COMPUTER MODELLING OF FLOW NETWORKS

FRANCES J. DONEGAN

M. PHIL

UNIVERSITY OF EDINBURGH

1988

Abstract

Pipe networks are a common feature of chemical and process plants, acting as conduits for main process fluids and for process utilities such as air, water and steam. For any pipe system it is often necessary to calculate flows and pressure drops throughout the network in order to assess the effect of changes in network structure (such as the addition or removal of certain pipes). Design and optimisation of pipe networks are dependent on reliable and accurate calculation of such flows and pressure drops.

These calculations are commonly performed using computer programs written specifically for the analysis of flow and pressure in pipe networks. This thesis firstly discusses factors which must be taken into account in the design of such programs. It subsequently describes the development and testing of three computer programs for the analysis of flow and pressure in steady- and unsteady-state pipe networks. In the thesis' conclusions the test results are discussed and recommendations are made for improvements to the computer programs.

Declaration

I declare that this thesis has been composed by me, and that the work contained herein is my own.

ACKNOWLEDGEMENTS

I should like to express my sincere thanks to Dr. J.W. Ponton for all his help, suggestions, support and encouragement during the course of this project. I should also like to thank Dr G. M. Alder of the Department of Mechanical Engineering, Edinburgh University, and Malcolm Preston and Chris Wells of ICI, Runcorn.

This project was carried out with the financial support of ICI.

TABLE OF CONTENTS

CHAPTER 1

FLOW NETWORK ANALYSIS : INTRODUCTION

Pipe networks are a common feature of chemical and process plants, acting as conduits for main process fluids and for process utilities such as air, water and steam. For any pipe system it is often necessary to calculate flows and pressure drops throughout the network in order to assess the effect of changes in system parameters (such as supply/source pressure) and changes in network structure (such as the addition or removal of certain pipes). Design and optimisation of pipe networks are dependent on reliable and accurate calculation of such flows and pressure drops.

These calculations are commonly performed using computer programs written specifically for the analysis of flow and pressure in pipe networks. A number are commercially available, including PIPENET **[35]**, PIPEPHASE **[36]** and FLONET (described in Chapter 3). Some programs have a wide range of facilities, such as a facility to handle two-phase flow in pipes, or a facility to analyse spray and sprinkler systems. Variation occurs in the method of data input and the way in which various network fittings (valves, orifice plates, pumps, compressors) must be described for data input. The essential requirement, however, of any program which performs the analysis of flow and pressure in pipe networks, is that the algorithm used to solve for flow and pressure drops must be robust and reasonably fast when networks of medium and large size are analysed. The algorithm needs to be robust because instabilities often can occur in a large network where flow in a few pipes is negligible in comparison with flow in the main network pipes. This can lead to the solution oscillating between two values.

The formulation and solution of equation sets which describe fluid flow in pipe networks have been extensively researched and documented. Early work concentrated on the analogy of fluid flow networks with electrical networks and used Kirchoff's laws to solve for flows and pressures. The main drawback to this method is that convergence of the solution is slow and not always guaranteed. Other methods have been developed which also use the electrical analogy. A major consideration when using any method of pipe network analysis is the ease and reliability with which the

set of equations describing the network may be solved. Research has focused on linearised network models for this reason.

This thesis describes work which was carried out on the topic of pipe network systems. The aim of the work was three-fold. Firstly, to make an appraisal of existing methods for modelling and solving steady-state pipe network problems. Secondly, to provide a computer tool to be used for the solution of such problems, by employing a modified version of a previously described linearisation method. Thirdly, to further develop this computer tool so that it would handle network constraints in the form of equations, as well as a description of the physical dimensions of the network. In addition, a further aim was to enable unsteady-state networks to be modelled by the computer program.

The first aim involved a survey of the available literature on analysis of steady-state pipe networks, including methods of describing network topology and the physical components incorporated into the network. Matrix and linear equation solving techniques were also briefly examined. Most recent methods describe a pipe network as a set of linearised equations and the computer program which was written to carry out the second aim used a combination of two of these methods, namely the Newton method and the Bending and Hutchison method. The program was tested on a number of sample networks of varying complexity. In some of these networks, flow in certain parts of the network was negligible in comparison with the rest of the network and this provided a good test of the program's ability to cope with potential instabilities in the final flow distribution throughout the network.

Aim three necessitated that a parser be written to handle additional data which is input as a set of linear equations. Syntax and consistency checking was incorporated in the parser. A number of sample networks were again used to test the program which successfully calculated the final flow distribution in all cases. The program was further modified to accept data input describing pressure vessels and also time step values, so that unsteady-state networks could be modelled. The sample unsteady-state problems supplied to test the program were all successfully solved.

The format of this thesis may be briefly described as follows :

- Chapter 2 gives an account of the literature survey of flow analysis in steady-state pipe networks.

- Chapter 3 describes the development of a computer program to analyse flow in steady-state pipe networks.

- Chapter 4 describes the sample network problems which were presented to the computer program and discusses the results obtained.

- Chapter 5 outlines the development of two computer programs which were developed from the program described in Chapter 3; the first intended to solve network problems where the data sets include network equations, the second intended to solve unsteady-state network problems.

- Chapter 6 discusses the performance of all three programs in relation to one particular sample network problem.

- Chapter 7 presents the conclusions of the thesis.

The notation for each chapter is given at the end of that chapter.

CHAPTER 2

LITERATURE REVIEW

2.1. Introduction

As stated in Chapter 1, the first aim of the work described in this thesis was to make an appraisal of existing methods for modelling and solving steady-state pipe network problems. This chapter presents a summary of the literature which was reviewed with this aim in mind.

The topics discussed in this chapter may be summarised as follows. Section 2.2 briefly describes how flow network problems are mathematically modelled. (It should be noted that the more general term 'flow network' is used interchangeably with 'pipe network' throughout this chapter and in the rest of the thesis). Section 2.3 discusses how the physical structure of a network is mathematically 'abstracted'. The next three sections are concerned with the analogy between fluid flow networks and electrical networks. These sections examine methods which have been originally used to model and solve electrical network problems, and their extension to fluid flow network problems. These methods use matrices to model the basic network and also to transform the network to one which is more easily analysed. The significant amount of matrix algebra involved in these methods, and those described in Section 2.7, illustrates the associated need for efficient matrix solution techniques when such methods are employed.

Section 2.8 discusses techniques which have been used to linearise the sets of non-linear equations relating flow and pressure in pipe networks. Section 2.9 presents a summary of all other network models covered in the literature survey.

Section 2.10 discusses how individual pipe line elements - pipes, pumps, various types of valves, compressors and pressure regulators - can be modelled. Sections 2.11 and 2.12 cover sparse matrix methods and the influence of supercomputers on the development of sparse matrix methods. The conclusions to this chapter are given in section 2.13.

2.2. Flow Network Representation

Flow networks are represented by sets of non-linear algebraic equations of the form :

$$f(x) = 0 \tag{2.1}$$

There are two ways of obtaining the solution of such equations :

1. Rearrange each equation to $x = \phi(x)$ and solve iteratively, so that a better estimate of x is obtained each time, on the left hand side.

2. Linearise the set of equations such that

   $$f(x) \text{ --> } Ax + B \tag{2.2}$$

   and solve this new set of linear equations, using established methods.

Either approach can include the rearrangement and/or decomposition of the set of equations into subsets.

In general there are two kinds of equation :

1. Mass balance

   $$\sum F_i = 0 \tag{2.3}$$

2. Flow/pressure drop equations

   $$F_i = f(P_k, P_j) \tag{2.4}$$

Solution methods can handle either

1. The full equation set

2. A reduced equation set formed by substituting (2.4) into (2.3) to get a node formulation (thus eliminating flows)

3. A reduced equation set formed by eliminating pressures (mesh formulation)

In the analysis of flow networks much use has been made of graph theory. Graph theory enables network relationships to be deduced and expressed in the form of matrix algebra. A short discussion of graph theory follows, as an introduction to the methods which are used to model flow networks.

2.3. Graph Theory

A flow, or pipe, network is a connected set of physical elements which permit or control the flow of fluid. Examples of such elements are pipes, pumps, control valves, non-return valves, pressure sources and reservoirs. The graph of such a network is a diagram showing the structure of the network. It consists of branches, which correspond to individual pipes, pumps or valves, and nodes, between which the branches run. A branch is said to be incident to its terminal nodes. The graph is directed if assumed directions of flow, or pressure rise, for example, are indicated. A graph is said to be connected if it is possible to move between any two nodes along the branches.

Any connected graph contains at least one tree. A tree is a set of branches connecting all the nodes without forming any meshes (or closed paths). The term 'basic mesh' describes any closed path formed from the tree by the inclusion of one non-tree branch (or link) in the graph. For example in Fig. 2.1, branches 1,2,3,4,5 constitute a tree (heavy lines). Consequently, this tree forms three basic meshes containing 3-4-6, 2-4-7 and 2-4-5-8.

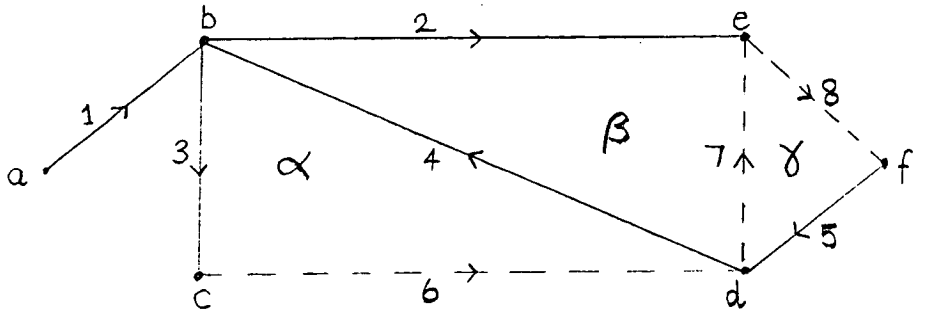


Fig. 2.1

It follows that if a connected graph has n nodes then any tree will contain $n-1$ branches, and the number of basic meshes is m, where $m = b - n + 1$ and b is the number of branches. In a directed graph the

meshes also have directions which may be defined by those of the defining links.

Graph/network characteristics can be represented by matrices. Table 2.1 shows the augmented branch-node incidence matrix A' for the graph in Fig. 2.1. An element a_{ij} of A' is 1, −1 or 0 if branch i is, respectively, incident towards, incident away from, or not connected to node j. The sum of elements in any row is zero and the columns are linearly dependent. Hence any column may be deleted ; the node corresponding to this column is called the 'datum node' and the matrix so formed constitutes the branch-node incidence matrix A of the graph.

Node

Branch	a	b	c	d	e	f
1	−1	1	.	.	.	.
2	.	−1	.	.	1	.
3	.	−1	1	.	.	.
4	.	1	.	−1	.	.
5	.	.	.	1	.	−1
6	.	.	−1	1	.	.
7	.	.	.	−1	1	.
8	.	.	.	.	−1	1

Table 2.1

Augmented branch-node incidence matrix for graph in Fig. 2.1

Mesh

Branch	α	β	χ
1	0	0	0
2	.	1	1
3	1	.	.
4	1	1	1
5	.	.	1
6	1	.	.
7	.	−1	.
8	.	.	1

Table 2.2

Branch-mesh incidence matrix for graph in Fig. 2.1

The basic meshes of a graph are described by its branch-mesh incidence matrix C. Table 2.2 shows this matrix for the graph (and tree) in Fig. 2.1. Any element c_{ij} of C is 1,-1 or 0 if branch i has, respectively, the same direction as, the opposite direction to, or is not included in mesh j.

It is readily shown that

$$A^T C = 0 \text{ and } C^T A = 0$$

This brief discussion of graph theory leads to the consideration of its applications in pipe-network modelling. One of the best-known methods for solving network problems, which uses graph theory to model the network, is the Hardy Cross method.

2.4. Hardy Cross

This method is based on Kirchoff's laws

1. The algebraic sum of the flows at any pipe junction is zero (this is a statement of the mass balance rule).

2. The algebraic sum of the pressure drops around any mesh of the network is zero.

To employ the Hardy Cross method it is necessary to construct 'circuit equations', using the matrices described in the section on graph theory.

A set of basic meshes for the network is selected, the branch-node and branch-mesh incidence matrices, A and C, are constructed and used in conjunction with the equations describing pressure-drop in a pipe and mass-balance at a node, to solve for pressures and flows in the network. The solution can be obtained in two ways, using either the 'mesh' method or the 'nodal' method.

Both methods have been discussed by Barlow and Markland [2] who have suggested modifications for improving convergence by either method. In their discussion they consider a network having P pipes connected between N nodes at M of which the head is specified and at each of the remaining $(M-N)$ nodes the outflow Q_n from the network is specified (Fig.

2.2).

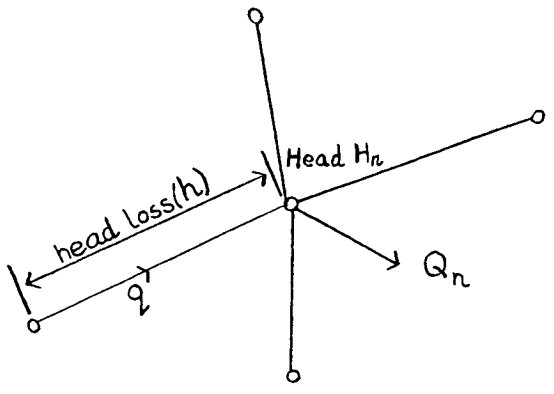


Fig. 2.2 Notation at a typical node of the network

In the mesh method, trial flowrates are assumed along each of the pipes, consistent with continuity of flow at the nodes. The head loss around each mesh is calculated with the assumed flowrates. In the unmodified Hardy Cross method, the flow corrections are applied one mesh at a time until Kirchoff's 2nd law is satisfied for all meshes. For network elements modelled by

$$\sigma_k = \alpha_k q_k |q_k|^{n-1} \tag{2.5}$$

where σ_k is head-loss through the element, q_k is flowrate and α_k depends on friction factor (and roughness, if the element is a pipe), the flow correction for mesh C_i is given by

$$\delta q_C = - \frac{\sum_{k \in C} \alpha_k q_k |q_k|^{n-1}}{\sum_{k \in C} n \alpha_k |q_k|^{n-1}} \tag{2.6}$$

where the summations are taken in a consistent direction around the mesh C_i. The exponent n is normally taken to be 2 (although it is somewhat less than 2 in the transition region of flow).

Rewriting eq.(2.6) in the simpler form of eq.(2.7), where ΔQ_C is the linear correction applied to the flowrate in each of the branches of a mesh around which the closing error in the head is ΔH_C .

$$\Delta Q_C = \frac{\Delta H_C}{\sum \left| \frac{nh}{q} \right|} \tag{2.7}$$

then Barlow and Markland's suggested improved correction may be expressed as

$$\Delta Q_C^2 \left[\sum_f \left| \frac{n(n-1)h}{2q^2} \right| - \sum_r \left| \frac{n(n-1)h}{2q^2} \right| \right] + \Delta Q_C \sum \left| \frac{nh}{q} \right| - \Delta H_C = 0 \tag{2.8}$$

where $\sum_f$ denotes summation over those branches in the mesh where the flow direction is in the direction in which the mesh is defined, and $\sum_r$ denotes summation over those branches where the flow is in the opposite direction. Assuming that $n = 2$ gives

$$\Delta Q_C^2 \left[\sum_f \left| \frac{h}{q^2} \right| - \sum_r \left| \frac{h}{q^2} \right| \right] + 2\Delta Q_C \sum \left| \frac{h}{q} \right| - \Delta H_C = 0 \tag{2.9}$$

Barlow and Markland state that over-correction, with a factor of about 1.25 has proved particularly valuable. They cite an example where, in a typical small network having 10 nodes and 13 pipes, the number of iterations was reduced from 13 to 7 when an over-correction factor of 1.2 was used. On much larger networks, values up to 1.4 have been used to advantage.

The major disadvantage of the mesh method is in the selection of basic meshes. The rate of convergence can be considerably affected by the set of meshes chosen. Barlow and Markland suggest choosing meshes directly from the network, according to general rules aimed at dispersing gross errors in flowrates rapidly over the whole network and removing local inaccuracies in pipes of subsidiary diameters. However this method is unsatisfactory when the networks are large and complex.

In the 'nodal' method, an assumed set of heads is successively corrected at each node in turn according to the expression

$$\Delta H_n = \frac{\Delta Q_n}{\sum \left| \frac{q}{nh} \right|^n} \tag{2.10}$$

In this, ΔH_n represents the increment of head H_n at the node in question and ΔQ_n is the amount by which the inflow rate along the pipes into the

node exceeds the specified outflow rate Q_n at that node.

Barlow and Markland's improved correction for the nodal method may be stated as

$$\frac{\Delta H_n^2}{8}\left[\sum_o\left|\frac{q}{h^2}\right| - \sum_i\left|\frac{q}{h^2}\right|\right] - \frac{\Delta H_n}{2}\sum\left|\frac{q}{h}\right| + \Delta Q_n = 0 \qquad (2.11)$$

where the expression $\sum_o$ indicates summation over all the pipes along which the flow is away from the node under consideration and $\sum_i$ indicates summation over all the pipes along which the flow is towards the node.

Barlow and Markland state that over-correction, using values in the range 1.1 - 1.3 again proves useful, but if too large a value is chosen, instability of the solution results. They also mention an extrapolation method. After a number of corrective iterations from an initial estimate, the total of flow errors at each of the nodes - $\sum|\Delta Q_n|$ - is compared with the initial total. The changes in the heads - $\sum\Delta H_n$ - are than extrapolated to make the total error equal to zero, on the assumption of linear dependence of errors and changes in heads. Barlow and Markland cite an example where, in a network having 14 nodes and 25 pipes, the number of iterations required for convergence is reduced from 55 without extrapolation to 27 with extrapolation.

In their survey of methods used in network flow analysis, Mah and Shacham [26] state that the general consensus is that the nodal method is slower than the mesh method, and Barlow and Markland mention slowness of convergence, when the network is ill-conditioned.

2.5. Network Transformations

Gay and Middleton [16] investigated methods of solution which appeared to be better adapted to computer techniques than the method of Hardy Cross. Their methods are based on the relationships which exist between nodal, branch and mesh quantities. The function of the previously described matrices A and C is to interrelate or transform these quantities. For example, if e' is the vector of nodal pressures (measured with reference to the datum node) then $e = A\,e'$ is the vector of branch pressure rises. Kirchoff's laws may be expressed using the relationships

$$A^T i = A^T C i' = 0 \tag{2.12}$$

$$C^T e = C^T A e' = 0 \tag{2.13}$$

where i is the vector of branch flows and i' is the vector of mesh flows.

Gay and Middleton visualised any branch of the network as consisting of three elements : an impedance element Z, a pressure source E and a flow source I For each branch the following relationships apply :

$$V = E + e \tag{2.14}$$

$$J = I + i \tag{2.15}$$

$$V = ZJ \tag{2.16}$$

$$J = YV \tag{2.17}$$

where E is the vector of branch pressure sources (i.e. pumps), I is the vector of branch flow sources, Z and Y are diagonal matrices and $Y = Z^{-1}$

From relationships (2.14)-(2.17) and the relationships between nodal, branch and mesh quantities, Gay and Middleton derived expressions for e', the vector of nodal pressures, and i', the vector of mesh flows.

$$e' = (A^T Y A)^{-1}(I' - A^T Y E) \tag{2.18}$$

$$i' = (C^T Z C)^{-1} C^T (E - ZI) \tag{2.19}$$

These two routes for a solution are analogous to the alternative methods developed by Hardy Cross. It can be seen that both require matrix inversion, which is a serious drawback for large networks. Gay and Middleton suggested that the computational problems could be reduced by applying the technique of diakoptics to the network problem. Diakoptics originated from the consideration of certain orthogonal transformations which could be applied to the network. The network may be converted to an all-node or an all-mesh network, described by the square transformation matrices C_1 or A_1, having dimensions $b \times b$ (where b is the number of branches).

Gay and Middleton considered an all-mesh network, that is, a network in which there are as many fictitious branches as there are non-datum

nodes in the original network. If i_p is the vector of 'primitive' branch flows ('primitive' signifying that each branch is connected to the datum node), then

$$i_p = C_1 \begin{vmatrix} I'_1 \\ i'_2 \end{vmatrix} = C_1 J' \tag{2.20}$$

where I'_1 and i'_2 are the flow vectors for an all-mesh network, I'_1 representing the nodal flow vector which is considered to be flowing in the fictitious meshes and i'_2 representing the mesh flows. From relationships (2.14)-(2.17) and the relationships between nodal (or fictitious mesh), branch and mesh quantities in an all-mesh network, two expressions for the nodal pressures may be derived.

$$e_1' = Y_1'^{-1}(I'_1 - Y'_2 E'_2) - E'_1 \tag{2.21}$$

$$e'_1 = (Z'_1 - Z'_2 Z_4'^{-1} Z'_3) I'_1 + Z'_2 Z_4'^{-1} E'_2 - E'_1 \tag{2.22}$$

in which the matrices E'_1 and E'_2 are partitions of E', the mesh pressure source vector for an all mesh network, into its nodal (or fictitious mesh) and mesh components. Y'_1 and Y'_2 are partitions of the admittance matrix Y' and Z'_1, Z'_2, Z'_3 and Z'_4 are partitions of the impedance matrix Z'.

2.6. Diakoptics

The method of diakoptics takes the concept of orthogonal transformations one step further. A transformation matrix can be constructed to relate any two systems containing the same number of equivalent branches. So, for two such systems, A and B,

$$J'_A = C_{AB} J'_B \tag{2.23}$$

where C_{AB} is the required transformation matrix.

The purpose of using the diakoptics method is to transform a network to an intermediate network, whose solution can be found, then to transform this solution into the solution of the given network. The matrix C_{AB} will be of a particularly simple form if the intermediate network is a 'cut' or 'torn' form of the original (the term 'cut' is used with reference to an 'all mesh' network which has been subdivided into different groups, and the term 'torn' indicates a similar subdivision of an 'all node' - or open

path - network). An 'all mesh' network may be 'cut' into two groups, containing 'cut segments' and 'cut branch segments'. The branch flows in the original 'all mesh' network and the new cut network are equated, giving rise to the transformation matrix C_{AB}. A solution may be found for the pressure vector, V'_A, in network A, which can then yield, by a further transformation, the vector of nodal pressures, e'_B for network B.

The work on orthogonal transformations and diakoptics was developed further by Gay and Preece in [17] and [18]. They examined the 'mesh' method of solution in both orthogonal transformations and diakoptics as an alternative to the 'nodal' method (i.e. solving the transformed networks for nodal pressures) presented by Gay and Middleton.

In the 'nodal' method the network is seen as consisting entirely of meshes. In the 'mesh' method the network is viewed as consisting of 'node to datum' (or open) paths. This leads to the construction of an incidence matrix B such that b_{ij} is equal to 1, -1 or 0 if branch i is included positively (directionwise), negatively, or not included in the node to datum path j. The positive direction of the node to datum path is away from the datum node.

Because of the referencing of the non-tree branches, the B matrix may be partitioned into tree and non-tree parts such that the non-tree part is a null matrix

$$B = \left|\begin{matrix} B_T \\ B_L \end{matrix}\right| = \left|\begin{matrix} B_T \\ 0 \end{matrix}\right| \qquad (2.24)$$

As in the nodal method the flows in the actual network may be related to those in the orthogonal network by the linear transformation.

$$J = \gamma J' \qquad (2.25)$$

This is a parallel to eq. (2.20). J is the vector of branch flows and J' is the vector of path flows. γ may be partitioned such that

$$\gamma = |B|C| \qquad (2.26)$$

where C is the branch-node incidence matrix, which may be further partitioned into tree and non-tree parts.

$$C = \left|\begin{matrix} C_T \\ C_L \end{matrix}\right| = \left|\begin{matrix} C_T \\ U \end{matrix}\right| \tag{2.27}$$

where U is a unit matrix.

In the mesh method, the mesh flows can be found from the following equation

$$I' = (C_T^T Z_T C_T + Z_L)^{-1}(E'_L - C_T^T Z_T B_T I') \tag{2.28}$$

where

$$E'_L = C_T^T E_T + E_L \tag{2.29}$$

I' is the vector of flows in the node to datum paths. Each node to datum path is assigned the flow which enters or leaves at the terminal node of the path. Z_T and Z_L are the tree and non-tree parts of the impedance matrix Z. E_T and E_L are the tree and non-tree parts of E, the matrix of branch pressure sources.

For networks which contain fewer meshes than branches, the matrices handled are smaller in the mesh method than in the nodal method (in which the numbers of branches and meshes are equal), due to the transformation used. The results of a program written by Gay and Preece to compare the nodal and mesh methods confirmed that the mesh method was faster for networks of this kind.

In the mesh method as it is applied to diakoptics, the relevant transformation matrix (corresponding to γ in eq. (2.25)) is α, where

$$\alpha = (\gamma^T)^{-1} \tag{2.30}$$

and

$$\alpha = |A|F| \tag{2.31}$$

A is the branch-node incidence matrix described previously and F is the non-tree branch-mesh incidence matrix. F may be partitioned into tree and non-tree parts.

$$F = \begin{vmatrix} 0 & T \\ U & L \end{vmatrix} \tag{2.32}$$

where 0 is the null matrix.

In Gay and Middleton's presentation of diakoptics using the nodal method, an all mesh network was 'cut' into 'cut segments' and 'cut branch segments'. In the mesh method, the network may be 'torn' into subnetworks. This is accomplished by the removal of combinations of tree and non-tree branches, provided that these combinations form continuous paths. Figs 2.2 and 2.3 show what is meant by 'tearing' the network.

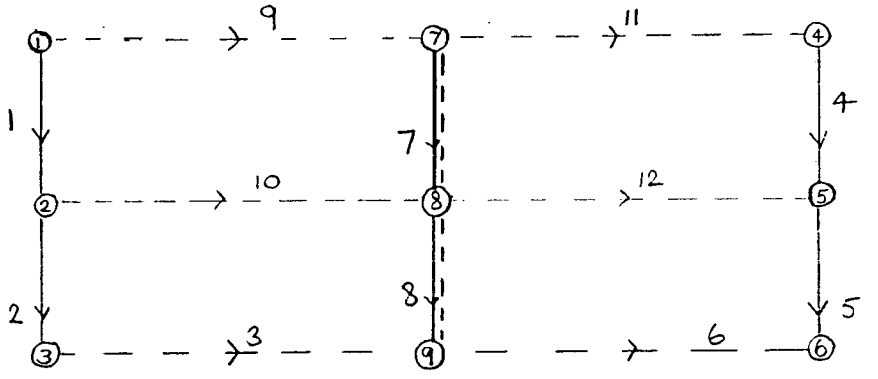


Fig. 2.2 - The connected network that is to be torn apart by the removal of the branches indicated by the heavy dashed line.

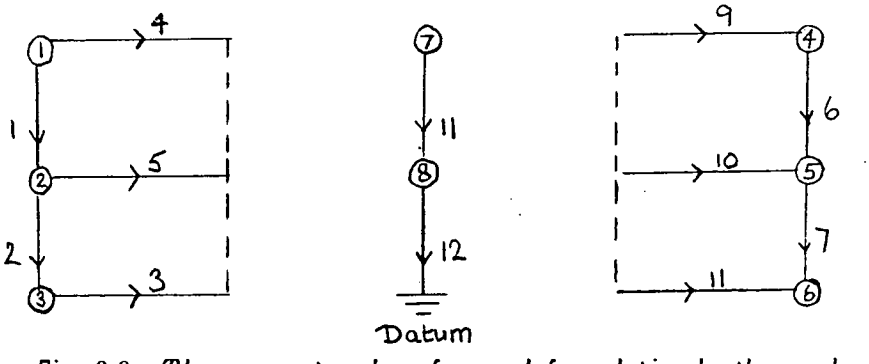


Fig. 2.3 - The torn network, referenced for solution by the mesh method

Each subnetwork is viewed as a collection of node-to-datum and mesh paths, while the removed branches are viewed as the tree of an additional subnetwork.

As was the case in orthogonal transformations, use of the mesh method in diakoptics leads to a solution for the network mesh flows, i', by using relationships (2.14) to (2.17) and the appropriate transformation

matrix α. Again, if the network is not heavily meshed, the mesh method as it has been tested by Gay and Preece, proves to be more effective. In both the nodal and mesh methods, the relevant authors suggest 'rule of thumb' techniques for cutting and tearing the network.

Gay and Middleton in their study compared the diakoptics method with that of Hardy Cross for a network containing 22 nodes and 38 branches. The ease with which network alteration effects could be tested using each method was investigated. They concluded that the diakoptics method converged faster and that changes in the network were more readily investigated using this method. However, they found that apparently small changes in the network affected the solution time considerably.

2.7. Other Matrix Methods

Further matrix methods for the solution of pipe network problems have been suggested by Mah and Shacham [26]. They used graph theory to investigate the possibility of grouping vertices (nodes) in such a way as to yield an advantageous formulation.

They used the concept of 'cut-sets' (not to be confused with 'cutting' in diakoptics) of a network, in their analysis. In a connected graph (described in the section on graph theory), a cut-set is a minimal set of branches whose deletion from the graph separates some vertices from others (resulting in an increase in the number of connected subgraphs). If the cut-set contains only one branch then it is called a bridge. In the case of a tree, every branch is a bridge – its removal creating two subgraphs containing subsets of vertices, V_A and V_B. Considering the graph again as a whole, the two subgraphs containing V_A and V_B are linked by a unique cut which contains one tree branch together with possibly some non-tree branches (or chords as they are referred to by Mah and Shacham). There are (N-1) such cuts corresponding to the (N-1) tree branches. If an incidence matrix is constructed based on branches incident with each of the (N-1) vertex (or node) subsets, V_A, V_B etc., then the cut-set matrix K is obtained. If the graph is undirected, i.e. no directions assumed for branches, then $K = \widetilde{K}$, where

$$\tilde{K} = |I_{N-1}|\tilde{B}| \tag{2.33}$$

I_{N-1} is the identity matrix of order (N−1) and $\tilde{B}$ is an (N−1) × m binary matrix corresponding to the chords (m is the number of chords).

The cut-set matrix $\tilde{K}$ is related to material balances around the (N−1) vertex subsets. The rows of $\tilde{K}$ are linear combinations of columns of $\tilde{A}$ (where A is the branch-node matrix as before – $\tilde{A}$ signifies that the graph is undirected).

Mah and Shacham state that, for an undirected graph

$$\tilde{K}\tilde{\Gamma}^T \cong 0 \quad \text{and} \quad \tilde{\Gamma}\tilde{K}^T \cong 0 \tag{2.34}$$

and

$$\tilde{B} \cong \tilde{T}^T \tag{2.35}$$

where $\tilde{\Gamma}$ is $\tilde{C}^T$ ($\tilde{C}$ is the branch-mesh incidence matrix for an undirected graph). $\tilde{\Gamma}$ may be expressed as

$$\tilde{\Gamma} = |\tilde{T}|I_C| \tag{2.36}$$

where I_C is an identity matrix of order C and $\tilde{T}$ is a m × (N−1) binary matrix corresponding to the tree branches.

For a directed graph, the parallel equation to eq. (2.35) is

$$B = -T^T \tag{2.37}$$

This relationship shows the link between basic meshes and cut-sets of a graph. Thus it may be argued that a spanning tree provides a convenient starting point for formulating a consistent set of governing equations for network problems.

If the flow rates associated with the tree branches and chords are denoted by q_T and q_C respectively, then the material balance for the (N−1) vertex subsets may be stated as

$$\hat{K}\hat{q} = I_{N-1}q_T - \hat{T}^T q_C = w \qquad (2.38)$$

where a component, w_i, of vector w represents the net output from the vertex subset V_{A_i}. The matrices $\hat{K}$ and $\hat{T}$ refer to 'internal' branches of the network only, i.e. any branches associated with external inputs and outputs to the network are ignored. From eq. (2.38)

$$q_T = w + \hat{T}^T q_C \qquad (2.39)$$

$$\hat{q} = \begin{pmatrix} q_T \\ q_C \end{pmatrix} = \begin{pmatrix} w \\ 0 \end{pmatrix} + \begin{pmatrix} \hat{T}^T \\ I_C \end{pmatrix} q_C = w' = \hat{\Gamma}^T q_C \qquad (2.40)$$

where w' is the vector of input/output flows. Eq. (2.40) gives, in effect, an expression for mesh flows.

Mah and Shacham do not cite any examples of the performance of this solution method in comparison with diakoptics. In both cases a spanning tree is the starting point for the problem formulation. However, for a given spanning tree, network equations may be more easily obtained via Mah and Shacham's method than by the more arbitrary cutting and tearing techniques of diakoptics. However, changes to the network are less readily investigated with the former method than with diakoptics.

2.8. Linearisation Methods

2.8.1. Bending and Hutchison Method

Bending and Hutchison [4] developed a method for calculating steady-state flows in networks of pipes and pumps, which they called the linearisation method. It is simpler in conception than Hardy Cross or diakoptics, requires smaller computation times and is more general, in that certain design-type calculations can be undertaken. Examples of these are problems in which input and output flowrates are determined so as to satisfy nodal pressure specifications.

For a network containing X_p pipes, X_n nodes, X_{pu} pumps and X_I input/outputs, the linearisation method involves the construction of the following set of equations.

(a) Mass balance over each node j

$$\sum_{i \in G_j} A_i V_i - \sum_{i \in R_j} A_i V_i + \sum_{i \in H_j} Q_i - \sum_{i \in S_j} Q_i - \sum_{i \in I_j} F_i = 0 \qquad (2.41)$$

X_n equations

(b) Pressure drop for each pipe

(i) If the flow is turbulent

$$P_k - P_l = 4C_i \rho \left(\frac{L}{D}\right)_i |V_i| V_i$$

(ii) If the flow is laminar

$$P_k - P_l = 32 \mu \left(\frac{L}{D^2}\right)_i V_i \qquad (2.42)$$

X_p equations

(c) Specified pressure drop for some (maybe all) pumps

$$P_k - P_l = x \qquad (2.43)$$

N_{pu} equations

(d) Specified input (or output) flowrates

$$F_i = x \qquad (2.44)$$

N_I equations

(e) Additional nodal pressure specifications – sufficient to completely define the problem

$$P_i = x \qquad (2.45)$$

$X_{pu} + X_I - N_{pu} - N_I$ equations

In (a) the sum for each member i of a set G_j is indicated by $\sum$. In (b) and (c) subscripts k and l refer to input and output nodal pressures.

The above set of equations is linear except for eq. (42.i). If an initial guess $V_i^{(0)}$ for the velocity is available, eq. (42a) can be rewritten as :

$$P_k - P_l = 4C_i \rho \left(\frac{L}{D}\right)_i |V_i^{(0)}| V_i^{(1)} \qquad (2.46)$$

This new set of equations is now linear and can be solved to obtain new values of pipe velocities, the process being repeated until convergence is attained. After each iteration the pipe velocities are taken to be the mean of the previous value and the calculated value. Bending and Hutchison

introduced this relaxation method, since the basic algorithm converges very slowly, due to the fact that pipe velocities oscillate about their basic values.

A problem arises from the size of the set of linear equations that need to be solved at each iteration. Clearly sparse matrix methods must be used if the linearisation method outlined by Bending and Hutchison is to have an advantage over the earlier mesh methods.

Bending and Hutchison applied their method to the network of Gay and Middleton [16] containing 22 nodes, 38 pipes and 6 input/outputs, and to variations of this network. They found that (for Gay and Middleton's network) the linearisation method gave faster convergence than the diakoptics method.

One conclusion which they reached in their analysis was that convergence does not seem to depend greatly on the network but only on the type of flow existing. Thus it may be stated that usually laminar flow problems converge in two iterations and mixed flow problems converge in 10–13 iterations.

The data input requirements of the linearisation method are more simple than for Hardy Cross or diakoptics. Also, although changes in the network topology require a complete recalculation, computation times of the order of only 3 seconds make this no great disadvantage.

2.8.2. Newton–Raphson Method

This method is based on the Taylor expansion of $f(x)$ about the kth iterate, x_k.

$$0 = f(x^*) = f(x_k) + \left(\frac{\partial f_i}{\partial x_j}\right)(x^* - x_k) + \ldots \qquad (2.47)$$

Neglecting higher order (nonlinear) terms in x^*-x_k and replacing x^* by the $(k+1)$th approximation, x_{k+1}, results in:

$$x_{k+1} = x_k - \left(\frac{\partial f_i}{\partial x_j}\right)^{-1} f(x_k) \tag{2.48}$$

The Jacobian, $\left(\frac{\partial f_i}{\partial x_j}\right)$, is to be evaluated at $x = x_k$

The chief merit of this method is its rapid rate of convergence starting with a set of good initial guesses.

The Newton-Raphson linearisation per se has been applied to the mesh formulation of a network by Lang and Miller [23]. They state that for the solution to converge there must be no discontinuities in calculated pressure drop or in the rate of change of pressure drop with flowrate. Many practical piping networks have laminar or transitional flow in cross-over piping between major flow streams which are turbulent so the friction-factor correlation must be smooth and continuous. Lang and Miller use the Churchill correlation. (Friction factor correlations are discussed in the section on modelling of pipeline network elements).

Referring to Fig. 2.4 the condition for a net zero pressure-drop around loop A may be stated as

$$\sum_i \xi_{A,i}\, \phi_i \left[m_i + \sum_{l'} \xi_{l',i} \sigma m_{l'}\right]^{n_i} = 0 \tag{2.49}$$

where $\sigma m_{l'}$ is the correction to flow in any given loop l', and ξ denotes the direction of flow in a pipe relative to the loop flow. For each loop the pressure condition may be expressed as the forcing function F_l where

$$F_l = \sum_l \phi_i m_i^{n_i} = 0 \tag{2.50}$$

m_i is the mass flow in each pipe i in the loop l, and ϕ_i and n_i are functions of the friction factor, f.

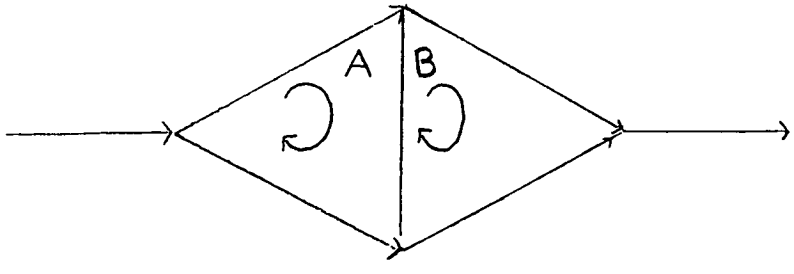


Fig. 2.4 Two-loop network

In order to use the Newton-Raphson linearisation, the F_l and σm_l

values are expressed in vector notation.

$$\vec{F} = (F_A, F_B) \tag{2.51}$$

$$\vec{\sigma m} = (\sigma m_A, \sigma m_B) \tag{2.52}$$

The solution procedure of Lang and Miller generates σm_i for the iteration $j+1$ by correcting σm_i from the previous iteration.

$$_{j+1}\vec{\sigma m} = {}_j\vec{\sigma m} - D^{-1}\vec{F} \tag{2.53}$$

where D is the Jacobian matrix whose elements are defined by

$$D_{i,i'} = \frac{\delta F_i}{\delta \sigma m_{i'}} \tag{2.54}$$

For the two loops in Fig. 2.4, the Jacobian is

$$D = \begin{vmatrix} \frac{\delta F_A}{\delta \sigma m_A} & \frac{\delta F_A}{\delta \sigma m_B} \\ \frac{\delta F_B}{\delta \sigma m_A} & \frac{\delta F_B}{\delta \sigma m_B} \end{vmatrix} \tag{2.55}$$

When $\sigma m = 0$, the iteration is complete.

Lang and Miller claim that this procedure has proved very reliable for analysing pipe networks involving all flow regimes. It does however require mesh selection prior to the calculation proper.

Mah and Shacham [26] state that the Newton-Raphson method also lends itself very readily to sensitivity analysis, that is, analysis of the way in which a network system behaves when certain specifications such as delivery pressures or nodal flow rates are changed. For many situations it is sufficient to determine the approximate behaviour from sensitivity information based on linearized approximation in the neighbourhood of the original design solution. Such an analysis is most readily carried out when the Newton-Raphson method is used for the steady-state solution of the network concerned.

If the network specifications and parameters are collectively denoted by u and the state variables are denoted by x, then the steady-state pipeline network equations are

$$f(x, u) = 0 \tag{2.56}$$

The effect of varying 'u' and at the same time satisfying eq. (2.56) is given by

$$df = \left(\frac{\partial f}{\partial x}\right)_u dx + \left(\frac{\partial f}{\partial u}\right)_x du = f_x dx + f_u du \tag{2.57}$$

or

$$\left(\frac{\partial f}{\partial x}\right)_u \left(\frac{\partial x}{\partial u}\right) = -\left(\frac{\partial f}{\partial u}\right)_x \tag{2.58}$$

where $(\partial x/\partial u)$ is an $(n \times m)$ matrix of partial derivatives of the n state variables with respect to the m 'external' variables (network specifications), referred to by Mah and Shacham as the sensitivity matrix. If the Newton-Raphson method is used to solve the network, the Jacobian matrix $(\partial f/\partial x)_u$ is already available.

Mah [25] uses the Newton-Raphson method, along with algorithms for node-arc reassignment and cycle selection, to solve mesh-formulated network problems. This formulation is solved by the product form of the inverse (a description is given in the section on matrix methods).

2.9. Other Work

Wood and Thorley [34] have written a BASIC computer program for pressure and flow analysis in pipe networks, which includes extended period simulations. They employ an algorithm called the SP method for the solution of the mesh equations (they use the mesh formulation of a network). Their algorithm makes use of gradient methods to handle the nonlinear flowrate terms in the pressure-flowrate equation for each network element. The algorithm is similar to the Hardy Cross mesh method except that corrections are applied to all meshes simultaneously instead of sequentially (SP stands for simultaneous path adjustment).

Pipeline network problems may, in principle, be solved by transient solution methods after allowing sufficient time steps for the solution to reach steady-state. Nahavandi and Catanzaro [27] made a comparison of a transient solution method with the Hardy Cross method of balancing flows.

For the particular 35-node and 45-branch hydraulic network problem tested, the transient solution method took 108 seconds compared with 134 seconds required by the Hardy Cross method.

Isaacs and Mills [22] have developed a linear theory method which is suitable for implementation on a mini- or microcomputer because the algorithm is simple. It uses a similar linearisation strategem to the Bending and Hutchison method. The nodal pressures are solved for simultaneously and the flows are then found from the flow-pressure equations using the calculated pressures. The matrix on the left hand side of the equations contains the coefficients for each branch which is involved in a particular nodal flow balance (or equation). The solution method used is iterative and at each step sucessive over-relaxation is used to solve the current set of equations.

The authors state that initialisation presented no problem, and that initial flow guesses ranging from 0.001 m^3/s to 1 m^3/s were used, without affecting the solution. They recognise the problem of zero flow and say that when the pressure drop across a branch is very low, that branch should be removed from the network.

A program for the analysis of flow networks was written by G.M. Alder [1] at Edinburgh University. The program runs interactively and the user has the option of using either the Hardy Cross method or the Newton method. The commands available include SOLVE to find the steady-state flows and pressures for the present network, or ADJUST to change the pipe diameters according to the pressure and flow requirements at the discharge nodes.

Chandrashekar [8] has written a program to analyse hydraulic networks consisting of pipes, pressure-reducing valves, non-return valves and booster pumps. The Newton-Raphson solution procedure is employed with the Hazen-Williams pipe pressure loss equation to find the nodal pressures. The program has been used to analyse several networks and the author claims that if several valves are present a correct solution may not be given, and problems of oscillation or slow convergence may also arise. Chandrashekar and Stewart [9] state that Newton's method is the fastest method for flow networks, but the step at which the inverse of the

Jacobian matrix is calculated is time-consuming. They observe that the fraction of non-zeros in the Jacobian may be as low as 2-5%, and they describe an LU decomposition method which takes advantage of this sparsity. The method required 10 iterations and 3 seconds for a network with 191 nodes and 287 pipes.

A large proportion of pipeline flow analysis methods are for hydraulic applications, but could be easily modified for compressible flow situations. Hutchison [21] has written a program for the simulation of steam distribution networks which is based on the linearisation method of Bending and Hutchison. Facilities are included for calculating steam properties and also for handling incondensible gases.

R. Liebe [24] has developed a method for finding the steady-state energy and flow distribution in arbitrary networks where, in addition to pumps, pipes and valves, the network contains components for the generation, transfer and removal of heat. In such networks the nodal properties are temperature and enthalpy, and in branches the quantities of interest are heat flow and fluid flow. Such networks are described by sets of coupled, partially non-linear equations. Liebe's method derives from an equivalent network model which employs lumped properties and quantities for nodes, branches and components - he describes the model as a 'discrete structure' model. He uses a Taylor expansion to linearise the fluid velocity/enthalpy relationship in all network branches ; the network as a whole is described by a set of equations in which nodal flow balances are expressed in terms of nodal enthalpies. Liebe's method uses an overall Gauss-Seidel iteration procedure to obtain nodal enthalpies. After each iteration step, a new coefficient matrix for nodal enthalpies is obtained. The elements of the coefficient matrix are further updated by an 'improvement step' before the next iteration. The values obtained in this improvement step are derived from the above-mentioned Taylor expansion of the branch fluid velocity/enthalpy relationship, using the current and previous values of enthalpy at the branch end-nodes. Within the overall Gauss-Seidel iteration procedure for nodal enthalpies, there are two sequential iteration procedures ; the first obtains the network heat-flow distribution by solving for nodal temperatures, and the second iteration procedure obtains the network fluid-flow distribution by solving for nodal enthalpies. Liebe claims

that the method has low sensitivity towards physical or numerical ill-conditioning, due to the formulation of nodal equations using scalar energy (enthalpy) type unknowns ; the coefficient matrix for nodal enthalpies is positive and diagonally dominant (this aspect is further discussed in the section on 'Iterative Methods'). Liebe cites a use of the method in the design optimization of an air coolant distribution system in a large, prototype water-wheel-generator. He states that it took only 5 to 10 iterations to produce nodal residual flows which were 1-2 % of the net nodal flow.

