# / (72 * a ^ 4 * b * c2 ^ 5)
KDx!(3, 8) = (c1 + 2 * a * c2) * Bform# / (36 * a ^ 4 * b * c2 ^ 5)
KDx!(3, 9) = -2 * (c1 + 2 * a * c2) * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(4, 4) = -7 * Cform# / (18 * a ^ 4 * b * c2 ^ 5)
KDx!(4, 5) = 7 * c1 * Bform# / (36 * a ^ 4 * b * c2 ^ 5)
KDx!(4, 6) = -2 * c1 * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(4, 7) = c1 * Bform# / (36 * a ^ 4 * b * c2 ^ 5)
KDx!(4, 8) = -1 * Cform# / (18 * a ^ 4 * b * c2 ^ 5)
KDx!(4, 9) = 4 * Cform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(5, 5) = -7 * Dform# / (72 * a ^ 4 * b * c2 ^ 5)
KDx!(5, 6) = Dform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(5, 7) = -1 * Dform# / (72 * a ^ 4 * b * c2 ^ 5)
KDx!(5, 8) = c1 * Bform# / (36 * a ^ 4 * b * c2 ^ 5)
KDx!(5, 9) = -2 * c1 * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(6, 6) = -2 * Dform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(6, 7) = Dform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(6, 8) = -2 * c1 * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(6, 9) = 4 * c1 * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(7, 7) = -7 * Dform# / (72 * a ^ 4 * b * c2 ^ 5)
KDx!(7, 8) = 7 * c1 * Bform# / (36 * a ^ 4 * b * c2 ^ 5)
KDx!(7, 9) = -2 * c1 * Bform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(8, 8) = -7 * Cform# / (18 * a ^ 4 * b * c2 ^ 5)
KDx!(8, 9) = 4 * Cform# / (9 * a ^ 4 * b * c2 ^ 5)
KDx!(9, 9) = -8 * Cform# / (9 * a ^ 4 * b * c2 ^ 5)
'
'*** copy values to lower triangle of matrix ***
'
FOR i% = 1 TO 9
   FOR j% = i% TO 9
      KDx!(j%, i%) = KDx!(i%, j%)
   NEXT j%
NEXT i%

END SUB

SUB Ky (a, b, KDy!())
'
'* * * this subroutine sets constants in Dy-matrix * * *
'
KDy!(1, 1) = 28
KDy!(1, 2) = 14
KDy!(1, 3) = -7
KDy!(1, 4) = 8
KDy!(1, 5) = -1
KDy!(1, 6) = 2
KDy!(1, 7) = 4
KDy!(1, 8) = -32
KDy!(1, 9) = -16
KDy!(2, 2) = 112
KDy!(2, 3) = 14
KDy!(2, 4) = -16
KDy!(2, 5) = 2
KDy!(2, 6) = 16
KDy!(2, 7) = 2
KDy!(2, 8) = -16
KDy!(2, 9) = -128
KDy!(3, 3) = 28
KDy!(3, 4) = -32
KDy!(3, 5) = 4
KDy!(3, 6) = 2
KDy!(3, 7) = -1
KDy!(3, 8) = 8
KDy!(3, 9) = -16
KDy!(4, 4) = 64
KDy!(4, 5) = -32
KDy!(4, 6) = -16
```

```
KDy!(4, 7) = 8
KDy!(4, 8) = -16
KDy!(4, 9) = 32
KDy!(5, 5) = 28
KDy!(5, 6) = 14
KDy!(5, 7) = -7
KDy!(5, 8) = 8
KDy!(5, 9) = -16
KDy!(6, 6) = 112
KDy!(6, 7) = 14
KDy!(6, 8) = -16
KDy!(6, 9) = -128
KDy!(7, 7) = 28
KDy!(7, 8) = -32
KDy!(7, 9) = -16
KDy!(8, 8) = 64
KDy!(8, 9) = 32
KDy!(9, 9) = 256
'
'*** copy values to bottom half of matrix ***
'
FOR i% = 1 TO 9
   FOR j% = i% TO 9
      KDy!(j%, i%) = KDy!(i%, j%)
   NEXT j%
NEXT i%

END SUB

SUB MatAdd (Matrixa(), Matrixb(), MatrixC!(), MatrixSize%)
'
'* * * matrix addition subroutine * * *
'
  FOR i% = 1 TO MatrixSize%
    FOR j% = 1 TO MatrixSize%
      MatrixC!(i%, j%) = Matrixa(i%, j%) + Matrixb(i%, j%)
    NEXT j%
  NEXT i%
END SUB

SUB MatInv (Matrixa(), Matrixb(), MatrixSize%, OK AS INTEGER)
'
'* * * matrix inversion subroutine * * *
'
  CONST ErrorBound! = .0000001                    'Mod. #1
  CONST False = 0
  CONST True = NOT False
  DIM MatrixC!(MatrixSize%, MatrixSize%)

  FOR i% = 1 TO MatrixSize%
    FOR j% = 1 TO MatrixSize%
      MatrixC!(i%, j%) = Matrixa(i%, j%)
      IF i% = j% THEN
        Matrixb(i%, j%) = 1!
      ELSE
        Matrixb(i%, j%) = 0!
      END IF
    NEXT j%
  NEXT i%
  FOR j% = 1 TO MatrixSize%
    i% = j%
    WHILE ABS(MatrixC!(i%, j%)) < ErrorBound!
      IF i% = MatrixSize% THEN
        OK = False
        EXIT SUB
      END IF
      i% = i% + 1
    WEND
    FOR k% = 1 TO MatrixSize%
      SWAP MatrixC!(i%, k%), MatrixC!(j%, k%)
      SWAP Matrixb(i%, k%), Matrixb(j%, k%)
    NEXT k%
    Factor! = 1! / MatrixC!(j%, j%)
    FOR k% = 1 TO MatrixSize%
      MatrixC!(j%, k%) = Factor! * MatrixC!(j%, k%)
      Matrixb(j%, k%) = Factor! * Matrixb(j%, k%)
    NEXT k%
    FOR M% = 1 TO MatrixSize%
```

```
      IF M% <> j% THEN
        Factor! = -MatrixC!(M%, j%)
        FOR k% = 1 TO MatrixSize%
          MatrixC!(M%, k%) = MatrixC!(M%, k%) + Factor! * MatrixC!(j%, k%)
          Matrixb(M%, k%) = Matrixb(M%, k%) + Factor! * Matrixb(j%, k%)
        NEXT k%
      END IF
    NEXT M%
  NEXT j%
  OK = True
END SUB

SUB MatMult (Matrixa(), Matrixb(), MatrixC!(), MatrixSize%)
'
'* * * matrix multiplication subroutine * * *
'
  FOR i% = 1 TO MatrixSize%
    FOR j% = 1 TO MatrixSize%
      TempSum! = 0!
      FOR k% = 1 TO MatrixSize%
        TempSum! = TempSum! + Matrixa(i%, k%) * Matrixb(k%, j%)
      NEXT k%
      MatrixC!(i%, j%) = TempSum!
    NEXT j%
  NEXT i%
END SUB

SUB MatShow (Matrixa(), MatrixSize%)
'
'* * * subroutine to display matrix * * *
'
  MatFormat$ = "##^^^^  "                    'Mod. #1
  FOR i% = 1 TO MatrixSize%
    FOR j% = 1 TO MatrixSize%
      PRINT USING MatFormat$; Matrixa(i%, j%);
    NEXT j%
    PRINT
  NEXT i%
  PRINT
END SUB

SUB MatSub (Matrixa(), Matrixb(), MatrixC!(), MatrixSize%)
'
'* * * matrix subtraction subroutine * * *
'
  FOR i% = 1 TO MatrixSize%
    FOR j% = 1 TO MatrixSize%
      MatrixC!(i%, j%) = Matrixa(i%, j%) - Matrixb(i%, j%)
    NEXT j%
  NEXT i%
END SUB

SUB MatTrans (Matrixa(), Matrixb(), MatrixSize%)
'
'* * * subroutine to transpose a matrix * * *
'
  FOR i% = 1 TO MatrixSize%
    FOR j% = 1 TO MatrixSize%
      Matrixb(j%, i%) = Matrixa(i%, j%)
    NEXT j%
  NEXT i%
END SUB