The field of dynamic modelling is not so well developed as that of steady-state, but commercial programs for unsteady-state analysis do exist. An example is the Pan network analysis program developed by Goldwater, Rogers and Turnbull [20] for the analysis of gas distribution networks. Bender [3] has developed a mathematical model for simulation of dynamic gas flows in networks including control loops. He uses the Lax-Wendroff scheme to solve the coupled hyperbolic partial differentiation equations which arise in the dynamic model.

2.10. Modelling of Pipeline Network Elements

An important factor in the analysis of pipeline network problems is the modelling of network elements, including pipes, pumps and various types of valves. The following subsections describe methods for modelling such elements.

2.10.1. Pipes

The modelling of pipes in network flow analysis is concerned mainly with the choice of friction factor correlation. If there are discontinuities in the pressure-drop equation used over different flow regimes this can lead to convergence difficulties.

Estimation of friction factor is usually done by using the Moody friction factor chart which is made up of the following equations. For laminar flow with $Re < 2100$, the Hagen-Poiseuille equation is used :

$$f_D = \frac{64}{Re} \tag{2.59}$$

where f_D is the Darcy friction factor which is four times the Fanning friction factor f_f.

For fully developed turbulent flow in smooth pipes with $3000 < Re < 3.4\times10^6$, Prandtl's equation is

$$\frac{1}{\sqrt{f_D}} = 2.0\log_{10}(Re\sqrt{f_D}) - 0.8 \tag{2.60}$$

For fully developed turbulent flow in rough pipes with $\left[(D/\varepsilon)/(Re\sqrt{f_f})\right] > 0.01$, Von Karman's equation is :

$$\frac{1}{\sqrt{f_D}} = 2.0\log_{10}\left(\frac{D}{\varepsilon}\right) + 1.74 \tag{2.61}$$

where ε is pipe roughness and D is diameter.

For transition flow where the friction factor varies with both Reynolds number and (ε/D), Colebrook's equation is the most commonly used :

$$\frac{1}{\sqrt{f}} = 1.74 - 2\log\left(\frac{2\varepsilon}{d} + \frac{18.7}{Re\sqrt{f}}\right) \tag{2.62}$$

This equation is valid up to a value of $\left[(D/\varepsilon)/Re\sqrt{f_f}\right] = 0.01$.

In fact the Colebrook equation covers the fully developed flow regions for smooth and rough pipes, as well as the transition region. However it is an implicit equation and requires iteration . Various explicit equations have subsequently been proposed. Chen **[11]** compared two of the explicit equations, the Wood and the Churchill, with his own suggested equation and concluded that the latter gave best agreement with Colebrook over a Reynolds number range of 4000 to 4×10^8 and a roughness ratio, (ε/D), range from 0.05 to 5×10^{-7}.

Chen's equation is :

$$\frac{1}{\sqrt{f_D}} = -2.0\log\left[\frac{\varepsilon}{3.7065D} - \frac{5.0452}{Re}\log\left(\frac{1}{2.8257}\left(\frac{\varepsilon}{D}\right)^{1.1098} + \frac{5.8506}{Re^{0.8981}}\right)\right] \tag{2.63}$$

2.10.2. Pumps and Compressors

With regard to the modelling of pumps, Gostoli and Spadoni [19] have extended the linearisation method of Bending and Hutchison to include pumps with variable head.

It is usual practice to represent the head-capacity curve of a centrifugal pump by a polynomial :

$$H = f(Q) = aQ^2 + bQ + h_0 \tag{2.64}$$

Gostoli and Spadoni propose that a linear characteristic equation be used to model the pump.

$$H = f(Q) = E_h - Z_h Q \tag{2.65}$$

in which

$$Z_h = \frac{f(Q_h)}{Q_0 - Q_h} \tag{2.66}$$

($E_h = Z_h Q_0$ and $0 < Q_h < Q_0$
(Q_0 is the maximum pump throughput)

Eq. (2.66) can be regarded as the equation of a linear element with a source E_h and impedance $Z_h > 0$. According to electrical network theory the network with a linear pump has thus a unique solution Q_{h+1}, and this can be found by the linearisation method.

Gastoli and Spadoni have used this linear pump model successfully in the solution of networks including several pumps. They state that singularities are never encountered until the impedances Z_h of the pumps are positive, and this is generally true in a wide range of flows.

Wood and Thorley [34] in their program for the analysis of pipe distribution systems allow a pump to be specified in two ways for data input. The useful power a pump puts into the system can be specified. This method of describing a pump is useful for a preliminary analysis or design when the specific characteristics of the pump are not known.

Alternately a pump can be described by points of operating data input to the program. An exponential curve is fitted to this data to obtain a

pump characteristic curve describing the pump operation, of the form :

$$E_p = H_1 - CQ^n \tag{2.67}$$

where E_p is the pump head, Q is the flowrate and H_1 is the pump shut-off head. C and n are determined by passing the curve through two points of operating data supplied to the program. The program handles out of range pump operation as follows. If flow reversal occurs then the pump operates at shut-off head. If the solution indicates that the pump is operating at a flowrate above that of the highest flowrate supplied in the input data, then the pump operates on a straight line with equation $E_p = A - SQ$, where S is the gradient of eq. (2.67) at the highest flowrate value supplied.

For compressors, Mah and Shacham **[26]** state that the modelling equation most commonly used is

$$q_{ij} = \frac{h_p}{\alpha_0\left(\frac{p_j}{p_i}\right) - \alpha_2} \tag{2.68}$$

where h_p is the compressor horsepower, q_{ij} is the flow through the compressor,p_i and p_j are the input and output pressures respectively and α_0, α_1 and α_2 are constants.

2.10.3. Pipe fittings

For pipeline fittings such as bends, valves, expansions and contractions, the head loss, h_L, is evaluated from

$$h_L = \frac{K_L V^2}{2g} \tag{2.69}$$

where K_L is the loss coefficient and V is a characteristic velocity in the fitting. If the network as a whole is modelled by linearised pipeline 'element' equations then eq. (2.69) will also be linearised according to the method used (Bending and Hutchison, Newton-Raphson, etc).

Eq. (2.69), however, applies only to the turbulent region of flow. Little is known about the behaviour of loss coefficients for pipe fittings in the laminar region. Edwards, Jadallah and Smith **[15]** have investigated this area and proposed that it is possible to present fittings loss data as

relationships between the loss coefficient and a generalised Reynolds number.

They performed experiments with various pipe fittings, including elbows, gate valves, 1 and 2 inch globe valves, sudden contractions and expansions and orifice plates. In all cases they were able to present their results in the form, $K_L = C/Re$.

2.10.4. Miscellaneous Pipeline Elements

Another common set of devices used as pipeline elements are pressure regulators. These are of two types : the downstream regulators (or pressure reducing valves) and the upstream regulators (or pressure retaining valves). Mah and Shacham **[26]** state that the idealized downstream regulator may be modelled by :

$$p_j = min(p_i, p_s) \tag{2.70}$$

where p_s is the regulator set-point pressure. The valve is closed when $p_j > p_i$ (p_i and p_j are the inlet and outlet pressures respectively). For the idealized upstream regulator,

$$p_i = max(p_j, p_s) \tag{2.71}$$

and the valve is closed when $p_i < p_s$.

Wood and Thorley **[34]** modelled these two types of regulators in their network flow analysis program. The downstream regulator was modelled as two nodes : at the upstream node the flow demand is set (within the program) equal to the flow through the regulator itself. The downstream node is a fixed pressure node in which the pressure is set equal to the set pressure plus the head due to elevation of the regulator. If flow reversal through the regulator occurs, a designated check valve downstream from the regulator will close. For the case of a pressure retaining valve, Wood and Thorley's simulation uses flow in reverse through a pressure reducing valve and the valve can operate in three modes :

1. The valve is fully open and the upstream pressure is above the set value.

2. The valve is throttled and the upstream pressure is regulated at the set value.

3. The valve is closed and the upstream pressure drops below its set value but cannot be controlled by the valve.

Wood and Thorley say that if the operation mode is unknown, then two simulations will be required to check all three possibilities.

In their program they also include the facilities to model variable pressure sources, storage tanks and pressure switches. These can be described briefly as follows.

Variable pressure source - as an example suppose a pressure main at 200 metres elevation has the following flowrate-pressure characteristics.

flowrate(litres/s)	available pressure(kPa)	head increase(m)
0	1000	102
18	690	70
25	572	58

This can be simulated by a feed line with a pump connected to a reservoir at elevation 200m. The pump characteristics are described by the flowrate-head data shown above where the head represents the pressure head of the source for the associated flowrate. This representation will simulate a variable pressure source which operates on a curve which passes through three specified points.

Storage tanks - represented by a fixed pressure node with the pressure specified as that due to the elevation of the fluid surface. For an extended period simulation the tank characteristics must be specified.

Pressure switches - this feature is used in extended period simulations and allows the open-closed status of lines to be controlled by the head at a specified node.

2.11. Sparse Matrix Methods

2.11.1. Gaussian Elimination

Most sparse matrix methods are derived from Gaussian elimination, so a brief description of the method is given.

The equations to be solved are :

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n &= b_n \end{aligned} \tag{2.72}$$

The first equation is stored for later use and the variable x_1 is eliminated from the remaining n−1 equations by subtracting an appropriate multiple of the other equations. If the original coefficients are given the notation

$$a_{ij}^{(1)} = a_{ij} \qquad i, j = 1,2, \ldots ,n \tag{2.73}$$

$$b_i^{(1)} = b_i \qquad i = 1,2, \ldots ,n \tag{2.74}$$

then the new coefficients are found by using the multipliers

$$m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} \qquad i = 2, 3, \ldots ,n \tag{2.75}$$

and forming the new elements

$$a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i1}a_{1j}^{(1)} \qquad \begin{matrix} i = 2,3, \ldots ,n \\ j = 1,2, \ldots ,n \end{matrix} \tag{2.76}$$

$$b_i^{(2)} = b_i^{(1)} - m_{i1}b_1^{(1)} \qquad i = 2,3, \ldots ,n \tag{2.77}$$

In this way, the first variable, x_1, is eliminated in the last n−1 equations.

If this procedure is repeated a further n−2 times, the remaining equation will have only one unknown and can be solved very easily.

At each stage in the process when the variable x_k is to be eliminated the multipliers formed are :

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \qquad i = k+1, k+2, \ldots ,n \tag{2.78}$$

and the new elements formed are :

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik}a_{kj}^{(k)} \qquad \begin{matrix} i = k+1, k+2, \ldots, n \\ j = k, k+1, \ldots, n \end{matrix} \tag{2.79}$$

$$b_i^{(k+1)} = b_i^{(k)} - m_{ik}b_k^{(k)} \qquad i = k+1, k+2, \ldots, n \tag{2.80}$$

The result of this elimination process is an upper-triangular set of equations given by

$$\begin{aligned} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \ldots + a_{1n}^{(1)}x_n &= b_1^{(1)} \\ a_{22}^{(2)}x_2 + \ldots + a_{2n}^{(2)}x_n &= b_2^{(2)} \\ a_{nn}^{(n)}x_n &= b_n^{(n)} \end{aligned} \tag{2.81}$$

where all the elements below the diagonal are zero. It is easy to solve these equations by a process of back substitution. The last equation has the solution

$$x_n = \frac{b_n^{(n)}}{a_{nn}^{(n)}} \tag{2.82}$$

and this value can then be substituted in the next lowest equation to give x_{n-1}. By working back up the equations the values of all the variables can be calculated.

The basic method can be improved upon by partial pivoting and scaling. The aim of partial pivoting is to minimize the build-up of errors. From Eqs. (2.79) and (2.80) it can be seen that one operation which occurs many times is multiplication by m_{ij}. In multiplying the number, any accumulated error which is present will also be multiplied by m_{ij}, therefore these multipliers should be made as small as possible, and certainly less than one, so that the errors are not magnified by the multiplication.

This can be achieved if the pivotal element $a_{kk}^{(k)}$ is the largest of all the elements $a_{ik}^{(k)}$ in the same column for $i > k$ since then

$$| m_{ij} | \leq 1 \qquad j < i \; ; \quad i = 2, \ldots, n \tag{2.83}$$

The partial-pivoting strategy on its own is inadequate ; the matrix should be scaled so that the rows are comparable in some defined way. This is usually done by normalizing in one of two ways. The rows can either be normalized by dividing the whole row by the element in the row which has the largest modulus so that the largest element of the new row is one, or

alternatively, each row can be divided by

$$d_i = \sqrt{\left(\sum_{j=1}^{n} a_{ij}^2\right)} \qquad (2.84)$$

Williams [32] states that although it is established that scaling can make a significant difference to the accuracy of the solution, there is no standard method of scaling which is universally accepted.

There are several variants of the standard Gaussian elimination method. In the Jordan elimination scheme the final form of the matrix after elimination is a diagonal form, in which each equation has only one variable. Therefore the back-substitution process is avoided and the values of the variables can be calculated directly. However Jordan elimination needs approximately $n^3/2$ operations compared to $n^3/3$ for Gaussian elimination.

There is another group of methods which can be described under the general heading of triangular decomposition ; these include the methods of Crout and Choleski. The computational scheme is based on a series of multipliers which reduce the matrix to triangular form followed by the process of back substitution. Reduction to triangular form means that matrix A can be expressed as :

$$A = L \,.\, U \qquad (2.85)$$

where U is an upper-triangular matrix and L is a lower-triangular matrix. Once these matrices have been found the set of equations is solved in two stages of forward- or back-substitution. If :

$$A\,X = L \,.\, U.\, X = B \qquad (2.86)$$

vector Y is found such that $L.Y = B$, then the equations $U.X = Y$ are solved.

The number of operations is the same as for Gaussian elimination.

In the case of a symmetric matrix it is possible to reduce the amount of computation and storage by taking advantage of the symmetry. If the diagonal elements of L and U are made equal then $U = L^T$ and only the elements of L need be calculated or stored. This is known as Choleski

factorization.

2.11.2. Product Form of the Inverse

This is also a matrix factorization and has been applied in one instance [25] to the solution of pipeline network problems after node and mesh reordering algorithms have been applied.

The following description of the PFI is taken from Brameller [6].

For the equation

$$A\ X = b \tag{2.87}$$

the solution is

$$X = A^{-1}\ b \tag{2.88}$$

In the product form of the inverse, A^{-1} is given by :

$$A^{-1} = T_n \ldots T_3 T_2 T_1 \tag{2.89}$$

The steps required to achieve this result can be illustrated by considering the following 3rd-order problem.

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix} = \begin{vmatrix} b_1 \\ b_2 \\ b_3 \end{vmatrix} \tag{2.90}$$

The elements below the diagonal element of the first column are eliminated by pre-multiplying the coefficient matrix A by a transformation matrix, T_1, where

$$T_1 = \begin{vmatrix} \frac{1}{a_{11}} & 0 & 0 \\ -\frac{a_{21}}{a_{11}} & 1 & 0 \\ -\frac{a_{31}}{a_{11}} & 0 & 1 \end{vmatrix} \tag{2.91}$$

This operation gives a new matrix $A^{(1)} = T_1 A$ where :

$$A^{(1)} = \begin{vmatrix} 1 & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} \end{vmatrix} \tag{2.92}$$

The elements a_{ij} of $A^{(1)}$ are obtained by the method used in Gaussian elimination, i.e.

$$a_{ij}^{(1)} = \frac{a_{i1}a_{1j}}{a_{11}} \qquad \begin{matrix} i = 2, \ldots, n \\ j = 2, \ldots, n \end{matrix} \tag{2.93}$$

Therefore eq. (2.87) has been transformed into a related set of equations which can be expressed as

$$A^{(1)}X = T_1 A X = T_1 b \tag{2.94}$$

This process can be continued using the second diagonal element of the new matrix $A^{(1)}$ as a pivot. Using the same technique, the off-diagonal elements of the second column of $^{(1)}$ can be reduced to zero and the diagonal element made unity by pre-multiplying the matrix $A^{(1)}$ by a transformation matrix T_2, where :

$$T_2 = \begin{vmatrix} 1 & \frac{a_{12}^{(1)}}{a_{22}^{(1)}} & 0 \\ 0 & \frac{1}{a_{22}^{(1)}} & 0 \\ 0 & \frac{a_{32}}{a_{22}^{(1)}} & 1 \end{vmatrix} \tag{2.95}$$

giving $A^{(2)} = T_2 A^{(1)} = T_2 T_1 A$ where

$$A^{(2)} = \begin{vmatrix} 1 & 0 & a_{13}^{(2)} \\ 0 & 1 & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(2)} \end{vmatrix} \tag{2.96}$$

Eq. (2.87) is now transformed to

$$A^{(2)}X = T_2 A^{(1)}X = T_2 T_1 A X = T_2 T_1 b \tag{2.97}$$

If this transformation process is continued, then for a nth order problem, eq. (2.87) becomes

$$A^{(n)}X = T_n \ldots T_2 T_1 AX = T_n \ldots T_2 T_1 b \qquad (2.98)$$

But $A^{(n)}$ has been reduced sequentially to a unit matrix, therefore eq. (2.98) is

$$X = T_n \ldots T_2 T_1 b \qquad (2.99)$$

and by comparing eqs. (2.87) and (2.99)

$$A^{-1} = T_n \ldots T_2 T_1 \qquad (2.100)$$

From eq. (2.100) it can be seen that this transformation process enables the inverse of the original matrix A to be obtained implicitly as the product of n factor or transformation matrices.

Each transformation matrix T_i (i = 1,2, .. ,n) is a unit matrix except for its i th column, therefore, in computer solutions, only the i th column need be stored ; all other elements of the matrix are known implicitly. In general sparse network problems, the i th column of T_i will also contain a large proportion of zero elements.

2.11.3. Sparse Matrix Codes

Duff **[13][14]** (with Stewart) has made some comparisons of code for the solution of sparse sets of linear eqations. The following is a brief list of programs with their description.

1. MA28 - pivots are selected using the Markowitz scheme with threshold pivot. There is an optional block triangularisation routine and the program can cope with singular systems.

2. YSMP - pivots are chosen from the main diagonal according to a minimum degree algorithm on A and A^T.

3. GNSOIN - generates a cycle-free code which performs Crout reduction when supplied with the pivot order.

4. SLMATH - generates pivot order using the Markowitz method with threshold pivoting and has the option of switching to a full matrix code when the active matrix is sufficiently full.

5. SSLEST - uses threshold pivoting. User options include the removal of all elements below a user-set value, restriction of pivots to the main diagonal, or restriction on the number of rows inspected before each pivot selection.

6. NSPIV - uses partial pivoting to solve a single set of equations. It preorders the rows in order of increasing number of non-zeros. The largest element from each row in turn is then used as pivot.

Bending and Hutchison [5] developed TRGB routines for solution of sparse matrices, further to their work on linearisation methods. The method is based on Gaussian elimination and has two stages. In the primary stage, the matrix problem is solved and a first-time 'operator list' is obtained. This is composed of the addresses of elements and the operations performed. If another system of the same topology is to be solved then the secondary stage will solve it in conjunction with the operator list. This reduces the computation time and is particularly appropriate for pipe network systems, since changing network parameters will alter the matrix coefficients, but not their position within the matrix. However, if a previously used pivot has become zero, or falls inside a pre-defined tolerance, then the primary stage is used again.

For the TRGB routines the pivot is selected by chosing the column with the fewest non-zero elements, or least 'file'. The rows which include the variable corresponding to the column of least file are searched, and the one with the fewest non-zero elements, or least rank, is chosen. The element at the intersection is the new pivot unless it is too small. The routines will only accept as many equations as there are variables and will reject any extra rows. If there are too few rows, or the matrix is singular, the TRGB routines will attempt to solve for as many variables as possible.

2.11.4. Iterative methods

Iterative methods often prove useful in the solution of linear systems. They use only the non-zero elements and so appear especially attractive in the solution of sparse sets of linear equations, since only the non-zero elements need be stored.

The simplest iterative method is that of Jacobi. The Gauss-Seidel method is an improved version of Jacobi's.

In order to see the way in which the two methods work, the coefficient matrix A may be split into three parts. These correspond to the set of diagonal elements, the elements above the diagonal, and the elements below the diagonal. Thus :

$$AX = (L + D + U)X = B \tag{2.101}$$

It is convenient to scale the equation by dividing through by the diagonal elements so that D becomes equal to the unit matrix I.

The Jacobi method results from transferring all terms to the right hand side except the diagonal terms, and iterating as follows.

$$X^{(r+1)} = (- L - U)X^{(r)} + B \qquad r = 0,1, \ldots \tag{2.102}$$

However, the Gauss-Seidel method introduces $x_1^{(r+1)}$, $x_2^{(r+1)}$, etc., on the right hand side as soon as they are available. Thus, the iteration equations become

$$X^{(r+1)} = -L.X^{(r+1)} - U.X^{(r)} + B$$

or

$$(1 + L)X^{(r+1)} = U.X^{(r)} + B \tag{2.103}$$

Williams [32] states that when both the Jacobi and the Gauss-Seidel methods converge, the Gauss-Seidel method converges faster than Jacobi.

If the iterative process converges slowly, the technique of over-relaxation may be employed. The values calculated from the Gauss-Seidel process are modified according to the following equation :

$$X^{(r+1)} = X^{(r)} + w(X^{(r+1)} - X^{(r)}) \tag{2.104}$$

$X^{(r+1)}$ is the value calculated by the Gauss-Seidel process.

If the above equation is written in the form

$$X^{(r+1)} = PX^{(r)} + C \tag{2.105}$$

and the final solution X is given by the equation

$$X = PX + C$$

then the error $E^{(r)} = X - X^{(r)}$ is given by

$$\begin{aligned} X - X^{(r+1)} &= P(X - X^{(r)}) \\ E^{(r+1)} &= PE^{(r)} \\ &= P^{r+1}E^{(0)} \end{aligned} \tag{2.106}$$

For convergence to be achieved, one necessary condition is that the eigenvalues of P should have modulus less than one. Williams [32] states that the condition $|\lambda_i| < 1$ is also a sufficient condition and, therefore, a knowledge of the eigenvalues of P will determine whether the iteration will converge. However, the eigenvalues themselves are difficult to evaluate.

One condition which is easily checked and which guarantees convergence is that of diagonal dominance of the original matrix A. A matrix is said to be strictly diagonally dominant if :

$$d_r < 1 \qquad r = 1,2, \ldots ,n$$

where

$$d_r = \frac{\sum_{j=1}^{n}{}' |a_{rj}|}{|a_{rr}|} \tag{2.107}$$

with the prime notation signifying that the value a_{rr} is omitted from the summation. If $d_r \leqslant 1$ for $r = 1,2, \ldots ,n$ and $d_r < 1$ for at least one value of r, then the matrix is said to be weakly diagonally dominant. This condition is sufficient for convergence of the iterative process. It should be noted here that, with respect to the network program described in Chapter 3, the coefficient matrix has $d_r = 1$ for all r, which signifies that convergence will not necessarily be achieved in all cases. However, as stated, this condition is sufficient, but not necessary.

Another condition which ensures convergence is when the matrix A is positive definite. A matrix is said to be positive definite if for every non-null vector X the quantity $X^T A > 0$. Since this property is more difficult to

investigate, the property of diagonal dominance is more often used to check if convergence can be guaranteed.

2.12. Supercomputers and sparse matrix strategies

The coming availability of cheap supercomputing power will greatly affect the areas of process simulation and design. Supercomputers differ architecturally from present-day "conventional" large mainframe computers and can potentially provide very large increases in speed relative to the conventional machine. However the amount by which speed can be increased depends on problem formulation and the solution strategy involved. Speed may be only doubled, or increased by a factor of twenty or more if the supercomputer architecture is well exploited.

The most significant architectural feature of the supercomputer is its ability to perform vector operations. The term vector operation is described by Calahan and Ames [7] as 'a sequence of identical arithmetic or logic operations performed on elements of one or more arrrays, invoked by a single instruction'. Thus, any algorithm which uses a high amount of vectorization in the course of its solution is well able to exploit this feature of supercomputers.

Stadtherr and Vegeais [28] have discussed various sparse matrix strategies which may be used on supercomputers.

One approach is the block-oriented approach, in which parts of the matrix are treated as if they were dense blocks of non-zeros. The blocks are so located that the system can be solved by performing block Gaussian elimination. The blocks are given descriptors that identify the size of the block and its position in the matrix. Because the blocks are considered full, the location of all elements is described completely by the block descriptors. The system is then solved by block Gaussian elimination. Because of the regular way in which the matrix is stored, the operations performed in this approach are vector operations.

The drawbacks to this method are that, although a high number of operations per second may be performed, many of these operations are carried out, unnecessarily, on zero elements, and difficulty arises in

pivoting to maintain numerical stabilty. If, in the course of performing threshold pivoting, it becomes necessary to exchange columns for reasons of numerical stability, then this could lead to transfers among a number of blocks and possibly the formation of new blocks because of fill-in. Thus, the overall performance of the block-oriented solver could be slowed down considerably.

Another approach is the continuous backsubstitution approach which tries to exploit the presence of contiguous non-zero elements in order to carry out vector operations. The CBS algorithm limits fill-in in the matrix to certain columns, called spike columns. Because these spike columns normally become completely filled-in in the CBS algorithm, they can be stored as a full vector. This means that computations with the spike columns can be done as vector operations. The method operates almost exclusively on non-zeros and also limits the amount of fill-in that can occur. However increased speed occurs only in the spike columns. The elements below the diagonal are indirectly indexed and cannot be operated on as vectors. Also, when column exchange is necessary in order to maintain numerical stability, the spike column must be put into indexed form and the pivot column must be "unindexed" into a contiguous vector.

Another method cited by Stadtherr and Vegeais is a variation of the frontal approach, which was developed for use in finite element problems. It takes advantage of the fact that each variable only appears in a few equations and that pivoting on a variable will only affect a small number of equations and variables. Only a small submatrix, called the frontal matrix, is stored at any time during the solution of the sparse matrix. In essence, this method takes advantage of a banded type of matrix structure.

The frontal matrix is fairly dense and may be treated as a full matrix, thus allowing the use of vector operations during elimination. Another advantage of this approach is that the amount of storage necessary for the frontal matrix and other needed arrays is small.

Stadtherr and Vegeais state that for the frontal method it is desirable to process small columns first and small rows last. Because of this, a reordering method should first be applied to the matrix. They have tested various reordering methods and conclude that the best overall method is

the BLOKS reordering (which they refer to in [28]).

A disadvantage to this method is that operations are done on zero elements (as was the case in the block-oriented approach). So while the operations performed are vector operations and are therefore performed at a much faster rate, some of this speed is wasted on unnecessary computations.

2.13. Conclusions

This literature review has described the development of methods for modelling steady-state flow networks, beginning with the method of Hardy Cross, which was the basis for much later work. The Hardy Cross method, in its basic and improved forms, requires mesh selection, which is time consuming, since it is largely an ad hoc process. Orthogonal transformation of the network also requires mesh selection, and diakoptics involves the abitrary cutting or tearing of the network, with only general guidelines available for the best way to carry out this operation. The methods described in the section on Graph Theory are also dependent on a mesh formulation of the network. Such methods as these are largely redundant. A number of linearisation methods were examined. The first of these, the Bending and Hutchison method, applies to networks which have been modelled by sets of equations describing mass conservation at network nodes (i.e. a nodal formulation). This model is simpler than mesh- or partition-type models and its use allows the effects of changes to network structure or conditions to be more easily demonstrated. Various different linearisation techniques, including Newton-Raphson or variations of the linearisation used by Bending and Hutchison, have been applied to mesh or nodal network models. The sets of equations describing nodal mass conservation or mesh pressure drop are typically large and sparse, and require the availability of efficient sparse matrix solvers. With the development of more efficient sparse matrix methods and improved computing power, the linearisation methods which have been applied to network problems appear the most favourable. They allow ease of specification and are very reliable, so long as the equations used for flow-pressure drop in pipes do not contain discontinuities.

With regard to future developments in flow network simulation, there

would appear to be scope for more reliable and accurate modelling of network elements in steady- and unsteady-state networks.

2.14. Notation

Flow Network Representation

F_i Mass flow along branch i

P_k, P_j pressures at nodes k and j (end nodes of branch i)

Graph Theory

A' augmented branch-node incidence matrix for graph of network

A branch-node incidence matrix for graph of network

C branch-mesh incidence matrix for graph of network

Hardy Cross

ΔH_C error in head around mesh

ΔH_n increment in head at node n

ΔQ_C linear correction applied to flowrate in mesh

ΔQ_n excess inflow/outflow at node n

q_k flowrate through network element

α_k coefficient in flow/head-loss equation

σ_k head-loss through network element

Network Transformations

C_1 square transformation matrix for network

E vector of branch pressure sources

E' vector of mesh pressure sources for all-mesh network

I vector of branch flow sources

I'_1 nodal flow vector for all-mesh network

J vector of total branch flows

J' vector of total mesh flows for all-mesh network

Y admittance matrix

Y' admittance matrix for all-mesh netwok

Z impedance matrix

Z'	impedance matrix for all-mesh network
e	vector of branch pressure rises
e'	vector of nodal pressures
i	vector of branch flows
i'	vector of mesh flows
i'_2	mesh flow vector for all-mesh network
i_p	vector of 'primitive' branch flows

Diakoptics

B	branch–'node to datum path' incidence matrix
B_T, B_L	tree and non-tree partitions of B
C_{AB}	transformation matrix relating networks A and B
I'	vector of flows in the node to datum paths
J'_A, J'_B	generalised flow vectors for networks A and B
J'	vector of path flows for all-path network
V'_A	pressure vector for network A
e'_B	vector of nodal pressures for network B
α	transformation matrix for all-mesh (diakoptic) network
γ	transformation matrix for all-path network

Other Matrix Methods

B	partition of K
K	cut-set matrix
$\tilde{K}$	cut-set matrix for undirected graph
$\tilde{T}$	partition of $\tilde{\Gamma}$
V_A, V_B	sets of vertices in subgraphs A and B
q_C	flow rate in chord
q_T	flow rate in tree branch
w	vector of flow balances for vertex subsets

w' vector of input/output flows for network

$\tilde{\Gamma}$ transpose of branch–mesh incidence matrix for undirected graph

Bending and Hutchison

C_i coefficient in pipe pressure/flow relationship

D_i pipe diameter

F_i Volumetric inflow/outflow at node i

G_j set of pipes connected to node j where flow direction is away from node j

H_j set of pumps connected to node j where flow direction is away from node j

I_j set of nodes where external network flows are input/output

L_i pipe length

N_I number of volumetric inflow/outflow specifications for network

N_{pu} number of pumps at which pressure drop is specified

P_i pressure at node i

P_k, P_l pressure at pipe input/output nodes

Q_i volumetric flowrate in pump i

R_j set of pipes connected to node j where flow direction is towards node j

S_j set of pumps connected to node j where flow direction is towards node j

V_i fluid velocity in pipe i

X_I number of network input/outputs

X_n number of nodes in network

X_p number of pipes in network

X_{pu} number of pumps in network

μ fluid viscosity

Newton-Raphson

D Jacobian matrix for forcing function F

F_l forcing function for network loop

m_i mass flow in each pipe i in network loop

σm_l correction to flow in network loop

ξ direction of flow in pipe relative to loop flow direction

Modelling of Pipeline Network Elements

D pipe diameter

Re Reynolds number

f_D Darcy friction factor

f_f Fanning friction factor

ϵ pipe roughness

Pumps (Gastoli and Spadoni)

H pump head

Q flowrate through pump

Q_0 maximum pump throughput

h_0 pump shut-off head

Pumps (Wood and Thorley)

E_p pump head

H_1 pump shut-off head

Compressors

h_p compressor horsepower

p_i, p_j compressor input and output pressures

q_{ij} flow through compressor

α_0, α_1, α_2 constants in flow/pressure equation

Pipeline fittings

h_L head loss

K_L loss coefficient

V characteristic velocity in pipe fitting

CHAPTER 3

THE DEVELOPMENT OF A PROGRAM FOR STEADY-STATE FLOW NETWORK MODELLING

3.1. Introduction

The second aim of this thesis, stated in Chapter 1, was to provide a computer tool to be used for the solution of steady-state flow network problems. In this chapter a description is given of the development of a FORTRAN program which was written to achieve this aim. Reference is made to methods mentioned in the literature survey of Chapter 2. Some of the concepts discussed in Chapter 2 are restated, in order to describe clearly the factors involved in program design.

3.2. Computer Modelling of Flow Networks : Overview

The modelling of steady-state flow networks by computer program depends on the way in which the network is 'abstracted' or numerically represented. Most flow network programs are based on representations which derive from one of the two following views of a network. The network may be seen as consisting of a set of meshes (the mesh formulation, described in Chapter 2), or viewed as a set of connected pipes and nodes. The choice of representation directly influences the choice of solution algorithm and this in turn has an effect on the range of problems which may efficiently be solved by the program, for example networks with very small flows in certain pipes may prove insoluble by a particular solution method. Clearly, it is desirable to have a solution algorithm which is robust for all types of flow regime and network topology.

3.3. Basis of Program Design

The computer program described in this chapter was based in part on an existing ICI flow network program ; certain features of that program, such as data input/output and physical properties utilities, were not changed, or modified only slightly. However the solution algorithm was replaced by one based on a different network abstraction. The original algorithm used the mesh formulation of a network ; the new algorithm applies to a network which is modelled as a set of linearised pipe pressure/flow equations. Several workers, e.g. Bending and Hutchison [4],

quoted in the literature survey, suggest that solution methods based on the 'linearised network' approach have superior convergence properties to those which are based on the mesh formulation. Consideration of the flow network literature thus decided that a version of the former method be used.

3.4. Choice of Network Representation

For the purposes of the program under discussion, a flow network is viewed as a connected set of nodes and links. Links correspond to actual physical pipeline elements which may be pipes, pumps or valves, however the latter two are considered as 'pseudo-pipes' by the program. The following is a summary of the network components and their properties.

- **Node** : Nodes in the network may be of two types - junction and pendant. A junction node refers to a point in the network adjoined by two or more links. A pendant node is a point in the network adjoined by only one link and which has been assigned either a fixed pressure or net inflow/outflow value. Physically a pendant node corresponds to a supply/demand point in the network, such as a reservoir, pressure source or input to a subsidiary network. The properties of a node are pressure and inflow/outflow. If a pressure or flow is not explicitly specified in the data set, then the inflow/outflow is set implicitly to zero.

- **Link** : A link, as mentioned above, can refer to either a pipe, pump or valve. Pumps and valves are considered, for the purposes of data input, to be pipes with zero length and bore equivalent to that of the actual pipe with which the pump or valve is physically associated in the network. Pipes (or pseudo-pipes) have the properties of length, bore, roughness ratio, fittings loss and temperature, of which only bore and temperature are non-zero for pumps and bore, fittings loss and temperature are non-zero for valves.

3.5. Flow Analysis

An analysis of flow in pipes is necessary as a precursor to the description of the algorithm used in the program. The algorithm is based on equations describing the flow/pressure drop relation for a single pipe.

Pressure drop in a pipe passing fluid is a result of :

a) friction or drag of the fluid on the pipe walls

b) losses due to fittings (localised changes of bore and changes of direction of flow)

c) kinetic energy changes

d) gravity heads due to change in height.

For incompressible flow, the pressure drop is related to the mean flow velocity by the Darcy-Weisbach or Fanning law. This law is only 'true' for fully turbulent flow, where the friction coefficient is constant, and it becomes the definition of friction or loss coefficients for other flow regimes.

$$P_i - P_o = \frac{\rho v^2}{2}\left(\frac{f l}{d} + k\right) \qquad (3.1)$$

Eq. (3.1) is a restatement of eq. (2.42(i)), with an added fittings loss term. For compressible flow, the density changes with pressure and eq. (3.1) can only be taken as true for short lengths of pipe. A similar expression to eq. (3.1) may be derived for compressible flow, using the continuity and momentum equations. The compressible flow equation applies for the isothermal flow of perfect gases (pressure $\propto$ density).

$$P_i - P_o = \frac{\rho v^2}{2}\left(\frac{f l}{d} + k + 2ln\left(\frac{\rho_i}{\rho_o}\right)\right) \qquad (3.2)$$

Eqs. (3.1) and (3.2) may be rewritten in terms of mass flows (the program herein described works internally in terms of mass flows) as :

$$P_i - P_o = \frac{F^2}{2s^2\rho}\left(\frac{f l}{d} + k\right) \qquad (3.3)$$

$$P_i - P_o = \frac{F^2}{2s^2\rho}\left(\frac{f l}{d} + k + 2ln\left(\frac{\rho_i}{\rho_o}\right)\right) \qquad (3.4)$$

The friction coefficient, f, is determined by the pipe wall roughness and the Reynolds number. In the program this is calculated from the Hagen-Poiseuille law for laminar flow and from the Colebrook-White equation for transitional and turbulent flow. The Colebrook-White equation is used when the Reynolds number is greater than or equal to 2500. Below 2500 the Hagen-Poiseuille law is used. The Hagen-Poiseuille law may be stated as :

$$f = \frac{64}{Re} \tag{3.5}$$

The Colebrook-White equation is (restating from Chapter 2, eq. (2.62))

$$\frac{1}{\sqrt{f}} = 1.74 - 2\log\left(\frac{2\varepsilon}{d} + \frac{18.7}{Re\sqrt{f}}\right) \tag{3.6}$$

The gravity heads due to sloping pipes are taken into account by adding the term $\rho g(h_o - h_i)$ to the pressure drop where ρ is the mean fluid density and h_i , h_o the node heights at inlet and outlet.

There now follows a description of the algorithm used in the program (the program will be referred to by the acronym **FLONET**). The flow diagrams for the FLONET program and subroutines are given in appendix ii.

3.6. FLONET algorithm

The algorithm involves setting up flow balances for all nodes in the network which have a specified or implicit inflow/outflow assigned to them (any node which has not been assigned a fixed pressure or inflow/outflow in the data set is assumed to have an inflow/outflow specification of zero). There is a requirement that at least one fixed nodal pressure be specified in the data set. (Nodal flow balances are not applied at fixed pressure nodes). Where nodal pressures are specified, their values are used to obtain an averaged 'initial' pressure specification for all other nodes in the network which are not fixed-pressure nodes.

The algorithm uses an iterative procedure to construct and solve a set of linear equations describing the flow/pressure drop relationship in each link. These equations are of the form

$$A_{ij}(P_i - P_j) + B_{ij} = F_{ij} \tag{3.7}$$

and are a linearised version of eqs. (3.3) and (3.4). Whatever linearisation method is used, it is necessary, at any rate, to have a value for the friction factor which appears in eqs. (3.3) and (3.4). The friction factor at each new iteration is calculated from the Reynolds number of the flow at the previous iteration. At the zeroth iteration there is, of course, no previous flow value, and so the algorithm incorporates an initialization step in which

the following points apply (for links which are pipes or valves).

1. The flow is assumed laminar and incompressible (thus equation (3.3) applies)

2. The fittings loss term 'k ' (if present) is 'absorbed' into the friction factor term (fl/d) by adding an extra length to the link length, l. This extra length is calculated according to a rule of thumb and is equal to $50 \times k \times d$, where d is the diameter of the link.

Substitution of $f=64/Re$ into the modified eq. (3.3) and rearrangement of terms gives an equation of the form of eq. (3.7) (with the terms B_{ij} equal to zero).

When the link is a pump, then the A_{ij}, B_{ij} and F_{ij} are obtained from an analysis of the supplied pump data describing the pump characteristic. A linearisation procedure is applied which permits the flow/pressure relationship for a pump (which is described in the input data by a set of operating points on the pump characteristic) to be expressed in the form of eq. (3.7). This linearisation procedure is outlined in flow diagram form in appendix ii (see flow diagrams for subroutines FPUMP,LPUMP,QPUMP and LNPUMP).

When the A_{ij}'s, B_{ij}'s and F_{ij}'s have been obtained for all links in the network then the flow balances are set up for every node in the network (excluding fixed-pressure nodes) using eq. (3.7) to express the flow entering or leaving a node via the links connected to it. For each node, j, then :

$$\sum_{i\,G} |A_{ij}(P_i - P_j) + B_{ij}| - \sum_{i\,H} |A_{ij}(P_j - P_i) + B_{ij}| = F_j \qquad (3.8)$$

Having thus set up the nodal flow balances, eq. (3.8) is modified such that all terms in P are on the left hand side and all constant terms are on the right hand side. The equation for each node j is then normalised by dividing through by the sum of the A_{ij}'s so that the coefficient of P_j becomes −1. These operations result in a set of linear equations in P to which are added expressions for the fixed pressure nodes, of the form :

$$-1.P_j = -P_{j_{fix}} \tag{3.9}$$

where P_j is the fixed pressure value assigned to node j. The left hand side of the set of equations represents a coefficient matrix for the vector of network nodal pressures. The pressures are solved for, using a direct linear solving method.

The new pressures are used, in the next iteration, to calculate new network flows, F_{ij} (using eqs. (3.3) or (3.4)). As already mentioned, the friction factor 'f' in eq. (3.3) or eq.(3.4) is calculated from the flow value in the previous iteration. New A_{ij} and B_{ij} must also be obtained. If the flow in any link is below $1*10^{-6}$ kg/s then the initialization step is again used, for that link. Otherwise the A_{ij} and B_{ij} are obtained by using one of two linearisation methods - the Newton-Raphson or the Bending and Hutchison method. These are fully explained in the next section. The process of calculating new pressures, using the new A_{ij} and B_{ij}, and hence obtaining updated flows, is repeated until convergence is achieved. The convergence criterion is that the discrepancy between the specified inflow/outflow to a node and the calculated inflow/outflow to that node is within a pre-specified tolerance, for every node in the network.

3.7. Linearisation methods

As already stated, two methods of linearising the flow/pressure drop equation in pipes (for transition and turbulent flow) were used, Newton-Raphson, and Bending and Hutchison.

If eq. (3.3) (or (3.4)) is written as

$$\Delta P_{ij} = F_{ij}^{\,2}.C \tag{3.10}$$

then it can be re-expressed as

$$F_{ij} = K.\sqrt{\Delta P_{ij}} \tag{3.11}$$

and it is this eq. (3.11) which is to be linearised using the Newton-Raphson technique. Actually, a slightly modified Newton-Raphson method is used in the program.

The Newton-Raphson approximation to a function, $f(x)$, may be stated as

$$f(x) = f(x_h) + \frac{f'(x_h)}{1!}x + \frac{f''(x_h)}{2!}x^2 + \ldots$$

For a linear approximation to a function, the first two terms are used, with $f(x_h)$ representing the actual value of the function at the point x_h about which linearisation is being attempted, and $f'(x_h)$ the value of the first derivative at that point. Thus if $\Delta P_{ij}^{\,0}$ were the value about which a linearisation of eq. (3.11) were being attempted then the Newton-Raphson approximation would be :

$$F_{ij} = K.\sqrt{\Delta P_{ij}^{\,0}} + \frac{K}{2\sqrt{\Delta P_{ij}^{\,0}}}.\Delta P_{ij} \qquad (3.12)$$

If this is compared with eq. (3.7), then $A_{ij} = K/(2\sqrt{\Delta P_{ij}^{\,0}})$ and $B_{ij} = K.\sqrt{\Delta P_{ij}^{\,0}}$. As already stated, a slightly modified Newton-Raphson method is used in the program. At the start of each iteration, k, F_{ij} is calculated from the pressures obtained in the previous iteration, using eq. (3.11). Thus :

$$F_{ij}^{(k)} = K.\sqrt{\left|\left(P_i^{(k-1)} - P_j^{(k-1)}\right)\right|} \qquad (3.13)$$

The coefficient $A_{ij}^{(k)}$ is calculated by the Newton-Raphson method, using the pressures from the previous iteration, $P_i^{(k-1)}$ and and $P_j^{(k-1)}$. But $B_{ij}^{(k)}$ is calculated as :

$$\left|B_{ij}^{(k)}\right| = F_{ij}^{(k)} - A_{ij}^{(k)}\left|\left(P_i^{(k-1)} - P_j^{(k-1)}\right)\right| \qquad (3.14)$$

(The appropriate sign is then assigned to B depending on the direction of flow between nodes i and j).

This modification to the method is necessary because of the fact that flows are always calculated by the non-linear equation (for transitional and turbulent flow regimes) and therefore, in eq. (3.12), the left hand side is equal to $K.\sqrt{\Delta P_{ij}^{\,0}}$. B_{ij}, in fact, becomes $(K/2).\sqrt{\Delta P_{ij}^{\,0}}$.

The Bending and Hutchison method may be described with reference to eq. (3.10), which may be rewritten in a linear, iterative form as :

$$F_{ij}^{(k)} = \frac{1}{C\left|F_{ij}^{(k-1)}\right|}.\Delta P_{ij}^{(k-1)} = K^*.\Delta P_{ij}^{(k-1)} \qquad (3.15)$$

However, due to the fact that the value of K^* could oscillate if the pressure drop ΔP_{ij} remained the same between two consecutive iterations,

then $F_{ij}^{(k)}$ is defined as

$$F_{ij}^{(k)} = \frac{1}{2}(F_{ij}^{(k)} + F_{ij}^{(k-1)}) \qquad (3.16)$$

i.e. the value of $F_{ij}^{(k)}$ is the average of that calculated in eq. (3.15) and the value, $F_{ij}^{(k-1)}$, calculated in the previous iteration.

A feature of the Bending and Hutchison linearisation method is that when applied to the inflows/outflows to a node from pipes connected to it, it forces a material balance at that node. With this in mind, the linearisation process used in FLONET was formulated as follows :

1. Use the Bending and Hutchison method for the first five iterations, to force a material balance at all network nodes, and thereby enhance convergence.

2. Use the Newton-Raphson method thereafter, up to iteration 25 (most of the test problems converged within 25 iterations). The Newton-Raphson linearisation is faster than the Bending and Hutchison method in terms of the number of iterations required to achieve convergence, though less robust in certain network problems.

3. If convergence has not been achieved after 25 iterations, switch to the Bending and Hutchison method. (In only one case, case 10, did the solution take longer than 25 iterations to converge and this was due to precision difficulties, which the Bending and Hutchison method was able to overcome ; if only the Newton-Raphson method was used, convergence was never achieved).

3.8. Intractable Problems

Some of the test cases whose solution was attempted using the FLONET algorithm, exhibited oscillating flow in certain pipes, which prevented convergence of the solution. In case 13, flow oscillation occurred in a pipe (23-43) which was between 4 and 8 times as long as the pipes adjacent to it in the network, and this difference could have made the network ill-conditioned. In case 14, oscillation occurred in the flow around a mesh (9-10-11-6-9).

Two subroutines were written to handle oscillating flow in a network -

QCON and CONCHK. The flow diagrams for these subroutines are given in appendix ii.

3.9. Conclusions

As will subsequently be shown in Chapter 4, the program FLONET performed significantly better on supplied network problems than the existing flow program on which it was based. The linearisation method used in the program was found to produce flow convergence in all the test networks. However, as mentioned in the section above, it was necessary to write additional subroutines to deal with the problem of flow oscillation, which occurred in two of the test networks.

A discussion of the supplied network problems and their solution using FLONET is presented in the next chapter. Further development of the steady-state program to allow data input in the form of linear equations, and to enable solution of dynamic problems, is detailed in Chapter 5.

3.10. Notation

A_{ij}, B_{ij}	coefficients in flow/pressure drop relationship
F	mass flow in pipe
F_{ij}	mass flow in pipe whose end nodes are i,j
F_j	mass inflow/outflow at nodes j
G	set of nodes whose pressures exceed that at node j
H	set of nodes whose pressures are less than that at node j
K, K^*	coefficients in flow/pressure drop relationship
P_i	pipe inlet pressure
P_o, P_j	pipe outlet pressure
$P_{j(fix)}$	fixed pressure at node j
ΔP_{ij}	pressure difference across link between nodes i,j
Re	Reynolds number of flow
d	diameter of pipe
f	friction factor
k	fittings loss
l	length of pipe
s	cross-sectional area of pipe
v	velocity of fluid flowing in pipe
ρ	density of fluid flowing in pipe
ρ_i, ρ_o	density of fluid at inlet/outlet of pipe

CHAPTER 4

PERFORMANCE TESTING OF STEADY-STATE FLOW NETWORK PROGRAM

4.1. Introduction

The performance of the steady-state flow network solver, **FLONET**, was evaluated by using it to solve 15 network test cases of varying size and topological complexity. The largest network tested consisted of 108 nodes and 142 pipes. Multiple pumps were incorporated in two of the larger networks. Seven of the test cases were compressible flow problems.

In the last chapter it was stated that **FLONET** was based partly on an existing steady-state flow network program. Solution of the test cases mentioned above had previously been attempted using this original program. Successful solution was not achieved in all cases. In the summary which follows, comparison is made between the performance of **FLONET** and its earlier version, for each test case.

4.2. Summary of test cases

Information relating to the test cases is given in appendix iii. Diagrams are presented for all the test cases, however data and results are only given for small and medium size networks. Graphs of maximum nodal flow residuals (after each program iteration) vs. time are given for all the test cases.

4.2.1. Case 1 : Simple Network

Case 1 is a very simple network which the old version of FLONET failed, however, to solve. With the pressure and flow specifications given in the data set, the new version of FLONET solved the problem in 7 iterations.

4.2.2. Case 2 : Simple Network Containing One Pump

Case 2 is composed of two test problems concerning the same network but with different data sets. In the first problem one reference pressure is specified at an external node (i.e. a node where flow enters or leaves the network) and at the other external nodes the network inflows/outflows are specified. (There is no need to specify inflow/outflow at the 'pressure' node as the program calculates the material balance for the network internally).

In the second problem reference pressures are specified at all external nodes.

The program can in fact handle a combination of external flow and pressure specifications, as will be shown in later test cases ; however, these two problems demonstrate the program's ability to cope with either type of specification.

For the first problem, where external flows were specified, the program took 9 iterations to converge. For the second problem, where external pressures were specified, the program took 6 iterations to converge.

4.2.3. Case 3 : Simple Single-Mesh Network Containing One Pump

The old FLONET program failed to solve this case, although it successfully solved the Case 2 problems, from which this one differs only by the presence of a cross pipe going from node A210 to A220.

This case was successfully solved by the new FLONET program, with convergence achieved after 6 iterations.

4.2.4. Case 4 : Network with Multiple Pumps

The first network in Case 4 demonstrated the problems which the old FLONET program had with pumps in parallel. The solution for this network was unobtainable using the old algorithm, although each of the parallel lines could be solved separately and the configuration without pumps could also easily be solved. Using the new algorithm the parallel pumps problem was easily solved in 5 iterations.

The solution of the third network in Case 4 converged after 1 iteration. The solution would have converged immediately with laminar flow in all pipes (which is incorrect in this particular instance) due to the strategy of assuming laminar flow in the network at the zeroth iteration. However, if the program detects flow convergence with laminar flow, before any iteration has occurred, then the internal convergence flag is reset and, although the nodal pressures may have converged, at least one more iteration is then required for the achievement of flow convergence.

The second network in Case 4 is a single line from the first network. As

may be expected, the number of iterations required for this network is the same as for the first - 5 iterations - due to the feature of parallelism.

4.2.5. Case 5 : Kiln Network

This case study examines a network which incorporates a kiln and associated piping. The diagram for this network is given in appendix iii, however, the data set and results are given in appendix vi (and referenced in Chapter 6). Chapter 6 presents a detailed discussion of this network in relation to the steady-state program and also to two further computer programs which are described in Chapter 5.

4.2.6. Case 6 : Steam System

This network is a simplified model of a works steam system. 43 flow conditions and 1 pressure condition are specified.

The solution converged in 8 iterations.

4.2.7. Case 7 : Network with Gravity Feed

This network is a model of a drainage system. It was found that the drain was not removing liquid fast enough, so it was hoped, by modelling the network using FLONET, to find some way of debottle-necking it.

As the flow is entirely gravity feed, this network is a good test of FLONET's ability to handle the effects of gravity.

The solution converged in 8 iterations.

4.2.8. Case 8 : Water System

This network models a fire safety water system. The previous version of FLONET failed to solve this network problem, predicting a negative pressure during the iteration procedure, although increasing all pressures by 10 bar (arbitrarily) led to successful solution.

The current version of FLONET successfully solved the original problem in 9 iterations.

4.2.9. Case 9 : Furnace Gas Distribution System

The problem in this case concerns the flow distribution through a network of furnace pipes. The gas flows through the furnace along a series of parallel pipes. The pipes are connected between two common headers (1-36) and (73-108).

Since the heat flux distribution is even in the furnace it is important to have approximately the same flow through each pipe. FLONET was to be used to examine the flow distribution of several network designs to see which would best be suited.

The old version of FLONET did not manage to balance the parallel flow system properly - the flow along pipe 1-37 differed markedly from that along 36-72. However, the new version of the program balanced the flows satisfactorily, as can be seen from the results. The solution converged in 8 iterations.

4.2.10. Case 10 : Subnetwork of Network 9

This network is a much smaller version of the network in Case 9.

With the old version of FLONET, the solution predicted that all the fluid is carried across the network through lines 1-5 and 4-8 (zero flow being predicted in lines 2-6 and 3-7).

This network presented a problem for the new version, with respect to the precision required to achieve a converged solution. As can be seen from the results, the nodal pressures which produce the specified outflow of 1 kg/s from the network (for a pressure of 30 bar at node 1) are very close in value, and this explains the number of iterations - 42 - required to achieve convergence. However the fact that convergence was achieved illustrates the robustness of the algorithm used.

4.2.11. Case 11 : Furnace Gas Distribution System

This network is similar to that in Case 9 and was one of the alternatives considered for the furnace gas piping. It differs from the network in Case 9 in that the upper and lower headers are rings.

The problem was solved in 8 iterations.

4.2.12. Case 12 : Water Supply System

This is a model of a works water supply system, with several inlet/outlet points at varying heights. The network includes 4 pumps (2 pairs in parallel).

FLONET easily handled the size (97 pipes) and complexity of this problem, with flow convergence being achieved in 9 iterations.

4.2.13. Case 13

This too is a model of a works water supply system. When an initial attempt was made made at the solution of this problem it was found that the specified system pressure was too low and several nodal pressures became negative. A check for negative nodal pressures was then included in the algorithm, with a warning message being printed to advise the user to increase the supply pressure, in this event.

After the supply pressure was increased, however, another problem was discovered, in that the occurrence of the maximum flow residuals after each iteration, oscillated between two adjacent nodes (23 and 24), without convergence being achieved. This necessitated the alteration of the solution algorithm to include steps to damp out oscillatating flows in pipes.

With these improvements the solution converged in 18 iterations.

4.2.14. Case 14 : Compressible flow network

This is another example of a network which gave problems with oscillating flows in pipes, although these yielded to the improvements in the solution algorithm, mentioned in the last case. The oscillating flows in the links in ring 9-10-11- 6-9 were damped out and the solution was achieved in 19 iterations. However, the program output indicated that the Mach number in one pipe in the network exceeds 0.2. The correlations used in the program are only valid up to a Mach number of 0.2 and therefore the results are to be viewed cautiously.

4.2.15. Case 15 : Brinefields network

This is a model of a subsection of a brinefields network. The data supplied is for a compressible fluid, for the purposes of testing FLONET's compressible flow algorithm in a complicated network.

As can be seen from the graph of flow residuals, oscillation of the solution occurred, with the largest flow residual being seen alternately at nodes BSB3 and WSB3. However the algorithm was able to deal with this oscillation, as in previous cases. The results indicate that, with the supply pressure of 40 bar given in the problem data, the Mach number in two of the network pipes exceeds 0.2. Reducing the supply pressure to 13 bar reduced the velocity to beneath the level where the Mach number exceeded 0.2, without affecting convergence properties.

The solution converged in 17 iterations.

4.3. Conclusions

Flow convergence was achieved in all the network test-cases submitted to **FLONET**, demonstrating the effectiveness of **FLONET**'s solution algorithm in general, and especially as compared with the solution algorithm of the earlier version of the program. The problems caused by oscillating flows in two of the test cases were successfully overcome by the smoothing procedure included in the problem. This procedure, however, should be further tested on other networks where the feature of flow oscillation is present, since the pattern of oscillation is likely to differ from case to case (e.g. alternate positive and negative flows at the same pipe, or maximum flow residual occurring alternately between two different nodes, adjacent or otherwise).

The program arrays holding flow and pressure values were declared as DOUBLE PRECISION ; Case 10 illustrated the necessity for this degree of precision.

The design and testing of the steady-state flow network program **FLONET** satisfied the second aim of this thesis. The third aim necessitated further development of the steady-state network program, to permit data input in the form of equations, and to enable solution of dynamic network problems. The next chapter describes the design and testing of program

modules for the purpose just outlined.

CHAPTER 5

EQUATION PARSER AND DYNAMIC NETWORK PROGRAM

5.1. Introduction

Further to the development of a program for the solution of steady-state network problems, additional design work was carried out in order to extend the program's range of application. Two types of extension to the program described in Chapter 4 were examined. The first type was concerned with the format of data input for steady-state network problems. The second type involved adding to the program a capacity to handle a certain class of dynamic network problems. This chapter describes the development of two computer programs which were the outcome of the additional design work. Both programs were used to obtain the solution of sample network problems and a discussion of the results is presented.

5.2. Data Input for Steady-state Network Problems

With regard to the first type of extension mentioned above, the current steady-state network solver described in Chapter 4 can only handle input data of the form :

P(n) = ... fixed value

F(n) = ... " "

i.e. nodal quantities of pressure and flow only can be specified

One example of additional flow network quantities which could be specified as input data would be fixed flows at any given link. This type of specification could be used when, for example, it is known that there is a particular coolant flow through an exchanger. Here the resistance of the link cannot be specified, the flow is achieved by closing a valve and the pressure drop across the exchanger is to be calculated. Another example would be the specification of a node which does not have a fixed height, as would be the case when the node represented a tank with a free surface.

A very useful extension would be to add the facility to provide, as data, any linear relationship amongst flows, or pressures, in the form of an additional

equation :

e.g. F(3) = F(4)

or F(2) = F(1) - F(6)

or P(5) = P(7) - 4

A computer program was written to analyse flow network problems which are specified in terms of linear equations involving pressures and flows. The steady-state program described in Chapter 3 was used as a basis for the new program, which is referred to as EQNET. Documentation for EQNET is presented in appendix iv. The former program required, firstly, the addition of a module to perform parsing of the network equations. Within the program the network is described by a matrix of linear equations which represent nodal flow balances. Additional modules were therefore required to incorporate the specified 'data' equations into this internal matrix. The network parser is described briefly in the following (full documentation for the network parser, EQPARSE, is given in appendix iv).

5.3. Input and Processing of Data by Equation Parser

Data is input to the program EQNET in a similar format to that used in the steady-state program, except that network equations are included after the physical properties data. The data set begins with a list describing the characteristics of the network links (pipes or valves). The first two numbers of each line in this list are the input and output node identifiers of the link. The third number is either 1 or -99, depending on whether the link is described by pipe data (length, bore, roughness ratio, etc) or by a network equation. The last number on each line in this list is the link temperature. The letter E at the end of a data set indicates that no more network equations are to be supplied.

The equation parser, EQPARSE, reads in one network equation (line) at a time and 'atomises' it, so that the equation is expressed as a set of separate entities or atoms. For each atom the parser generates 3 entries - type, label and value. The type of an atom indicates whether the atom represents a node pressure, link flow, node inflow/outflow or a numerical constant. An atom label is the identifier of the node or link to which the atom refers (the label is equal to -1 for numerical constants). For atoms which refer to node pressure/flow or

link flow, the value entry is the coefficient of the flow or pressure term in the equation. Where atoms refer to numerical constants, the value entry is the actual value of the constant.

The following description outlines the main steps of the parsing algorithm.

1. Read in next equation line to be processed. If the first character in the line is "E", then exit from the module.

2. Initialise to zero the counters for numbers of atoms and left and right parentheses in the equation. Set a flag to indicate that the "=" sign has not yet been encountered in the current equation line. Initialise to 1 the coefficient of P, Q, F and constant terms.

3. (Start of cycle to atomise each equation line - maximum number of atoms allowed in each equation is 10). Look at next unread character in equation line and set this to be the 'current character'. (First time round, see if line begins with "+" or "-" sign and set 'sign flag' accordingly).

4. If the current character is a digit or a decimal point, go to step 11.

5. If the current character is a "+" or "-" character, check the current settings of the two sign flags (one of which refers to terms inside parentheses) and alter the settings if necessary. If the current character is an "=" sign then set a flag to indicate that this has been encountered.

6. If the current character is a left or right parenthesis, then increment the appropriate counter.

7. If the current character is "P" (signifying a pressure term), go to step 13.

8. If the current character is "Q" (signifying a link flow term), go to step 14.

9. If the current character is "F" (signifying a nodal inflow/out flow term), go to step 15.

10. If the current character is none of the previous items, then write error message to terminal and exit from the parser module.

11. Read the characters following the current character until a non-numerical character (i.e. not a digit, exponent sign or decimal point) is encountered. Decode the character string to a real number. If the non-numerical character following this string is a "+", "-", "=", ")" or a newline character, then the character string represents an atom which is a

numerical constant. (The atom's type, label and value are stored in the arrays **itype**, **ivalue** and **rvalue**). If the string represents an atom, then increment the counter for the number of atoms and return to step 3.

12. If the non-numerical character following the character string is a left parenthesis, this indicates that the character string is a coefficient. Go to step 13, 14, 15 or 3, depending on whether the character after the parenthesis is "P", "Q", "F" or none of these (in which case it will be a digit, unless an error occurs).

13. Increment the counter for the number of atoms. Set the atom type to 1 (indicates pressure term). Set the atom value equal to the current value of the coefficient term (this will be either 1, or the number obtained in step 11) multiplied by the current values of the 'sign' and 'equals' flags. Extract the string delimited by parentheses, which follows the "P", "Q" or "F" characters. Go to step 16.

14. Same as step 13, except that the atom type is set to 2 (indicates link flow term).

15. Same as step 13, except that the atom type is set to 3 (indicates node inflow/outflow term).

16. Decode the character string obtained in step 13, 14 or 15 to obtain the identifier of the node or link associated with the pressure ("P") or flow ("Q" or "F") term. Set the atom label equal to this identifier (the identifier is an integer value). Return to step 3 and repeat until the end of the current line is reached. If the end of the current line is reached, then return to step 1.

Once the equation lines have been read and parsed, then the values stored in arrays **itype**, **ivalue** and **rvalue** have to be transferred to the matrix arrays *A* and *B* used in the linear-equation solving routine which obtains values for flow and pressure throughout the network. This transfer is accomplished in routines **SETUPM** and **SET UP RM**, the listings for which are given in appendix iv. The program then procedes in a similar manner to the steady-state program **FLONET**, described in Chapter 3.

There now follows a description of the network problems used to test the flow network program with incorporated network equation parser, and a discussion of the results in each case. The data sets and results are presented in appendix v.

5.4. Test Problems for Flow Network Program with Equation Parser

5.4.1. Problem 1 : All Network Data Supplied as Equations

The network for this problem is shown in Fig. 5.1 in appendix v. It is a simple four node, three branch network for which the input data is given entirely as a set of linear equations. The data set for this problem is listed as Network 5.1 in appendix v. At two of the external nodes the pressure is fixed and at the other node the outflow is specified. Linear valve constants are given for all three branches. In the first three lines of the data set, -99 indicates that an equation will be supplied for the flow/pressure relationship in the branch whose end nodes are specified in the first two columns of each row. The results for this problem are listed after the data set (for Network 5.1), in appendix v.

5.4.2. Problem 2 : "Mixed" Input, i.e. Equations and Data List

This network is identical to the previous one, but the data set contains two rows of physical pipe data as well as equations. This was intended to test the program's ability to handle network problems with "mixed" data sets. The data set and results for this problem are listed under the heading of Network 5.2 in appendix v.

5.4.3. Problem 3 : Mixed Data Input for HF_3 Network

This network problem, and its solution using EQNET, is mentioned here only in passing. Chapter 6 describes how an analysis of this network was made, using three different computer programs, including EQNET. Therefore a full description of the data set for this network, and a discussion of the results, is deferred till then.

5.5. Performance of Network Program EQNET

The three supplied problems tested the effectiveness of the program EQNET as an analysis tool for the networks described by linear equations. The parser module successfully processed the input equations in each problem, and the 'atomised' equations were then successfully placed in the program's internal matrix of network equations, using modules SETUPM and SET UP RM.

5.6. Dynamic Modelling of Flow Networks

As a conclusion to the work carried out on the analysis and solution of flow network problems, a 'dynamic' version of the steady-state program described in Chapter 3 was written and tested. This 'dynamic' version was intended to solve a limited class of unsteady-state network problems. Four sample networks were used to test the program's ability to profile flow/pressure control with time. The input data format is identical to that for the steady-state program except that non-zero values are assigned to nodal capacities and valve (linear) characteristics.

Before discussing the dynamic version of the program and the solutions of the four test problems, a short summary is given of the theory which was used in the design of the program.

5.7. Theory in Dynamic Flow Network Modelling

At a node in a dynamic flow network (a 'dynamic' node) the net sum of flows into the node is not zero.

$$\sum_k F_i \neq 0 = \frac{dm_k}{dt} \tag{5.1}$$

where m_k is the mass stored at node k.

Assuming compressible flow and ideal gas, then

$$P_k V_k = \frac{m_k}{w_k} R T_k$$

$$\rightarrow \quad \frac{dm_k}{dt} = \frac{w_k}{RT_k} . V_k \frac{dP_k}{dt} = C . \frac{dP_k}{dt} \tag{5.2}$$

Therefore, eq. (5.1) may be stated in the form

$$\sum_k F_i = \frac{dP_k}{dt} C \tag{5.3}$$

Eq. (5.3) may be written in finite form as

$$\sum_k F_i(P_k^t) = \frac{P_k^t - P_k^0}{\delta t} . C \tag{5.4}$$

Thus, at a dynamic node, the sum of flows may be expressed as

$$\ldots\left|\sum_{k} k_i P_j^t - \left(\frac{P^t}{\delta t}j\right)\right| \ldots = - C\frac{P^0}{\delta t}j = C' \qquad (5.5)$$

(leaving out the pressure terms due to other nodes connected to node k).

The four network problems used to test the dynamic program - listed in appendix iv as **DYNET** - incorporate flow and/or pressure control valves. The controllers in each case are of the simplest type - proportional controllers. The equation used in the program to model the controller is :

$$k = k_s\mu(x_s - x) + k_r \qquad (5.6)$$

where

x is the measured variable (a flow or a pressure)

x_s is its fixed setpoint value

k_r is a manual reset (valve position at zero error)

k is the controller output

$(k_s\mu)$ expresses the controller "gain"

The main steps in the dynamic flow program DYNET are summarised in the following :

1. Obtain the flow distribution in the network at time zero using subroutine FLOWS.
2. Increment the time step (by the user-specified value).
3. Using the value of the controlled flow/pressure at the previous time step to obtain a 'k' value for the controller valve(s), solve iteratively for flows and pressures in the network at the current time value.
4. If steady-state has been obtained, print the results, plot graphs and stop.
5. If steady-state has not been achieved, go to step 2 and repeat.

Step 3 is carried out using the following set of subroutines (which are listed in appendix iv).

1. SET UP K : Get 'k' values for all branches in network
2. SET UP A : Construct matrix of linearised network equations.
3. PRESSURES : Solve for pressures in network at current time value.
4. FLOWS : Obtain flows in network at current time value.

There now follows a description of the network problems used to test the dynamic flow network program.

5.8. Test Problems for Dynamic Version of Flow Network Program

5.8.1. Problem 1 : Simple Flow Control

The network diagram for this problem is shown in Fig. 5.2 in appendix v. The flow through the line is to be regulated to 5 kg/s. The data set is listed as Network 5.3 and the graph of flow vs. time is shown in Fig 5.4.

5.8.2. Problem 2 : Simple Pressure Control

The network diagram for this problem is shown in Fig. 5.3 in appendix v. The pressure in the pressure vessel at node 2 is to be regulated to 12 bar. The data set is listed as Network 5.4 and the graph of pressure (at node 2) vs. time is shown in Fig. 5.5. The network pressure and flow values at steady-state are listed after the data set.

5.8.3. Problem 3 : Flow and Pressure Control in Compressor Network

The network for this problem is shown in Fig. 5.6. The pressure at node 4 is to be regulated to 7 bar and the flow in the line between nodes 3 and 4 is to be regulated to 0.6 kg/s. The data set for this problem is listed as Network 5.5. The steady-state values of pressure and flow are listed after the data set. The graphs of flow vs. time (for line (3,4)) and pressure vs. time (for node 4) are shown at Figs. 5.7 and 5.8 respectively.

5.8.4. Problem 4 : Flow and Pressure Control in HF Network

As was the case in problem 3 for the program EQNET, a discussion of this network problem, and its solution using DYNET, is deferred till the next chapter.

5.9. Conclusions

The dynamic network program solved, and produced graphic output for, the four test problems presented to it. The HF network problem demonstrated the program's ability to solve medium-size network problems involving flow and pressure control, and the compressor problem showed the the program could successfully handle networks involving recycle loops.

5.10. Notation

P(n)	pressure at node n
F(n)	inflow/outflow at node n
$\sum_k F_i$	Sum of all mass flows F_i into node k
P_k	Pressure at node k
V_k	Capacity at node k
w_k	Molecular weight of fluid flowing into node k
R	Gas Constant
$\sum_k F_i(P_k^t)$	Sum of flows F_i into node k at time t
P_k^t	Pressure at node k at time t
P_k^0	Pressure at node k at time 0

CHAPTER 6

STEADY-STATE AND DYNAMIC ANALYSIS OF KILN NETWORK

6.1. Introduction

Chapter 5 described the development of two new flow network programs which were based on the steady-state program **FLONET**. This chapter is devoted to a discussion of a particular test network which was analysed using **FLONET** and also **EQNET** and **DYNET** (with suitable modifications to the steady-state data set for the latter two programs).

6.2. HF Kiln Network

The network, referred to as test case 5 in Chapter 4, incorporates an HF kiln and associated piping (the network diagram is given in appendix iii). The kiln is heated by passing hot gas through 4 jackets placed round the kiln. The fuel gas (methane) is burnt with a large excess of air in the burner. Hot flue gas then separates into 4 streams, each of which then passes through a jacket. Once through the jackets the streams recombine and pass back via a recycle fan. Gas is purged off after the fan, this being replaced at a gas inlet in the burner feed line.

The flow through the jackets can be altered by adjusting any of the 9 butterfly valves within the network. Valves are situated on the inlet and outlet lines of each jacket and on the inlet line of the recycle fan.

The purpose of using the steady-state program, **FLONET**, here was to develop an appropriate operating schema for controlling the gas flows through the jackets using the 9 valves. The pressure in the system must be kept close to atmospheric to prevent either suction of cold air into the system or loss of gas out of it.

6.3. Steady-State Solution using FLONET

In the **FLONET** data set two node conditions were specified - the pressures at the external nodes '28' and '27' were set to atmospheric pressure. The solution to the network then converged in 6 iterations. (The data set and results for this test case are given in appendix vi).

The results for this network show that the flows in cross-lines '11'-'14', '13'-'16' and '12'-'15' are negligibly small in comparison with flows in the rest of the network and, in the case of the first two, are in fact negative. The results thus demonstrate that these lines are redundant in terms of controlling the flows through the jackets and indicate that they should be removed from the network.

6.4. Steady-State Solution using EQNET

Two versions of the original problem were solved using EQNET. In the first version, two lines which listed pipe physical data were replaced by equations. In the second version, a design specification was included. The flow in link (9,13) was specified as being equal to a fraction (0.8) of the total flow in links (6,10), (7,11) and (8,12). (The data set is listed as Case 5b in appendix vi).

The first version of the problem (Case 5a in appendix vi) is similar to the "mixed" network described in Chapter 5 - in neither problem do the equations in the data set represent design constraints. Case 5a was successfully solved, as might be expected, given the prior solution of the "mixed" network problem.

In the second version, as stated above, the list of equations included a flow constraint. Successful solution of this problem was achieved using EQNET. This illustrates the usefulness of the equation parsing facility in allowing such design constraints,which could not be specified in the original program.

6.5. Dynamic Solution using DYNET

In the dynamic version of test case 5, the steady-state data set was modified such that capacity was assigned to nodes 10, 11, 12, 13, 29 and 30, and links (2,24) and (30,23) were designated as control valves of pressure and flow respectively. The dynamic simulation involves disturbing the pressure at node 28 at random time intervals and noting the effect of this disturbance on the pressure at node 29 (node 29 is a pressure vessel whose pressure is controlled by the valve in link (2,24)).

Fig 6.2 in appendix vi shows the pressure variation with time at node 29. The upper graph shows the randomly generated pressure variation which is applied to the pressure at node 28 at randomly generated time intervals (over a period of 20 seconds). The lower graph shows pressure versus time at node 29.

The specified set pressure for the pressure vessel at node 29 is 1.0013 bar. The maximum pressure disturbance introduced was ± 500 Newtons and the time step for the simulation was 0.2 seconds. The problem was run for 2 seconds before pressure disturbance was instigated and the maximum allowed time interval for non-disturbance was set at 2.5 seconds.

The lower graph in Fig. 6.2 shows that after each perturbation of pressure at node 28, pressure control is exerted (by the control valve in link (2,24)) to force the pressure at node 29 towards the set value of 1.0013 bar. Only proportional control was applied, so there is an offset above 1.0013 bar.

6.6. Conclusions

Test case 5, representing an HF kiln heating network, was originally supplied as a steady-state problem, to be solved by **FLONET**. Having obtained a successful solution of the problem by the use of **FLONET**, the data set was modified to allow further analysis of the network using the programs **EQNET** and **DYNET**.

EQNET was applied to two versions of the steady-state problem. The original data set was modified by replacing pipe data by linear equations in pressure and flow. In the second of the two versions, a design specification, in the form of a linear equation, was included. Successful solution was obtained in both cases. The latter version demonstrated the enhanced capability of the equation parser to handle design constraints, in comparison with the data processing module of **FLONET** which can handle only fixed values of nodal pressures and flows.

DYNET was used to simulate pressure control at a node in the network where random pressure fluctuation was generated.

The solution of the original network problem, and of suitably modified versions of it, by **FLONET**, **EQNET** and **DYNET** demonstrated firstly the improvements made to the program **FLONET**, as represented by the programs **EQNET** and **DYNET**. Secondly, diverse aspects of flow network modelling were illustrated by the use of the three different programs.

CHAPTER 7

CONCLUSIONS

The work done in this project may be summarised in two parts. In the first part the aim was to produce a computer program which could be used in the analysis of steady-state flow network systems. Given fixed values of pressure and inflow/outflow at certain points in the network, the program was to solve for flows in all network branches and pressures at branch junctions. A network model was adopted which described the network in terms of a set of linearised equations containing flow and pressure terms. Two different linearisation methods - the Bending and Hutchison method and the Newton-Raphson method - were used in an iterative procedure to obtain flows and pressures at steady-state. The computer program was tested on a number of network problems, ranging from simple networks containing of the order of 10 pipes and nodes, to large, densely interconnected networks containing about 150 pipes and nodes and including pumps/fans and non-return valves. The program performed well on all the test cases provided. Modifications, however, had to be made to the solution algorithm when two of the test cases exhibited flow oscillation in certain network branches. A 'smoothing' procedure was added to the program to handle oscillation of branch flows. In general, convergence was fast and solution was obtained in less than 10 iterations for the medium size test networks (between 10 and 30 nodes and pipes), and in about 20 iterations for the larger networks tested.

With respect to the steady-state flow network program, further work could be done to determine the optimum switch-point from the Bending and Hutchison linearisation to the Newton-Raphson linearisation. Also, the program's ability to cope with networks where the solution oscillates needs to be tested to a greater extent. In the two test cases where oscillation of the solution occurred, it appeared to be caused by a disproportionately small flow in one part of the network which was adjacent to, or intrinsic in, a network mesh. The program should be tested further on networks of this type.

The second part of the project concerned work done on a network parser and on a program which could solve unsteady-state problems. The network parser was tested on two small network problems, one containing a mixture of written equations and numeric data. The HF network, tested first using the

steady-state program, was modified, with some pipe data being replaced by equations. The network parser satisfactorily handled the network specifications in the form of equations. Only simple equations were handled - the parser should be expanded to cope with more complicated equation forms, and more testing should be done to illustrate how much information should be specified in the equation list, so that over- or under-specification of the network does not occur.

The dynamic network program was tested on two simple pressure/flow control problems and on a network containing a compressor, and finally on the HF network mentioned above. Satisfactory solution of flows and pressures at steady-state was obtained in all cases. Further work could be done to link this program with the network parser so that equation lists could be used to describe all or part of the network.

Viewed as a whole, the project illustrated a variety of requirements which must be met in a computer program designed to analyse flow networks. The program which was written to solve steady-state flow networks successfully solved all the test cases which were supplied ; network specification in the form of equation lists was tested using a network parser, and the steady-state program was extended to include a capability for dynamic analysis.

With reference to the three aims stated in Chapter 1, the project can be considered to have been successful in all of these, particularly with regard to the second aim. The computer program which was designed for the the solution of steady-state network problems was a modified version of an earlier program. The earlier program had been unable to obtain a solution for several of the test networks solved by the new program. The robustness of the algorithm used in the new program was thus effectively demonstrated.

To sum up, this project has examined methods of modelling fluid flow networks by computer, and has successfully applied a number of these methods in the design of three computer programs to solve flow network problems.

I. Bibliography

1. ALDER G.M., "Pipe Net - A computer program for the analysis of hydraulic networks", Dept. of Mechanical Engineering, Edinburgh University, November 1980

2. BARLOW J.F. and MARKLAND E., "Computer analysis of pipe networks", Proc. Institution of Civil Engineers, 43 (June), pp249-259, 1969

3. BENDER E., Simulation of dynamic gas flows in networks including control loops", Computers & Chemical Engineering, Vol. 3, pp611-613, 1979

4. BENDING M.J. and Hutchison H.P., "The calculation of steady-state incompressible flow in large networks of pipes", Chemical Engineering Science, Vol. 28, pp1857-1864, 1973

5. BENDING M.J. and Hutchison H.P., "TRGB Routines", Appendix to [4]

6. BRAMELLER A., ALLAN R.N. and HAMAM Y.M., "Sparsity", Pitman Publishing, pp 58-99, 1976

7. CALAHAN D.A. and AMES W.G., "Vector Processors : Models and Applications", IEEE Transactions on Circuits and Systems, Vol CAS-26 No. 9 (September), pp715-725, 1979

8. CHANDRASHEKAR M., "Extended sets of components in pipe networks", Journal of the Hydraulics Division, Proc ASCE, Vol. 106, No. HY1, pp 133-149, January 1980

9. CHANDRASHEKAR M. and STEWART K.H., "Sparsity Oriented Analysis of Large Pipe Networks", Journal of the Hydraulics Division, Proc ASCE, Vol. 101, No. HY4, pp 341-355, April 1975

10. CHEN H. and STADTHERR M.A., "On solving large sparse nonlinear equation systems", Computers & Chemical Engineering, Vol. 8, pp1-7, 1984

11. CHEN N.S. "An explicit equation for friction factor in pipe", Ind. Eng. Chem. Fundam., Vol. 18, No. 3, pp296-297, 1984,

12. CROSS H. "Analysis of Flow in Networks of Conduits or Conductors" Univ. of Illinois, Bull. 286, Nov. 1946

13. DUFF I.S., "A Survey of Sparse Matrix Research", Proceedings of the IEEE, Vol. 65, No. 4, pp 500-535, 1977

14. DUFF I.S. and STEWART G.W., "Practical Comparisons of Codes for the solution of Sparse Linear Systems", Sparse Matrix Proceedings 1978 Society for Industrial & Applied Mathematics, Philadelphia

15. EDWARDS M.F., JADALLAH M.S.M. and SMITH R, "Head losses in pipe fittings at low Reynolds numbers", Trans. Institution Chem. Engineers, January 1985

16. GAY B. and MIDDLETON P., "The solution of pipe network problems", Chemical Engineering Science, Vol. 26, pp109-123, 1971

17. GAY B. and PREECE P.E., "Matrix methods for the solution of fluid network problems : Part I - Mesh methods", Trans. Institution Chem. Engineers, Vol. 53, No. 1, pp 12-15, 1975

18. GAY B. and PREECE P.E., "Matrix methods for the solution of fluid network problems : Part II - Diakoptic methods", Trans. Institution Chem. Engineers, Vol. 55, No. 1, pp 38-45, 1977

19. GOSTOLI C. and SPADONI G., "Linearisation of the head-capacity curve in the analysis of pipe networks including pumps", Computers & Chemical Engineering, Vol. 9, No.1, pp89-92, 1985

20. GOLDWATER M.H., ROGERS K. and TURNBULL D.K., "The Pan Network Analysis Program - its development and use", The Institution of Gas Engineers, Communication 1005, November 1976

21. HUTCHISON H.P., "Simulation of Steam Distribution Networks", Paper : Source - Chem. Eng. Dept Literature

22. ISAACS L.T. and MILLS K.G., "Linear Theory Methods for Pipe Network Analysis", Journal of the Hydraulics Division, Proc ASCE, Vol. 106, No. HY7, pp 1191-1201, July 1980

23. LANG F.D. and MILLER B.L., "Use of friction-factor correlation in pipe-network problems", Chem. Engineering, Vol. 88, No. 13, pp 95-97, 1981

24. LIEBE R., "Numerical Simulation of Nonlinear Networks with Complex Feedbacks" CEF 87 : XVIII Congress, The Use of Computers in Chemical Engineering EFCE

25. MAH R.S.H., "Pipeline Network Calculations using sparse computation techniques", Chem. Eng. Science, Vol. 29, pp 1629-1638, 1974

26. MAH R.S.H. and SHACHAM M., "Pipeline Network Design and Synthesis", Advances in Chem. Engineering, Vol. 10, 1978

27. NAHAVANDI A.N. and CATANZARO G.V., "Matrix Method for Analysis of Hydraulic Networks", Journal of the Hydraulics Division, Proc ASCE, Vol. 99, No. HY1, pp 47-63, January 1973

28. STADTHERR M.A. and VEGEAIS J.A., "Process flowsheeting on supercomputers", Chem. Eng. Dept., Univ of Illinois, 1985

29. TAYLOR R.G., "Fluid Flow Network Analysis (A computer program)", Research & Development Dept, Theoretical Exploratory Group, ICI July 1970, File No. A.127,564

30. VARGA R.S., "Matrix Iterative Analysis", Prentice-Hall, 1962

31. WILCOCKSON R.B., "Pipeline Simulation", Pipeline, No. 21, pp 10-12, November 1985

32. WILLIAMS P.W., "Numerical Computation", Nelson, pp 57-72

33. WISHART R. "The Solution of Flow Network Problems", Literature Survey, Dept. of Chem. Engineering, Edin. Univ., 1983

34. WOOD D.J. and THORLEY A.R.D., "BASIC Computer Program for the analysis of pressure and flow distribution systems including extended period simulations", Dept. of Mech. Eng., The City University, London, 1983

35. PIPENET Computer Aided Design Centre, Cambridge

36. PIPEPHASE SimSCi Simulation Sciences Inc. Stockport, Cheshire

II. Flow Diagrams for FLONET main program and modules

In the flow diagrams shown in this appendix, the following conventions apply. Boxes which are connected by horizontal lines entering or leaving the sides of the box signify sequential instructions at the same program level. Boxes which are connected by vertical lines signify nested program levels. The symbol 0 in a box indicates that the box contains a conditional statement and that lower-level statements beneath the current box will only be executed if the conditional statement is true. The symbol * in a box signifies that all lower-level statements connected to the current box will be executed iteratively, for the number of cycles specified in the box, or until the exit condition is satisfied.

1. Program **FLONET**

2. **NLINK** : Gets number of links to each network node.

3. **PMPCAR** : Gets number of points supplied on each pump characteristic.

4. **PMPNET** : Gets pump equation (calls FPUMP, LPUMP or QPUMP), depending on the number of points supplied as data for each pump.

5. **FPUMP** :Expresses pump equation as "flow = constant" (only one point on pump characteristic has been supplied as data).

6. **LPUMP** : Expresses pump equation as "$(P_{in} - P_{out}) = A * flow + B$" (two points on pump characteristic have been supplied as data).

7. **QPUMP** : Expresses pump equation as "$(P_{in} - P_{out}) = A * flow^2 + B$" (three or more - up to five - points on pump characteristic have been supplied as data).

8. **LINNET** : Calculates new network flows at the current values of nodal pressures, and linearises the flow/pressure relationship in all network links. Calls LNPUMP and LINPIP.

9. **LNPUMP** : Obtains pump flow/pressure relationship of the form "$flow = A * (P_{in} - P_{out}) + B$".

10. **LINPIP** : Obtains pipe (or valve) flow/pressure relationship of

the form "flow = A * (P_{in} - P_{out}) + B".

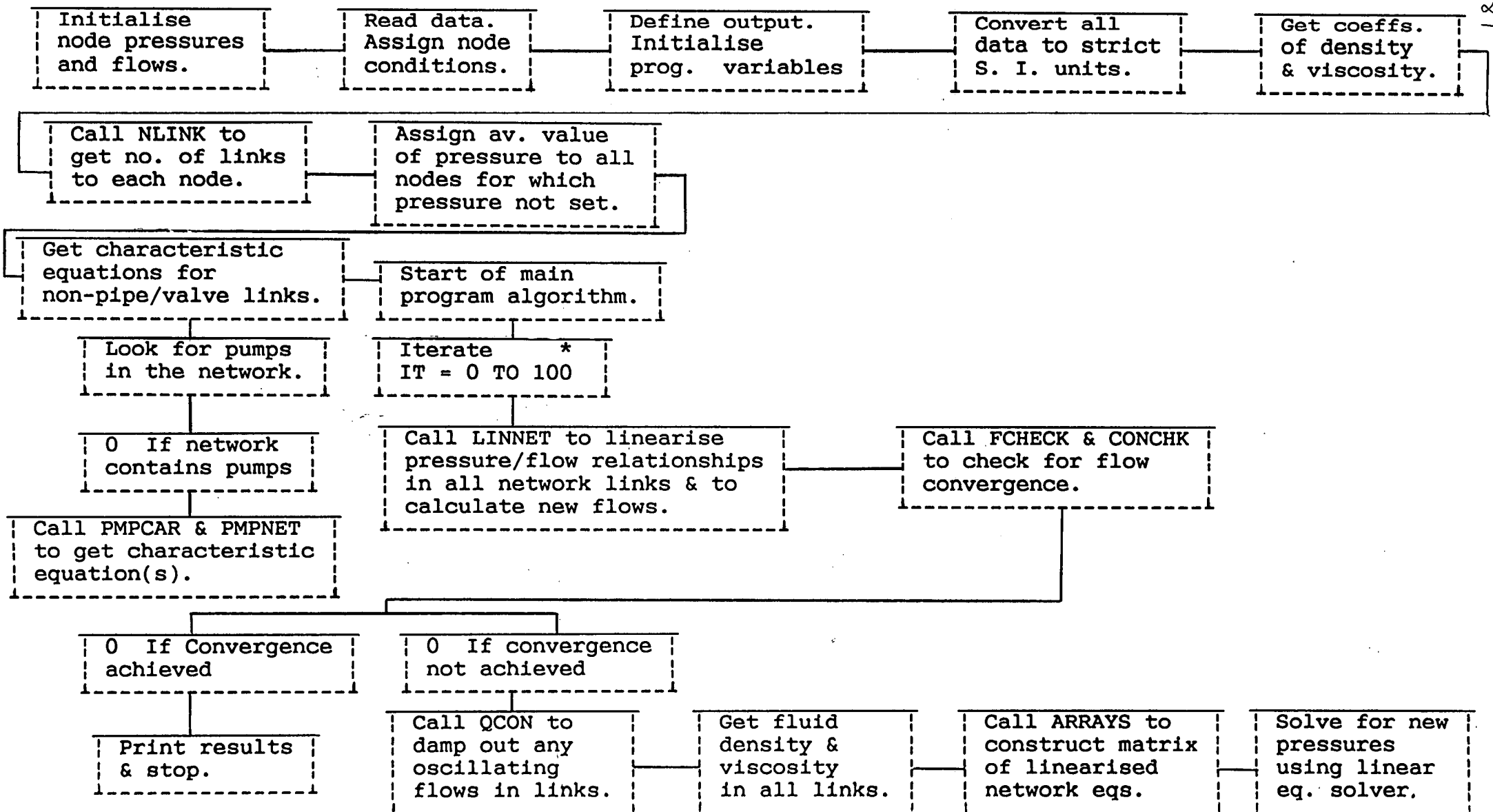
11. **FCHECK** : Checks whether absolute flow convergence has been achieved (flow residuals at all nodes are < 0.001 kg/s).

12. **CONCHK** : Examines behaviour of maximum flow residuals over number of iterations and sets flags for links in which flow oscillation occurs.

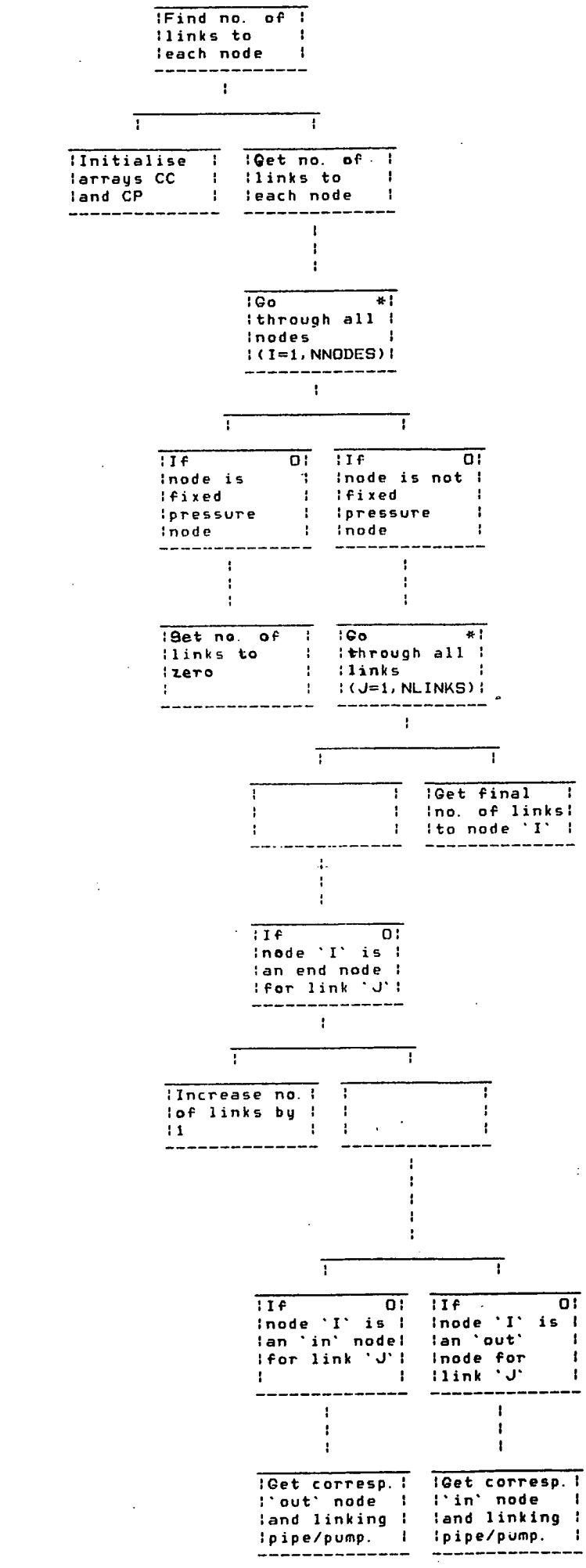
13. **QCON** Averages out oscillating flows in network links.