SUB MatXConst (Matrix!(), NewMatrix!(), kl, MatSize%)
'
'* * * subroutine to multiply matrix by a constant * * *
'
FOR i% = 1 TO MatSize%
  FOR j% = 1 TO MatSize%
    NewMatrix!(i%, j%) = Matrix!(i%, j%) * kl
  NEXT j%
NEXT i%
END SUB
```

Appendix B

A hydraulic network and its data file format for ALGNET

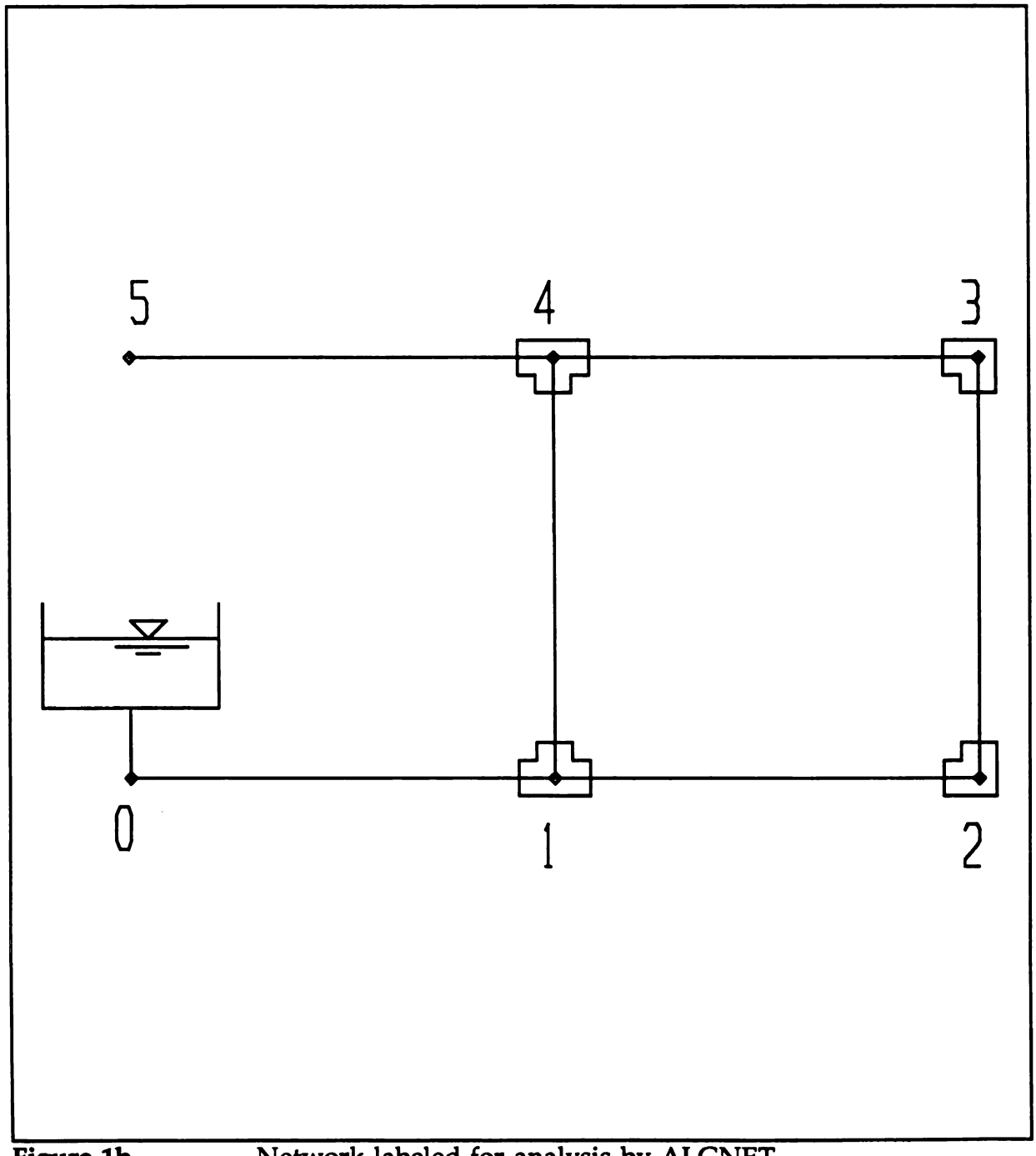


Figure 1b Network labeled for analysis by ALGNET

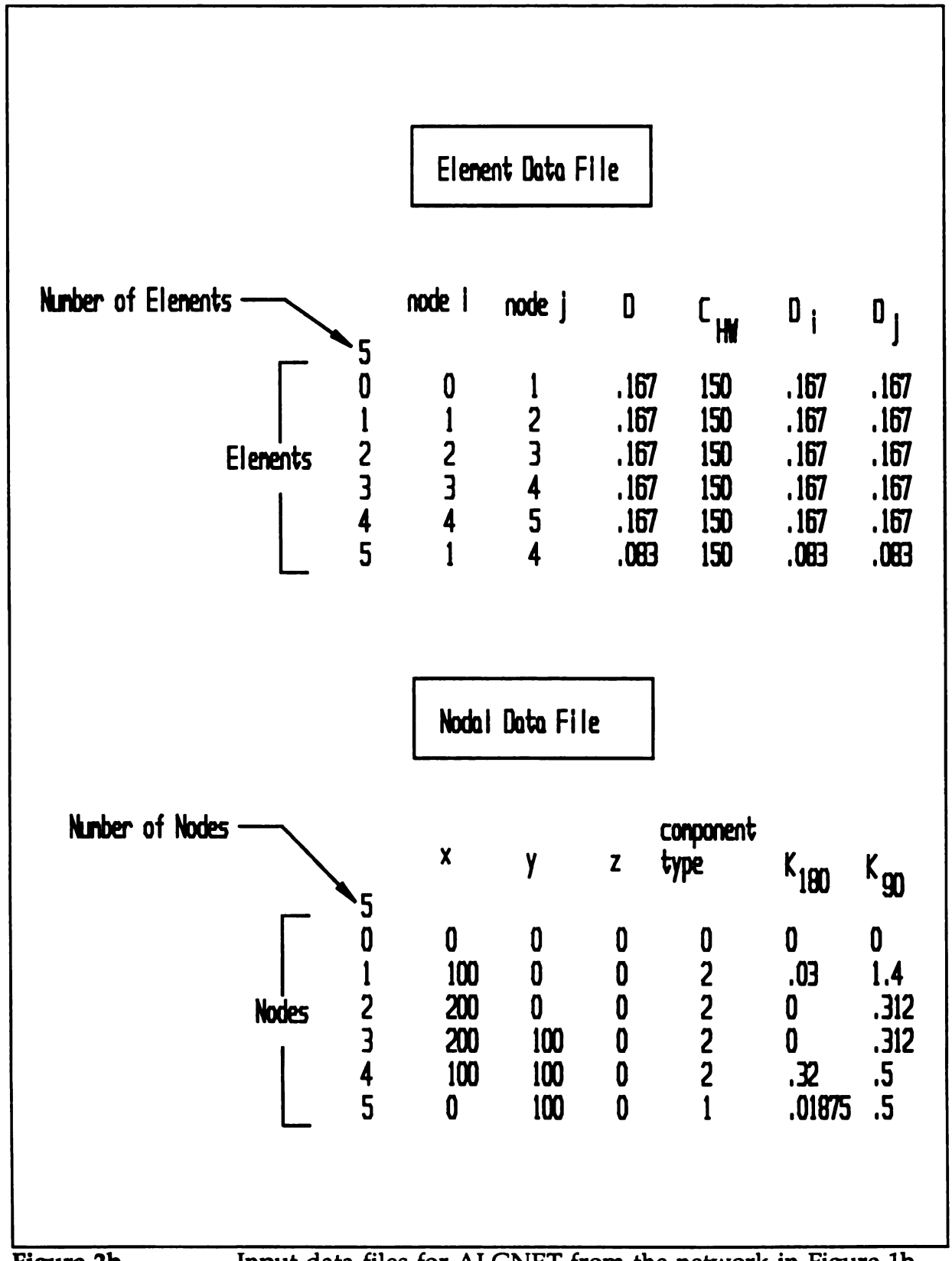


Figure 2b Input data files for ALGNET from the network in Figure 1b.

A hydraulic network and its solution with ALGNET

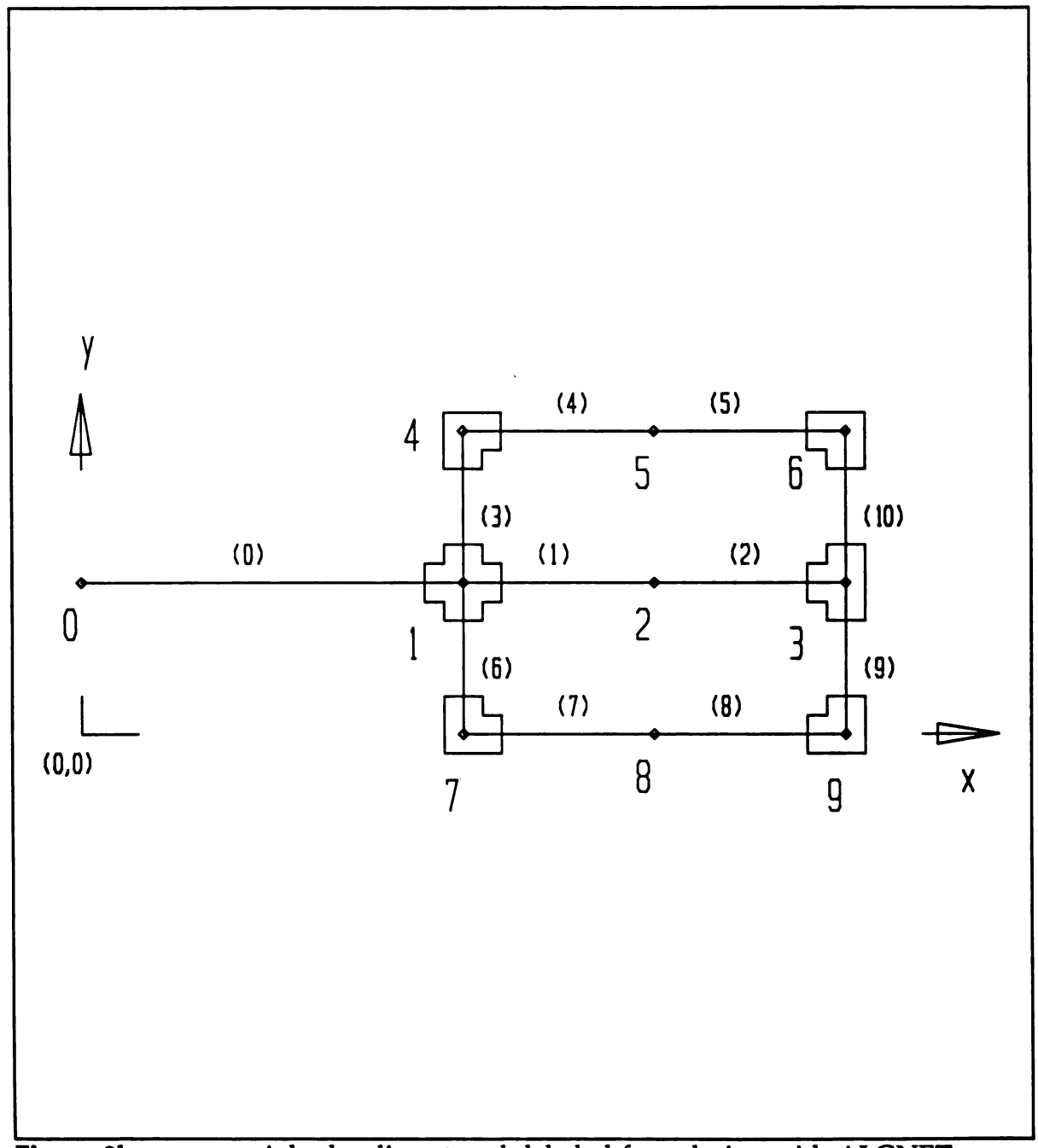


Figure 3b A hydraulic network labeled for solution with ALGNET. Demonstrated here is ALGNET's ability to handle components with more than three fittings (note the cross at node 1).

Element Data File