14. **ARRAYS** : Constructs matrix of linnearised flow/pressure equations for all network links. The matrix also contains equations which specify fixed pressures or inflows/outflows at nodes.

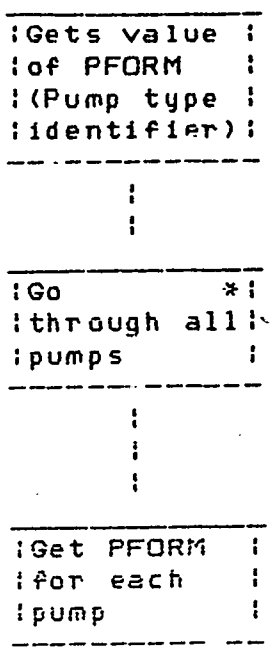
FLOW DIAGRAM FOR PROGRAM FLONET



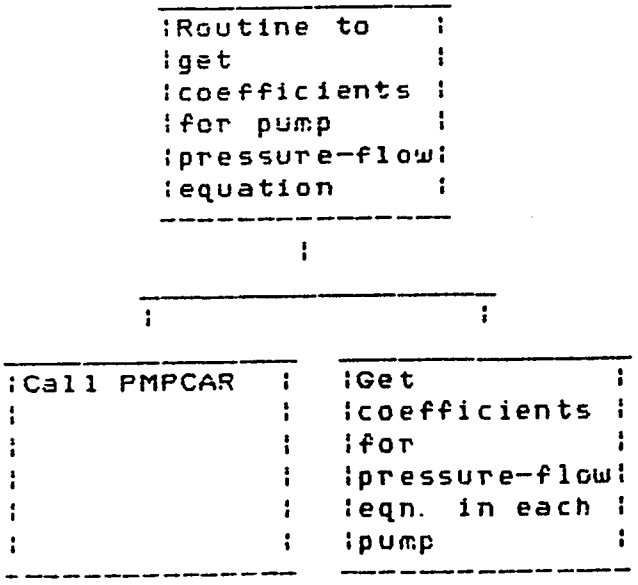
NLINK(NNODES, NLINKS, MAX, IN, OUT, LINK, CC, CP, PP, FEXX)



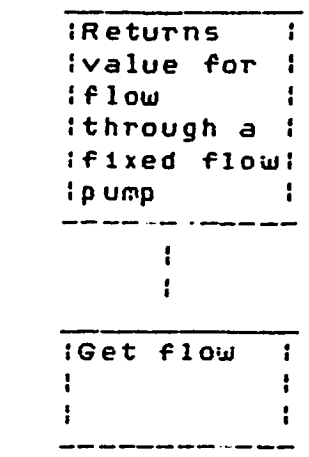
PMPCAR(NPTS, PFORM, NPUMP)



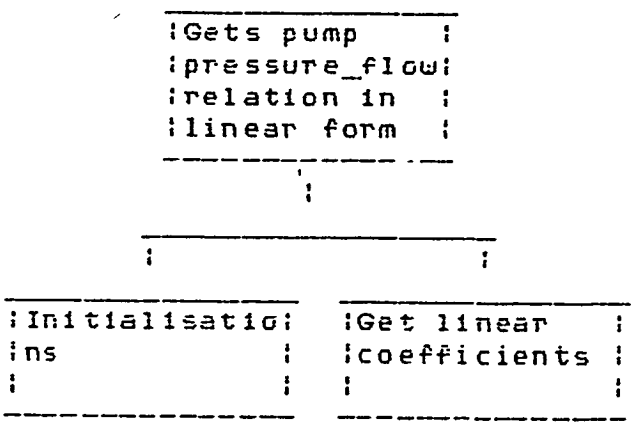
PMPNET(PCHAR, C1, C2, NPTS, PFORM, NPUMP)



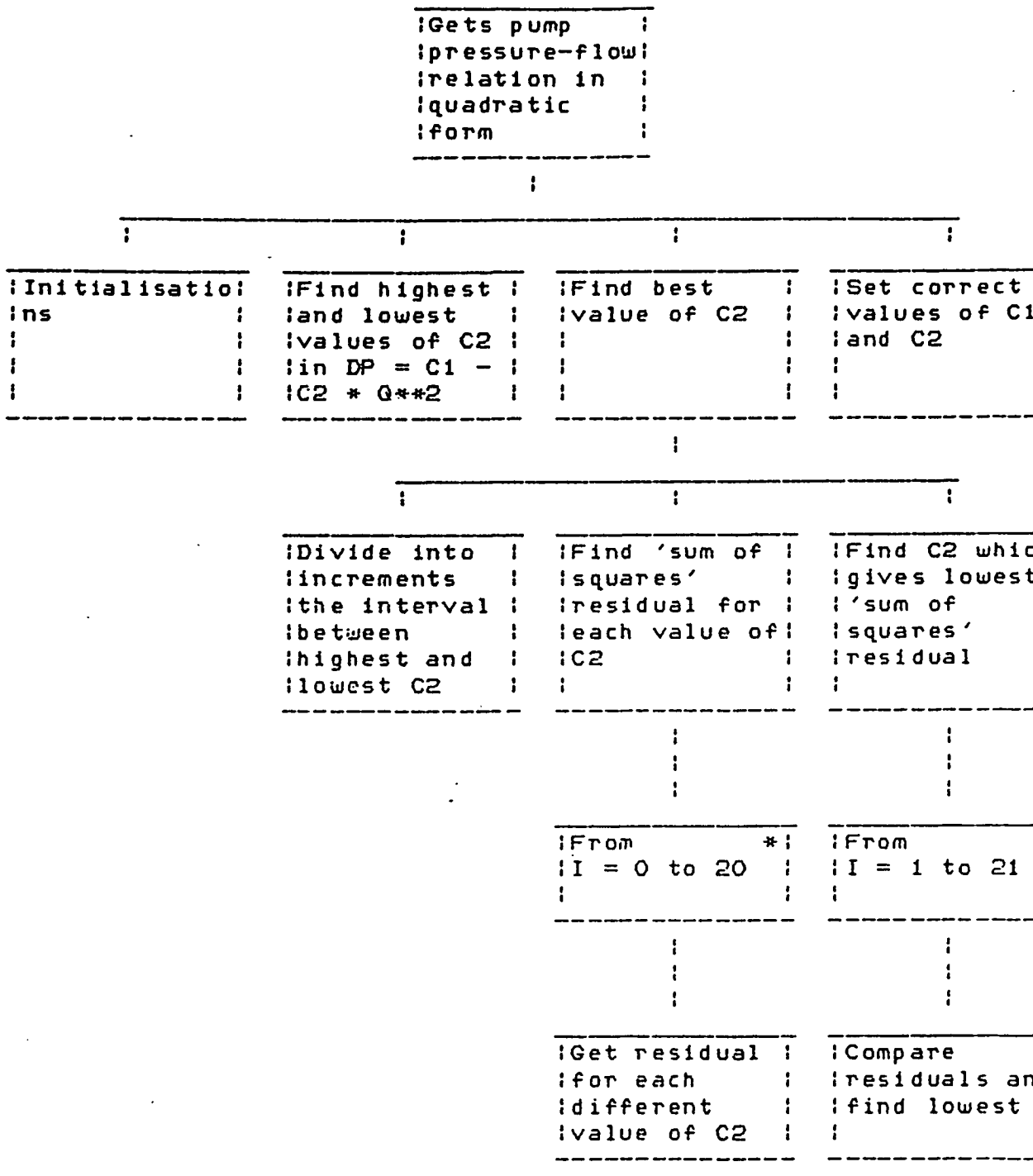
FPUMP(PCHAR, C1, C2, NPUMP)



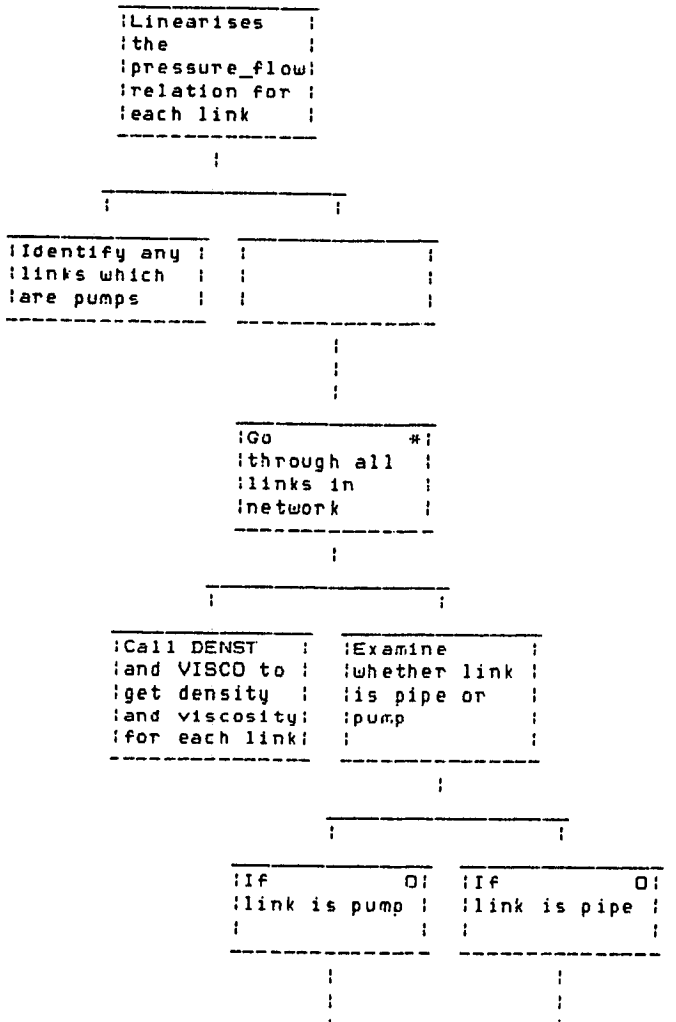
LPUMP(PCHAR, C1, C2, NPUMP)



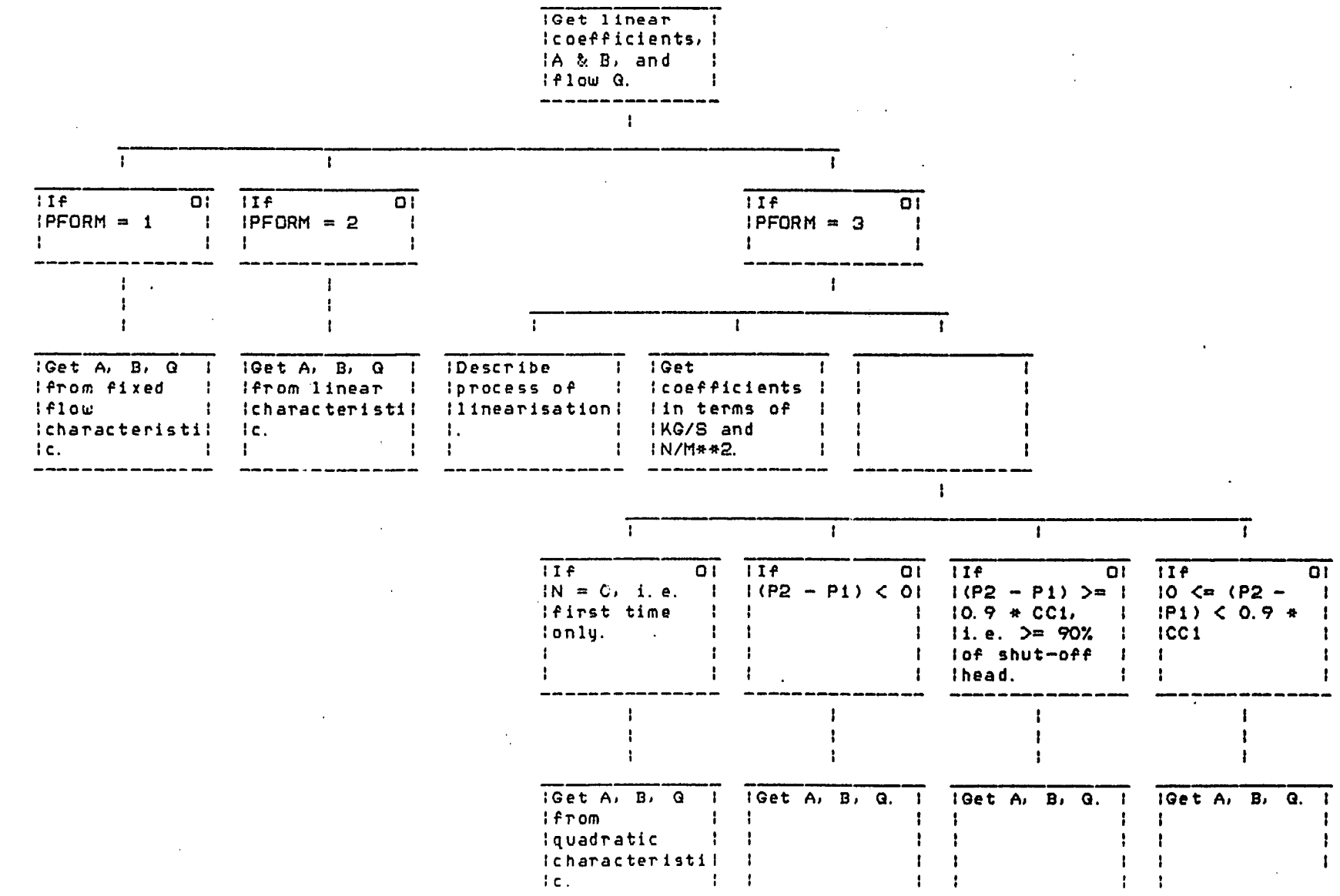
QPUMP(PCHAR, C1, C2, NPTS, NPUMP)



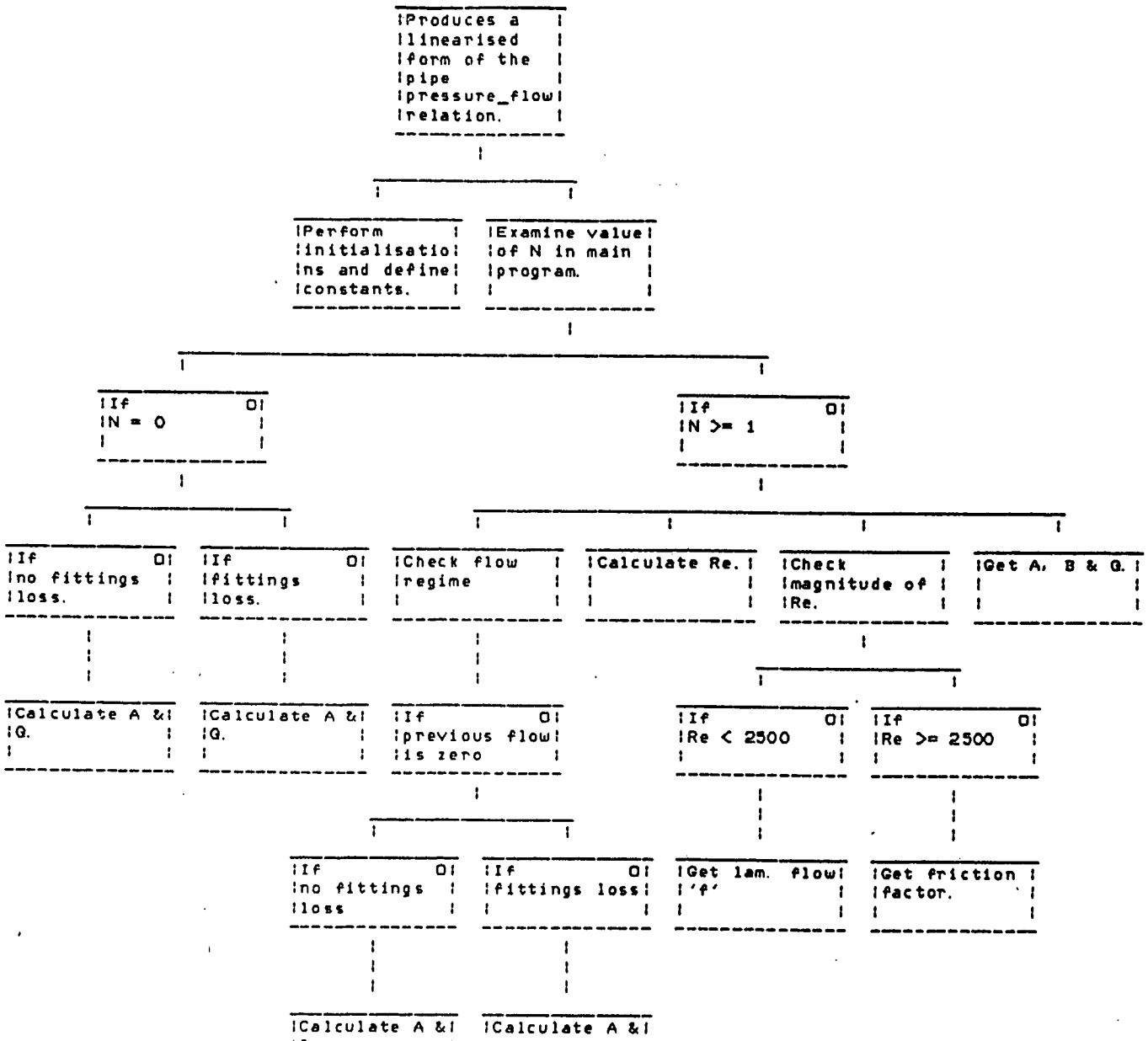
LINNET(PP, PLENG, BORE, ROUGH, FITLOS, TEMP, HEIGHT, CD, CV, C1, C2, A, B, Q, QO, PLOSS,
+IN, OUT, IPBR, NP, NPUMP, IT, LFLOW, MAX, PFORM, NFLUID)



UNPUMP(P1, P2, C1, C2, DEN, PFORM, N, Q, A, B)

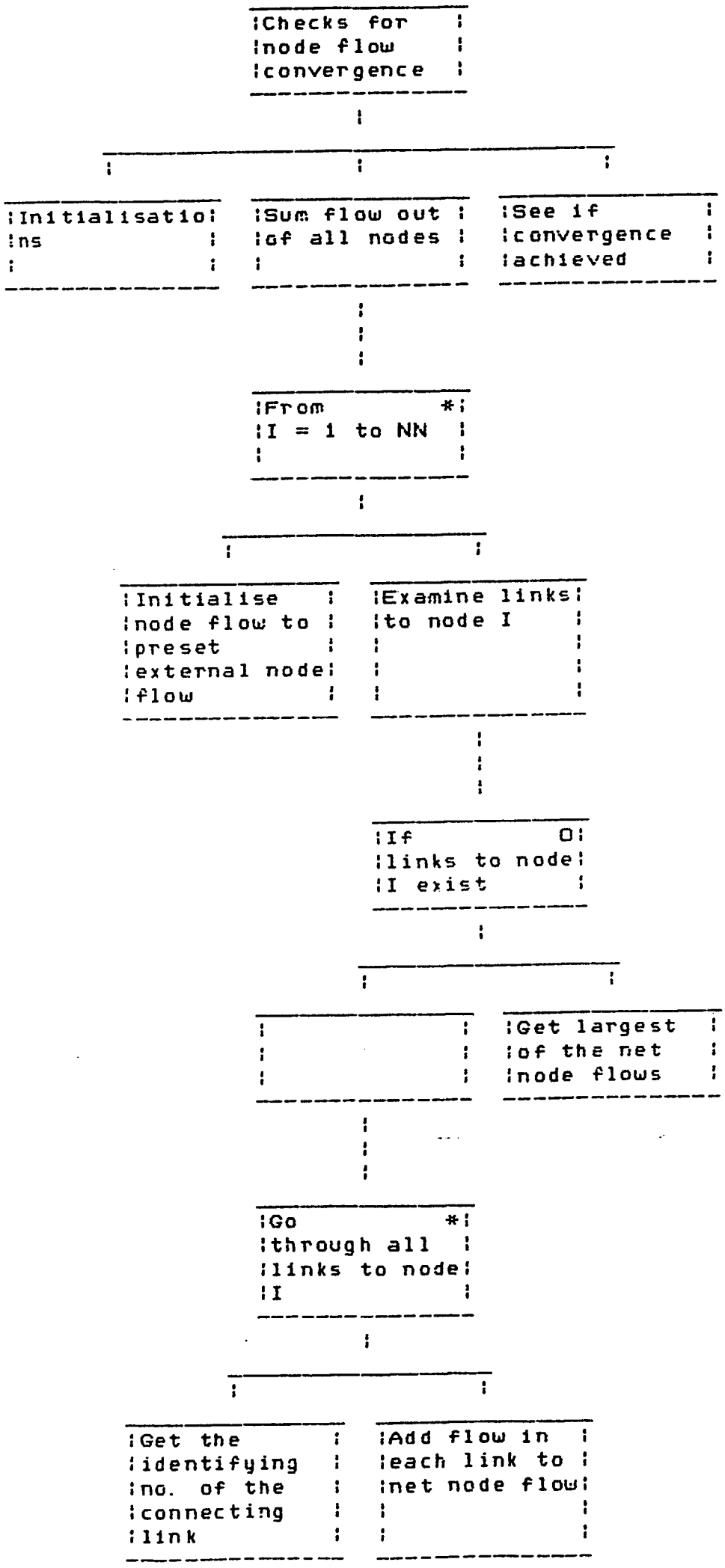


LINPIP (P1, P2, L, D, DENAV, DENY1, DENY2, VIS, RK, FIT, N, Q, QO, A, B, PLO, LFLOW)

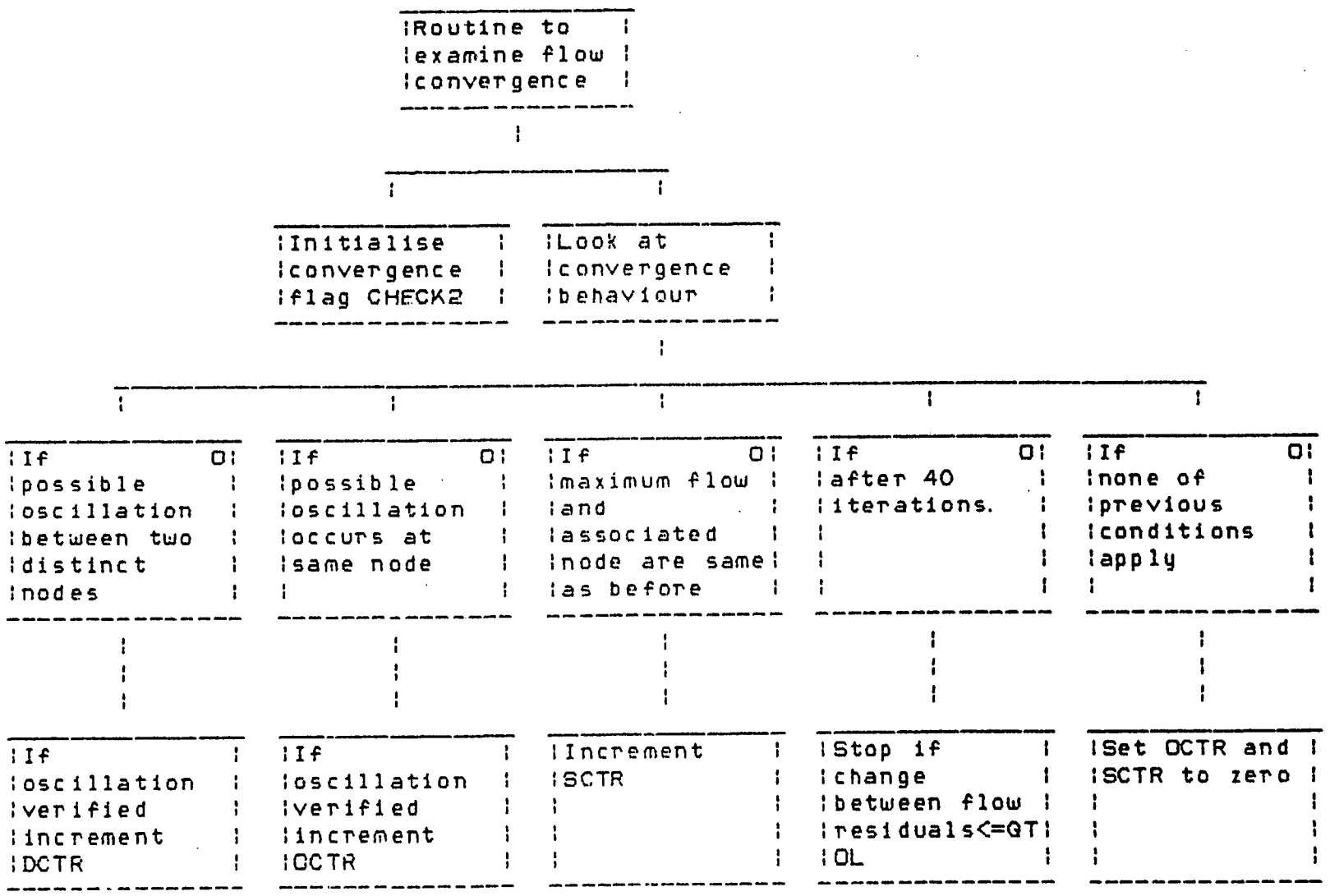


FCHECK(G,FEXX,LINK,IN,OUT,CP,NN,LFLOW,MAX,FTOL,HFTOT,HFNOD,CHEC

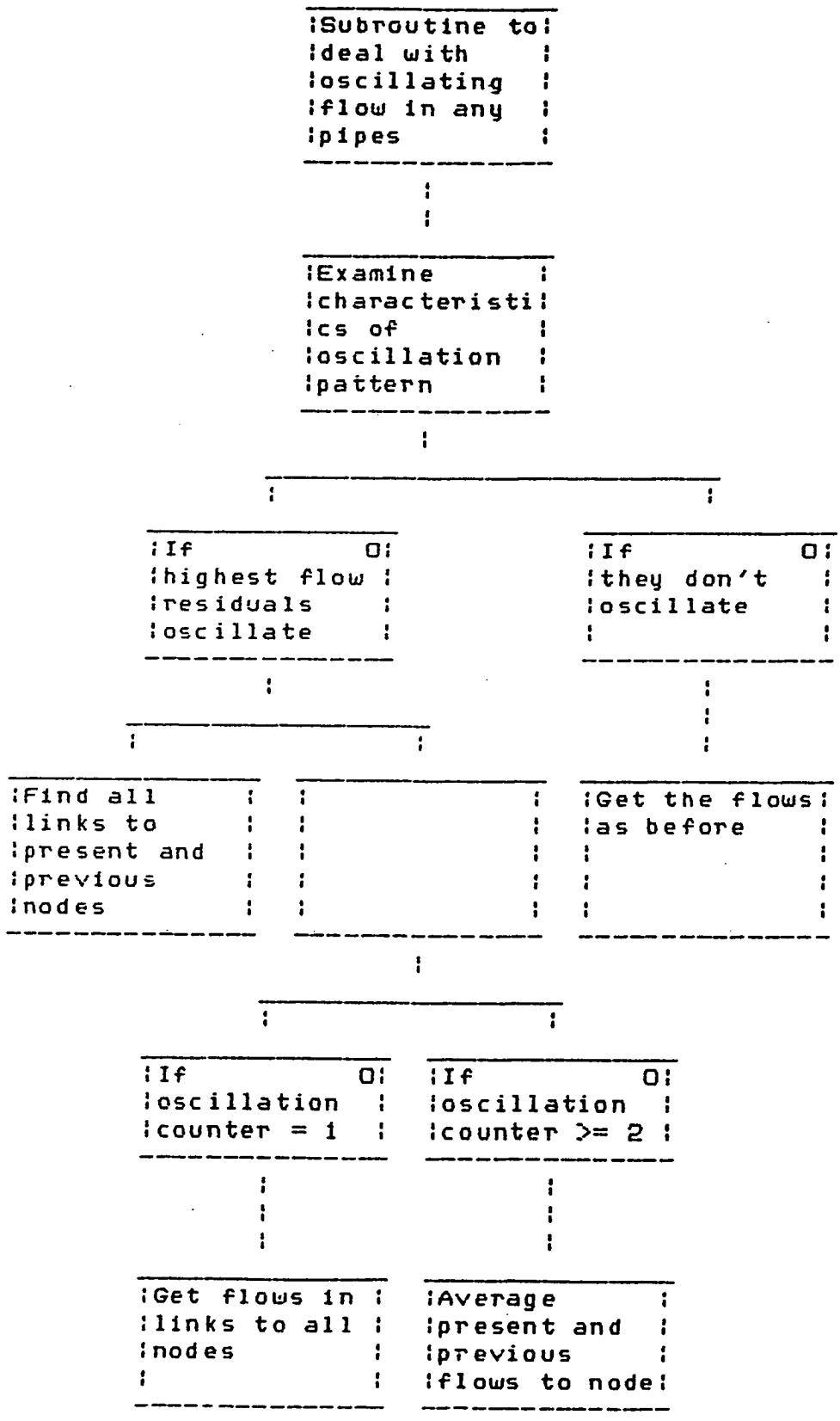
RCHEC



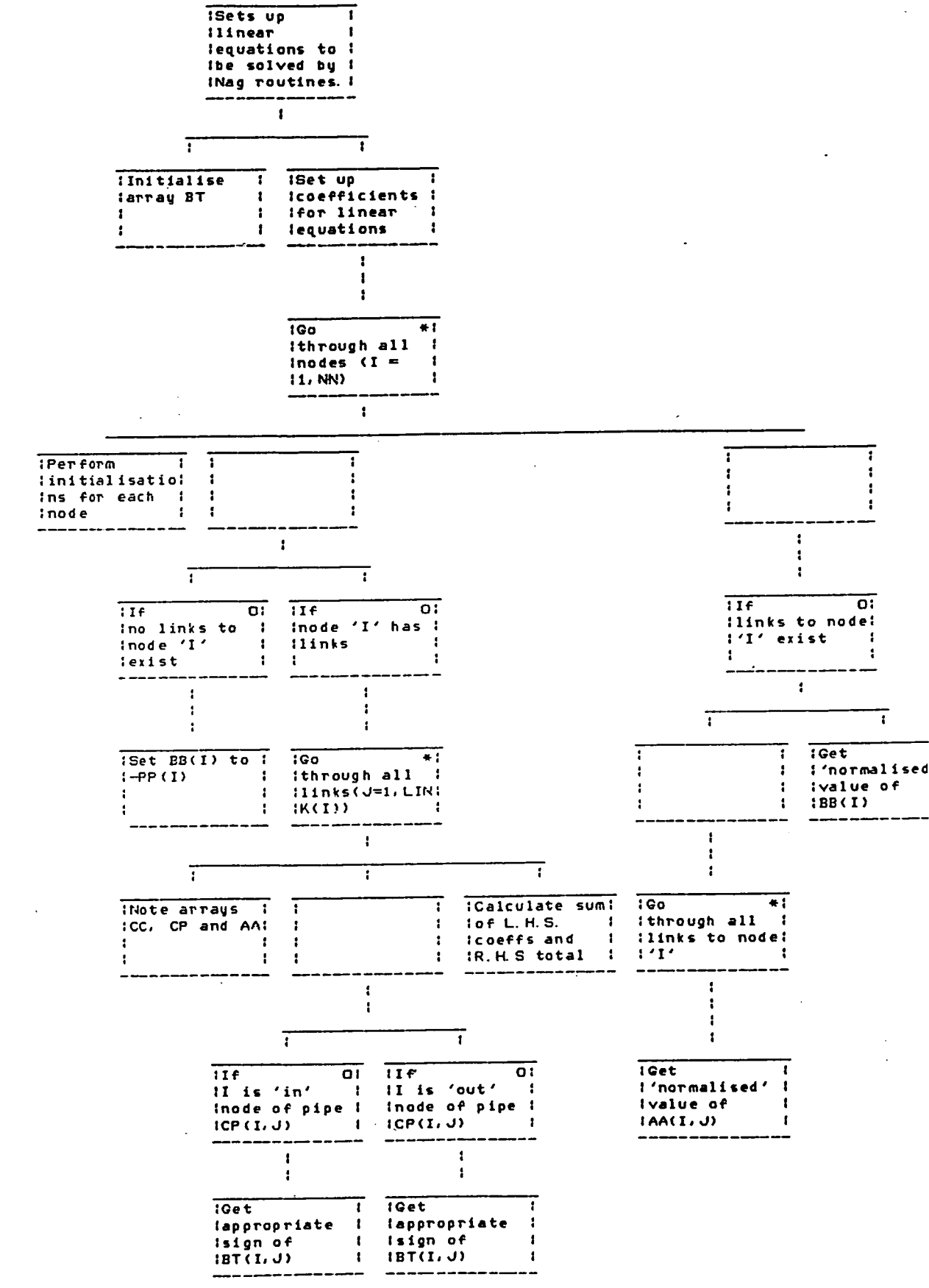
CONCHK(HFNOD, PHFNOD, SHFNOD, HFTOT, PHFTOT, SHFTOT, QTOL,
SCTR, OCTR, DCTR, OFLAG, PFLAG, IT, CHECK2, NOR, FOR, DOR)



QCON(Q, Q1PV, HFTOT, PHFTOT, LINK, CP, DCTR, OCTR, HFNOD, PHFNOD, MAX)



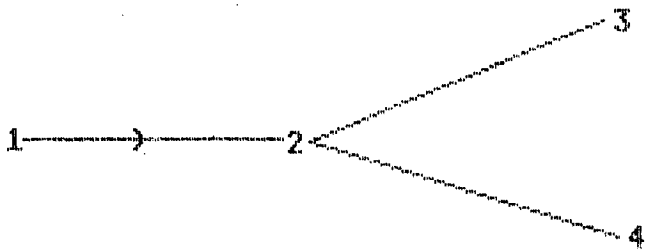
ARRAYS(NN, MAX, CC, CP, IN, LINK, A, B, AA, BB, PP, FEXX, DEN, HEIGHT)



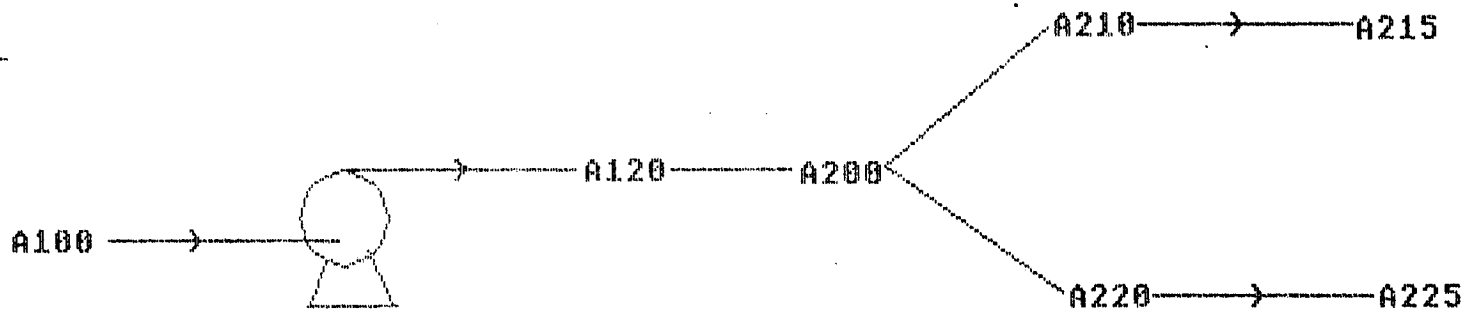
III. Data, Results and Diagrams for Steady-State Networks

This appendix lists information referred to in Chapter 4. Diagrams are presented for all network test cases. Graphs of maximum nodal flow residuals vs. iteration step are also given for all test cases (except for Case 4(iii)). Data sets and results are given for Cases 1, 2, 3, 4, 7, 8 and 10.

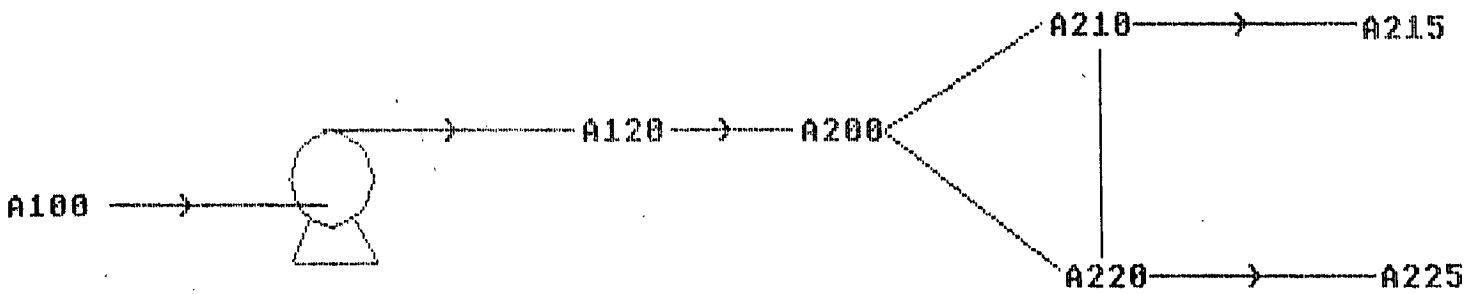
CASE1



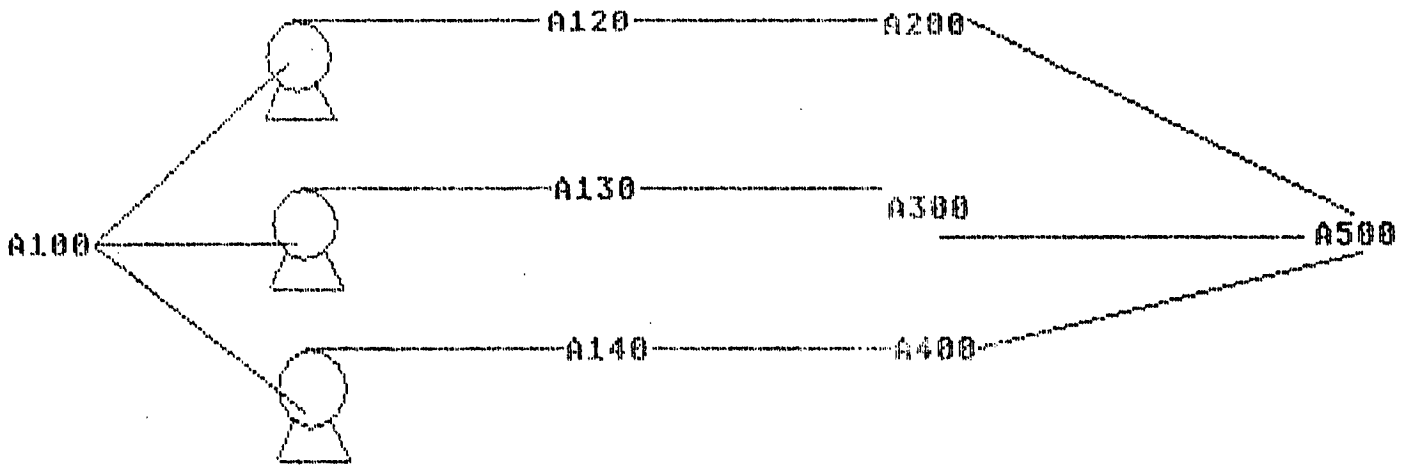
CASE2



CASE3

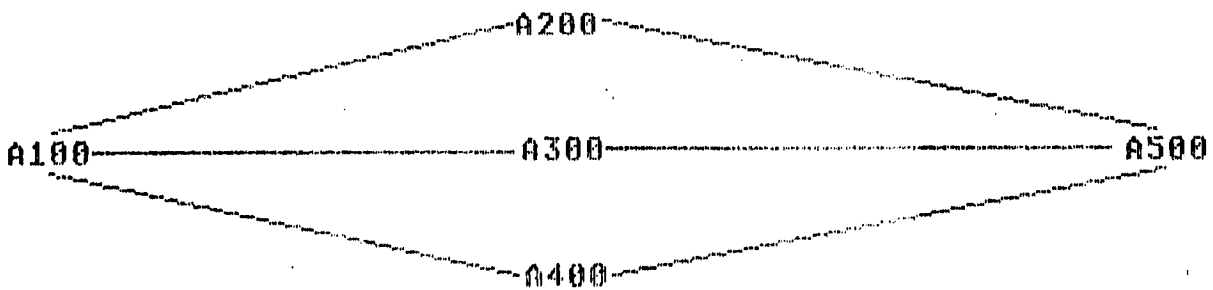


CASE4



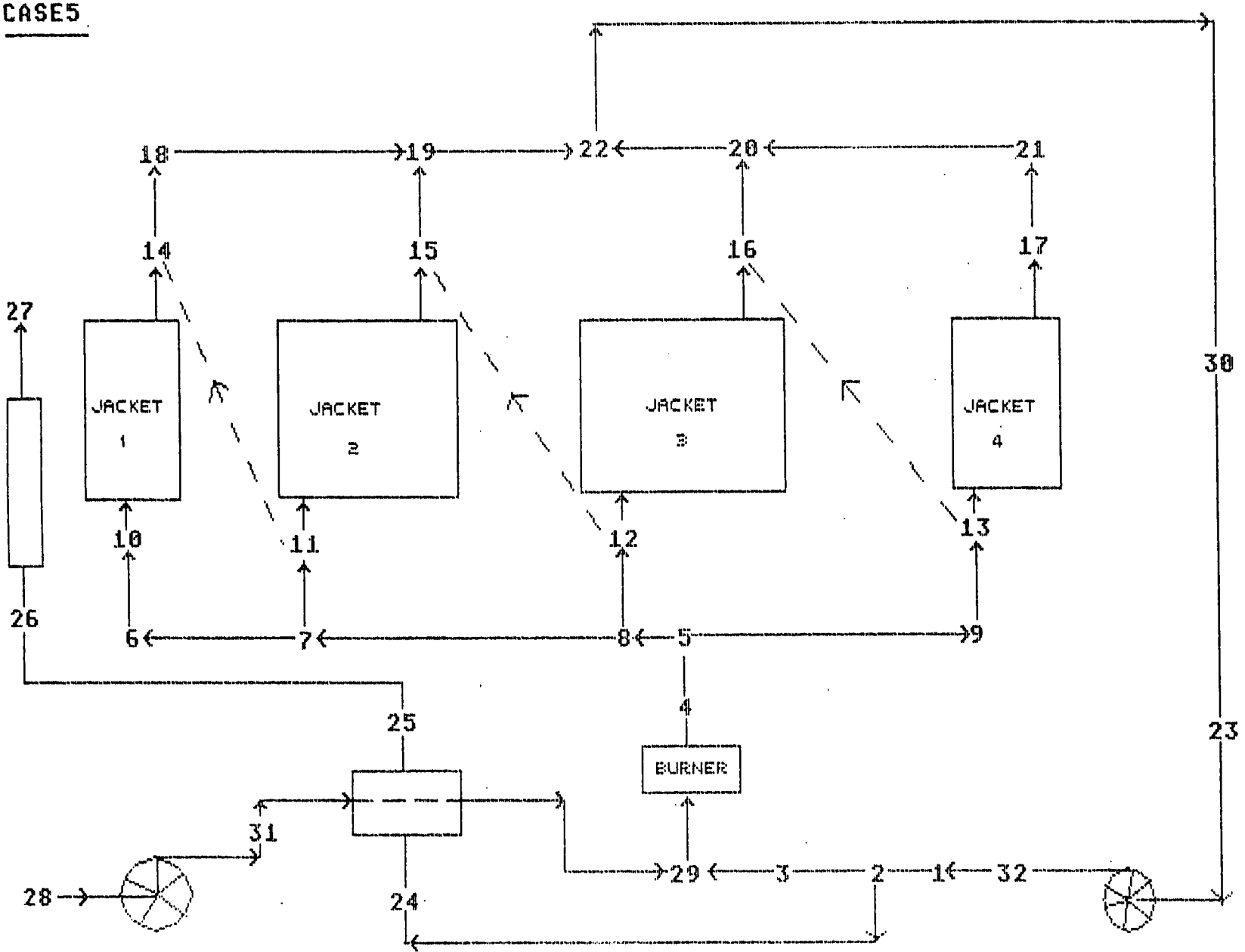
(i)

(ii)