```
10
0     0     1     .167   140    .167   .167
1     1     2     .167   140    .167   .167
2     2     3     .167   140    .167   .167
3     1     4     .167   140    .167   .167
4     4     5     .167   140    .167   .167
5     5     6     .167   140    .167   .167
6     1     7     .167   140    .167   .167
7     7     8     .167   140    .167   .167
8     8     9     .167   140    .167   .167
9     3     9     .167   140    .167   .167
10    3     6     .167   140    .167   .167
```

Nodal Data File

```
9
0     0     20    0     0     0     0
1     40    20    0     2     .6    1.2
2     60    20    0     1     .05   .5
3     80    20    0     2     .6    1.2
4     40    40    0     2     .6    1.2
5     60    40    0     1     .05   .5
6     80    40    0     2     .6    1.2
7     40    0     0     2     .6    1.2
8     60    0     0     1     .05   .5
9     80    0     0     2     .6    1.2
```

Boundary Condition File

```
8     40
```

Data files for use with ALGNET. Boundary Conditions file specifies that head is 40 at node 8. See Figure 1c in Appendix C for explanation of files.

Output from ALGNET not including Boundary Condition

Head at Node (0): 100

Head at Node (1): 32.48771
Head at Node (2): 23.80511
Head at Node (3): 23.44133
Head at Node (4): 28.25964
Head at Node (5): 23.18241
Head at Node (6): 23.32144
Head at Node (7): 28.25959
Head at Node (8): 23.18239
Head at Node (9): 23.32143

Coefficient of Uniformity = 86.73108 %
Converged after 14 iterations.
Total time of convergence = 2.580078 seconds

Note that the appropriate symetry is calculated:

$$H_4 = H_7$$

$$H_5 = H_8$$

$$H_6 = H_9$$

Output from ALGNET including Boundary Condition

Head at Node (0): 100

Head at Node (1): 45.09686
Head at Node (2): 37.48613
Head at Node (3): 37.5452
Head at Node (4): 40.83437
Head at Node (5): 35.71997
Head at Node (6): 36.708
Head at Node (7): 42.77199
Head at Node (8): 40
Head at Node (9): 38.99974

Coefficient of Uniformity = 92.27956 %
Converged after 12 iterations.
Total time of convergence = 2.860352 seconds

Note that the Boundary Condition is met: $H_8 = 40$

A hydraulic network and its solution with DIFNET

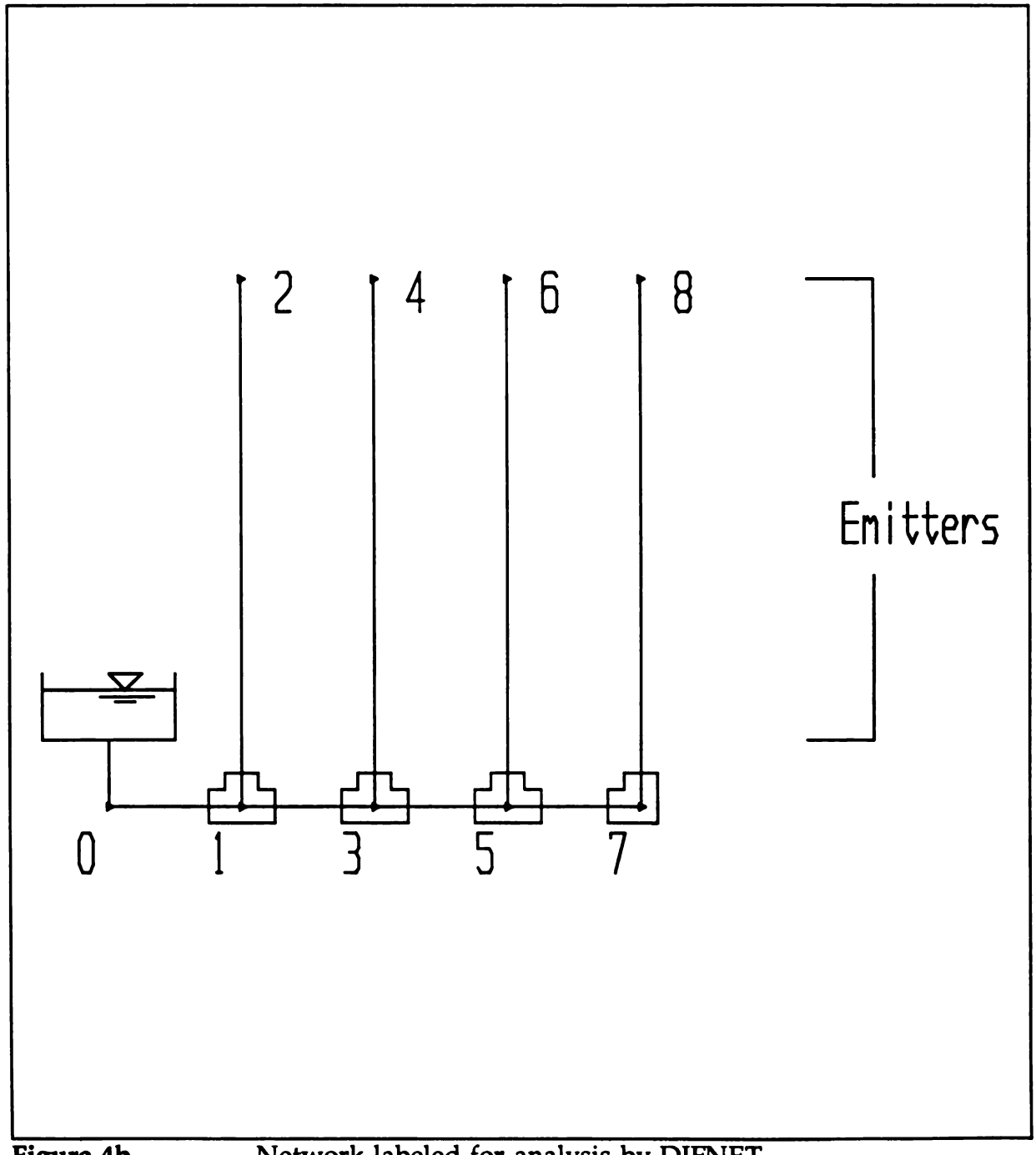


Figure 4b Network labeled for analysis by DIFNET

Element data file

Number of emitters along element

i	j	D	C_{HW}	D_i	D_j	↓
7						
0	0	1	.25	150	.25	.25	0
1	1	2	.25	150	.25	.25	4
2	1	3	.25	150	.25	.25	0
3	3	4	.25	150	.25	.25	4
4	3	5	.25	150	.25	.25	0
5	5	6	.25	150	.25	.25	4
6	5	7	.25	150	.25	.25	0
7	7	8	.25	150	.25	.25	4

Except for the last column, this data file has the same format as that for ALGNET.

Nodal data file

x	y	z	ct	k_{180}	k_{90}
8					
0	0	0	0	0	0	0
1	5	0	0	2	.5	.8
2	5	20	0	1	.05	.5
3	10	0	0	2	.5	.8
4	10	20	0	1	.05	.5
5	15	0	0	2	.5	.8
6	15	20	0	1	.05	.5
7	20	0	0	2	.5	.8
8	20	20	0	1	.05	.5

This data file has the same format as that for ALGNET

Output from DIFNET1

Head at node (0): 100
Head at node (1): 71.52046
Head at node (2): 59.99211
Head at node (3): 36.04306
Head at node (4): 30.07906
Head at node (5): 22.65207
Head at node (6): 18.83755
Head at node (7): 19.40233
Head at node (8): 16.11636

Output from DIFNET2

Head at node (0): 100
Head at node (1): 71.91549
Head at node (2): 60.27168
Head at node (3): 35.51284
Head at node (4): 29.61253
Head at node (5): 21.98885
Head at node (6): 18.26993
Head at node (7): 18.74747
Head at node (8): 15.55783

Appendix C

Calculation of average head along a lateral

The shape of the energy grade line along a lateral element may be approximated by the equation developed by Wu and Gitlin (1975):

$$\frac{H_i - h}{H_i - H_j} = 1 - \left(1 - \frac{s}{L}\right)^{m+1}$$

where, H_i = head at node i (upstream node)

H_j = head at node j (downstream node)

h = head at any point along the lateral element

s = position on lateral element (local coordinate)

L = length of lateral element

m = velocity exponent (1.852 for Hazen-Williams, 2 for Darcy-Weisbach)

Solving for h gives:

$$h = H_i - (H_i - H_j)\left[1 - \left(1 - \frac{s}{L}\right)^{m+1}\right]$$

Average head is then solved for by letting $u = 1 - \frac{s}{L}$ and integrating from u=0 to u=1:

$$h_{ave} = \int_0^1 h \, du = \left[\frac{1}{m+1}\right] H_i + \left[1 - \frac{1}{m+1}\right] H_j$$

Appendix D

A comparison of stability

In this section, the stability of each method is evaluated by inspection of its convergence curve. Those curves which rapidly approach an asymptote represent methods with rapid convergence. Those which oscillate before converging represent methods which are unstable. A convergence curve for each method follows.

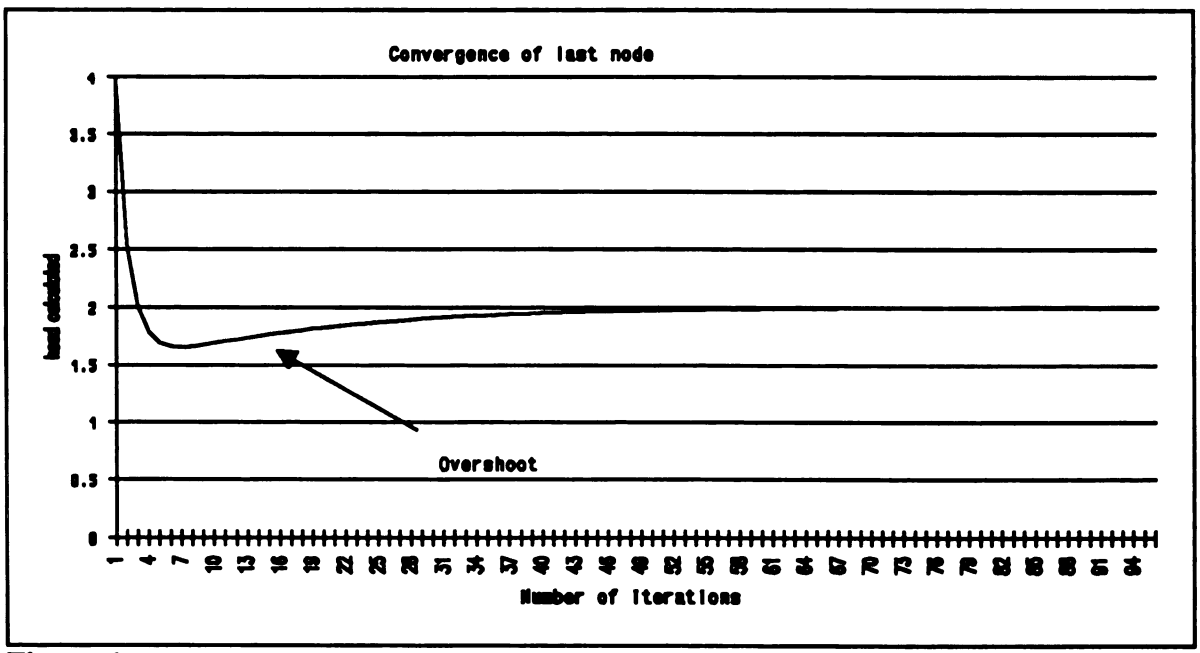


Figure 1d Convergence of last node in an irrigation network of 36 nodes, using ALGNET1. Note slight oscillation before convergence; this method is somewhat unstable for larger networks.

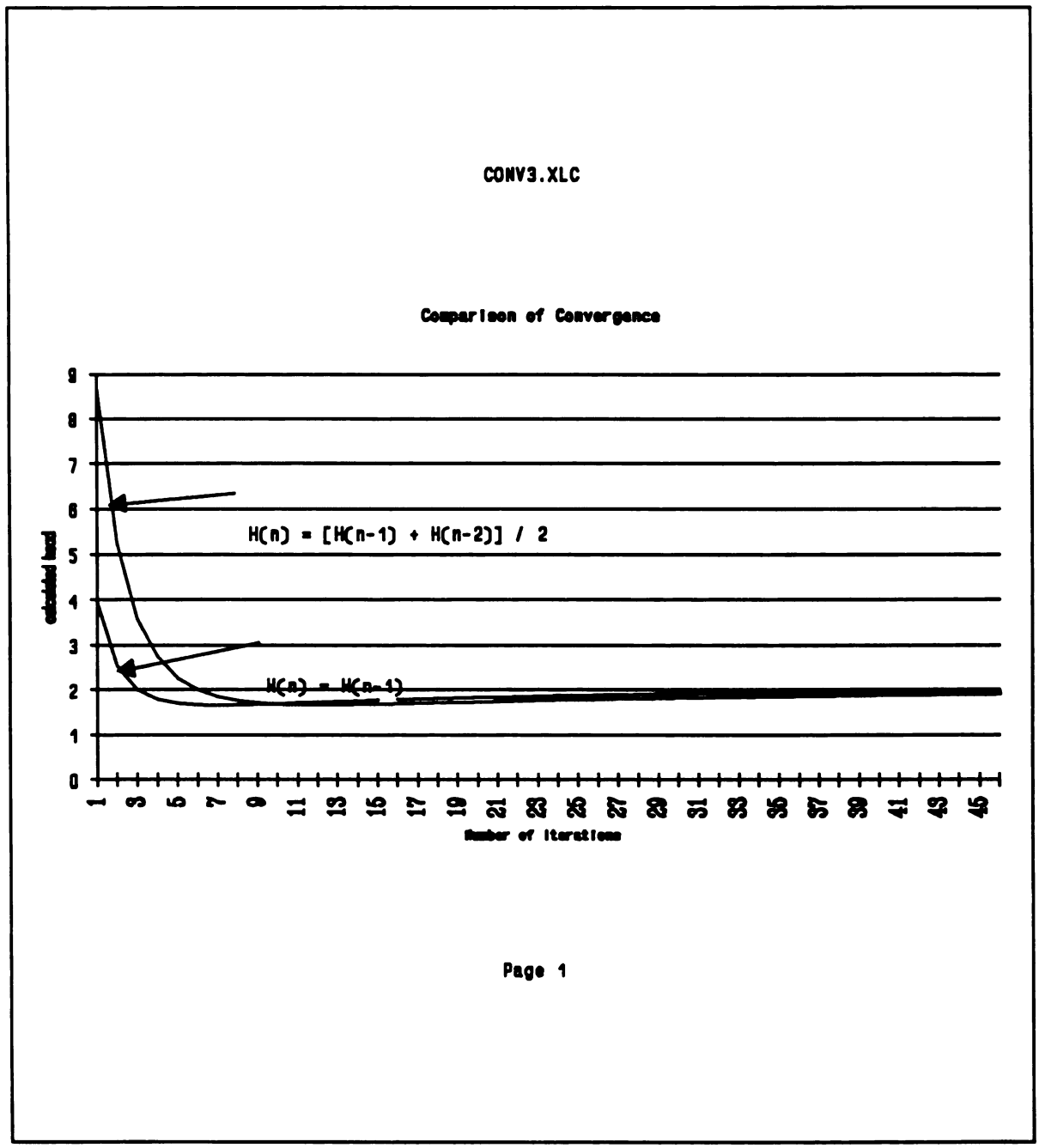


Figure 2d Effect of averaging values from previous two iterations to calculate linearizing constants for current iteration. Oscillation is not damped much and convergence is slower.

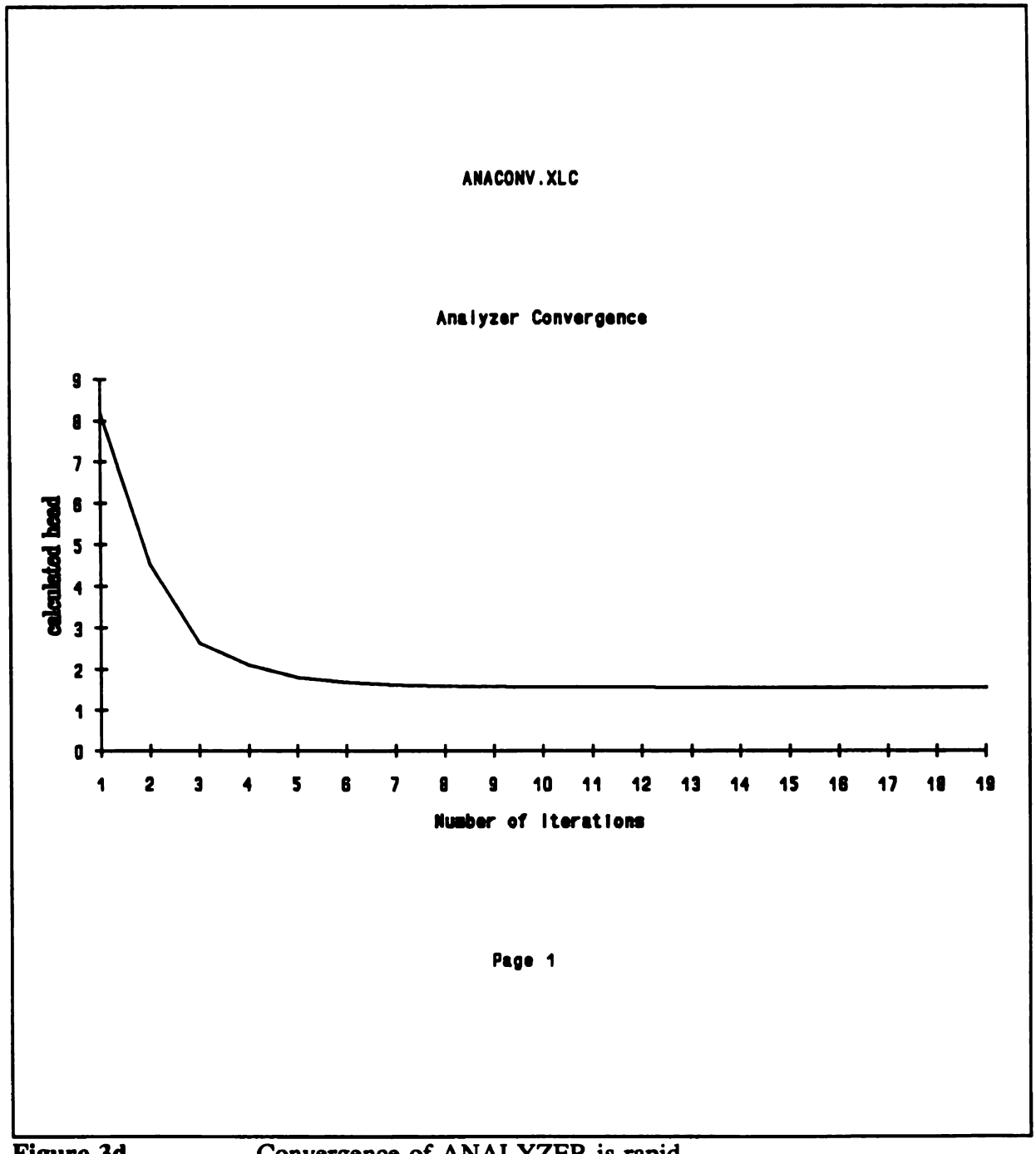


Figure 3d Convergence of ANALYZER is rapid.

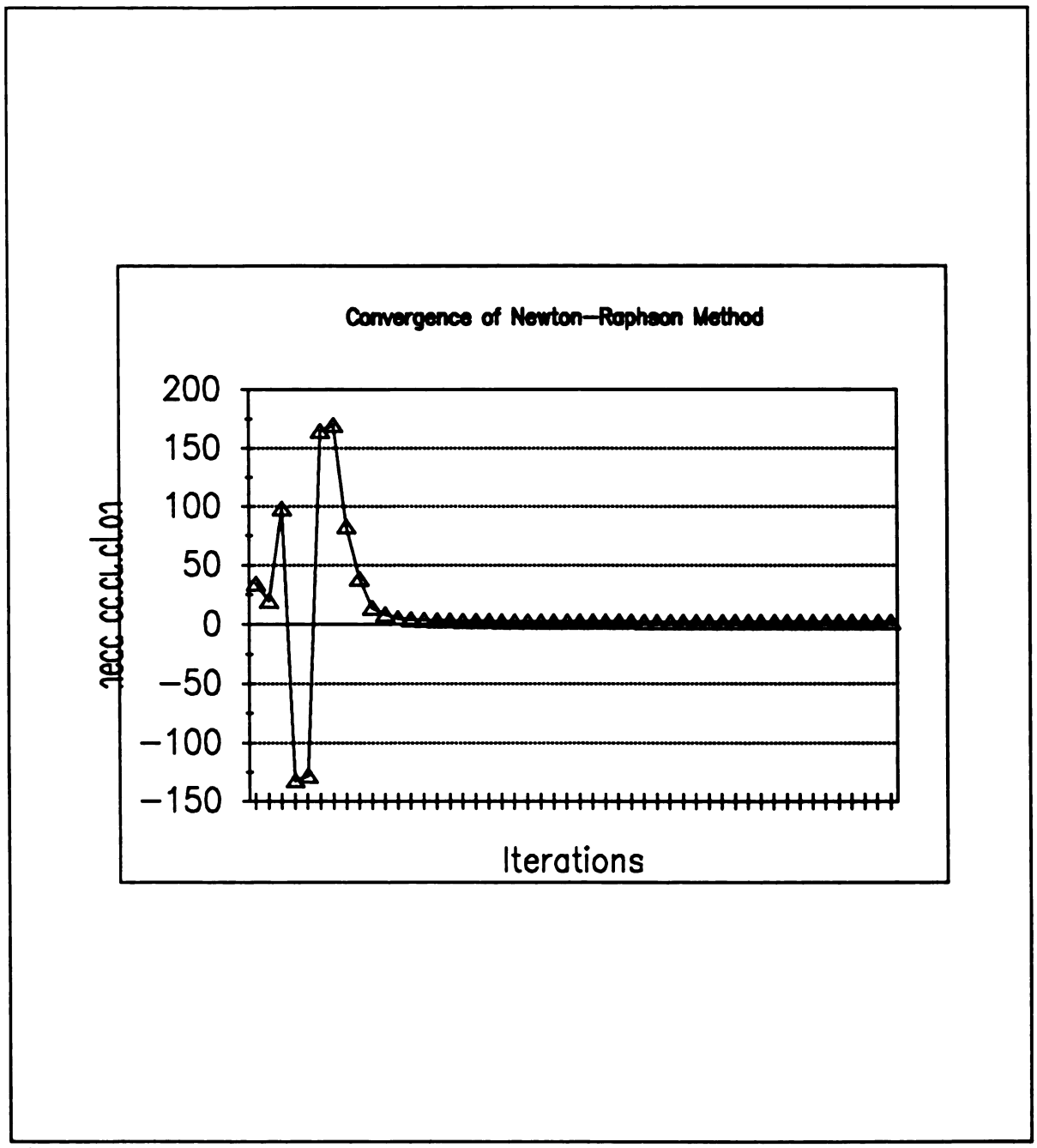


Figure 4d. Convergence of Newton-Raphson method for the last node in a 20-node network. While this method is highly unstable, it did eventually converge in this case.