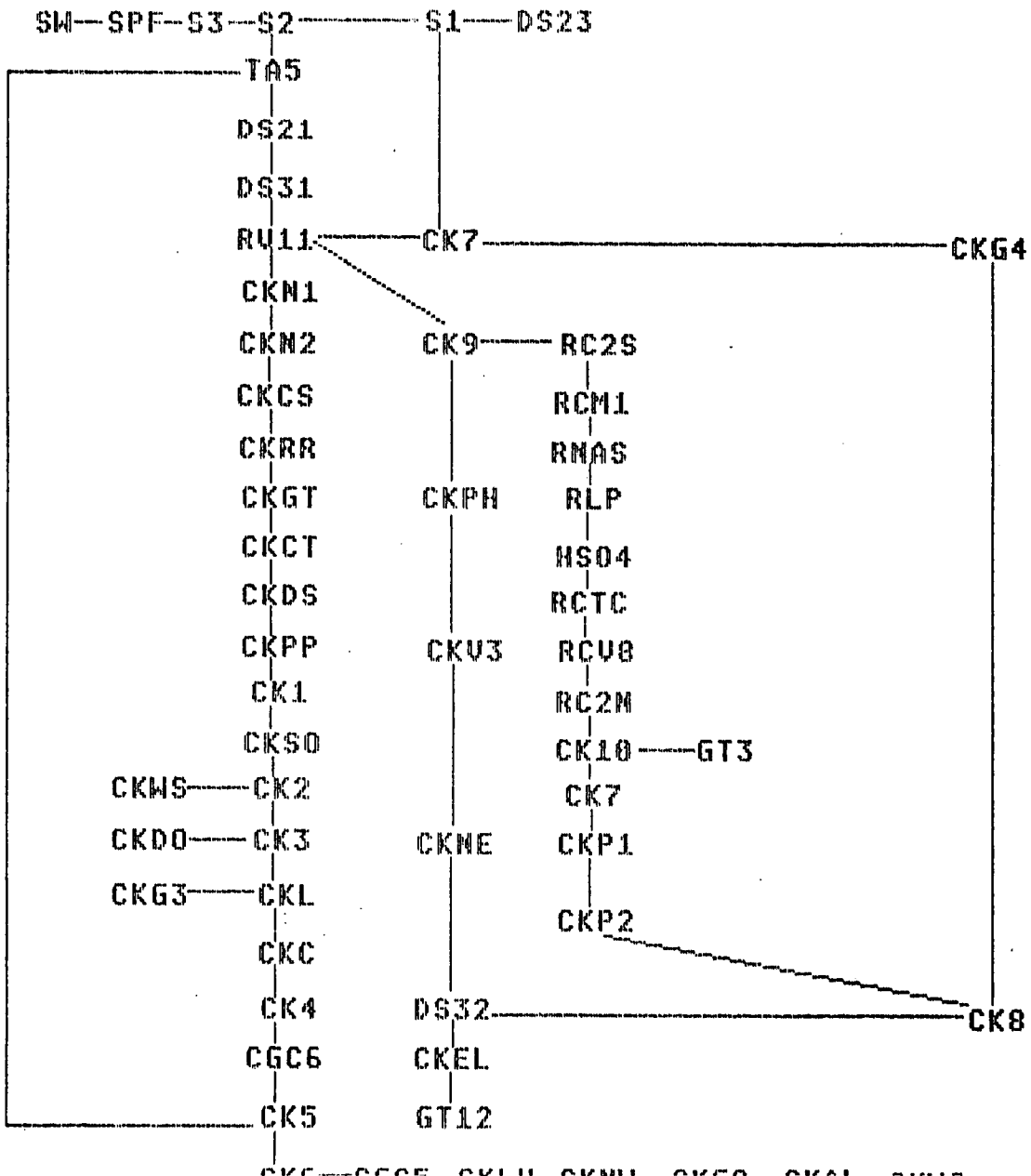


(iii)

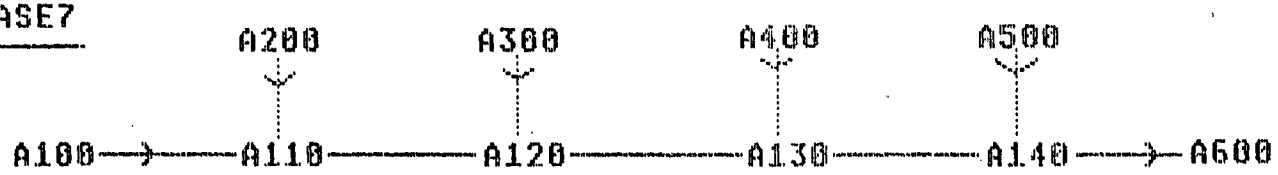
CASE5


18
19
22
20
21
14
15
16
17
27
JACKET 1
JACKET 2
JACKET 3
JACKET 4
30
10
11
12
13
26
6
7
8
5
9
25
4
BURNER
23
31
28
29
3
2
1
32
24

CASE6

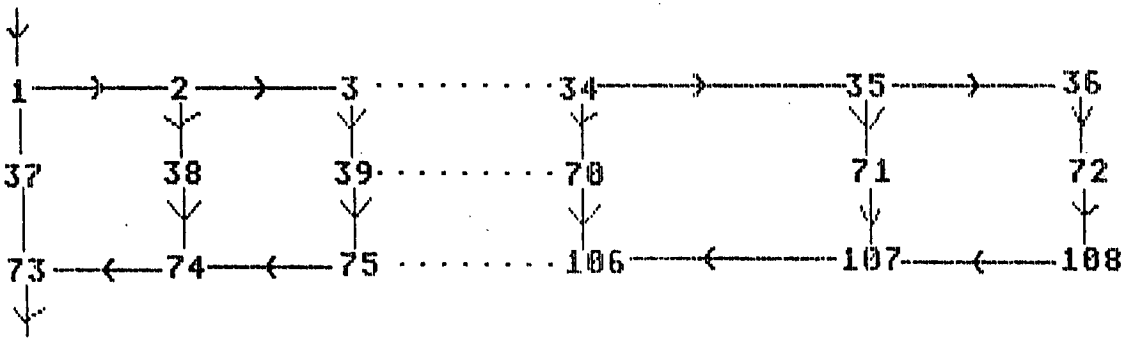

SW
SPF
S3
S2
S1
DS23
TA5
DS21
DS31
RV11
CK7
CKG4
CKN1
CKN2
CK9
RC2S
CKCS
RCM1
CKRR
RNAS
CKGT
CKPH
RLP
CKCT
HS04
CKDS
RCTC
CKPP
CKV3
RCV8
CK1
RC2M
CKS0
CK10
GT3
CKWS
CK2
CK7
CKDO
CK3
CKNE
CKP1
CKG3
CKL
CKP2
CKC
CK4
DS32
CK8
CGC6
CKEL
CK5
GT12

CASE7

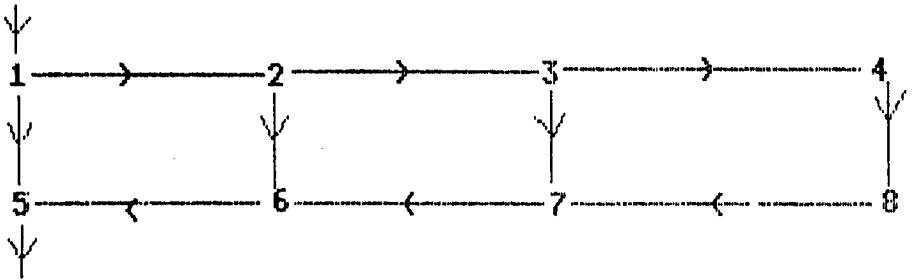


CASE8

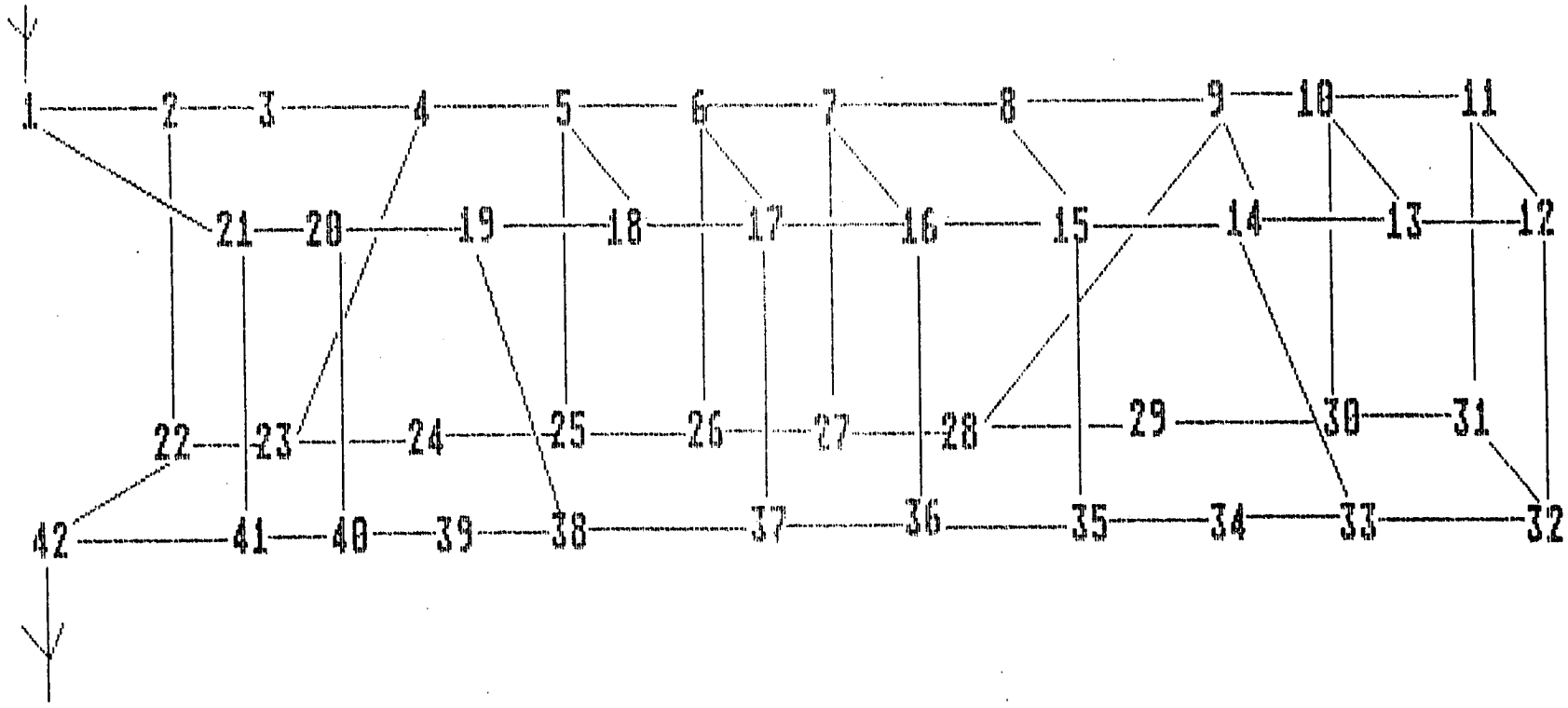
CASE9

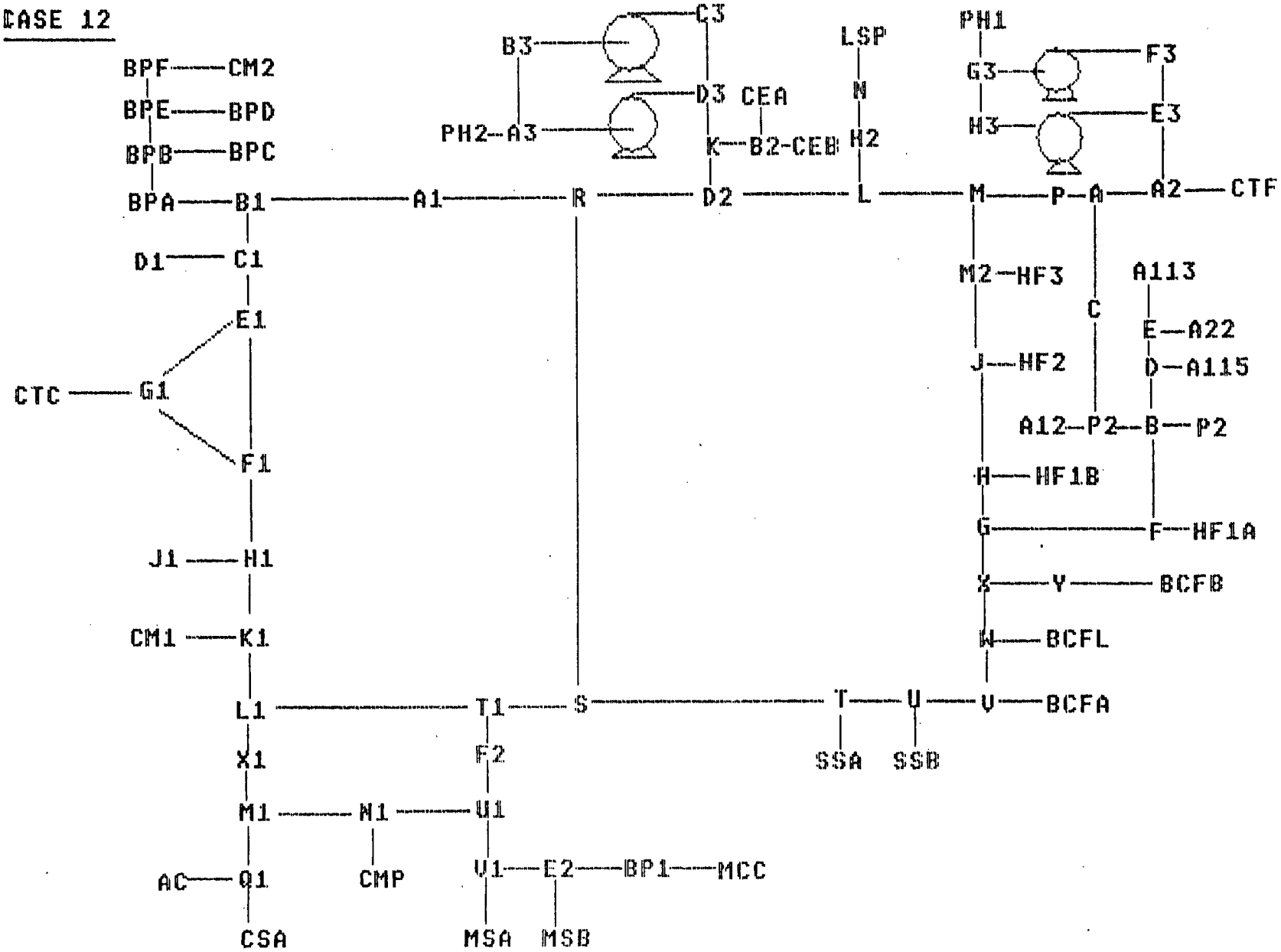


CASE10

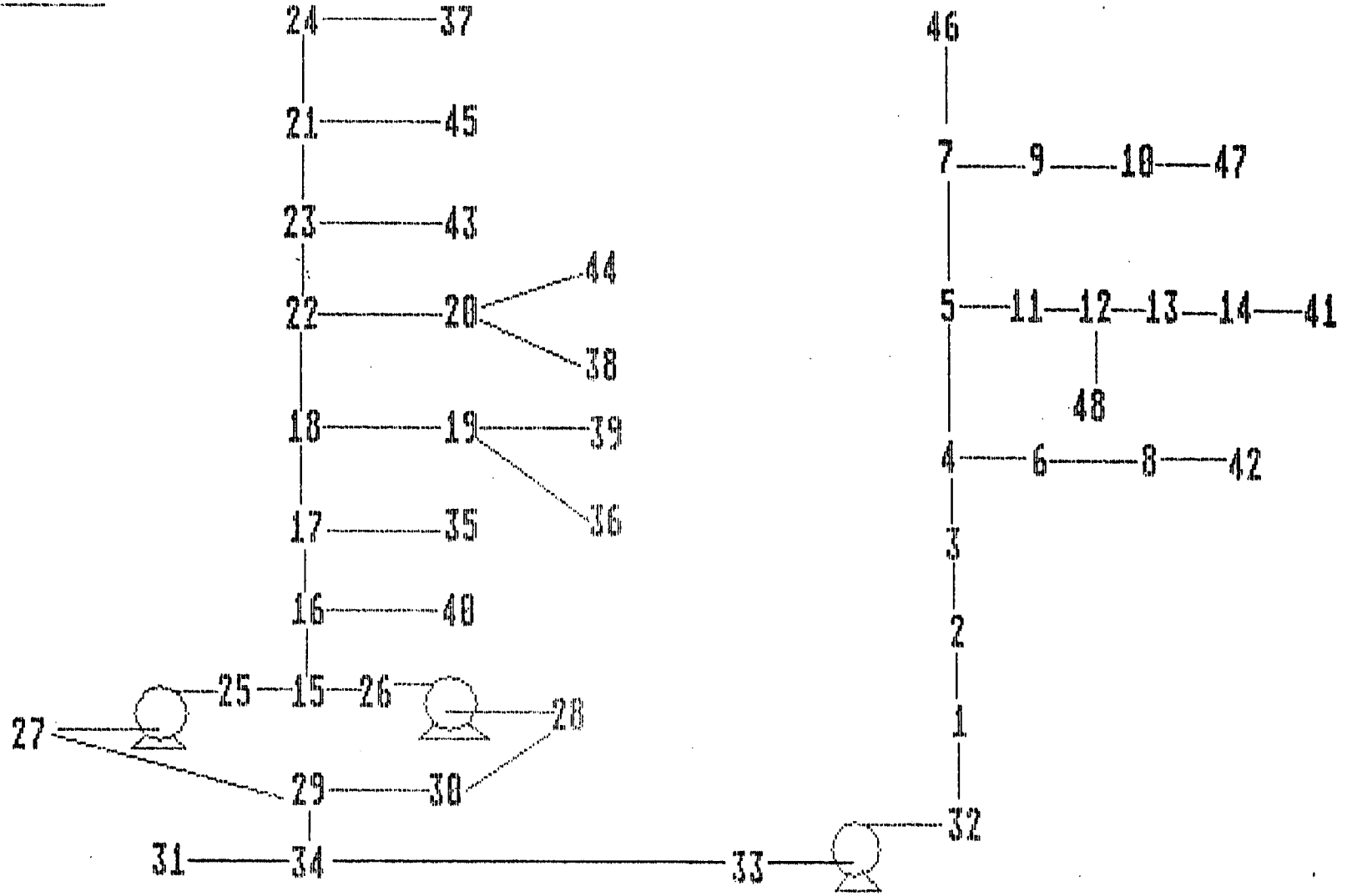


CASE 11

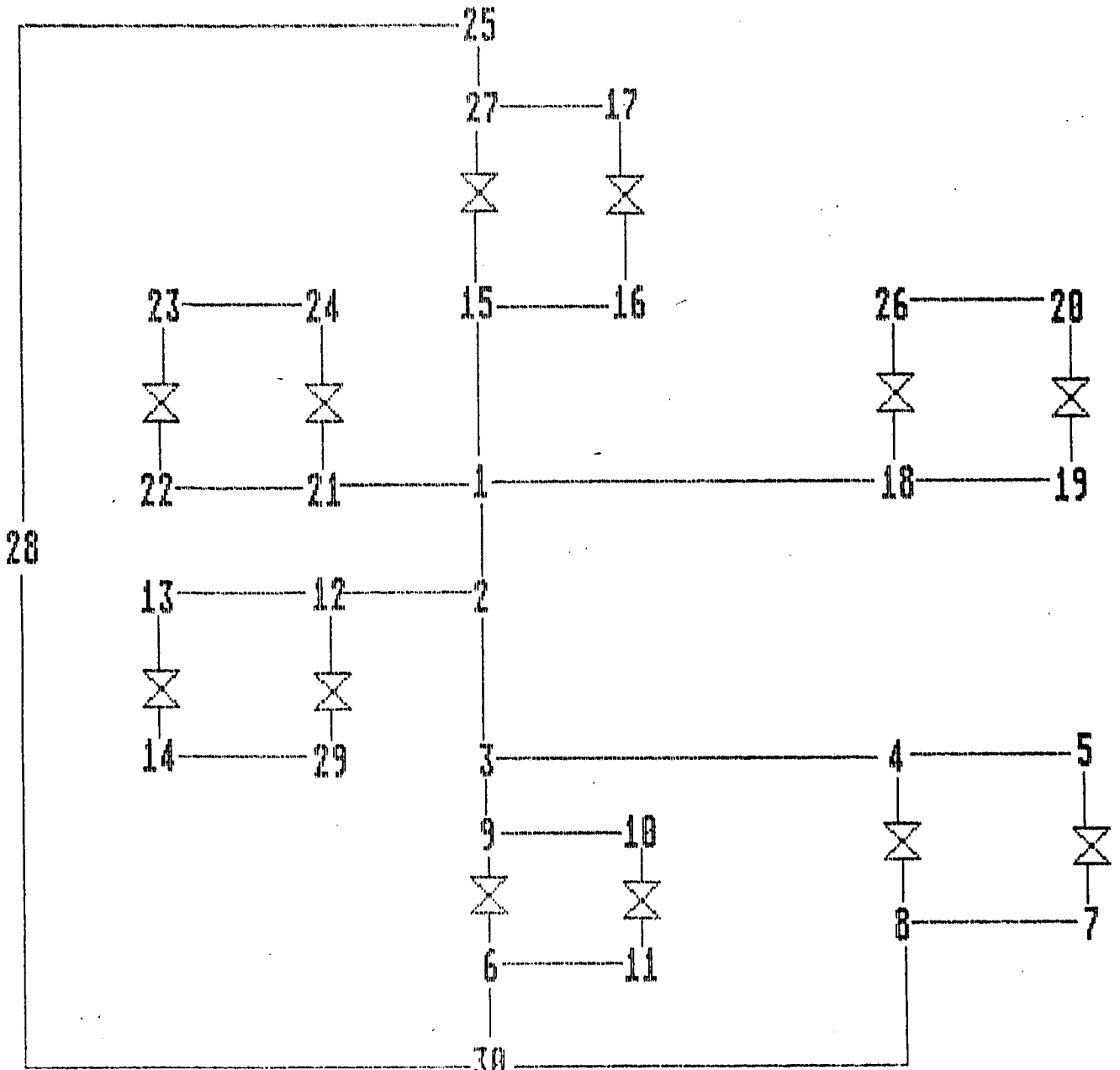

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42


CASE 12
BPF
CM2
BPE
BPD
BPB
BPC
BPA
B1
A1
R
D2
L
M
P
A
A2
CTF
B3
C3
D3
CEA
PH2
A3
K
B2-CEB
LSP
N
H2
PH1
G3
F3
H3
E3
D1
C1
E1
CTC
G1
F1
J1
H1
CM1
K1
M2
HF3
C
A113
E
A22
J
HF2
D
A115
A12
P2
B
P2
H
HF1B
G
F
HF1A
X
Y
BCFB
W
BCFL
V
BCFA
L1
T1
S
T
U
SSA
SSB
X1
F2
M1
N1
U1
AC
Q1
CMP
V1
E2
BP1
MCC
CSA
MSA
MSB

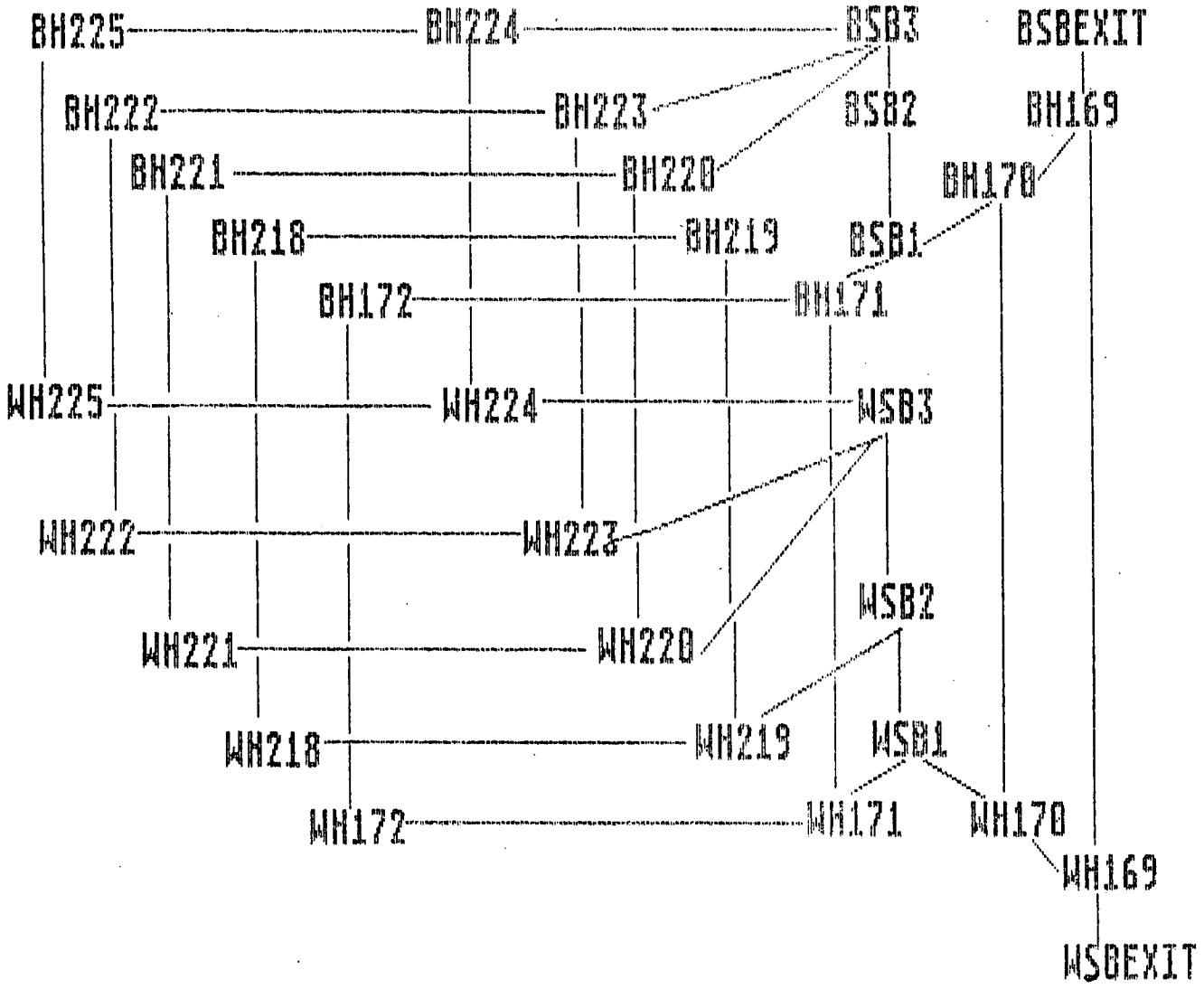
CASE13


24
37
21
45
23
43
44
22
20
38
18
19
39
36
17
35
16
40
25
15
26
28
27
29
30
31
34
33
46
7
9
10
47
5
11
12
13
14
41
48
4
6
8
42
3
2
1
32

CASE14


25
27
17
23
24
15
16
26
20
22
21
1
18
19
28
13
12
2
14
29
3
4
5
9
10
8
7
6
11
30

CASE15


BH225
BH224
BSB3
BSBEXIT
BH222
BH223
BSB2
BH169
BH221
BH220
BH170
BH218
BH219
BSB1
BH172
BH171
WH225
WH224
WSB3
WH222
WH223
WSB2
WH221
WH220
WH218
WH219
WSB1
WH172
WH171
WH170
WH169
WSBEXIT

CASE 1

DATA AND RESULTS UNITS -

MASS FLOWRATES: KG / S

PRESSURES: BARS ABS

DENSITY: KG / CU.M

VISCOSITY: CENTIPOISE

PIPE BORE: MILLIMETRES

PIPE LENGTH AND NODE HEIGHT: METRES

TEMPERATURE: CELSIUS

MEAN FLOW VEL: M / S

PIPING DETAILS DATA -

NUMBER OF PIPES = 3

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS (RELATIVE)	FIT. LOSS COEFF.	MEAN TEMP.
1	2	1.00	100.000	0.00100	1.000	1.0
2	3	1.00	100.000	0.00100	1.000	1.0
2	4	1.00	100.000	0.00100	1.000	1.0

SPECIFIED FLOW AND PRESSURE CONDITIONS -

NUMBER OF CONDITIONS = 3

NODE LABEL	FLOW INTO NODE	PRESSURE	NODE HEIGHT ABOVE STANDARD LEVEL
1	0.000	1.0000	0.0
3	-1.000	0.0000	0.0
4	-1.000	0.0000	0.0

FLUID PROPERTIES DATA -

TYPE OF FLUID : LIQUID

PRESSURE	TEMPERATURE	DENSITY	VISCOSITY
1.000	1.0	1000.0	1.0000
2.000	2.0	1000.0	1.0000
3.000	3.0	1000.0	1.0000

CASE 1

NODE LABELS XXXX --> XXXX		NODE PRESSURES (BARS)		FLOW (KG / S)	VELOCITY (M / S)	REYNOLDS NUMBER
1	2	1.0000	0.9996	2.0004	0.2547	25471
2	3	0.9996	0.9995	1.0004	0.1274	12738
2	4	0.9996	0.9995	1.0004	0.1274	12738

CASE 2 : Flows Specified

DATA AND RESULTS UNITS -

MASS FLOWRATES: KG / S

PRESSURES: BARS ABS

DENSITY: KG / CU.M

VISCOSITY: CENTIPOISE

PIPE BORE: MILLIMETRES

PIPE LENGTH AND NODE HEIGHT: METRES

TEMPERATURE: CELSIUS

MEAN FLOW VEL: M / S

PIPING DETAILS DATA -

NUMBER OF PIPES = 6

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS (RELATIVE)	FIT. LOSS COEFF.	MEAN TEMP.
A100	A120	0.00	50.000	0.00000	0.000	40.0
A120	A200	100.00	50.000	0.00050	2.000	40.0
A200	A210	50.00	25.000	0.00025	-5.000	40.0
A200	A220	50.00	25.000	0.00025	-5.000	40.0
A210	A215	25.00	30.000	0.00030	6.000	30.0
A220	A225	25.00	30.000	0.00030	6.000	30.0

SPECIFIED FLOW AND PRESSURE CONDITIONS -

NUMBER OF CONDITIONS = 5

NODE LABEL	FLOW INTO NODE	PRESSURE	NODE HEIGHT ABOVE STANDARD LEVEL
A100	0.000	1.0100	0.0
A210	0.000	0.0000	5.0
A215	-1.000	0.0000	5.0
A220	0.000	0.0000	5.0
A225	-1.200	0.0000	5.0

PUMP CHARACTERISTIC DATA -

NUMBER OF PUMPS = 1

PUMP 1 PIPE A100 TO A120

HEAD	FLOW
25.00	0.0000
23.77	0.0060
21.33	0.0120

FLUID PROPERTIES DATA -

TYPE OF FLUID : LIQUID

PRESSURE	TEMPERATURE	DENSITY	VISCOSITY
1.380	45.0	998.25	0.59273
1.725	35.0	992.10	0.71811
2.068	55.0	983.93	0.50006

CASE 2 : Flows Specified

NODE LABELS XXXX --> XXXX		NODE PRESSURES (BARS)		FLOW (KG / S)	VELOCITY (M / S)	REYNOLDS NUMBER
A100	A120	1.010	3.396	2.2000	1.1259	86004
A120	A200	3.396	3.121	2.1999	1.1259	86000
A200	A210	3.121	1.693	0.9999	2.0470	78180
A200	A220	3.121	1.315	1.1999	2.4564	93817
A210	A215	1.693	1.454	1.0000	1.4014	53440
A220	A225	1.315	0.980	1.2000	1.6817	64128

CASE 2 : Pressures Specified

DATA AND RESULTS UNITS -

MASS FLOWRATES: KG / S

PRESSURES: BARS ABS

DENSITY: KG / CU.M

VISCOSITY: CENTIPOISE

PIPE BORE: MILLIMETRES

PIPE LENGTH AND NODE HEIGHT: METRES

TEMPERATURE: CELSIUS

MEAN FLOW VEL: M / S

PIPING DETAILS DATA -

NUMBER OF PIPES = 6

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS (RELATIVE)	FIT. LOSS COEFF.	MEAN TEMP.
A100	A120	0.00	50.000	0.00000	0.000	40.0
A120	A200	100.00	50.000	0.00050	2.000	40.0
A200	A210	50.00	25.000	0.00025	-5.000	40.0
A200	A220	50.00	25.000	0.00025	-5.000	40.0
A210	A215	25.00	30.000	0.00030	6.000	30.0
A220	A225	25.00	30.000	0.00030	6.000	30.0

SPECIFIED FLOW AND PRESSURE CONDITIONS -

NUMBER OF CONDITIONS = 5

NODE LABEL	FLOW INTO NODE	PRESSURE	NODE HEIGHT ABOVE STANDARD LEVEL
A100	0.000	1.0100	0.0
A210	0.000	0.0000	5.0
A220	0.000	0.0000	5.0
A215	0.000	0.6000	5.0
A225	0.000	0.4000	5.0

PUMP CHARACTERISTIC DATA -

NUMBER OF PUMPS = 1

PUMP 1 PIPE A100 TO A120

HEAD	FLOW
25.00	0.0000
23.77	0.0060
21.33	0.0120

FLUID PROPERTIES DATA -

TYPE OF FLUID : LIQUID

PRESSURE	TEMPERATURE	DENSITY	VISCOSITY
1.380	45.0	998.25	0.59273
1.725	35.0	992.10	0.71811
2.068	55.0	983.93	0.50006

CASE 2 : Pressures Specified

NODE LABELS XXXX --> XXXX		NODE PRESSURES (BARS)		FLOW (KG / S)	VELOCITY (M / S)	REYNOLDS NUMBER
A100	A120	1.010	3.384	2.6607	1.3617	104015
A120	A200	3.384	2.992	2.6608	1.3618	104019
A200	A210	2.992	0.986	1.2948	2.6505	101231
A200	A220	2.992	0.827	1.3661	2.7966	106810
A210	A215	0.986	0.600	1.2944	1.8140	69176
A220	A225	0.827	0.400	1.3657	1.9140	72988

CASE 3

DATA AND RESULTS UNITS -

MASS FLOWRATES: KG / S

PRESSURES: BARS ABS

DENSITY: KG / CU.M

VISCOSITY: CENTIPOISE

PIPE BORE: MILLIMETRES

PIPE LENGTH AND NODE HEIGHT: METRES

TEMPERATURE: CELSIUS

MEAN FLOW VEL: M / S

PIPING DETAILS DATA -

NUMBER OF PIPES = 7

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS (RELATIVE)	FIT. LOSS COEFF.	MEAN TEMP.
A210	A220	50.00	25.000	0.00025	2.000	40.0
A100	A120	0.00	50.000	0.00000	0.000	40.0
A120	A200	100.00	50.000	0.00050	2.000	40.0
A200	A210	50.00	25.000	0.00025	-5.000	40.0
A200	A220	50.00	25.000	0.00025	-5.000	40.0
A210	A215	25.00	30.000	0.00030	6.000	30.0
A220	A225	25.00	30.000	0.00030	6.000	30.0

SPECIFIED FLOW AND PRESSURE CONDITIONS -

NUMBER OF CONDITIONS = 5

NODE LABEL	FLOW INTO NODE	PRESSURE	NODE HEIGHT ABOVE STANDARD LEVEL
A100	0.000	1.0100	0.0
A210	0.000	0.0000	5.0
A220	0.000	0.0000	5.0
A215	0.000	0.6000	5.0
A225	0.000	0.4000	5.0

PUMP CHARACTERISTIC DATA -

NUMBER OF PUMPS = 1

PUMP 1 PIPE A100 TO A120

HEAD	FLOW
25.00	0.0000
23.77	0.0060
21.33	0.0120

FLUID PROPERTIES DATA -

TYPE OF FLUID : LIQUID

PRESSURE	TEMPERATURE	DENSITY	VISCOSITY
1.380	45.0	998.25	0.59273
1.725	35.0	992.10	0.71811
2.068	55.0	983.93	0.50006

CASE 3

NODE LABELS XXXX --> XXXX		NODE PRESSURES (BARS)		FLOW (KG / S)	VELOCITY (M / S)	REYNOLDS NUMBER
A210	A220	0.924	0.896	0.1450	0.2968	11337
A100	A120	1.010	3.384	2.6589	1.3607	103941
A120	A200	3.384	2.993	2.6589	1.3608	103944
A200	A210	2.993	0.924	1.3233	2.7090	103464
A200	A220	2.993	0.896	1.3357	2.7342	104428
A210	A215	0.924	0.600	1.1783	1.6513	62970
A220	A225	0.896	0.400	1.4801	2.0742	79097

CASE 4(i)

DATA AND RESULTS UNITS -

MASS FLOWRATES: KG / S

PRESSURES: BARS ABS

DENSITY: KG / CU.M

VISCOSITY: CENTIPOISE

PIPE BORE: MILLIMETRES

PIPE LENGTH AND NODE HEIGHT: METRES

TEMPERATURE: CELSIUS

MEAN FLOW VEL: M / S

PIPING DETAILS DATA -

NUMBER OF PIPES = 9

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS (RELATIVE)	FIT. LOSS COEFF.	MEAN TEMP.
A100	A120	0.00	50.000	0.00000	0.000	40.0
A100	A130	0.00	50.000	0.00000	0.000	40.0
A100	A140	0.00	50.000	0.00000	0.000	40.0
A120	A200	100.00	50.000	0.00050	2.000	40.0
A130	A300	100.00	50.000	0.00050	2.000	40.0
A140	A400	100.00	50.000	0.00050	2.000	40.0
A200	A500	100.00	50.000	0.00050	2.000	40.0
A300	A500	100.00	50.000	0.00050	2.000	40.0
A400	A500	100.00	50.000	0.00050	2.000	40.0

SPECIFIED FLOW AND PRESSURE CONDITIONS -

NUMBER OF CONDITIONS = 2

NODE LABEL	FLOW INTO NODE	PRESSURE	NODE HEIGHT ABOVE STANDARD LEVEL
A100	0.000	2.0000	0.0
A500	0.000	2.0000	0.0

PUMP CHARACTERISTIC DATA -

NUMBER OF PUMPS = 3

PUMP 1 PIPE A100 TO A120

HEAD	FLOW
25.00	0.0000
23.77	0.0060
21.33	0.0120

PUMP 2 PIPE A100 TO A130

HEAD	FLOW
25.00	0.0000
23.77	0.0060
21.33	0.0120

PUMP 3 PIPE A100 TO A140

HEAD	FLOW
25.00	0.0000
23.77	0.0060
21.33	0.0120

FLUID PROPERTIES DATA -

TYPE OF FLUID : LIQUID

PRESSURE	TEMPERATURE	DENSITY	VISCOSITY
1.380	45.0	998.25	0.59273
1.725	35.0	992.10	0.71811
2.068	55.0	983.93	0.50006

CASE 4(i)

NODE LABELS XXXX --> XXXX		NODE PRESSURES (BARS)		FLOW (KG / S)	VELOCITY (M / S)	REYNOLDS NUMBER
A100	A120	2.000	4.323	4.7233	2.4173	184647
A100	A130	2.000	4.323	4.7233	2.4173	184647
A100	A140	2.000	4.323	4.7233	2.4173	184647
A120	A200	4.323	3.161	4.7241	2.4177	184676
A130	A300	4.323	3.161	4.7241	2.4177	184676
A140	A400	4.323	3.161	4.7241	2.4177	184676
A200	A500	3.161	2.000	4.7241	2.4177	184676
A300	A500	3.161	2.000	4.7241	2.4177	184676
A400	A500	3.161	2.000	4.7241	2.4177	184676

CASE 4(ii)

DATA AND RESULTS UNITS -

MASS FLOWRATES: KG / S

PRESSURES: BARS ABS

DENSITY: KG / CU.M

VISCOSITY: CENTIPOISE

PIPE BORE: MILLIMETRES

PIPE LENGTH AND NODE HEIGHT: METRES

TEMPERATURE: CELSIUS

MEAN FLOW VEL: M / S

PIPING DETAILS DATA -

NUMBER OF PIPES = 3

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS (RELATIVE)	FIT. LOSS COEFF.	MEAN TEMP.
A10	A100	0.00	50.000	0.00000	0.000	40.0
A100	A200	100.00	50.000	0.00050	2.000	40.0
A200	A500	100.00	50.000	0.00050	2.000	40.0

SPECIFIED FLOW AND PRESSURE CONDITIONS -

NUMBER OF CONDITIONS = 2

NODE LABEL	FLOW INTO NODE	PRESSURE	NODE HEIGHT ABOVE STANDARD LEVEL
A10	0.000	2.0000	0.0
A500	0.000	2.0000	0.0

PUMP CHARACTERISTIC DATA -

NUMBER OF PUMPS = 1

PUMP 1 PIPE A10 TO A100

HEAD	FLOW
25.00	0.0000
23.77	0.0060
21.33	0.0120

FLUID PROPERTIES DATA -

TYPE OF FLUID : LIQUID

PRESSURE	TEMPERATURE	DENSITY	VISCOSITY
1.380	45.0	998.25	0.59273
1.725	35.0	992.10	0.71811
2.068	55.0	983.93	0.50006

CASE 4(ii)

NODE LABELS XXXX --> XXXX		NODE PRESSURES (BARS)		FLOW (KG / S)	VELOCITY (M / S)	REYNOLDS NUMBER
A10	A100	2.000	4.323	4.7233	2.4173	184647
A100	A200	4.323	3.161	4.7241	2.4177	184676
A200	A500	3.161	2.000	4.7241	2.4177	184676

CASE 4(iii)

DATA AND RESULTS UNITS -

MASS FLOWRATES:	KG / S
PRESSURES:	BARS ABS
DENSITY:	KG / CU.M
VISCOSITY:	CENTIPOISE
PIPE BORE:	MILLIMETRES
PIPE LENGTH AND NODE HEIGHT:	METRES
TEMPERATURE:	CELSIUS
MEAN FLOW VEL:	M / S

PIPING DETAILS DATA -

NUMBER OF PIPES = 6

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS (RELATIVE)	FIT. LOSS COEFF.	MEAN TEMP.
A100	A200	100.00	50.000	0.00050	2.000	40.0
A100	A300	100.00	50.000	0.00050	2.000	40.0
A100	A400	100.00	50.000	0.00050	2.000	40.0
A200	A500	100.00	50.000	0.00050	2.000	40.0
A300	A500	100.00	50.000	0.00050	2.000	40.0
A400	A500	100.00	50.000	0.00050	2.000	40.0

SPECIFIED FLOW AND PRESSURE CONDITIONS -

NUMBER OF CONDITIONS = 2

NODE LABEL	FLOW INTO NODE	PRESSURE	NODE HEIGHT ABOVE STANDARD LEVEL
A100	0.000	2.0000	0.0
A500	0.000	1.0000	0.0

FLUID PROPERTIES DATA -

TYPE OF FLUID : LIQUID

PRESSURE	TEMPERATURE	DENSITY	VISCOSITY
1.380	45.0	998.25	0.59273
1.725	35.0	992.10	0.71811
2.068	55.0	983.93	0.50006

CASE 4(iii)

NODE LABELS XXXX --> XXXX		NODE PRESSURES (BARS)		FLOW (KG / S)	VELOCITY (M / S)	REYNOLDS NUMBER
A100	A200	2.000	1.500	3.2815	1.6794	128282
A100	A300	2.000	1.500	3.2815	1.6794	128282
A100	A400	2.000	1.500	3.2815	1.6794	128282
A200	A500	1.500	1.000	3.2815	1.6794	128282
A300	A500	1.500	1.000	3.2815	1.6794	128282
A400	A500	1.500	1.000	3.2815	1.6794	128282

CASE 7

DATA AND RESULTS UNITS -

MASS FLOWRATES: KG / S

PRESSURES: BARS ABS

DENSITY: KG / CU.M

VISCOSITY: CENTIPOISE

PIPE BORE: MILLIMETRES

PIPE LENGTH AND NODE HEIGHT: METRES

TEMPERATURE: CELSIUS

MEAN FLOW VEL: M / S

PIPING DETAILS DATA -

NUMBER OF PIPES = 9

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS (RELATIVE)	FIT. LOSS COEFF.	MEAN TEMP.
A100	A110	8.23	100.000	0.01000	5.550	20.0
A200	A110	3.96	100.000	0.01000	5.550	20.0
A110	A120	6.10	100.000	0.01000	0.000	20.0
A300	A120	4.57	100.000	0.01000	5.550	20.0
A120	A130	12.20	100.000	0.01000	0.000	20.0
A400	A130	4.57	100.000	0.01000	5.550	20.0
A130	A140	4.27	100.000	0.01000	0.000	20.0
A500	A140	3.96	100.000	0.01000	5.550	20.0
A140	A600	2.93	100.000	0.01000	1.650	20.0

SPECIFIED FLOW AND PRESSURE CONDITIONS -

NUMBER OF CONDITIONS = 6

NODE LABEL	FLOW INTO NODE	PRESSURE	NODE HEIGHT ABOVE STANDARD LEVEL
A100	0.000	1.0000	1.2
A200	0.000	1.0000	1.2
A300	0.000	1.0000	1.2
A400	0.000	1.0000	1.2
A500	0.000	1.0000	1.2
A600	0.000	1.0000	-1.5

FLUID PROPERTIES DATA -

TYPE OF FLUID : LIQUID

PRESSURE	TEMPERATURE	DENSITY	VISCOSITY
1.380	45.0	998.25	0.59273
1.725	35.0	992.10	0.71811
2.068	55.0	983.93	0.50006

CASE 7

NODE LABELS XXXX --> XXXX		NODE PRESSURES (BARS)		FLOW (KG / S)	VELOCITY (M / S)	REYNOLDS NUMBER
A100	A110	1.000	1.116	2.4560	0.3054	31864
A200	A110	1.000	1.116	2.7341	0.3400	35472
A110	A120	1.116	1.111	5.1901	0.6455	67336
A300	A120	1.000	1.111	3.9924	0.4965	51797
A120	A130	1.111	1.080	9.1835	1.1421	119146
A400	A130	1.000	1.080	8.3700	1.0409	108592
A130	A140	1.080	1.040	17.5544	2.1832	227750
A500	A140	1.000	1.040	11.9788	1.4898	155412
A140	A600	1.040	1.000	29.5334	3.6729	383165