Appendix E

An alternative development for the two-dimensional flow problem

Perhaps it makes sense to look at the two-dimensional flow problem in terms of velocity. Continuity then takes the form,

$$\frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z} = 0$$

where velocity in the x-direction is defined as,

$$V_x = \left[\frac{1}{a_x}\frac{\partial h}{\partial x}\right]^{\frac{1}{m}}$$

and $$a = \left(\frac{\pi}{4}\right)^m \frac{k}{C_{HW}^m D^{(4.87-2m)}}$$

The head difference between point (x,y) and (x+dx,y) may then be thought of as dependent on the flow path joining these points:

$$\Delta h_x = \frac{\partial h}{\partial x}dx = a_x V_x^m dx + 2a_y V_y^m y$$

or, rearranging and solving for V_x,

so that,

$$V_x = \left[\frac{1}{a_x}\frac{\partial h}{\partial x} - \frac{2a_y V_y^m y}{a_x dx}\right]^{\frac{1}{m}}$$

$$\frac{\partial V_x}{\partial x} = \frac{1}{ma_x}\left[\frac{1}{a_x}\frac{\partial h}{\partial x} - \frac{2a_y V_y^m y}{a_x dx}\right]^{(\frac{1}{m}-1)}\frac{\partial^2 h}{\partial x^2}$$

Substituting the equality,

$$V_y^m = \frac{1}{a_y}\frac{\partial h}{\partial y}$$

yields,

$$\frac{\partial V_x}{\partial x} = \frac{1}{ma_x^{1/m}}\left[\frac{\partial h}{\partial x} - \frac{2\left(\frac{\partial h}{\partial y}\right)y}{dx}\right]^{(\frac{1}{m}-1)}\frac{\partial^2 h}{\partial x^2}$$

Derivation of the y-term is straight forward:

$$V_y = \left[\frac{1}{a_y}\frac{\partial h}{\partial y}\right]^{\frac{1}{m}}$$

so that,

$$\frac{\partial V_y}{\partial y} = \frac{1}{ma_y^{1/m}}\left[\frac{\partial h}{\partial y}\right]^{(\frac{1}{m}-1)}\frac{\partial^2 h}{\partial y^2}$$

The z-term from the continuity equation will represent the "velocity" with which water is leaving the system through the emitters (total flow out of the system divided by the system's surface area). This term will be taken to be the total flow from the element divided by the element's area:

$$\frac{\partial V_z}{\partial z} = \frac{\sum q_e}{A} = \frac{n_e n_1 k h'^x}{dxdy}$$

Again, the resulting equation takes the general form,

$$D_x \frac{\partial^2 h}{\partial x^2} + D_y \frac{\partial^2 h}{\partial y^2} + G\,h + Q = 0$$

where,

$$D_x = \frac{1}{m a_x^{1/m}} \left[\frac{\partial h}{\partial x} - \frac{2\left(\frac{\partial h}{\partial y}\right)y}{dx} \right]^{\left(\frac{1}{m}-1\right)} \Bigg|_{n-1}$$

$$D_y = \frac{1}{a_y^{1/m}} \left[\frac{\partial h}{\partial y} \right]^{\left(\frac{1}{m}-1\right)} \Bigg|_{n-1}$$

and,

$$G = \frac{n_e n_1 k \bar{h}^{(x-1)}}{dxdy} \Bigg|_{n-1}$$

or,

$$Q = \frac{n_e n_1 k \bar{h}^x}{dxdy}\Bigg|_{n-1}$$