CASE 8

DATA AND RESULTS UNITS -

MASS FLOWRATES: KG / S

PRESSURES: BARS ABS

DENSITY: KG / CU.M

VISCOSITY: CENTIPOISE

PIPE BORE: MILLIMETRES

PIPE LENGTH AND NODE HEIGHT: METRES

TEMPERATURE: CELSIUS

MEAN FLOW VEL: M / S

PIPING DETAILS DATA -

NUMBER OF PIPES = 7

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS (RELATIVE)	FIT. LOSS COEFF.	MEAN TEMP.
A	B	0.00	260.000	0.00000	0.000	10.0
B	AA	1.00	260.000	0.00000	0.000	10.0
AA	1	180.00	260.000	0.00096	3.400	10.0
1	2	30.00	260.000	0.00096	1.200	10.0
2	3	130.00	206.000	0.00121	0.200	10.0
3	4	70.00	206.000	0.00121	1.300	10.0
4	H13	1.00	206.000	0.00121	1.300	10.0

SPECIFIED FLOW AND PRESSURE CONDITIONS -

NUMBER OF CONDITIONS = 2

NODE LABEL	FLOW INTO NODE	PRESSURE	NODE HEIGHT ABOVE STANDARD LEVEL
A	0.000	1.0000	0.0
H13	0.000	7.2000	0.0

PUMP CHARACTERISTIC DATA -

NUMBER OF PUMPS = 1

PUMP 1 PIPE A TO B

HEAD	FLOW
125.00	0.0000
109.30	0.0430
106.60	0.0757
94.25	0.1038
73.71	0.1290

FLUID PROPERTIES DATA -

TYPE OF FLUID : LIQUID

PRESSURE	TEMPERATURE	DENSITY	VISCOSITY
1.000	10.0	1000.0	1.3000
2.000	20.0	1000.0	1.0000
3.000	30.0	1000.0	0.8000

CASE 8

NODE LABELS XXXX --> XXXX		NODE PRESSURES (BARS)		FLOW (KG / S)	VELOCITY (M / S)	REYNOLDS NUMBER
A	B	1.000	9.143	116.9339	2.2025	440501
B	AA	9.143	9.142	116.9386	2.2026	440519
AA	1	9.142	8.722	116.9337	2.2025	440500
1	2	8.722	8.636	116.9337	2.2025	440500
2	3	8.636	7.806	116.9336	3.5086	555970
3	4	7.806	7.286	116.9337	3.5086	555971
4	H13	7.286	7.200	116.9339	3.5086	555972

CASE 10

DATA AND RESULTS UNITS -

MASS FLOWRATES: KG / S

PRESSURES: BARS ABS

DENSITY: KG / CU.M

VISCOSITY: CENTIPOISE

PIPE BORE: MILLIMETRES

PIPE LENGTH AND NODE HEIGHT: METRES

TEMPERATURE: CELSIUS

MEAN FLOW VEL: M / S

PIPING DETAILS DATA -

NUMBER OF PIPES = 10

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS (RELATIVE)	FIT. LOSS COEFF.	MEAN TEMP.
1	2	0.36	477.800	0.00009	0.000	409.0
2	3	0.36	477.800	0.00009	0.000	409.0
3	4	0.36	477.800	0.00009	0.000	409.0
1	5	0.40	477.800	0.00009	0.000	409.0
2	6	0.40	477.800	0.00009	0.000	409.0
3	7	0.40	477.800	0.00009	0.000	409.0
4	8	0.40	477.800	0.00009	0.000	409.0
8	7	0.36	477.800	0.00009	0.000	409.0
7	6	0.36	477.800	0.00009	0.000	409.0
6	5	0.36	477.800	0.00009	0.000	409.0

SPECIFIED FLOW AND PRESSURE CONDITIONS -

NUMBER OF CONDITIONS = 2

NODE LABEL	FLOW INTO NODE	PRESSURE	NODE HEIGHT ABOVE STANDARD LEVEL
1	0.000	30.0000	0.0
5	-1.000	0.0000	0.0

FLUID PROPERTIES DATA -

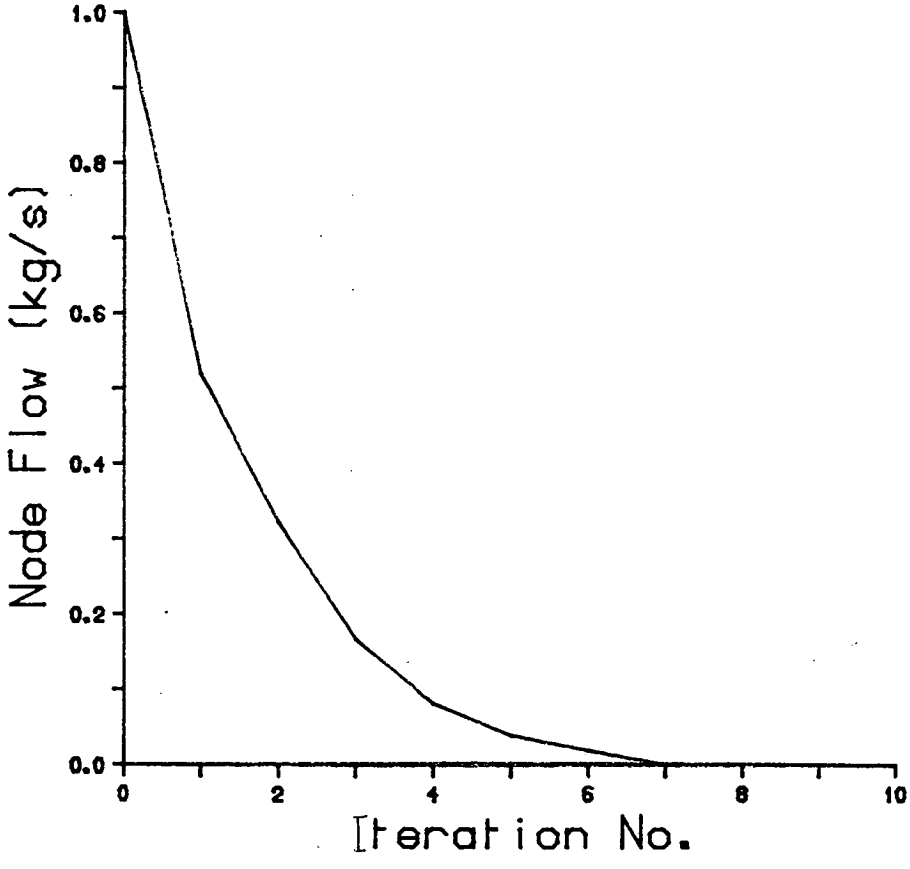
TYPE OF FLUID : GAS
RATIO OF SPECIFIC HEATS = 1.100

PRESSURE	TEMPERATURE	DENSITY	VISCOSITY
29.980	420.0	12.950	0.18800E-01
25.840	460.0	10.270	0.19800E-01
25.840	460.0	10.270	0.19800E-01

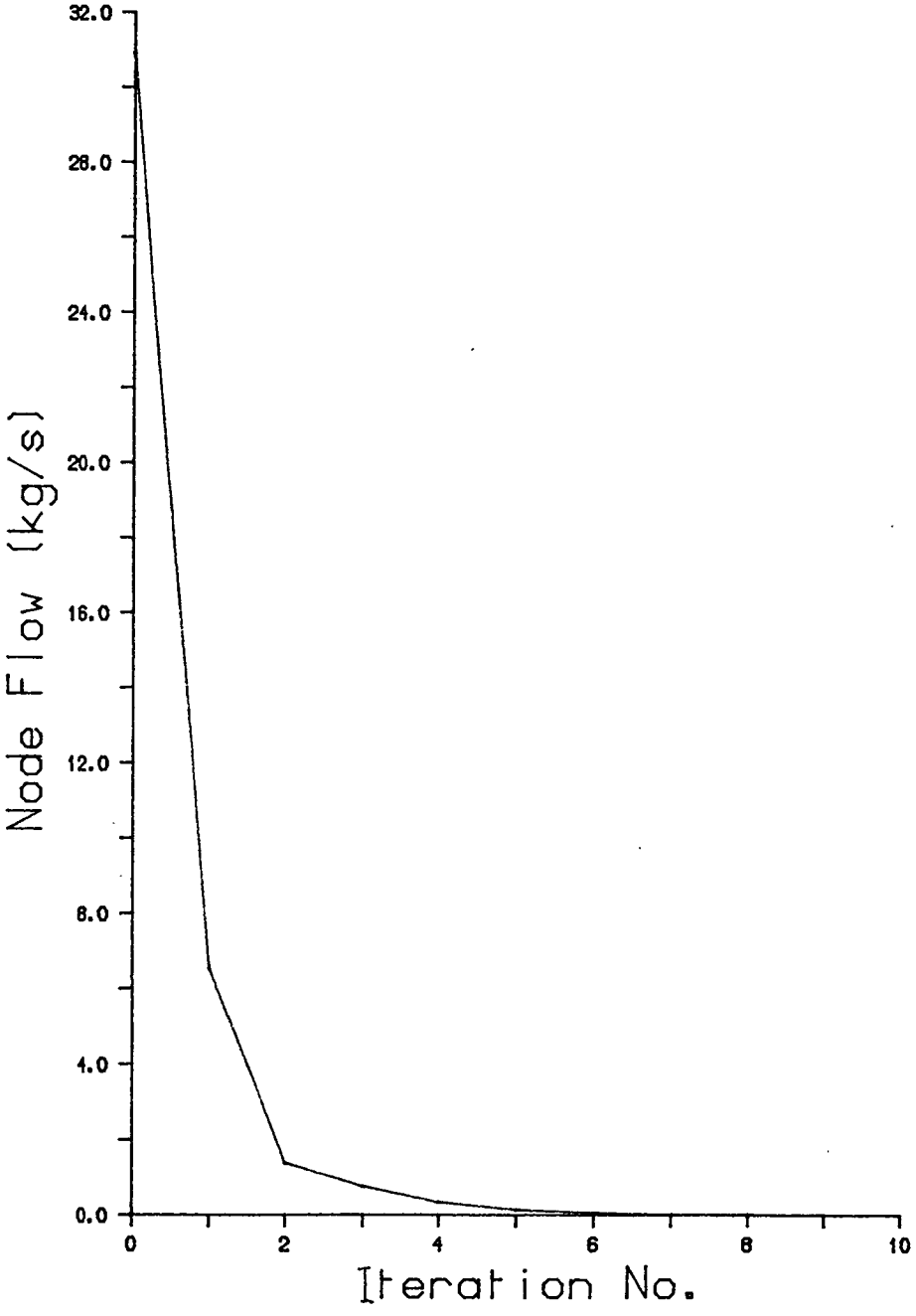
CASE 10

NODE LABELS XXXX --> XXXX		NODE PRESSURES (BARS)		FLOW (KG / S)	VELOCITY (M / S)	REYNOLDS NUMBER
1	2	30.00000000	29.99999997	0.3911	0.1656	56269
2	3	29.99999997	29.99999997	0.1512	0.0640	21761
3	4	29.99999997	29.99999997	0.0545	0.0231	7847
1	5	30.00000000	29.99999993	0.6089	0.2579	87610
2	6	29.99999997	29.99999996	0.2399	0.1016	34514
3	7	29.99999997	29.99999996	0.0975	0.0413	14034
4	8	29.99999997	29.99999996	0.0537	0.0228	7731
8	7	29.99999996	29.99999996	0.0537	0.0228	7730
7	6	29.99999996	29.99999996	0.1512	0.0640	21758
6	5	29.99999996	29.99999993	0.3911	0.1656	56269

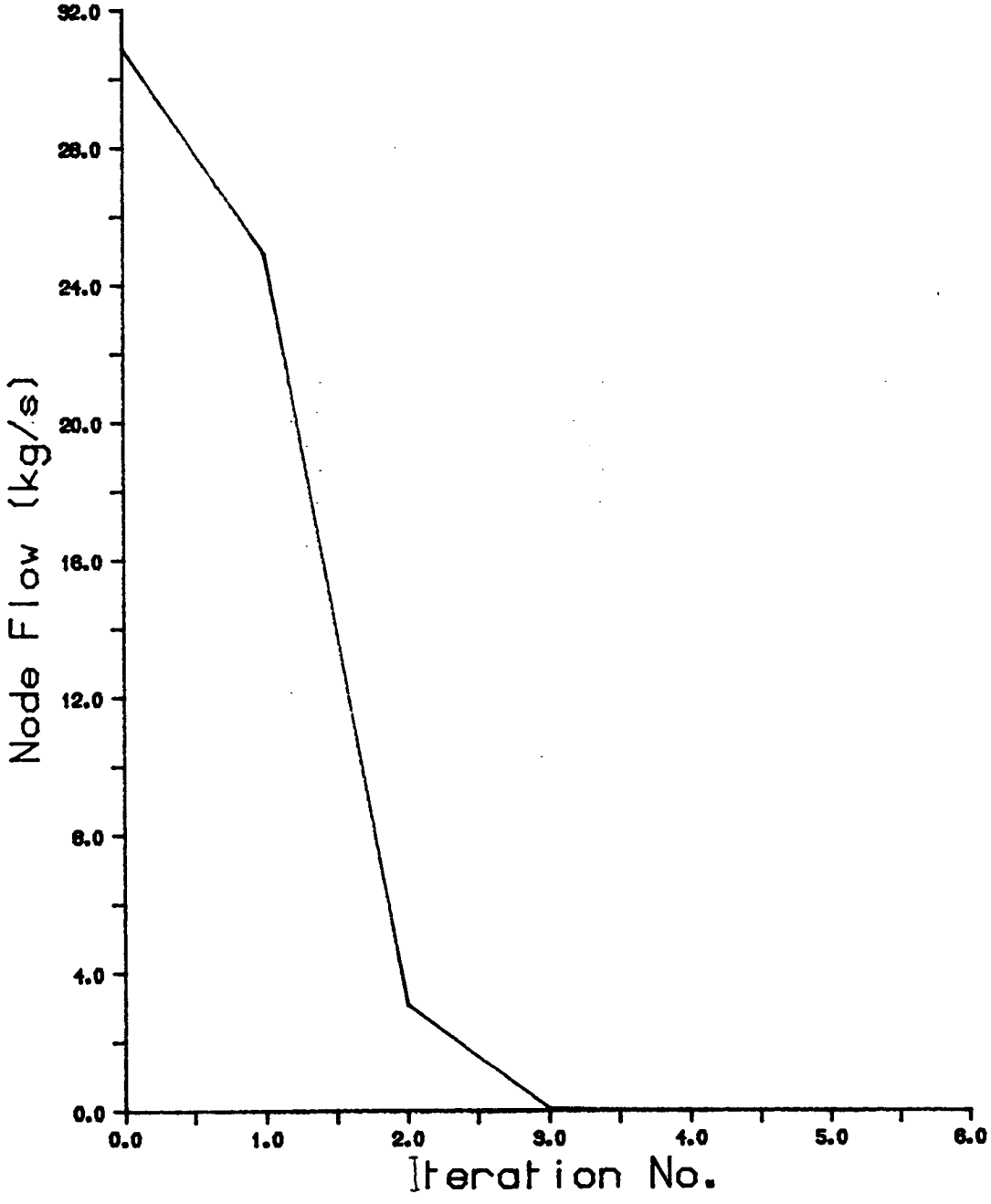
Case 1: Simple Network



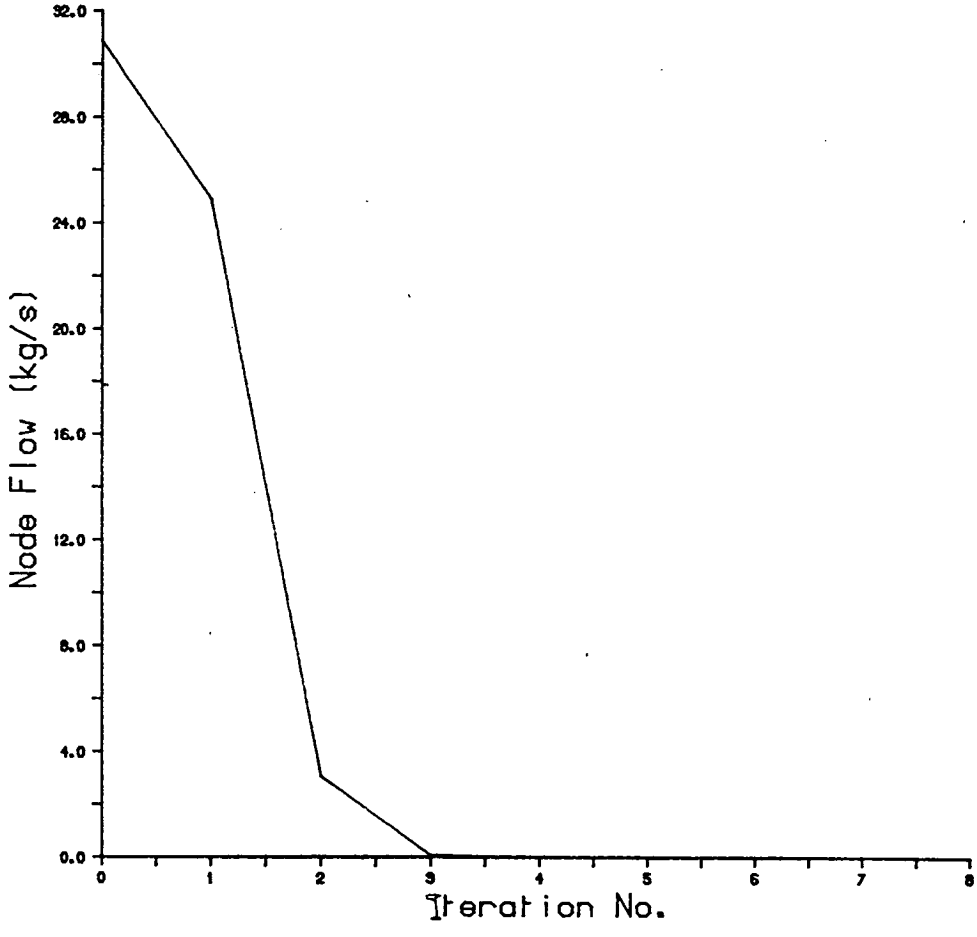
Case 2: Flows Spec.



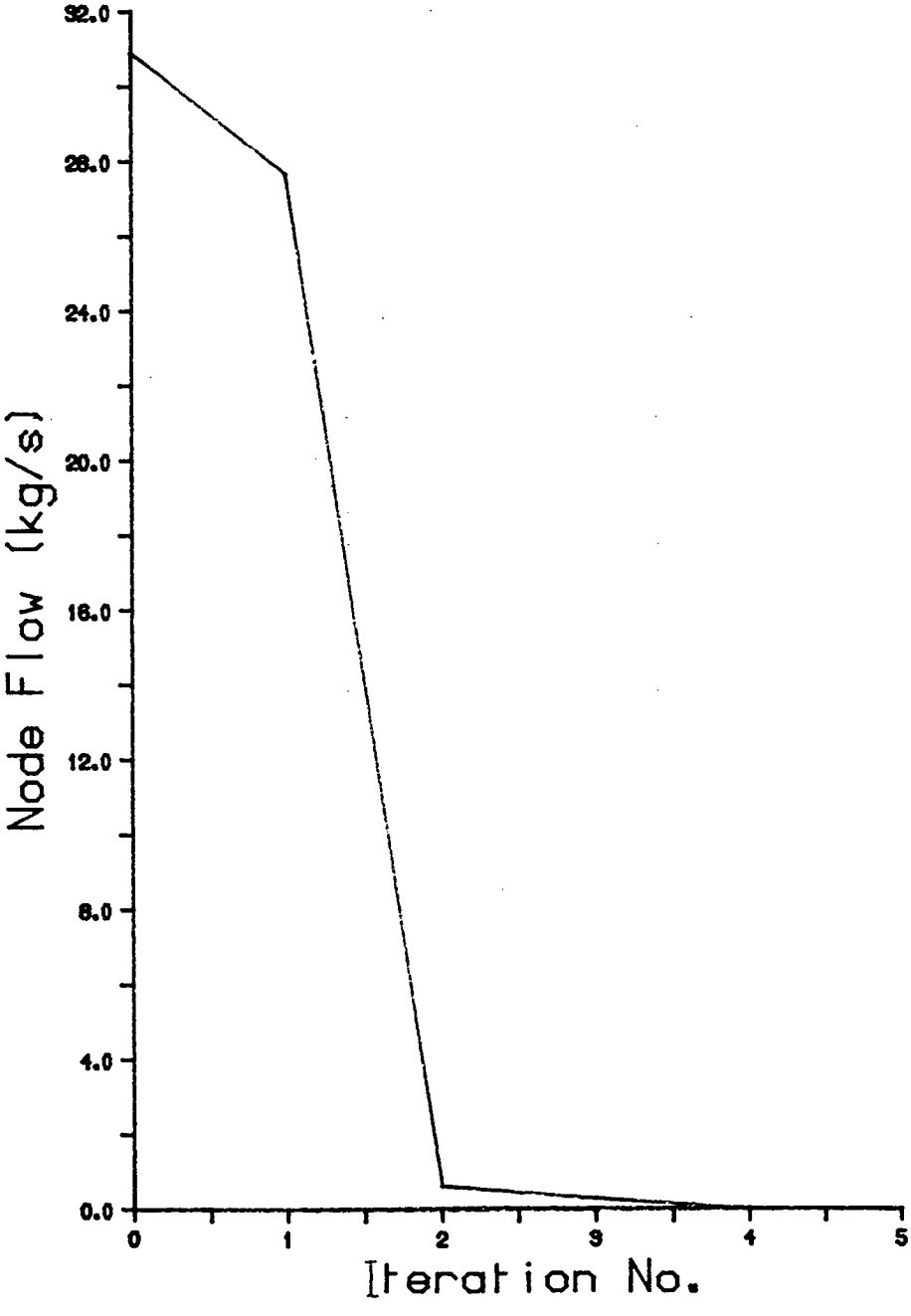
Case 2: Pres. Spec



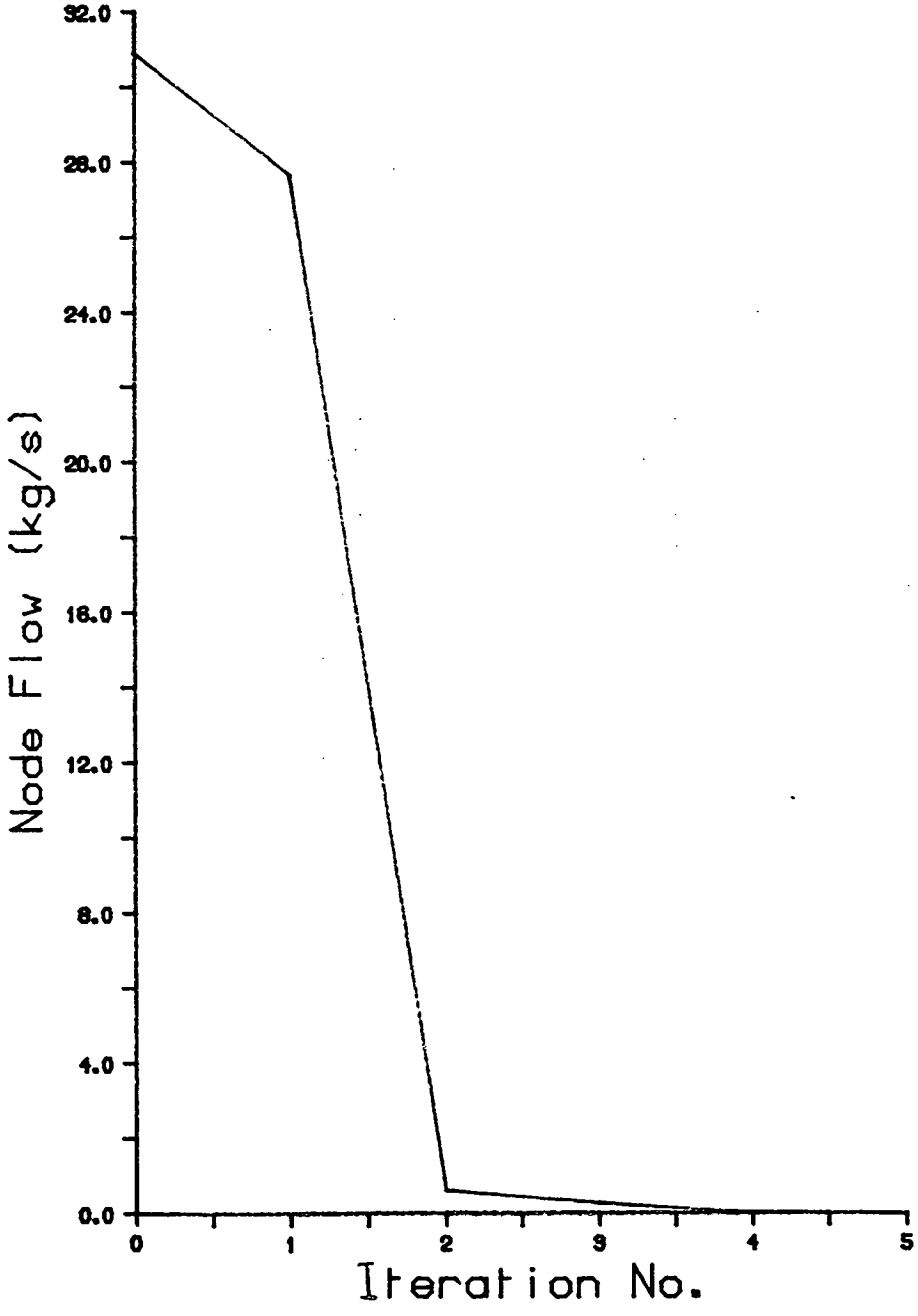
Case 3: Single-Mesh Network containing one pump



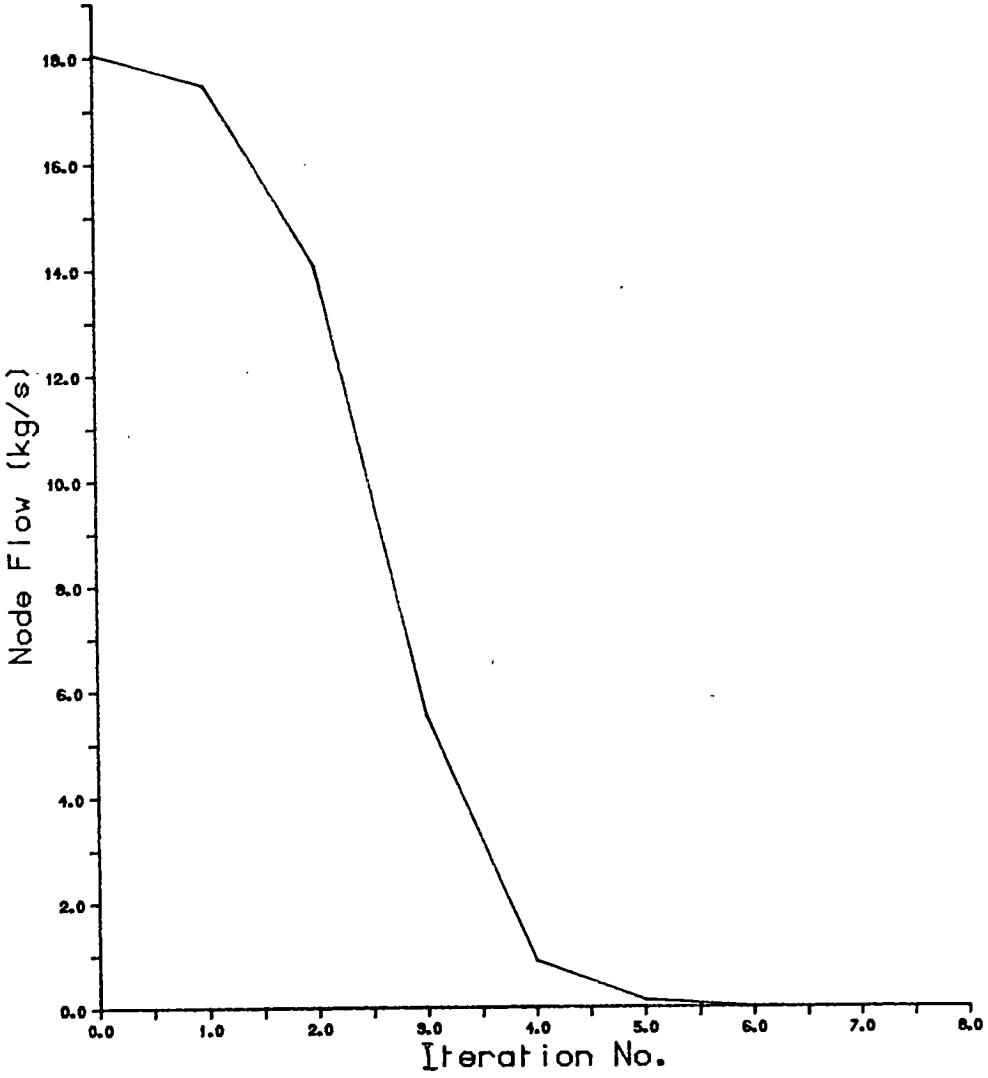
Case 4(i): Pumps in parallel



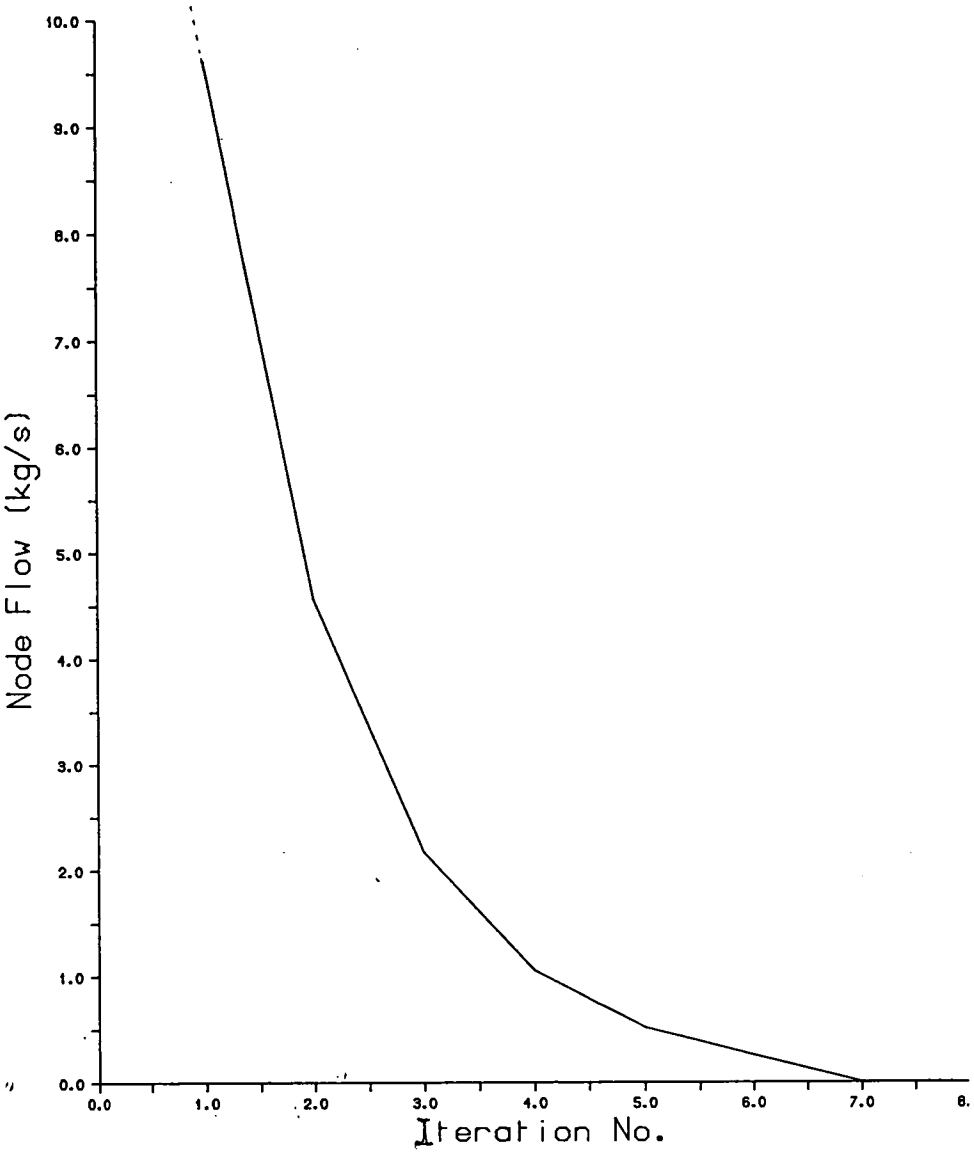
Case 4(ii): Line From Network 4(i)



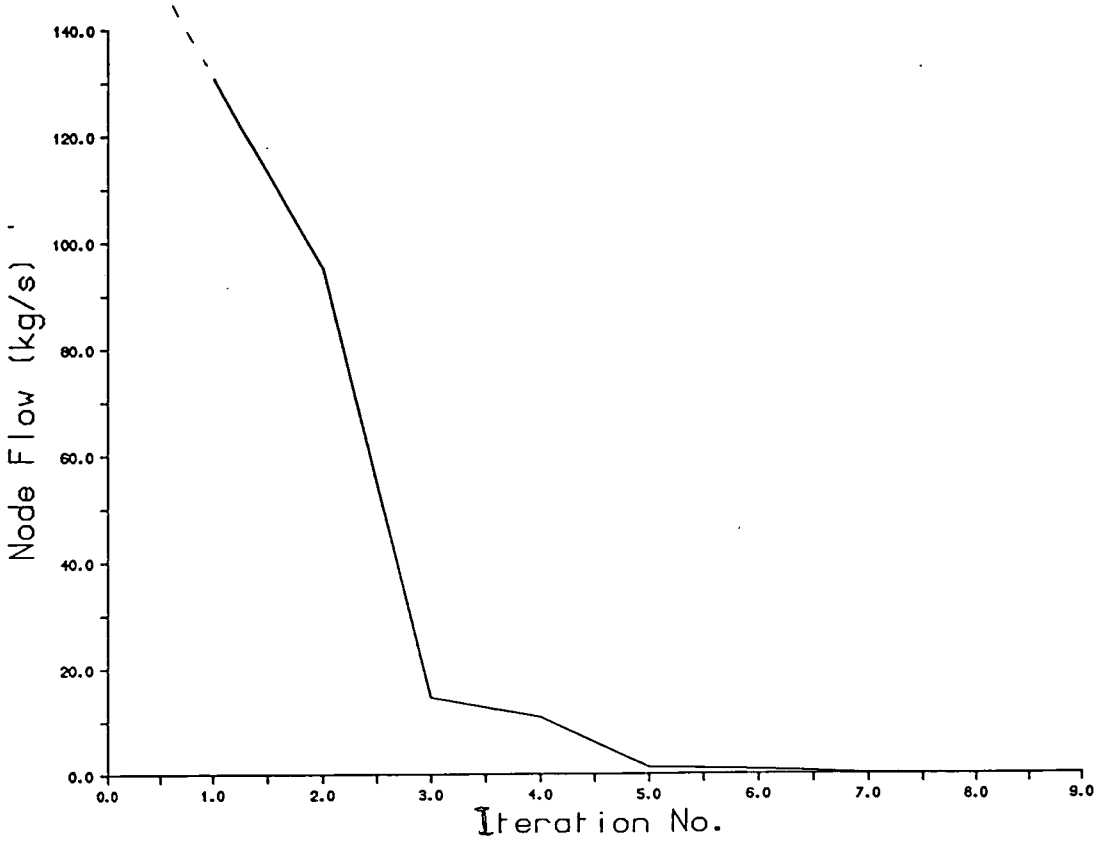
Case 6: Steam System



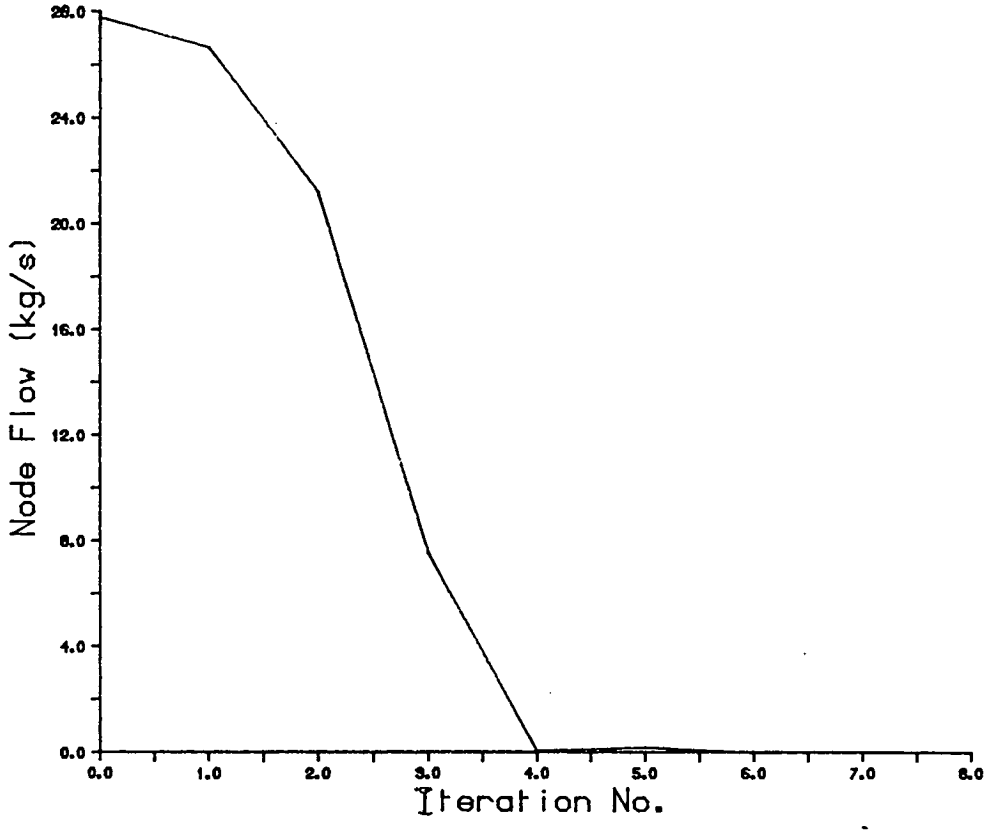
Case 7: Network with Gravity Feed



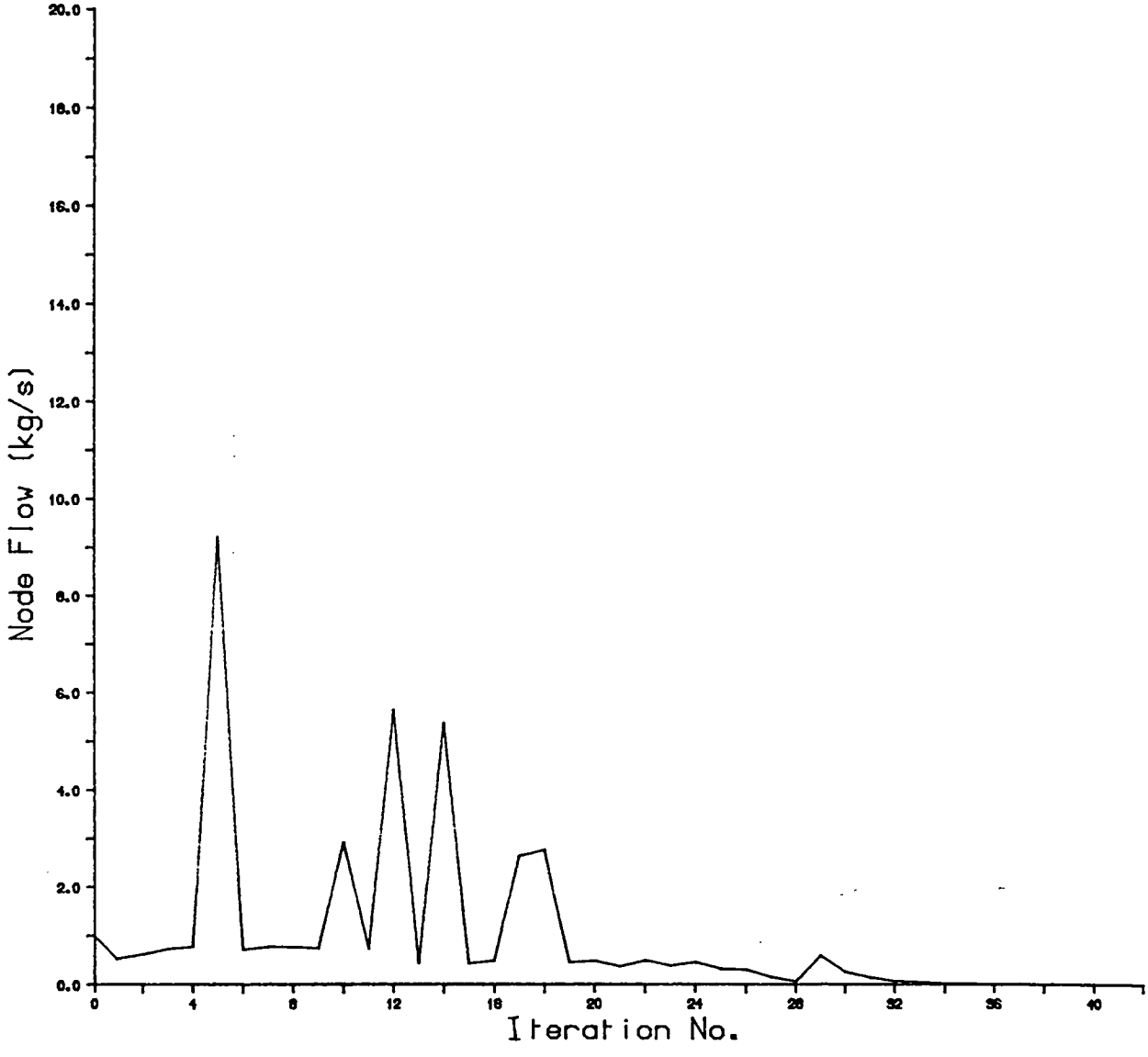
Case 8: Water System



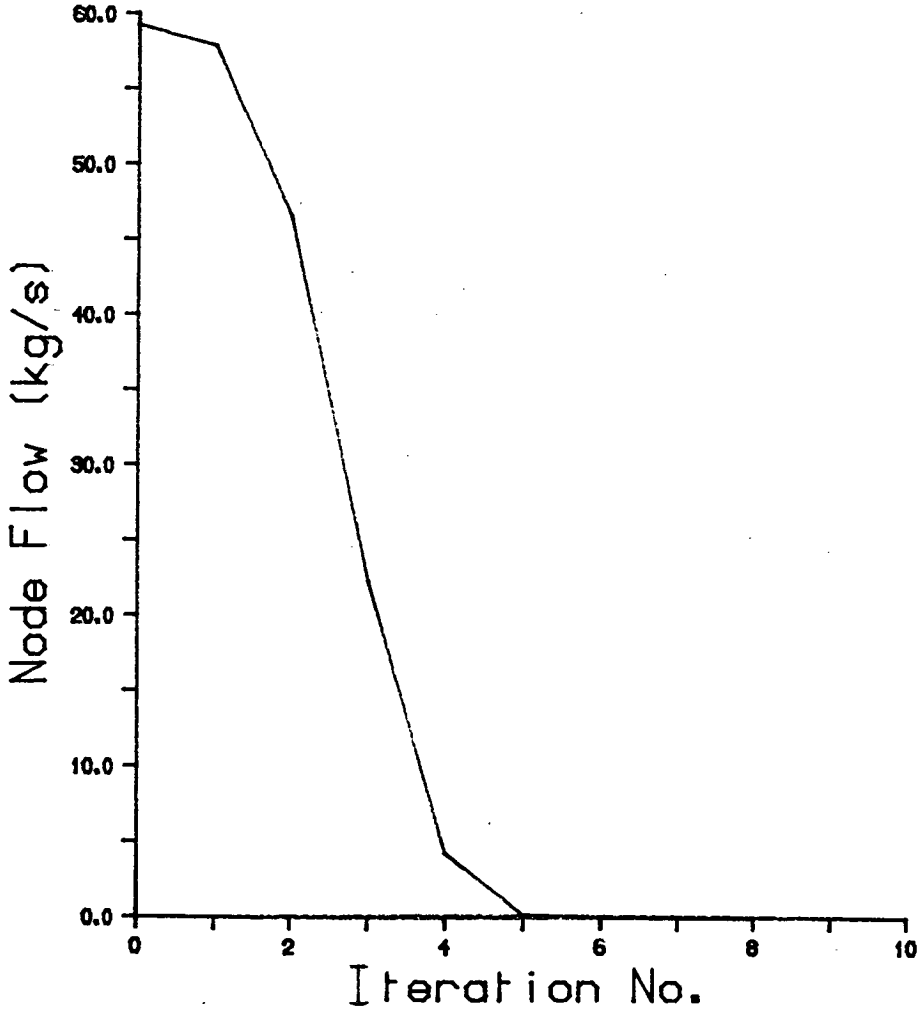
Case 9: Furnace Gas Distribution System



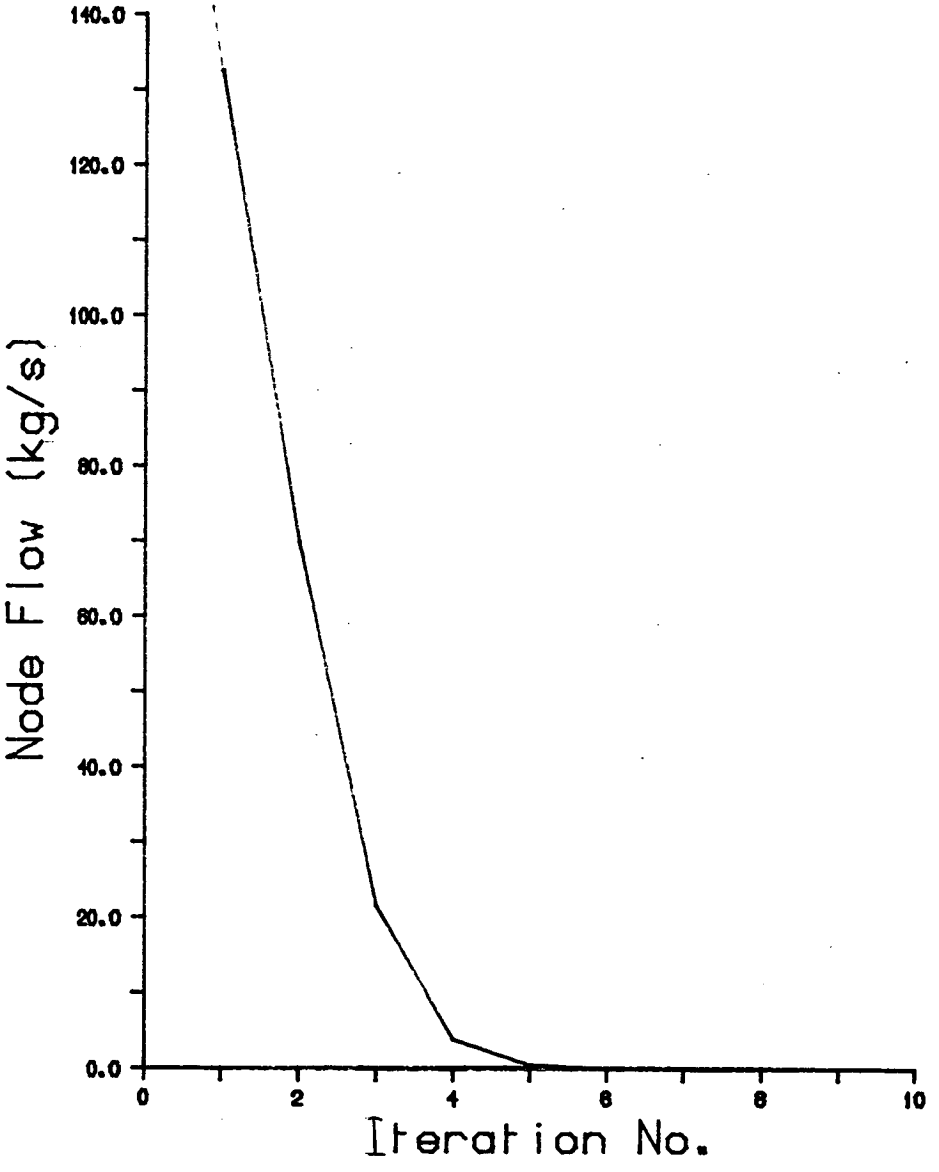
Case 10: Subnetwork of Network 9



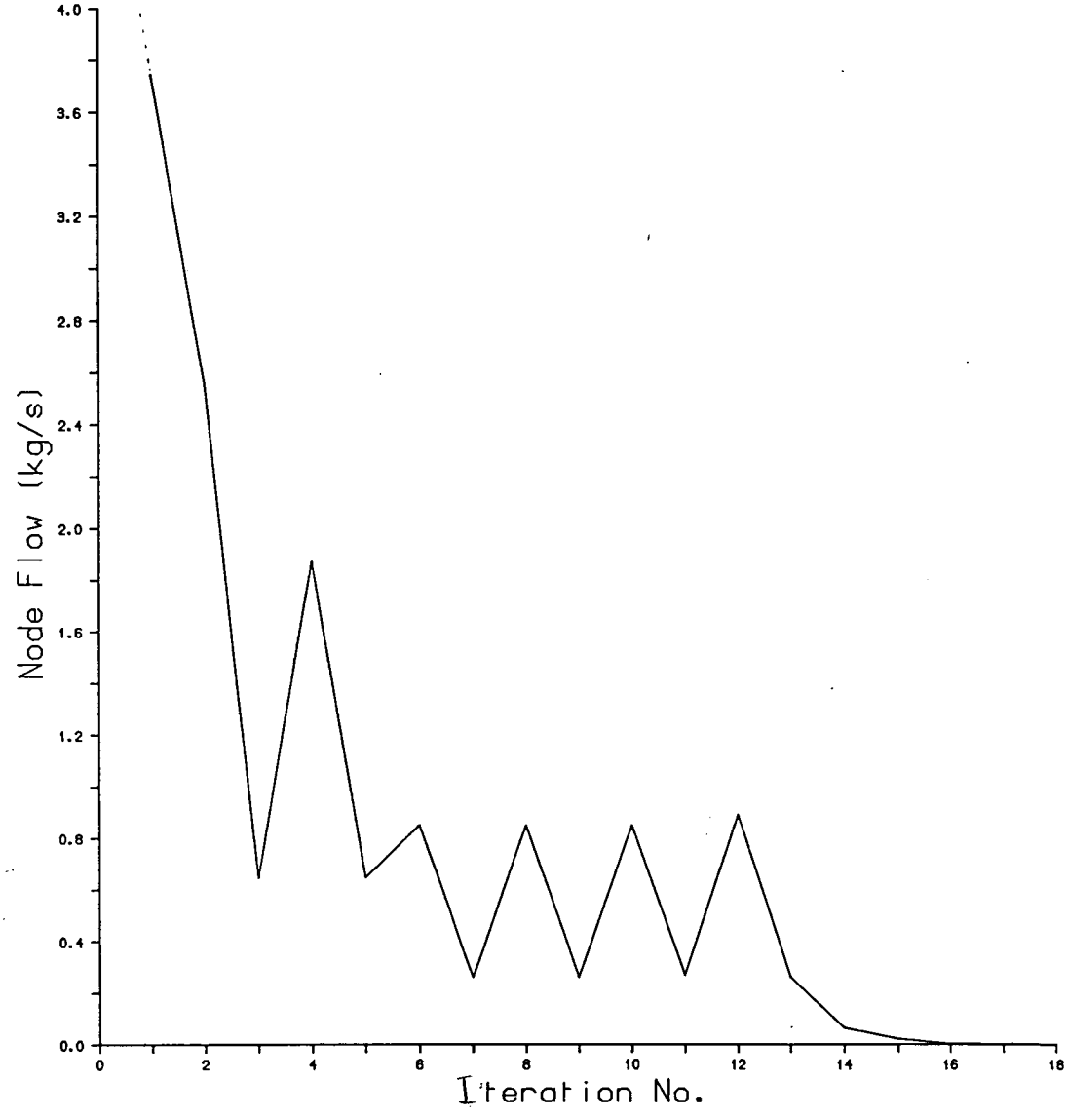
Case 11: Furnace Gas Distribution System



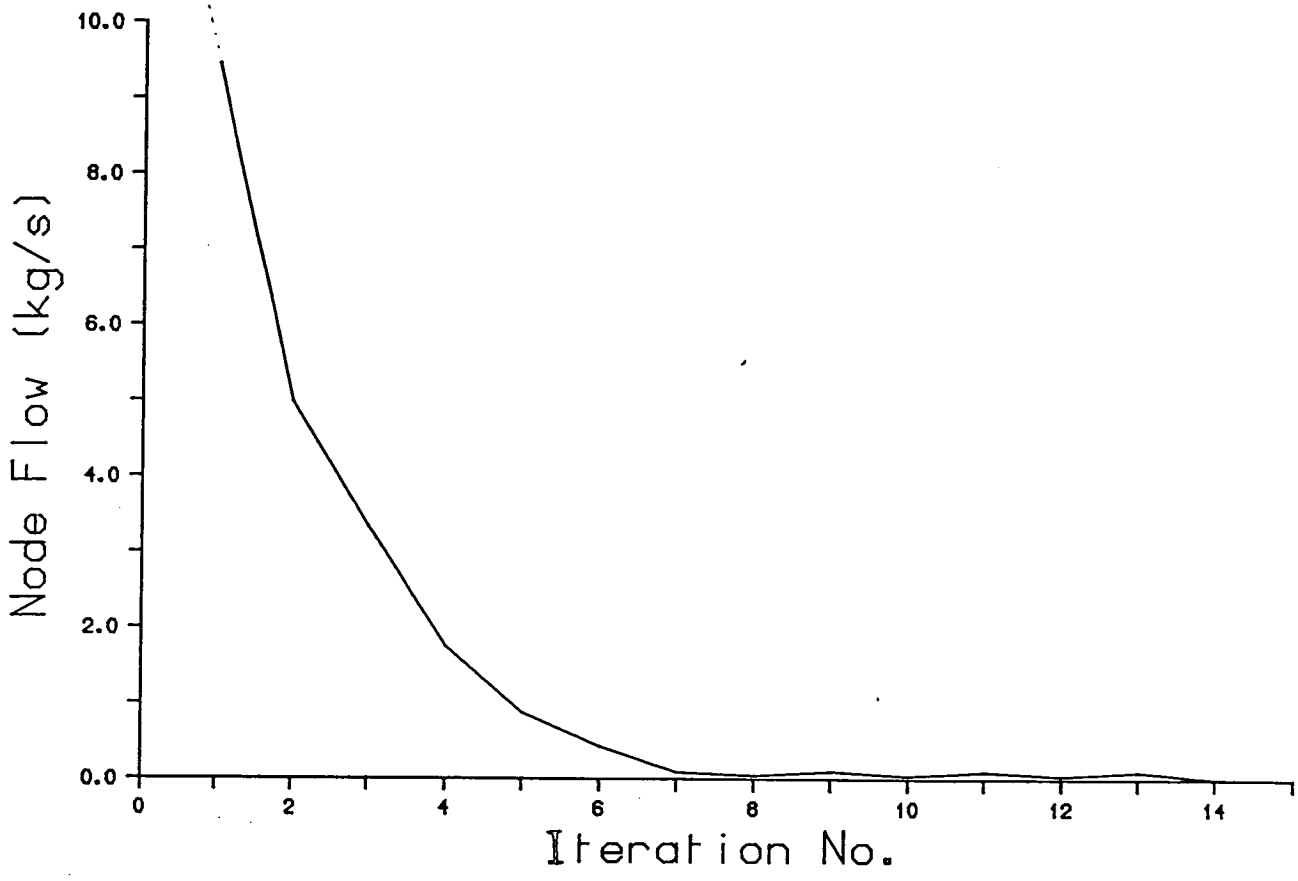
Case 12: Water Supply System



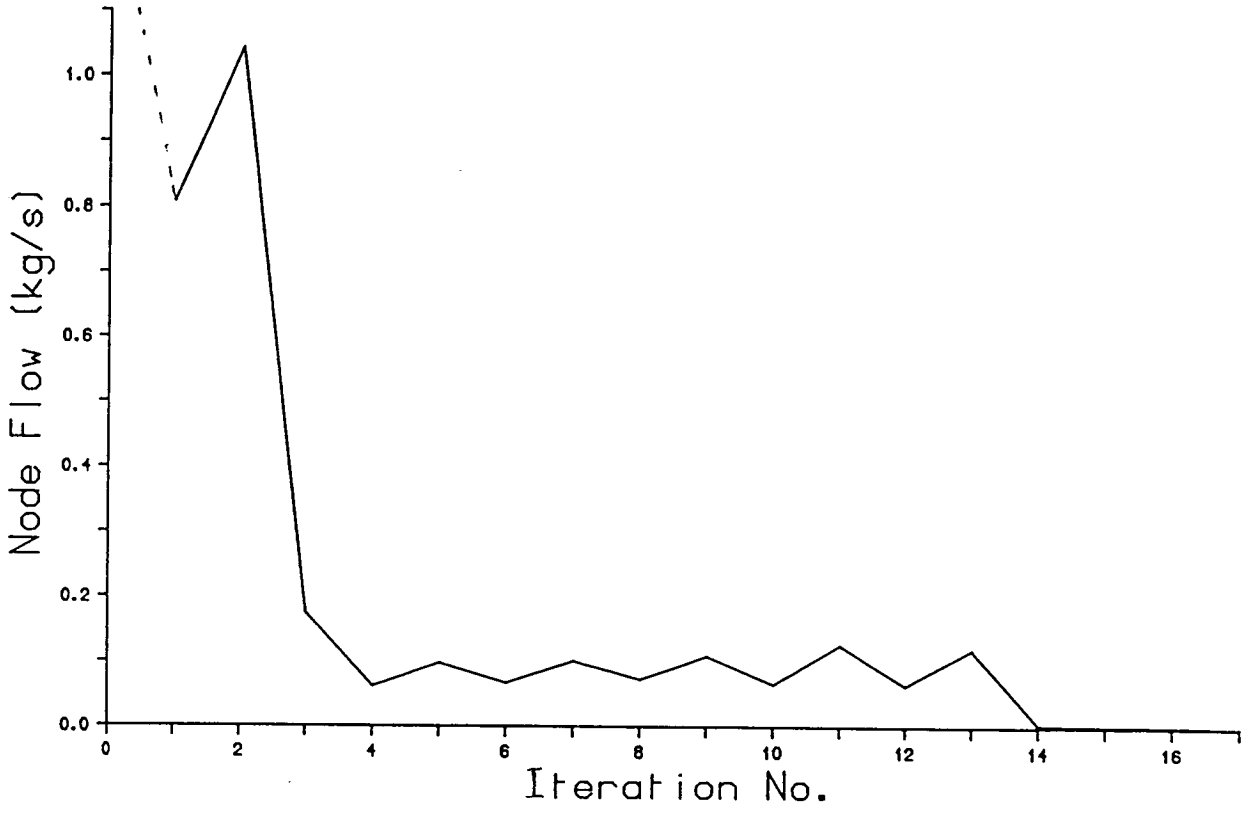
Case 13: Water Supply System



Case 14: Compressible Flow Network



Case 15: Brinefields network



IV. Listings for programs/modules referred to in Chapter 5

This appendix gives the listings for the following programs or modules:

The programs listed in this appendix are written in IMP80.
(Ref : "IMP80 Language Manual",
Felicity Stephens & John Murison,
Edinburgh Regional Computing Centre, 1981)

Program Notes :

The symbol '@' signifies the exponent (E).

In converting temperature values from celsius to kelvin, the factor +273 is used.

The value used for acceleration due to gravity (g) is 9.81.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Program to solve pipe network problems. Data input may include network
! equations as an alternative to 'number lists' specifying pipe/valve
! characteristics and pressure/flow conditions at nodes.
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
begin
!
!-----------------------------------------------------------------------
externalroutinespec pressures(longrealarrayname a,b,p,f,fo,c
integer nn,nf,y,integerarrayname pset,qset)
!
externalroutinespec set up a(longrealarrayname k,kb,p,tp,fn,ncap,c
a,b,den,ht,realarrayname qrterms,integerarrayname qterms,qtctr,c
u,d,pfix,longreal delta,tcon,integer qlctr,nn,nf,string(20) linmeth)
!
externalroutinespec nlink(integer nnodes,nlinks,c
integerarrayname in,out,ncode,link,cc,cp,longrealarrayname pp,fexx)
!
externalroutinespec fcheck(longrealarrayname q,c
fexx,integerarrayname pfix,link,in,cp,c
integer nn,longreal ftol,longrealname hftot,integername hfnod,check)
!
externalroutinespec flows(longrealarrayname p,kv,k,kb,f,fo,l,da,c
rk,ft,denav,den,ht,vis,cd,cv,cl,c2,temp,realarrayname qrterms,c
integerarrayname qtctr,qterms,ipbr,u,d,tlink,pform,integer c
pass,printit,nf,nn,npump,nfluid,string(20) linmeth)
!
externalroutinespec idenst(longrealarrayname cd,longrealname c
dens,longreal press,temp,integer nfluid)
!
externalroutinespec idfit(longrealarrayname d,p,t,longrealname c
cdl,cd2,cd3,integer nfluid)
!
externalroutinespec ipmpnet(longrealarrayname pchar,cl,c2,c
integerarrayname npts,pform,integer npump)
!
externalroutinespec ivfit(longrealarrayname v,p,t,longrealname c
cvl,cv2,cv3,integer nfluid)
!
externalroutinespec ivisco(longrealarrayname cv,longrealname c
visc,longreal press,temp,integer nfluid)
!
externalroutinespec ilnpump(longreal pl,p2,cl,c2,den,c
integer pform,n,longrealname q,a,b)
!
externalroutinespec setupm(longrealarrayname p,fnum,qn,k,kv,kb,a,rhs,c
realarrayname qrterms,rvalue,integerarrayname qterms,qtctr,pset,qset,c
pfix,ffix,qfix,in,out,tlink,ivalue,itype,integername qlctr,ierror,mrows,
integer pass,nn,nf,sum)
!
externalroutinespec set up rm(longrealarrayname ppi,p,k,kb,f,c
fn,qn,a,b,integerarrayname node,pset,qset,pfix,ffix,qfix,in,out,c
integer nn,nf,integername sum,mrows)
!
externalroutinespec getdata(longrealarrayname p,c
kv,l,da,rk,ft,temp,fn,ht,ndtp,tpres,tvisc,tden,ttemp,pchar,mu,ncap,sfp,c
```

```
integerarrayname node,ltno,in,out,ffix,pfix,npts,ipbr,tlink,ncode,c
itype,ivalue,u,d,realarrayname rvalue,integername c
nn,nf,npump,nfluid,ierror,longrealname ptot,rav,stringname filename)
!
externalroutinespec set up k(longrealarrayname f,fo,flst,k,kb,kv,p,plst,c
l,da,denav,temp,vis,ht,ft,rk,cv,cd,cl,c2,mu,sfp,integerarrayname c
ltno,in,out,tlink,pform,ipbr,integer nf,npump,c
nfluid,pass,string(20) linmeth)
!
externalroutinespec emas3prompt(stringname s)
!
externalroutinespec emas3(stringname comm,parms,c
integername flag)
!
!-------------------------------------------------------------------------
!      ***** <<<<<   MAIN PROGRAM   >>>>> *****
!
!  main arrays...
!   a - matrix for linearised equations
!   b - constant vector for  ..   ..
!   p - new pressures to be calculated
!   po- last pressures
!   kv - valve consts for flow=kv*sqrt(delta p)
!   k - linearised valve constants
!   f - new flows
!   fo - last flows
!   fn - node specified flows
!
!
!  'structure' arrays...
!   u(i) - number of node upstream on branch i
!   d(i) -   ..     ..   downstream ..  ..
!   pfix(i) - is 1 if pressure at node i is fixed specification,
!             0 if variable
!   ffix(i) - is 1 if flow at node i is fixed specification,
!             0 if variable
!              N.B. flow into node is +ve, out of node is -ve
!
longrealarray a(1:100,1:100),p,b,fo,f(1:100),po,ppi,ncap,tp,ht,c
plst,flst,l,da,rk,ft,qn,fn,den,nodtemp,temp,mu,sfp(1:40)
longrealarray k,kv,kb,denav,vis(1:40)
longrealarray tpres,tvisc,tden,ttemp(1:3),pchar(1:10,1:10)
longrealarray cl,c2(1:10)
longrealarray cv,cd(1:3)
realarray rvalue(1:40,1:10),qrts(1:10,1:5)
integerarray in,out,ffix,qfix,pfix,tlink,ncode(1:40),npts,ipbr(1:10)
integerarray itype,ivalue(1:40,1:10),node(1:40)
integerarray u,d(1:40)
integerarray cp,cc(1:40,1:6),link(1:40)
integerarray ltno,pset(1:40),qset(1:40)
integerarray pbr,pform(1:10)
integerarray qts(1:10,1:5),qtcount(1:10)
integer nn,nf,npump,ierror,i,j,mv
longreal ptot,rav,time,hftot,ftol,delta,tcon
integer hfnod,check,rcheck,qlcount
integer nfluid{-ve for gas, 0 or +ve for liquid}
integer ll, mm, ntotal, tc, y,HH,zz,mcc,qq,ks
integer sum, mrows, pass, eflag, pcount, zw
string(20) filename
string(40) outfile
```

```
!
ownstring (20) linmeth="newton" {initial solution method}
!
ftol=0.000001
!
!-------------------------------------------------------------------------
! CHECK P : This routine checks pressures for convergence
!           and returns 0 if sum of absolute changes is less
!           than specified limit. Also updates po().
!
integerfunction check p(longrealarrayname p,po,integer nn)
!
! in... p(),po(),nn
! out.. po()
integer i
longreal sum
sum=0
for i=1,1,nn cycle
  sum=sum+mod(p(i)-po(i))
  po(i)=p(i)
repeat
if sum<0.1 then result=0
printstring("press Error = ") ; printfl(sum,7) ; newline
result=1
!
end
!-------------------------------------------------------------------------
!
! initialise values which will be returned by the parser routine
for i=1,1,40 cycle
qset(i)=0
ppi(i)=0
for j=1,1,10 cycle
itype(i,j)=0 ; ivalue(i,j)=0
rvalue(i,j)=0
repeat
repeat
qtcount(i)=0 for i=1,1,10
for i=1,1,10 cycle
  for j=1,1,5 cycle
    qts(i,j)=0
    qrts(i,j)=0
  repeat
repeat
!
!
! Initialise pump parameter values
for i=1,1,10 cycle
pform(i)=0
c1(i)=0 ; c2(i)=0
repeat
!
qfix(i)=0 for i=1,1,40
!
! Get input data file
!
filename="name of file : "
emas3prompt(filename)
readstring(filename)
emas3("define","2,.out",eflag)
```

```
outfile="name of output file : "
emas3prompt(outfile)
! Get name of output file for results
!
readstring(outfile)
emas3("define","11,".outfile,eflag)
!
!set value of delta and tcon
tcon=0.0
delta=0.0
!
!initialise values of ncap (the capacity of each node in m**3)
!
ncap(zz)=0.0 for zz=1,1,40
!
! Call routine to read network data (and parse network equations if prese
getdata(p,kv,l,da,rk,ft,temp,fn,ht,nodtemp,tpres,tvisc,tden,c
ttemp,pchar,mu,ncap,sfp,node,ltno,in,out,ffix,pfix,npts,ipbr,tlink,c
ncode,itype,ivalue,u,d,rvalue,c
nn,nf,npump,nfluid,ierror,ptot,rav,filename)
!
!set value of tp
tp(zz)=0.0 for zz=1,1,nn
!
! assign values to elements of array ppi for nodes whose p's are fixed
!
for i=1,1,nn cycle
if pfix(i)=1 then ppi(i)=p(i)
repeat
!
! get no. of links to each node
!
nlink(nn,nf,in,out,ncode,link,cc,cp,p,fn)
!
! assign average node pressure (returned in 'ptot' by routine getdata)
! to nodes which have not been assigned inititial (fixed) pressures
!
      for y=1,1,nn cycle
      if p(y)=0 then start
      p(y)=ptot
      po(y)=ptot
      ppi(y)=ptot
      finish else start
      ppi(y)=p(y)
      po(y)=p(y)
      finish
      repeat
!
!     assign initial values to hfnod and hftot
!     hfnod is the node identifier of the node with the highest excess
!     inflow/outflow after each iteration
!     hftot is the value of the excess inflow/outflow.
!
      hfnod=0
      hftot=0
!
selectinput(0)
!
!get pump characteristics if there are pumps in the network
!
```

```
if npump>0 then start
ipmpnet(pchar,c1,c2,npts,pform,npump)
finish
!
!get viscosity and density fit details
        ivfit(tvisc,tpres,ttemp,cv(1),cv(2),cv(3),nfluid)
        idfit(tden,tpres,ttemp,cd(1),cd(2),cd(3),nfluid)
!
! initialise the constants in the pipe flow/pressure equations
!
for mv=1,1,nf cycle
k(mv)=0.0 ; kb(mv)=0.0
repeat
!
! call routine flows to get initial flow distribution in network
!
flows(p,kv,k,kb,f,fo,l,da,rk,ft,denav,den,ht,vis,cd,cv,cl,c2,temp,qrts,c
qtcount,qts,ipbr,u,d,tlink,pform,0,0,nf,nn,npump,nfluid,linmeth)
!
selectinput(0)
!
ntotal=nn+nf
!
for i=1,1,100 cycle ;! ---------------- start iteration ----------------
pass=1
if i=1 then start
pass=0
finish
!
! Get current pressure and flow values for input to routine set up k
!
plst(zw)=p(zw) for zw=1,1,nn
flst(zw)=f(zw) for zw=1,1,nf
!
set up k(f,fo,flst,k,kb,kv,p,plst,l,da,denav,temp,vis,ht,ft,rk,c
cv,cd,cl,c2,mu,sfp,ltno,u,d,tlink,pform,ipbr,nf,npump,nfluid,pass,linmeth)
!
set up rm(ppi,p,k,kb,f,c
fn,qn,a,b,node,pset,qset,pfix,ffix,qfix,u,d,nn,nf,sum,mrows)
!
! first time round call setupm to insert prespecified network equations ir
! full matrix of flow network equations
!
if pass=0 then setupm(ppi,fn,qn,k,kv,kb,a,b,qrts,rvalue,qts,qtcount,c
pset,qset,pfix,ffix,qfix,u,d,tlink,ivalue,itype,qlcount,ierror,mrows,c
pass,nn,nf,sum)
!
if i=1 then start
p(zz)=ppi(zz) for zz=1,1,nn
po(zz)=ppi(zz) for zz=1,1,nn
finish
!
selectoutput(2)
!
set up a(k,kb,p,tp,fn,ncap,a,b,denav,ht,qrts,qts,qtcount,u,d,pfix,c
delta,tcon,qlcount,nn,nf,linmeth)
!
pressures(a,b,p,f,fo,nn,nf,0,pset,qset)
!
flows(p,kv,k,kb,f,fo,l,da,rk,ft,denav,den,ht,vis,cd,cv,cl,c2,temp,qrts,c
```

```
qtcount,qts,ipbr,u,d,tlink,pform,pass,0,nf,nn,npump,nfluid,linmeth)
!     check for convergence
! check for flow convergence
!
fcheck(f,fn,pfix,link,u,cp,nn,ftol,hftot,hfnod,check)
!
newline; printstring("error = "); print(hftot,3,8)
printstring(" at node "); write(hfnod,3)
if (check = 0 and i>2) or i>40 then exit
!
repeat ;!              ---------------  next iteration ----------------
!
newline ; write(i,3)
flows(p,kv,k,kb,f,fo,l,da,rk,ft,denav,den,ht,vis,cd,cv,cl,c2,temp,qrts,c
qtcount,qts,ipbr,u,d,tlink,pform,l,l,nf,nn,npump,nfluid,linmeth)
!
selectoutput(2)
printstring("
  Used ") ; write(i,4) ; printstring(" iterations")
!
closestream(2)
selectoutput(11)
flows(p,kv,k,kb,f,fo,l,da,rk,ft,denav,den,ht,vis,cd,cv,cl,c2,temp,qrts,c
qtcount,qts,ipbr,u,d,tlink,pform,l,l,nf,nn,npump,nfluid,linmeth)
closestream(11)
!
endofprogram
```

```
!
!-----------------------------------------------------------------------
!
!  routine getdata : reads in data for a pipe network,presented as a list
!  pipe/pump/valve physical characteristics.
!
!  read in...
!  fixed pressures (pfix)
!  branches and associated kv's or piping data (u,d,kv,l,da,rk,ft,temp)
!  and set nn and nf
!
!-----------------------------------------------------------------------
!
externalroutine getdata(longrealarrayname p,kv,l,da,rk,ft,temp,fn,c
ht,nodtemp,tpres,tvisc,tden,ttemp,pchar,mu,ncap,sfp,c
integerarrayname node,ltno,in,out,ffix,pfix,npts,ipbr,tlink,ncode,c
itype,ivalue,u,d,realarrayname rvalue,integername c
nn,nf,npump,nfluid,ierror,longrealname ptot,rav,stringname filename)
!
externalroutinespec inoden(integer label,nnodes,integername index,c
integerarrayname node)
externalroutinespec indlis(integerarrayname lindex,node,instr,outstr,c
in,out,integername nnodes,npipes)
externalroutinespec eqparse(longrealarrayname kv,integerarrayname node,c
in,out,tlink,itype,ivalue,realarrayname rvalue,integername nn,nf,ierror)
externalroutinespec emas3(stringname comm,parms,integername flag)
!
integer i,f1,f2,f3,nnum,ii,jj,pcount,srflag,eflag,nnodes,qq,nlab
real nspec, pspec
integerarray uu,dd(1:40)
integerarray lindex,dlindex(1:80)
!
      ptot=0 {initialise sum of pressures}
      pcount=0 {initialise counter}
for i=1,1,40 cycle
      pfix(i)=0
      ffix(i)=0
      p(i)=0
      fn(i)=0
      ht(i)=0
      tlink(i)=-99
      ltno(i)=0
      mu(i)=0
      ncap(i)=0
      sfp(i)=0
!
repeat
!
!Enter branch details
!upstream node no., downstream node no.,
!if full piping details supplied answer > 0,
!if only constant is supplied answer 0 for non-linear, -1 for linear const
!if equation to be read in for this link later, enter <-1
!terminate with < 0
nnodes=0
nf=0
!
emas3("define","12,".filename,eflag)
selectinput(12)
```

```
for i=1,1,40 cycle
read(f1)
if f1<0 then exit
read(f2) ; if f2<0 thenexit
read(f3) ;
u(i)=(f1) ; d(i)=(f2)
uu(i)=u(i) ; dd(i)=d(i)
nf=nf+1
   if f3>0 then start
!signify that the link is either a pipe with full piping details
!supplied or it is a pump (the value of tlink is changed later
!in this routine if the link is a pump)
      tlink(i)=1
!read length (assumed to be in meters)
read(l(i))
!read pipe diameter (assumed to be in mm), and convert it to m
read(da(i)); da(i)=da(i)*1@-3
!read roughness ratio
read(rk(i))
!read fittings loss
read(ft(i))
!read temperature and convert it from celcius to kelvin
      read(temp(i)); temp(i)=temp(i)+273.0
!
   finish else if f3=0 or f3=-1 then start
!
!signify that link is a valve (valve const. supplied)
!
      tlink(i)=f3; {tlink=-1 signifies linear k, tlink=0 means non-linear
read(kv(i)); read(mu(i)) ; read(ltno(i)) ; read(sfp(i))
if ltno(i)>0 then sfp(i)=sfp(i)*100000
read(temp(i)) ; temp(i)=temp(i)+273.0
  finish else start
!
! the characteristics of this link are described in an equation later.
read(temp(i)) ; temp(i)=temp(i)+273.0
  kv(i)=0
  finish
repeat
!
!  get a list of all the nodes
  for qq=1,1,40 cycle
   in(qq)=0 ; out(qq)=0
   node(qq)=0
  repeat
   dlindex(i)=0 for i=1,1,40
   for qq=1,1,nf cycle
   lindex(qq*2-1)=u(qq) ; lindex(qq*2)=d(qq)
   repeat
   indlis(dlindex,node,uu,dd,in,out,nnodes,nf)
   for qq=1,1,40 cycle
   lindex(qq)=dlindex(qq)
   repeat
nn=nnodes
!Enter node conditions which are fixed
!             Answer >= 0 to prompt 'node spec ?' if specifying
!             a condition, else answer < 0
!
for i=1,1,40 cycle
read(nspec)
```

```
      if nspec < 0 thenexit
      read(nnum)
!  check that the node label is in the list
       inoden(nnum,nn,nlab,node)
!
!     the nodes are classified according to the following codes
!     code                  type of node
!     ----                  ------------
!      1                  fixed pressure node
!      2                  fixed flow node
!      3                  height specified
!
      read(ncode(nnum));!read the node code
      read(ncap(nnum));!   read the capacity of the node in m**3
      read(ht(nnum));!   read the height of the node in m
      if ncode(nnum)=2 then start; !fixed flow node
!
      pcount=pcount+1
read(fn(nnum)); ! note that flow in is +ve, flow out of node is -ve
      fn(nnum)=-fn(nnum)
         ffix(nnum)=1
      finish else start
      read(p(nnum))
      if p(nnum)>0 then start
      pcount=pcount+1
         p(nnum)=p(nnum)*1.0@5 ;! convert bar to n/m**2
         ptot=ptot+p(nnum)
      if ncode(nnum)=1 then pfix(nnum)=1
      finish
      finish
      read(nodtemp(nnum)) ; ! read the node temperature in C
      nodtemp(nnum)=nodtemp(nnum)+273.0 ; ! convert to kelvin
      repeat
!read in number of pumps in network
!
read(npump)
!
      if npump>0 thenstart
      for ii=1,1,npump cycle
!get the number of points on the characteristic.
      read(npts(ii))
      repeat
!
      for ii=1,1,npump cycle
      read(uu(ii)); read(dd(ii))
!
      for jj=1,1,nf cycle
!set value of tlink for appropriate link number.
!also set value of ipbr (link number of pump in network)
      if uu(ii)=u(jj) and dd(ii)=d(jj) then start
      tlink(jj)=2
!tlink(..) = 2 signifies a pump
      ipbr(ii)=jj
      finish
      repeat
!head/flow data
         for jj=1,1,npts(ii) cycle
!read head and flow values on pump characteristic
         read(pchar(ii,jj*2-1)); read(pchar(ii,jj*2))
         repeat
```

```
      repeat
      finish
!
      rav=0
!
!get the physical properties data
      for ii=1,1,3 cycle
      read(tpres(ii)); read(ttemp(ii))
      read(tden(ii)); read(tvisc(ii))
!convert bar to pascal, celcius to kelvin, cp to kg/ms
      tpres(ii)=tpres(ii)*1.0@5
      ttemp(ii)=ttemp(ii)+273.0
      tvisc(ii)=tvisc(ii)*1@-3
      rav=rav+(tpres(ii)/(tden(ii)*ttemp(ii)))
      repeat
!get value of nfluid (-ve for gas, 0 or +ve for liquid)
      read(nfluid)
!
! get the average value of the gas constant, rav.
!
      rav=rav/3
!
! see if should quit at this point ...
!
      read(f3)
!
if f3<0 then start
ptot=ptot/pcount
return
finish else start
eqparse(kv,node,in,out,tlink,itype,ivalue,rvalue,nn,nf,ierror)
finish
!
end
endoffile
```

```
!
!  ROUTINE EQPARSE : parses input equations describing network
!
externalroutinespec s to r(string(40) s,realname x,integername srflag)
externalroutinespec ucstrg(stringname s)
externalroutinespec inoden(integer label,nnodes,integername index,c
integerarrayname node)
!
externalroutine eqparse(longrealarrayname kv,integerarrayname node,c
in,out,tlink,itype,ivalue,realarrayname rvalue,integername nn,nf,ierror)
!
! For each atom, 4 entries are generated.
!      itype - indicates if item is P,Q,F or constant
!      ivalue - label indicating location of node/link in network
!      rvalue - coefficient of P,Q,F or value of constant
!
! itype         meaning           ivalue     rvalue
! -----         -------           ------     ------
!   1        pressure term     node label    coeff
!   2        link flow term    link label    coeff
!   3        node flow term    node label    coeff
!   4        constant term          -1       value
!
!  The node identifiers are F (f) and P (p).
!  The link identifier is Q (q).
!  There are also identifiers for individual nodes and links.
!  Links are identified in terms of the nodes between which they run.
!
!
integer eqstat,j,ollen,stsign,lbctr,rbctr
integer natoms,sign,srflag,index1,index2,index,eqlen,ll,dpt,am
integer istr1,istr2,istr,qq,i,m,jj,errcnt
real x,xstr1,xstr2,xstr
string(1) lcr,ccr,lbr,rbr,nlcar
string(1) array oper(1:3),nos(1:10)
string(20) numstr,str,str1,str2
string(80) line,oline,nline,errmes
!
!  set the newline character
   nlcar="
"
!  set the array of operator values
   oper(1)="+"
   oper(2)="-"
   oper(3)="="
!
!  set the string values for brackets
   lbr="("
   rbr=")"
!
!  initialise the array of integers
   nos(1)="1" ; nos(2)="2" ; nos(3)="3"
   nos(4)="4" ; nos(5)="5" ; nos(6)="6"
   nos(7)="7" ; nos(8)="8" ; nos(9)="9"
   nos(10)="0"
!
!  start reading in equation lines (max of 20 lines is expected)
!.....................................................................
   for m=1,1,20 cycle
```

```
!
   ierror=0
   errmes=""
   line=""
   nline=""
   oline=""
   ccr=""
!
!  call routine ucstr to read the line
   ucstrg(line)
!  get the line length
   eqlen=length(line)
   if line="E" or line = "e" then return
!
   lbctr=0
   rbctr=0
   natoms=0
!
!  initialise eqstat at start of read. eqstat signifies whether
!  an "=" has been encountered yet in the line.
!
   eqstat=-1
!
!  cycle for the max no. of atoms expected in equation line.
!  initialise the string variable holding the previous character.
   lcr=""
   oline=line
!  initialise the sign to "+"
!
!  initialise the coefficient of P,Q,and F terms.
   x=1
!
   sign=1 ; stsign=1
   for i=1,1,10 cycle ; ! start of main cycle to read line
!.....................................................................
!
!  no blanks allowed
   if oline->(" ").oline  then ierror=-1 and -> errorl
!
!  see if line commences with '+' or '-'
!
   if i=1 then start
     if substring(oline,1,1)="+" then start
        oline -> ("+").oline ; sign=1 ; lcr="+"
     finish
     if substring(oline,1,1)="-" then start
        oline -> ("-").oline ; sign=-1 ; lcr="-"
     finish
   finish
!
numtest : !test for numbers
!  see if next char is a number
   if oline="" then exit
   ccr=substring(oline,1,1)
   for j=1,1,10 cycle
   if ccr=nos(j) then -> numlab
   repeat
   if ccr="." then start
     -> numlab
   finish
```

```
!
!  if the next character is not a number :
!  it may be an operator
!
   if ccr="+" or ccr="-" or ccr="=" then start
!
   if lcr="+" or lcr="-" or lcr="=" then ierror = -2 and -> errorl
   if lcr="+" or lcr="-" or lcr="=" then ierror = -2 and -> errorl
     if ccr="+" and lbctr>rbctr then stsign=sign else stsign=1
     if ccr="+" then sign=1 and lcr="+" and oline->(ccr).oline
     if ccr="-" and lbctr>rbctr then stsign=sign else stsign=1
     if ccr="-" then sign=-1 and lcr="-" and oline->(ccr).oline
     if ccr="=" then eqstat=1 and lcr="=" and oline->(ccr).oline
   if ccr="=" then sign=1 and x=1
   -> newlab
   finish
!
!  test if left bracket is present.
   if ccr=lbr then start
    oline -> (ccr).oline
    lbctr=lbctr+1
    -> newlab
   finish
!
!  test if right bracket is present.
   if ccr=rbr then start
    rbctr=rbctr+1
    if rbctr>lbctr then ierror=-3 and -> errorl
    oline -> (ccr).oline
    -> newlab
   finish
!  test whether the next char is an identifier
   if ccr="P" or ccr="Q" or ccr="F" then start
     if lcr="" or lcr="+" or lcr="-" or lcr="=" then start
      if ccr="P" then nline=oline and -> plab
      if ccr="Q" then nline=oline and -> qlab
      if ccr="F" then nline=oline and -> flab
     finish else ierror = -4 and -> errorl
   finish
!
!  if none of these things, then error
   ierror = -5
   -> errorl
!
numlab : !numbers
!-------------------------------------------------------------------------
!
!  initialise decimal point counter
   dpt=0
!  initialise exponent counter
   am=0
!
   numstr=""
!  look at the rest of the line
   ollen=length(oline)
!
!  has the line been completed ?
   if ollen=0 then -> newlabl
!
   for jj=1,1,ollen cycle
```

```
      ccr=substring(oline,jj,jj)
      for j=1,1,10 cycle
    if ccr=nos(j) and jj=ollen then numstr=numstr.ccr and -> numdec
    if ccr=nos(j) and jj#ollen then numstr=numstr.ccr and -> ncont
      repeat
      if ccr="." then start
        if dpt=1 then ierror=-6 and -> errorl  else start
          dpt=1
          numstr=numstr.ccr
          lcr="."
          -> ncont
        finish
      finish
      if ccr="@" then start
        if am=1 then ierror=-7 and -> errorl else start
          am=1
          numstr=numstr."@"
          lcr="@"
!  look at nline to see if a valid number follows the exponent
          -> ncont
        finish
      finish
!
   if am=1 then start
!  if the next char is a "+" or "-" after an "@".
     if lcr="@" and (ccr="+" or ccr="-") then start
      numstr=numstr.ccr
      lcr=ccr
      -> ncont
     finish
     if (lcr="+" or lcr="-") and (ccr="+" or ccr="-") c
     then ierror=-8 and -> errorl
   finish
!
!  if the next char is none of these things, then decode the number
   if jj=ollen and ccr=lbr then ierror=-9 and -> errorl
      lcr=ccr
      -> numdec
!
ncont : repeat
!-------------------------------------------------------------------------
numdec : !decode the number
!
   s to r(numstr,x,srflag)
!  look at rest of line following number.
!
   if lcr=lbr then start
    lbctr=lbctr+1
    nline=substring(oline,jj+1,ollen)
    -> plab
   finish
!
  if jj#ollen then c
   nline=substring(oline,jj+1,ollen) else nline=""
   ll=length(nline)
if ll#0 then ccr=substring(nline,1,1) else ccr=nlcar
!  is number a constant ?
!
 if ccr="+" or ccr="-" or ccr="=" or ccr=")" or ll=0  then start
   natoms=natoms+1
```

```
   itype(m,natoms)=4  ; ivalue(m,natoms)= -1
   rvalue(m,natoms)=stsign*x*sign*eqstat
  if ccr="-" then sign=-1 else sign=1
  if ccr="=" then eqstat=1 and x=1
   if jj=ollen then exit
   lcr=ccr
   oline=nline
!  atom finished. continue atom cycle
      -> newlab
   finish
!
!  the number is a coefficient
!
   if nline -> (lbr).nline then start
!
!.....................................................................
plab : ! pressure
!
   if nline -> ("P(").nline then start
   natoms=natoms+1
   itype(m,natoms)=1
   rvalue(m,natoms)=stsign*x*sign*eqstat
   nline -> str.(")").nline
!
   -> nodelab
   finish
!
!.....................................................................
!  may be link flow identifier
!
qlab : ! flow in link
   if nline -> ("Q(").nline then start
   natoms=natoms+1
   itype(m,natoms)=2
   rvalue(m,natoms)=stsign*x*sign*eqstat
   nline -> str.(")").nline
   -> nodelab
!
   finish
!.....................................................................
flab : !
!  may be node flow identifier
!
   if nline -> ("F(").nline then start
   natoms=natoms+1
   itype(m,natoms)=3
   rvalue(m,natoms)=stsign*x*sign*eqstat
   nline -> str.(")").nline
   finish
!
!.....................................................................
nodelab : !
!  at this point call the special routine for labels, inoden
!
!  if identifier is Q there are two labels to be matched
   if itype(m,natoms)=2 then start
   str -> str1.(",").str2
   s to r(str1,xstr1,srflag)
   s to r(str2,xstr2,srflag)
   istr1=int(xstr1) ; istr2=int(xstr2)
```

```
    inoden(istr1,nn,index1,node)
    inoden(istr2,nn,index2,node)
    for qq=1,1,nf cycle
    if (index1=in(qq) and index2=out(qq)) then c
    ivalue(m,natoms)=qq and exit
    if (index1=out(qq) and index1=in(qq)) then c
    ivalue(m,natoms)=-(qq) and exit
    repeat
   finish else if itype(m,natoms)=1 or itype(m,natoms)=3 then c
   start
    s to r(str,xstr,srflag)
    istr=int(xstr)
    inoden(istr,nn,index,node)
    ivalue(m,natoms)=node(index)
    finish
    lcr=")"
!
!   if it's another number.
!
!   see if next char is a number
    oline=nline
    finish
!.....................................................................
newlab : repeat
!
newlab1 : repeat
!
error1 : !error handling
     if ierror  < 0 then start
       if ierror = -1 then errmes="Blank character not allowed"
       if ierror = -2 then errmes="Invalid character after operator"
       if ierror = -3 then errmes="Brackets not matching"
       if ierror = -4 then errmes="Next character should be P, Q or F"
       if ierror = -5 then errmes="Invalid character"
       if ierror = -6 then errmes="Error in position of decimal point"
       if ierror = -7 then errmes="Error in position of exponent"
       if ierror = -8 then errmes="Invalid character after exponent"
       if ierror = -9 then errmes="Brackets not closed"
       newline ; printstring(errmes." in line "); write(m,3); newline
     finish
!
end
endoffile
```