These results were encoded in computer program LAGRANG2 which utilizes the Lagrangian element.

Maclaurin expansion of D_x

Integration of the D_x term from Chapter IV may be facilitated by replacing D_x by its Maclaurin expansion. Consider the expansion:

$$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k$$

D_x can be rearranged to take the appropriate form:

$$D_x = \frac{D_{x0}}{dy}\left(\frac{1}{1-p}\right) \quad where \quad p = -\frac{D_{x0}}{D_y} y$$

so that,

$$D_x = \frac{D_{x0}}{dy} \sum_{k=0}^{\infty} p^k$$

This series would then be truncated at an appropriate k so as to approximate D_x with reasonable accuracy. Because this expansion has the constraint, $|p| < 1$, the Natural

Coordinate System must be chosen and D_{x0} must be less than or equal to D_y. If D_{x0} is greater than D_y, which is unfortunately the case for conventional irrigation systems, the integration limits must be changed.

LIST OF REFERENCES

Bralts, V.F., D.M. Edwards, and I. Wu. 1987. *Drip irrigation design and evaluation based on the statistical uniformity concept*. **Advances in Irrigation**, Vol. 4, Hillel. Academic Press Inc.

Bralts, V.F. and L.J. Segerlind. 1985. *Finite element analysis of drip irrigation submain units.* Transactions of the ASAE 28(3):809-814.

Bralts, V.F., S. Kelly, W.H. Shayya and L.J. Segerlind. 1993. *Finite element analysis of microirrigation hydraulics using a virtual emitter system.* Accepted for publication in the TRANSACTIONS of the ASAE 36(3):717-725.

Dhatt, G. and G. Touzot. 1984. The Finite Element Method Displayed. John Wiley and Sons, New York.

Finkel, H. 1982. CRC Handbook of Irrigation Technology, CRC Press Inc., Boca Raton, Florida. Vol. 1, Ch. 8, pp. 171-193.

Haghighi, K., V.F. Bralts and L.J. Segerlind. 1988. *Finite element formulation of tee and bend components in hydraulic pipe network analysis.* Transactions of the ASAE 31(6): 1750-1758.

Haghighi, K., R. Mohtar, V.F. Bralts and L.J. Segerlind. 1992. *A linear model for pipe network components.* Published in Computers and Electronics in Agriculture, Elsevier Scientific Publishing Co., Amsterdam, The Netherlands 7:301-321.

Holy, M. 1981. *Irrigation systems and their role in the food crisis.* International Commission on Irrigation and Drainage. International Conference for Cooperation in Irrigation, N.D. Gulhati Memorial Lecture. Sept. 1981.

Jeppson, R.W. 1976. Analysis of Flow in Pipe Networks. Ann Arbor Science Publishers, Inc. Ann Arbor, Michigan.

Kelly, S.F. 1989. *Finite Element Analysis of Drip Irrigation Hydraulics Using Quadratic Elements and A Virtual Emitter System.* MS Thesis, Dept. of Agricultural Engineering, Michigan State University.

Kunze, R.J. and W.H. Shayya. 1990. *FINDIT: a new 3-directional infiltration model with graphics.* ASAE paper presented at the "1990 International Winter Meeting sponsored by the American Society of Agricultural Engineers," Dec. 18-21, 1990, Chicago, IL.

Mohtar, R.H., V.F. Bralts and W.H. Shayya. 1991. *A finite element model for the analysis and optimization of pipe networks.* TRANSACTIONS of the ASAE 34(2):393-401.

Okosun, T.Y. 1993. How Much Longer? All Out Publishing. Holland, Michigan.

Power, J.W. 1986. *Sharing irrigation know-how with developing countries.* **Agricultural Engineering**, Sept. 1986, pp. 15-18.

Segerlind, L.J. 1984. Applied Finite Element Analysis. John Wiley and Sons.

Shayya, W.H., R.H. Mohtar, and V.F. Bralts. 1988. *ANALYZER: A computer model for the hydraulic analysis of pipe-networks.* Department of Agricultural Engineering, Michigan State University, East Lansing, Michigan.

Turner II, B.L. and Peter D. Harrison. 1983. Pulltrouser Swamp. University of Texas Press. Austin, Texas.

Villemonte, J.R. 1977. *Some basic concepts on flow in branching conduits.* Journal of the Hydraulics Division, ASCE 103(HY7):685-697.

von Bernuth, R.D. and T. Wilson. 1989. *Friction factors for small diameter plastic pipes.* J. Hydr. Engrg., ASCE 115(2):183-192.

Wiseman, Frederick M. 1989. *Agriculture and vegetation dynamics of the Maya collapse in Central Peten.* Peabody Library Press, Harvard.

Wood, D.J. and C.O. Charles. 1973. *Closure of hydraulics network analysis using linear theory.* Journal of the Hydraulics Division, ASCE 99(HY11):2129.

Wood, D.J. and A.G. Rayes. 1981. *Reliability of Algorithms for Pipe Network Analysis.* Journal of the Hydraulics Division, ASCE 107(HY10):1145-1161.

Wood, D.J. and C.O. Charles. 1972. *Hydraulic network analysis using linear theory.* Journal of the Hydraulics Division, ASCE 98(HY7):1157-1170.

Wood, D.J. 1980. Users Manual for computer analysis of flow in pipe networks including extended period simulations. Office of Engineering Continuing Education, University of Kentucky, Lexington, KY.

Wu, I.P., and Gitlin, H.M. 1975. *Energy gradient line for drip irrigation laterals.* Journal of the Irrigation and Drainage Division of the American Society of Civil Engineers. 10(IR4), 321-326. Proceedings Paper no. 11750.

Wu, I.P., and Gitlin, H.M. 1974. *Design of drip irrigation lines.* HAES Technical Bulletin. University of Hawaii, Honolulu, (96) 29.

Wu, I., T.A. Howell, and E.A. Hiler. 1979. *Hydraulic design of drip irrigation systems.* HAES Technical Bulletin 105, University of Hawaii, Honolulu, Hawaii.