```
external c
routine set up rm(longrealarrayname ppi,p,k,kb,f,fn,qn,a,b,c
integerarrayname node,pset,qset,pfix,ffix,qfix,in,out,c
integer nn,nf,integername sum,mrows)
!
!This routine solves for pressures and flows by setting up the equations
!for flows into nodes and flows into pipes as separate entities.
!
! in...
! nn, nf, p, k, f, fn, pfix, ffix, qfix, in, out
!
! out...
! a, b, pset, fset
!
integer nl,i,j,s,ii,nfl,ln,flag,hh
integerarray flowval,flowdir(1:6),lmark(1:nf)
!
!get total number of equations (=no. of links + no. of nodes)
      nl=nn+nf
!
      flowval(i)=0 for i=1,1,6
      flowdir(i)=0 for i=1,1,6
      for i=1,1,nl cycle
      a(i,j)=0 for j=1,1,nl
      b(i)=0
      repeat
!
      pset(i)=0 for i=1,1,nn; qset(i)=0 for i=1,1,nf
!     lmark is a marker for each pipe. It is set to 1 once the flow
!     equation in that pipe has been inserted into the matrix. This
!     is in order that there can be no repetitions when the same
!     pipe is encountered again.
!
      lmark(i)=0 for i=1,1,nf
!
!     sum is the number of entities in the flow/pressure vector which
!     have been 'set' so far. ln is the line position marker.
!
      sum=0; ln=0
!
!     go through all the nodes
      cycle hh=1,1,nn
      ii=node(hh)
      if pfix(ii)=1 thenstart;! fixed pressure node
      ln=ln+1
!     first examine to see if this pressure is already in the
!     vector of flows and pressures (i.e. has it been 'set' yet)
!     if the entity is a new one then note its position in the
!     flow/pressure vector (as indicated by the value of 'sum')
      if pset(ii)=0 then sum=sum+1 and pset(ii)=sum
      a(ln,pset(ii))=1
      b(ln)=ppi(ii)
      finish
!
      if pfix(ii)=0 thenstart
      nfl=0
!     go through all links
      cycle i=1,1,nf
!     examine which links are connected to node ii
```

```
      if ii=out(i) or ii=in(i) thenstart
!     if pipe is previously 'unmarked' then mark it and set
!     flag to off.
   if lmark(i)=0 then lmark(i)=1 and flag=0 else flag=1
!  if flag is 'off' increase line no.
      if flag=0 and k(i)=0 and qfix(i)=1 then ln=ln+1
      if flag=0 and k(i)#0 then ln=ln+1
      nfl=nfl+1; flowval(nfl)=i
      if ii=out(i) then flowdir(nfl)=1 else flowdir(nfl)=-1
!     get b(ii)
      if flag=0 and k(i)#0 then b(ln)=-kb(i)
!
      if pset(ii)=0 then sum=sum+1 and pset(ii)=sum
!     is ii downstream or upstream node of pipe i?
        if ii=out(i) thenstart
         if pset(in(i))=0 then sum=sum+1 and pset(in(i))=sum
      if k(i)#0 then start
         if flag=0 then a(ln,pset(ii))=-k(i)
         if flag=0 then a(ln,pset(in(i)))=k(i)
      finish
        finish elsestart
         if pset(out(i))=0 then sum=sum+1 and pset(out(i))=sum
      if k(i)#0 then start
         if flag=0 then a(ln,pset(ii))=k(i)
         if flag=0 then a(ln,pset(out(i)))=-k(i)
      finish
        finish
      if qset(i)=0 then sum=sum+1 and qset(i)=sum
      if flag=0 and k(i)=0 and qfix(i)=1 then a(ln,qset(i))=1 c
      and b(ln)=qn(i)
      if flag=0 and k(i)#0 then a(ln,qset(i))=-1
      finish
!
      repeat
!     sum flows at a node if appropriate
      if nfl>=1 and ffix(ii)=1 then start
       ln=ln+1
       a(ln,qset(flowval(j)))=flowdir(j) for j=1,1,nfl
       b(ln)=fn(ii)
      finish
!
      finish
      repeat
!
   mrows=ln
!
      end
endoffile
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       ROUTINE SETUPM
!       --------------
!       Adds equations in the data input file to the matrix of
!       network equations.
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
external routine setupm(longrealarrayname p,fnum,qn,k,kv,kb,a,rhs,c
realarrayname qrterms,rvalue,integerarrayname qterms,qtctr,pset,qset,c
pfix,ffix,qfix,in,out,tlink,ivalue,itype,integername qlctr,ierror,mrows,
integer pass,nn,nf,sum)
!
!This routine is a modified version of 'set up reqn', which solves
!for pressures and flows by setting up the equations for flows into
!nodes and flows in pipes as separate entries. In this routine,
!the matrix created from network data is added to, using additional
!network equations specified by the user.
!
! in...
! itype, ivalue, rvalue
!
! out...
! a, b, pset, qset
!
integer nl,i,j,s,mm,ii,nfl,ln,flag,nodl,nod2
integer inls,onls,nsp,modp,ppl,pp2,m,npts,nfts,nqts
integer ncts,mdim,jj,kk,natoms,index,cpindex,nodiql,nodiq2
integer pcon,nindex,pindex,pinctr,fldir,nodiq
real cpincf,pincf,pplcf,pp2cf
real cfl,cf2
longrealarray irhs(1:100)
integerarray flowval,flowdir(1:6),lmark(1:nf)
!
!  initialise the error flag, ierror
   ierror=0
!
!  initialise the number of link flows which are fixed
   qlctr=0
   qtctr(i)=0 for i=1,1,10
   for mm=1,1,10 cycle
     qterms(mm,i)=0 for i=1,1,5
     qrterms(mm,i)=0 for i=1,1,5
   repeat
!
!  the dimension of the (square) matrix = no. of links + no. of nodes.
   mdim = nn+nf
!
   for m=1,1,20 cycle ; !read each equation line
     if itype(m,1)=0 then -> nextcont
!
!increment no. of rows in matrix
   if pass=0 then mrows=mrows+1
!check that current no. of rows does not exceed mdim.
   if mrows>mdim then ierror=-1 and -> errorl
!
   a(mrows,i)=0 for i=1,1,mdim
   irhs(mrows)=0
   npts=0 ; nqts=0 ; nfts=0 ; ncts=0 ; natoms=0
!
```

```
   for i=1,1,10 cycle
    if itype(m,i)=0 then exit
    natoms=natoms+1
    if itype(m,i)=1 then npts=npts+1
    if itype(m,i)=2 then nqts=nqts+1
    if itype(m,i)=3 then nfts=nfts+1
    if itype(m,i)=4 then ncts=ncts+1
   repeat
!
!-------------------------------------------------------------------------
!  is the equation a fixed pressure specification ; i.e. does it
!  contain only one pressure term and some constant terms ?
!
   if npts=1 and nqts=0 and nfts=0 and ncts>0 then start
!
      irhs(mrows)=0
      for j=1,1,natoms cycle
         if itype(m,j)=1 then start
!  look at the node index.
          index=ivalue(m,j)
          if pset(index)=0 then sum=sum+1 and pset(index)=sum
          a(mrows,pset(index))=rvalue(m,j)
          pfix(index)=1
         finish else start
          irhs(mrows)=irhs(mrows)+rvalue(m,j)
         finish
      repeat
!
   if irhs(mrows)>0 and a(mrows,pset(index))<0 then c
   a(mrows,pset(index))=0-a(mrows,pset(index))
   if irhs(mrows)<0 and a(mrows,pset(index))>0 then c
   irhs(mrows)=0-irhs(mrows)
   if irhs(mrows)<0 and a(mrows,pset(index))<0 then c
   ierror=-12 and -> error1
   p(index)=irhs(mrows)*1@5
   rhs(mrows)=irhs(mrows)*1@5
   continue
   finish
!-------------------------------------------------------------------------
!  is the equation a pseudo fixed pressure spec ; i.e. does it
!  contain two pressure terms and some constant terms ?
!
   if npts=2 and nqts=0 and nfts=0 and ncts>0 then start
!
      pinctr=0
      irhs(mrows)=0
      pp1=0 ; pp2=0; pp1cf=0 ; pp2cf=0
      for j=1,1,natoms cycle
         if itype(m,j)=1 then start
         pindex=0 ; pincf=0
         pinctr=pinctr+1
!  look at the node index.
          index=ivalue(m,j)
          if pset(index)=0 then sum=sum+1 and pset(index)=sum
          a(mrows,pset(index))=-rvalue(m,j)
          if pp1=0 and pfix(index)=0 then start
          pp1=-(index)
           pp1cf=rvalue(m,j)
          if pinctr=1 then pindex=index and pincf=rvalue(m,j)
          continue
```

```
            finish
            if ppl=0 and pfix(index)=1 then start
            cpindex=index ; cpincf=rvalue(m,j)
            finish
            if ppl#0 and pfix(index)=0 then pp2=-(index) and c
            pp2cf=rvalue(m,j)
          finish else start
!
            irhs(mrows)=irhs(mrows)+rvalue(m,j)
      finish
      repeat
!
!  see if both nodes are of unspecified pressure
!
   if ppl<0 and pp2<0 then start
     index=-(ppl) ; pfix(index)=pp2
     index=-(pp2) ; pfix(index)=ppl
   finish else start
     if ppl<0 then start
       nindex=-(ppl) ; pfix(nindex)=1
       if pindex>0 then start
        if pplcf<0 and rvalue(m,j)>0 then p(nindex)=c
        p(index)+irhs(mrows)
        if pplcf>0 and rvalue(m,j)<0 then p(nindex)=c
        p(index)-irhs(mrows)*1@5
       finish else start
        if pplcf<0 and cpincf>0 then p(nindex)=c
        p(cpindex)+irhs(mrows)
        if pplcf>0 and cpincf<0 then p(nindex)=c
        p(cpindex)-irhs(mrows)*1@5
       finish
     finish
   finish
!
   rhs(mrows)=irhs(mrows)*1@5
   -> contline
   finish
!-------------------------------------------------------------------------
! is the equation a flow specification for a pipe ?
!
   if npts=2 and nqts=1 and nfts=0 and ncts>=0 then start
!
!  get the identifiers of the end nodes.
!
   rhs(mrows)=0
   nod1=0 ; nod2=0 ; cf1=0 ; cf2=0
!
   for i=1,1,10 cycle
   if itype(m,i)=1 then start
     if nod1=0 then nod1=ivalue(m,i) and cf1=rvalue(m,i)else c
     nod2=ivalue(m,i) and cf2=rvalue(m,i)
   finish else if itype(m,i)=2 then start
   index=ivalue(m,i) ; if index<0 then index=-(index)
   ppl=in(index) ; pp2=out(index)
   finish else rhs(mrows)=rhs(mrows)+rvalue(m,i)
   repeat
!
!  check that the two nodes are connected by the given pipe no.
       if (nod1=ppl and nod2=pp2) or c
       (nod1=pp2 and nod2=ppl) then start
```

```
        if cf1#-(cf2) then ierror=-7 and -> error1
finish else start
!
!
ierror=-8
-> error1
finish
!
!  check that a 'k' value has not been assigned to this pipe ; if so
!  then assume that the coefficient (cf1,cf2) is the
!  required 'k' and set up the standard equation for that pipe.
!
          if k(index)#0 then ierror=-9 and -> error1
          if pset(nod1)=0 then sum=sum+1 and pset(nod1)=sum
          k(index)=cf2
          tlink(index)=-1 ; kv(index)=k(index)
          kb(index)=0
          a(mrows,pset(nod1))=-cf1
          a(mrows,pset(nod2))=-cf2
          if qset(index)=0 then sum=sum+1 and qset(index)=sum
          a(mrows,qset(index))=-1
!
continue
   finish
!
!-------------------------------------------------------------------------
! Q terms only
!
   if nqts>=1 and ncts>=0 and nfts=0 and npts=0 then start
   inls=0
   onls=0
   rhs(mrows)=0
!
     if nqts=1 then start ; !look at the end nodes
      for jj=1,1,natoms cycle
       if itype(m,jj)=2 then start
! see if the end nodes of the pipe are pendant nodes
        modp=ivalue(m,jj)
        fldir=1
        if modp<0 then modp=-(modp) and fldir=-1
        for kk=1,1,nf cycle
         if in(modp)=in(kk) or c
         in(modp)=out(kk) then inls=inls+1
         if out(modp)=in(kk) or c
         out(modp)=out(kk) then onls=onls+1
        repeat
        if inls=1 and onls=1 then ierror=-9 and -> error1
        if inls=1 or onls=1 then start
         if inls=1 then nsp=in(modp) else c
         nsp=out(modp)
         if pfix(nsp)=1 or ffix(nsp)=1 then ierror=-10 and -> error1
!
!  put the line in the coeff matrix
!
         if qset(modp)=0 then sum=sum+1 and qset(modp)=sum
         a(mrows,qset(modp))=-rvalue(m,jj)
         ffix(nsp)=1
        finish
       if inls>1 or onls>1 then start
!
```

```
! put the line in the coeff matrix
!
      if qset(modp)=0 then sum=sum+1 and qset(modp)=sum
      a(mrows,qset(modp))=-rvalue(m,jj)
      qfix(modp)=1
       finish
       finish else rhs(mrows)=rhs(mrows)+rvalue(m,jj)
      repeat
     if a(mrows,qset(modp))>0 then start
     if fldir=1 and onls=1 then fnum(nsp)=rhs(mrows)
     if fldir=-1 and onls=1 then fnum(nsp)=-rhs(mrows) and c
     rhs(mrows)=-rhs(mrows)
     if fldir=1 and inls=1 then fnum(nsp)=-rhs(mrows) c
     and rhs(mrows)=-rhs(mrows)
     if fldir=-1 and inls=1 then fnum(nsp)=rhs(mrows)
     if fldir=1 and (inls>1 or onls>1) then qn(modp)=rhs(mrows)
     if fldir=-1 and (inls>1 or onls>1) then qn(modp)=-rhs(mrows)
     finish
     finish
!
!if there are two 'q' terms.
!
   if nqts>1 then start
!
!  increment the counter for the number of fixed flows
!
     qlctr=qlctr+1
   nodiq=0 ; nodiq1=0 ; nodiq2=0
   mm=0
   for jj=1,1,natoms cycle
    if itype(m,jj)=2 then start
     mm=mm+1
     modp=ivalue(m,jj)
     fldir=1
     if modp<0 then modp=-(modp) and fldir=-1
!get the identifier of flow which is defined in terms of other network f
     if mm=1 then qtctr(qlctr)=modp
     if mm>1 then qterms(qlctr,mm-1)=modp
!
!put the line in the coefficient matrix
!
     if qset(modp)=0 then sum=sum+1 and qset(modp)=sum
     a(mrows,qset(modp))=-rvalue(m,jj)
     if mm>1 then qrterms(qlctr,mm-1)=-rvalue(m,jj)
!    if pfix(in(modp))=0 and (in(modp)=nodiq or c
!    ffix(in(modp))=0) then c
!    ffix(in(modp))=1 and nodiq=in(modp)
!    if pfix(out(modp))=0 and (out(modp)=nodiq or c
!    ffix(out(modp))=0) then c
!    ffix(out(modp))=1 and nodiq=out(modp)
     if nodiq1=0 then nodiq1=nodiq else nodiq2=nodiq
    finish else rhs(mrows)=rhs(mrows)+rvalue(m,jj)
   repeat
   if nodiq#0 and nodiq1=nodiq2 then fnum(nodiq)=rhs(mrows)
   finish
   finish
contline : !continue
!-------------------------------------------------------------------------
      repeat ; !finish reading each equation line
!
```

```
nextcont : !next
!   check that all k's have been assigned.
      for i=1,1,nf cycle
         if k(i)=0 then ierror=-1 and exit
      repeat
!
   for ii=1,1,nn cycle
!
!  check that flow balances have been set up for all nodes
!  where the pressure has not been assigned.
!
     if pfix(ii)=0 and ffix(ii)=0 then start
      mrows=mrows+1
      rhs(mrows)=0
      for jj=1,1,nf cycle
       if ii=in(jj) or ii=out(jj) then start
        if ii=in(jj) then a(mrows,qset(jj))=-1 c
        else a(mrows,qset(jj))=1
        ffix(ii)=1
       finish
      repeat
     finish
!
!  check that, for any node which has pressure specified in terms
!  of pressure at another node, that the pressure specification is
!  now specific
!
     if pfix(ii)<0 then start
       pcon=-(pfix(ii))
       if pfix(pcon)=1 then pfix(ii)=1 else c
       ierror=-6 and -> error1
     finish
   repeat
!
error1 : !error label
end
endoffile
```

```
!
!
!              ***** DYNAMIC NETWORK PROGRAM *****
!
begin
!
!-----------------------------------------------------------------------
externalroutinespec flprint(longrealarrayname p,k,kb,f,c
integerarrayname u,d,integer nn,nf)
!
externalroutinespec pressures(longrealarrayname a,b,p,f,fo,c
integer nn,nf,y,integerarrayname pset,qset)
!
externalroutinespec set up a(longrealarrayname k,kb,p,tp,fn,ncap,c
a,b,den,ht,realarrayname qrterms,integerarrayname qterms,qtctr,u,d,pfix,c
longreal delta,tcon,integer qlctr,nn,nf,string(20) linmeth)
!
externalroutinespec emas3cputime(longrealname time)
!
externallongrealfnspec logten(longreal x)
!
externalroutinespec nlink(integer nnodes,nlinks,c
integerarrayname in,out,pfix,link,cc,cp,longrealarrayname pp,fexx)
!
externalroutinespec fcheck(longrealarrayname q,c
fexx,integerarrayname pfix,link,in,cp,c
integer nn,longreal ftol,longrealname hftot,integername hfnod,check)
!
externalroutinespec flows(longrealarrayname p,kv,k,kb,f,fo,l,da,rk,ft,c
denav,den,ht,vis,cd,cv,cl,c2,temp,realarrayname qrterms,c
integerarrayname qtctr,qterms,ipbr,u,d,tlink,pform,integer pass,printit,c
nf,nn,npump,nfluid,string(20) linmeth)
!
externalroutinespec iaux(longrealname a,rhs,pp,integername nn,nz,c
nm,licn,lirn,icn,irn,ikeep,ivect,jvect,iw,idisp,c
rpt,longrealname anag,w)
!
externalroutinespec idenst(longrealarrayname cd,longrealname c
dens,longreal press,temp,integer nfluid)
!
externalroutinespec idfit(longrealarrayname d,p,t,longrealname c
cd1,cd2,cd3,integer nfluid)
!
externalroutinespec ipmpnet(longrealarrayname pchar,c1,c2,c
integerarrayname npts,pform,integer npump)
!
externalroutinespec ivfit(longrealarrayname v,p,t,longrealname c
cv1,cv2,cv3,integer nfluid)
!
externalroutinespec ivisco(longrealarrayname cv,longrealname c
visc,longreal press,temp,integer nfluid)
!
externalroutinespec ilnpump(longreal p1,p2,c1,c2,den,c
integer pform,n,longrealname q,a,b)
!
externalroutinespec set up rm(longrealarrayname ppi,p,k,kb,f,c
fn,qn,a,b,integerarrayname node,pset,qset,pfix,ffix,qfix,in,out,c
integer nn,nf,integername sum,mrows)
!
```

```
externalroutinespec getdata(longrealarrayname p,kv,l,da,rk,ft,c
temp,fn,ht,ntemp,tpres,tvisc,tden,ttemp,pchar,mu,ncap,sfp,c
integerarrayname node,ltno,in,out,ffix,pfix,npts,ipbr,tlink,ncode,c
itype,ivalue,u,d,realarrayname rvalue,integername c
nn,nf,npump,nfluid,ierror,longrealname ptot,rav,stringname filename)
!
externalroutinespec set up k(longrealarrayname f,fo,flst,k,kb,kv,p,c
plst,l,da,denav,temp,vis,ht,ft,rk,cv,cd,cl,c2,mu,sfp,integerarrayname c
ltno,in,out,tlink,pform,ipbr,integer nf,npump,c
nfluid,pass,string(20) linmeth)
!
externalroutinespec rsolex eqn(longrealarrayname a,b,p,f,fo,anag,c
w,integer nn,nf,y,nm,integername nz,licn,lirn,integerarrayname c
u,d,node,pset,qset,icn,irn,ikeep,iw,idisp,ivect,jvect,string(20) linmeth)
!
externalroutinespec emas3prompt(stringname s)
!
externalroutinespec emas3(stringname comm,parms,c
integername flag)
!
externalroutinespec opensq(integer m)
!
externalrealfnspec random(integername i, integer n)
!
!-------------------------------------------------------------------------
!      ***** Main program starts here *****
!            ------------------------
!
!      ***** beginning of declarations *****
!
!  main arrays...
!   a - matrix for linearised equations
!   b - constant vector for  ..   ..
!   p - new pressures to be calculated, or delta p's in Newton solution
!   po- last pressures
!   kv - valve consts for flow=kv*sqrt(delta p)
!   k - linearised valve constants
!   f - new flows.  NB max 2* no.of nodes
!   fo - last flows
!   fn - node specified flows
!
!
!  'structure' arrays...
!   u(i) - number of node upstream on branch i
!   d(i) -   ..     ..   downstream ..  ..
!   pfix(i) - is 1 if pressure at node i is fixed specification,
!              0 if variable
!   ffix(i) - is 1 if flow at node i is fixed specification,
!              0 if variable
!               N.B. flow into node is +ve, out of node is -ve
!
longrealarray a(1:40,1:40),p,plst,b(1:40),po,ppi,ncap,ptp,tp,ht,c
l,da,rk,ft,fn,den,temp,ntemp,mu,sfp(1:40)
longrealarray k,kv,kb,fo,f,flst,denav,vis(1:40)
longrealarray tpres,tvisc,tden,ttemp(1:3),pchar(1:10,1:10)
longrealarray cl,c2(1:10)
longrealarray cv,cd(1:3)
realarray rvalue(1:40,1:10),qrts(1:10,1:5)
integerarray ltno,pset(1:40),qset(1:40)
integerarray pbr,pform(1:10),qts(1:10,1:5)
```

```
integerarray cp,cc(1:40,1:6),link(1:40),qtcount(1:10)
integerarray in,out,ffix,pfix,tlink,ncode(1:40),npts,ipbr(1:10)
integerarray itype,ivalue(1:40,1:10),node(1:40)
integerarray u,d(1:40)
longreal ptot
longreal time2, time, tcon, hftot, ftol, rav
real delta,deltac,tnext,xmax,xrandom,deltatime,deltaclast,deltaco
real tmax
string(40) outfile
integer nm,hfnod,check,rcheck,oflag,nodemax,ntstep
integer nfluid{-ve for gas, 0 or +ve for liquid}
integer ll,mm,tc,y,zz,mcc,ks,irandom,nrandom
integer sum, mrows,pass,eflag,pcount,pcset,pc2s
integer nn,nf,npump,ierror,i,j,zw,zy,qlcount,ff
string(20) filename
string(1) ans
!
!
ownintegerseed1=1234567
ownintegerseed2=7654321
ownreal switch=10 {..after ? iterations switch to Newton}
ownreal eqset=0 {..solve full set or short set of eqns}
ownreal eps=0.001 {small number for 'compressibility'}
ownstring (20) linmeth="hutchison" {initial solution method}
ownstring (20) solmeth="shortset" {solve for pressures only}
!
!-------------------------------------------------------------------------
! CHECK P : This routine checks pressures for convergence
!           and returns 0 if sum of absolute changes is less
!           than specified limit. Also updates po().
!
integerfunction check p(longrealarrayname p,po,integer nn)
!
! in... p(),po(),nn
! out.. po()
integer i
longreal sum
sum=0
for i=1,1,nn cycle
  sum=sum+mod(p(i)-po(i))
  po(i)=p(i)
repeat
!
! current limit is 0.00001 N/m**2
!
if sum<0.00001 then result=0
!newline
!printstring("press Error = ") ; !printfl(sum,7) ; !newline
result=1
!
end
!-------------------------------------------------------------------------
!
nm=10
ftol=0.0000001
!
!
qlcount=0
for i=1,1,10 cycle
  qtcount(i)=0
```

```
  for j=1,1,5 cycle
    qts(i,j)=0; qrts(i,j)=0
  repeat
repeat
!
for i=1,1,40 cycle
for j=1,1,10 cycle
itype(i,j)=0 ; ivalue(i,j)=0
rvalue(i,j)=0
repeat
p(i)=0
f(i)=0
ppi(i)=0
repeat
!
for i=1,1,10 cycle
pform(i)=0
c1(i)=0 ; c2(i)=0
repeat
!
!
filename="name of file : "
emas3prompt(filename)
readstring(filename)
emas3("define","2,.out",eflag)
outfile="name of output file : "
emas3prompt(outfile)
! Get name of output file for results
!
readstring(outfile)
emas3("define","11,".outfile,eflag)
emas3("define","20,cfkout",eflag)
emas3("define","21,cfout",eflag)
emas3("define","22,cpout",eflag)
emas3("define","23,cpkout",eflag)
emas3("define","25,dy2list",eflag)
!
getdata(p,kv,l,da,rk,ft,temp,fn,ht,ntemp,tpres,tvisc,tden,c
ttemp,pchar,mu,ncap,sfp,node,ltno,in,out,ffix,pfix,npts,ipbr,tlink,c
ncode,itype,ivalue,u,d,rvalue,c
nn,nf,npump,nfluid,ierror,ptot,rav,filename)
!get no. of links to each node
nlink(nn,nf,in,out,ncode,link,cc,cp,p,fn)
!
      for y=1,1,nn cycle
      if ncap(y)>0 then ncap(y)=ncap(y)/(rav*ntemp(y))
      if p(y)=0 then start
      p(y)=ptot
      po(y)=ptot
      ppi(y)=ptot
      finish else start
      ppi(y)=p(y)
      po(y)=p(y)
      finish
      repeat
!
!     assign initial value to hfnod
      hfnod=0
      hftot=0
!
```

```
!get pump characteristics if there are pumps in the network
if npump>0 then start
ipmpnet(pchar,c1,c2,npts,pform,npump)
finish
!
!get viscosity and density fit details
        ivfit(tvisc,tpres,ttemp,cv(1),cv(2),cv(3),nfluid)
        idfit(tden,tpres,ttemp,cd(1),cd(2),cd(3),nfluid)
!
for y=1,1,nf cycle
k(y)=0.0 ; kb(y)=0.0
repeat
flows(p,kv,k,kb,f,fo,l,da,rk,ft,denav,den,ht,vis,cd,cv,cl,c2,temp,c
qrts,qtcount,qts,ipbr,u,d,tlink,pform,0,0,nf,nn,npump,nfluid,linmeth)
!
!      ****** start of program run with defined data set *****
!
cycle
emas3cputime(time)
!
selectinput(0)
emas3prompt("Switch to Newton:") ; read(switch)
emas3prompt("Label of node for pressure control:"); read(nodemax)
if nodemax>0 then start
emas3prompt("Maximum pressure variation at this node (Newtons):")
read(xmax);xmax=xmax*2
finish
emas3prompt("time step : "); read(delta)
      deltatime = 0
      deltac=0
      deltaco=0
if nodemax>0 then start
 emas3prompt("non-disturbance time(secs) : "); read(deltatime)
finish
emas3prompt("no of time steps : "); read(ntstep)
emas3prompt("value of tmax : "); read(tmax)
      tcon=0; ! initialise time counter
      tp(zz)=0 for zz=1,1,nn
      oflag=1;! set convergence flag to off
!
!
!-------------------------------------------------------------------------
!      ***** read data into file for plotting with "EASYGRAPH" *****
!
!     only print data for node whose pressure is being controlled.
      for j=1,1,nf cycle
      if ltno(j)>0 then start
      selectoutput(22)
!     write in time and pressure as data pairs
      newline
      print(deltac,5,2); print(p(ltno(j))*0.00001,4,10)
      newline
      finish
      repeat
!
!     closestream(22)
      selectoutput(21)
!
!     only print data for link through which flow is controlled
      for j=1,1,nf cycle
```

```
      if ltno(j)<0 then start
      ltno(j)=0-ltno(j)
!     write in time and flow as data pairs
      print(deltac,5,2); print(f(ltno(j)),4,5)
      ltno(j)=0-ltno(j)
      newline
      finish
      repeat
!
!      closestream(21)
      selectoutput(20)
!
      for j=1,1,nf cycle
      if ltno(j)<0 then start
!     write in time and valve constant as data pairs
      print(deltac,5,2); print(kv(j),2,10)
      newline
      finish
      repeat
!
!      closestream(20)
      selectoutput(23)
!
      for j=1,1,nf cycle
      if ltno(j)>0 then start
!     write in time and valve constant as data pairs
      print(deltac,5,2); print(kv(j),2,10)
      newline
      finish
      repeat
!
!      closestream(23)
!
!*** set values of flst and plst for next time round
flst(zw)=f(zw) for zw=1,1,nf
plst(zy)=p(zy) for zy=1,1,nn
!***
!
      deltaclast=0
      tnext=deltatime
      for tc=1,1,ntstep cycle;! start of time cycle
      deltac=deltac+delta
! see if time value has exceeded tnext
      if nodemax>0 then start
      if (deltac-deltaclast)>tnext then start
        deltaclast=deltac
        tnext=random(seed1,0)*tmax
newline ; printstring("time = ");print(deltac,5,2);printstring("secs")
newline ; printstring("new generated time interval = ");print(tnext,4,2)
        xrandom=random(seed2,0)*xmax
! if xrandom >= 200 then xrandom -> +ve fluctuation in pressure
! if xrandom < 200 then xrandom -> -ve fluctuation in pressure
        xrandom = xrandom - 200
        printstring("  x = ");print(xrandom*1@-5,2,3)
        if nodemax>0 then start
newline; printstring("old pressure was ");print(p(nodemax),7,3)
          p(nodemax)=p(nodemax)+xrandom
newline; printstring("new pressure is ");print(p(nodemax),7,3)
        finish
      finish
```

```
      finish
      for j=1,1,nn cycle
      if ncap(j)>0 then ptp(j)=tp(j) and tp(j)=p(j)
      repeat
!     newline;!printstring("iteration no. ");!write(tc,3)
!     newline;! printstring("time step = ");!print(deltac,5,3)
      tcon=delta
!     if tc=1 then tcon=0
!
for i=1,1,100 cycle ;! ---------------- start iteration ----------------
!
!  set the value of 'pass' for routines 'setupk' and 'flows'
!
pass=1
if tc=1 and i=1 then pass=0
if tc>1 and i=1 then pass=0
!
!
selectoutput(2)
set up k(f,fo,flst,k,kb,kv,p,plst,l,da,denav,temp,vis,ht,ft,rk,c
cv,cd,cl,c2,mu,sfp,ltno,u,d,tlink,pform,ipbr,nf,npump,nfluid,pass,linmeth
selectoutput(2)
!
set up a(k,kb,p,tp,fn,ncap,a,b,denav,ht,qrts,qts,qtcount,u,d,pfix,c
delta,tcon,qlcount,nn,nf,linmeth)
!
pressures(a,b,p,f,fo,nn,nf,0,pset,qset)
   if i>switch then linmeth="newton"
flows(p,kv,k,kb,f,fo,l,da,rk,ft,denav,den,ht,vis,cd,cv,cl,c2,temp,c
qrts,qtcount,qts,ipbr,u,d,tlink,pform,pass,0,nf,nn,npump,nfluid,linmeth)
!
!     check for convergence
fcheck(f,fn,pfix,link,u,cp,nn,ftol,hftot,hfnod,check)
!
if (check = 0 and i>2) or i>100 then start
!newline ;! printstring("check=0")
!newline;! printstring("error = ");! print(hftot,3,5)
!printstring(" at node ");! write(hfnod,3)
!newline
if (check = 0 and i>2) or i>100 then exit
!if check p(p,po,nn) = 0 or i>100  thenexit
finish
!
repeat ;!              ---------------  next iteration ----------------
!
! set up k(f,fo,flst,k,kb,kv,p,plst,l,da,denav,temp,vis,ht,ft,rk,c
!cv,cd,cl,c2,mu,sfp,ltno,u,d,tlink,pform,ipbr,nf,npump,nfluid,pass,linmetl
!
flows(p,kv,k,kb,f,fo,l,da,rk,ft,denav,den,ht,vis,cd,cv,cl,c2,temp,c
qrts,qtcount,qts,ipbr,u,d,tlink,pform,1,0,nf,nn,npump,nfluid,linmeth)
!
!*** set values of flst and plst for next time round
flst(zw)=f(zw) for zw=1,1,nf
plst(zy)=p(zy) for zy=1,1,nn
!***
emas3cputime(time2)
!printstring("cpu secs = ");!print(time2-time,5,3)
!printstring("
!  Used ") ;! write(i,4) ;! printstring(" iterations")
!newline
```

```
!
!-------------------------------------------------------------------------
!read data into file for plotting with "EASYGRAPH"
!
!       closestream(2)
!      only print data for node whose pressure is being controlled
       for j=1,1,nf cycle
       if ltno(j)>0 then start
       selectoutput(22)
!      write in time and pressure as data pairs
       print(deltac,5,2); print(p(ltno(j))*0.00001,4,10)
       newline
       finish
       repeat
!
!       closestream(22)
       selectoutput(21)
!
!      only print data for link through which flow is controlled
       for j=1,1,nf cycle
       if ltno(j)<0 then start
!      write in time and flow as data pairs
       ltno(j)=0-ltno(j)
       print(deltac,5,2); print(f(ltno(j)),4,5)
       ltno(j)=0-ltno(j)
       newline
       finish
       repeat
!
!       closestream(21)
       selectoutput(20)
!
       for j=1,1,nf cycle
       if ltno(j)<0 then start
!      write in time and valve constant as data pairs
       print(deltac,5,2); print(k(j),2,10)
       newline
       finish
       repeat
!
!       closestream(20)
       selectoutput(23)
!
       for j=1,1,nf cycle
       if ltno(j)>0 then start
!      write in time and valve constant as data pairs
       print(deltac,5,2); print(k(j),2,10)
       newline
       finish
       repeat
!
!       closestream(23)
       selectoutput(2)
!-------------------------------------------------------------------------
!
       for zz=1,1,nn cycle
       if ncap(zz)>0 then start
       if mod(ptp(zz)-tp(zz))<0.001 then oflag=0
       if tc=1 then oflag=1
       finish
```

```
      repeat
      if oflag=0 then c
      start
      printstring("
      end of time-step cycle")
      newline
      exit
      finish
!
      repeat; ! repeat for time step cycle
selectinput(0)
emas3prompt("Continue (Y or N) ?"); skipsymbol ; readitem(ans)
         if ans ="n" or ans="N" then c
         printstring("         Finish") and -> printlabel
!
!        set all pressures to original values
         p(ll)=ppi(ll) for ll=1,1,nn
      newline
      emas3prompt("change parameters (Y or N) ?")
      skipsymbol;readitem(ans)
      newline
      if ans="y" or ans="Y" then start
      emas3prompt("link or node parameter?")
      read(kv(l)); read(mu(l)); read(ltno(l)) ; read(sfp(l))
      finish
!
printlabel:
!closestream(2)
selectoutput(ll)
flows(p,kv,k,kb,f,fo,l,da,rk,ft,denav,den,ht,vis,cd,cv,cl,c2,temp,c
qrts,qtcount,qts,ipbr,u,d,tlink,pform,l,l,nf,nn,npump,nfluid,linmeth)
!closestream(ll)
if ans ="n" or ans="N" then stop
!
repeat
!
endofprogram
```

```
!-----------------------------------------------
!
! SET UP K : Routine to get the 'k' values for each link.
!            The 'k' values are obtained from the nonlinear
!            'kv' values. The equation set up for each link
!            is ; Q = K * (P(in) - P(out)) + KB
!
external c
routine set up k(longrealarrayname f,fo,flst,k,kb,kv,p,plst,l,c
da,denav,temp,vis,ht,ft,rk,cv,cd,c1,c2,mu,sfp,integerarrayname ltno,in,c
out,tlink,pform,ipbr,integer nf,npump,nfluid,pass,string(20) linmeth)
!
externalroutinespec ilnpump(longreal p1,p2,c1,c2,den,c
integer pform,n,longrealname q,a,b)
!
externalroutinespec idenst(longrealarrayname cd,longrealname c
dens,longreal press,temp,integer nfluid)
!
externalroutinespec ivisco(longrealarrayname cv,longrealname c
visc,longreal press,temp,integer nfluid)
!
externallongrealfnspec logten(longreal x)
!
! in... kv(),p(),fo(),nf
! out.. k()
!
integer i ,j, vv,ks
longreal flow,dp,pi,re,ff,pav
longrealarray den(1:40)
pi=3.14159
!
! limit sets minimum pressure difference or flow below which
! linearisation is not attempted.
!
constreal limit=0.0001
for i=1,1,nf cycle
      pav=0.5*(p(in(i))+p(out(i)))
!
!     get average density in the pipe
      idenst(cd,denav(i),pav,temp(i),nfluid)
!     get density at either end of the pipe
      idenst(cd,den(in(i)),p(in(i)),temp(i),nfluid)
      idenst(cd,den(out(i)),p(out(i)),temp(i),nfluid)
!
!     get viscosity in the pipe
      ivisco(cv,vis(i),pav,temp(i),nfluid)
      dp=(p(in(i))+9.81*denav(i)*ht(in(i)))-(p(out(i))c
      +9.81*denav(i)*ht(out(i)))
!
      dp=mod(dp)
  if dp<limit then dp=limit
  flow=mod(fo(i))
  if mod(flow)<limit then flow=limit
!
!for tlink=-1, k does not have to be linearised
!
      if tlink(i)=-1 and kv(i)#0 then start
      kb(i)=0
      if ltno(i)>0 then start
```

```
! is upstream or downstream pressure being controlled ?
!
if sfp(i)>0 then start
!
       k(i)=kv(i)+mu(i)*(sfp(i)-plst(ltno(i)))
!
finish else if sfp(i)<0 then start
!
       k(i)=kv(i)+mu(i)*(plst(ltno(i))+sfp(i))
!
finish
!
       if pass=0 then k(i)=kv(i)
       finish else if ltno(i)=0 then start
       k(i)=kv(i)
       kb(i)=0
       if p(in(i))<p(out(i)) then kb(i)=-kb(i)
       finish else if ltno(i)<0 then start
       vv=0-ltno(i)
       k(i)=kv(i)+mu(i)*(sfp(i)-flst(vv))
       kb(i)=0
       if pass=0 then k(i)=kv(i)
       if flst(vv)<0.0000001 then k(i)=kv(i)
       finish
      finish else if tlink(i)=-1 and kv(i)=0 then start
        k(i)=0
        kb(i)=0
      finish
      if tlink(i)=0 thenstart
       if ltno(i)>0 then start
! is upstream or downstream pressure being controlled ?
!
if sfp(i)>0 then start
!
       k(i)=kv(i)+mu(i)*(sfp(i)-plst(ltno(i)))
!
finish else if sfp(i)<0 then start
!
       k(i)=kv(i)+mu(i)*(plst(ltno(i))+sfp(i))
!
finish
!
       if pass=0 then k(i)=kv(i)
       k(i)=k(i)/2/sqrt(dp)
       kb(i)=k(i)*(sqrt(dp)/2)
       if p(in(i))<p(out(i)) then kb(i)=-kb(i)
       finish else if ltno(i)=0 then start
       k(i)=kv(i)/2/sqrt(dp)
       kb(i)=kv(i)*(sqrt(dp)/2)
       if p(in(i))<p(out(i)) then kb(i)=-kb(i)
       finish else if ltno(i)<0 then start
       vv=0-ltno(i)
       k(i)=kv(i)+mu(i)*(sfp(i)-flst(vv))
       if pass=0 then k(i)=kv(i)
       k(i)=k(i)/2/sqrt(dp)
       kb(i)=k(i)*(sqrt(dp)/2)
       if p(in(i))<p(out(i)) then kb(i)=-kb(i)
       finish
      finish
!
```

```
      if tlink(i)=1 then start
       if pass=0 then start
!get laminar 'k'
      kv(i)=pi/(2*vis(i))*denav(i)*da(i)**4/((l(i)+50*ft(i)*da(i))*64)
      k(i)=kv(i)
      kb(i)=0
      finish else start
!     get Reynolds number
      re=(flow*4)/(pi*da(i)*vis(i))
!
!     laminar or turbulent flow?
      if re<2500 thenstart
!     get friction factor for laminar flow
      ff=64/re
      finish else start
!     get friction factor for turbulent flow
!     use Chen explicit equation
      ff=(rk(i)**1.1098)/2.8257+(5.8506/(re**0.8981))
ff=-2*logten((rk(i)/3.7065)-(5.0452/re)*logten(ff))
ff=(1/ff)**2
      finish
!     get k(i)
kv(i)=2*mod(log(den(in(i))/den(out(i))))
kv(i)=1/(ff*l(i)/da(i)+ft(i)+kv(i))
!!!
if denav(i)<0 then start
  newline;printstring("up");write(in(i),4);print(p(in(i)),7,3)
  newline;printstring("down");write(out(i),4);print(p(out(i)),7,3)
finish
!!!
kv(i)=(pi/2)*da(i)**2*sqrt(denav(i)/2)*sqrt(kv(i))
  if linmeth="newton" then start
 k(i)=kv(i)/(2*sqrt(dp))
kb(i)=mod((kv(i)/2)*sqrt(dp))
if p(out(i))>p(in(i)) then kb(i)=0-kb(i)
      finish
if linmeth="hutchison" then k(i)=(kv(i)**2)/flow and kb(i)=0
      finish
      finish
!
!     get the pump number corresponding to this link number
      if tlink(i)=2 then start
      for j=1,1,npump cycle
      if ipbr(j)=i then start
      ilnpump(p(in(i)),p(out(i)),c1(j),c2(j),denav(i),c
      pform(j),pass,f(i),k(i),kb(i))
      finish
      repeat
      finish
repeat
!
end
endoffile
```

```
externalroutine set up a(longrealarrayname k,kb,p,tp,fn,ncap,a,b,c
den,ht,realarrayname qrterms,integerarrayname qterms,qtctr,u,d,pfix,c
longreal delta,tcon,integer qlctr,nn,nf,string(20) linmeth)
!
!  Create the a matrix of linearised equations and its vector b
!  from the linearised flow/pressure relations involving k.
!
integer i,j,f1,f2,fd1,fd2,vv
integer fflag,q2def,jj,mm,uu
!
for i=1,1,nn cycle
  a(i,j)=0 for j=1,1,nn
  b(i)=fn(i)
repeat
!
for i=1,1,nf cycle
  fflag=0
  if qlctr>0 then start
    for jj=1,1,qlctr cycle
      if i=qtctr(jj) then fflag=jj
    repeat
  finish
  if fflag=0 then start
    f1=u(i) ; f2=d(i)
    a(f1,f1)=a(f1,f1)-k(i)
    a(f2,f2)=a(f2,f2)-k(i)
    a(f1,f2)=a(f1,f2)+k(i)
    a(f2,f1)=a(f2,f1)+k(i)
        b(f1)=b(f1)+kb(i)
        b(f2)=b(f2)-kb(i)
        b(f1)=b(f1)-(k(i)*den(i)*9.81*(ht(f2)-ht(f1)))
        b(f2)=b(f2)-(k(i)*den(i)*9.81*(ht(f1)-ht(f2)))
  finish else start
!look at the related flows
!*************************************************************************
    for mm=1,1,5 cycle
      if qterms(fflag,mm)>0 then start
!examine whether these flows themselves are defined in terms
!of other flows
        q2def=0
        for vv=1,1,qlctr cycle
          if qterms(fflag,mm)=qtctr(vv) then q2def=vv
        repeat
        if q2def=0 then start
          if k(qterms(fflag,mm))>0 then start
            f1=u(qtctr(fflag)) ; f2=d(qtctr(fflag))
            fd1=u(qterms(fflag,mm)) ; fd2=d(qterms(fflag,mm))
            a(f1,fd1)=a(f1,fd1)-(-1)*qrterms(fflag,mm)*k(qterms(fflag,mm))
            a(f1,fd2)=a(f1,fd2)+(-1)*qrterms(fflag,mm)*k(qterms(fflag,mm))
            a(f2,fd2)=a(f2,fd2)-(-1)*qrterms(fflag,mm)*k(qterms(fflag,mm))
            a(f2,fd1)=a(f2,fd1)+(-1)*qrterms(fflag,mm)*k(qterms(fflag,mm))
            b(f1)=b(f1)+(-1)*qrterms(fflag,mm)*kb(qterms(fflag,mm))
            b(f2)=b(f2)-(-1)*qrterms(fflag,mm)*kb(qterms(fflag,mm))
          finish
        finish else start
          for uu=1,1,5 cycle
            if qterms(vv,uu)>0 then start
              if k(qterms(vv,uu))>0 then start
                f1=u(qtctr(vv)) ; f2=d(qtctr(vv))
```

```
                fd1=u(qterms(vv,uu)) ; fd2=d(qterms(vv,uu))
        a(f1,fd1)=a(f1,fd1)-qrterms(fflag,mm)*qrterms(vv,uu)*k(qterms(vv,u
        a(f1,fd2)=a(f1,fd2)+qrterms(fflag,mm)*qrterms(vv,uu)*k(qterms(vv,u
        a(f2,fd2)=a(f2,fd2)-qrterms(fflag,mm)*qrterms(vv,uu)*k(qterms(vv,u
        a(f2,fd1)=a(f2,fd1)+qrterms(fflag,mm)*qrterms(vv,uu)*k(qterms(vv,u
        b(f1)=b(f1)+qrterms(fflag,mm)*qrterms(vv,uu)*kb(qterms(vv,uu))
        b(f2)=b(f2)-qrterms(fflag,mm)*qrterms(vv,uu)*kb(qterms(vv,uu))
              finish
            finish
          repeat
        finish
      finish
    repeat
!*************************************************************************
  finish
repeat
!
for i=1,1,nn cycle
  if pfix(i)=1 thenstart
    a(i,j)=0 for j=1,1,nn
    b(i)=p(i) ; a(i,i)=1
  finish
  if ncap(i)#0 and tcon>0 then start
    a(i,i)=a(i,i) - ncap(i)/delta
!
    b(i)=b(i)-tp(i)*ncap(i)/delta
  finish
repeat
end
endoffile
```

```
external routine flows(longrealarrayname p,kv,k,kb,f,fo,l,da,rk,ft,c
denav,den,ht,vis,cd,cv,cl,c2,temp,realarrayname qrterms,c
integerarrayname qtctr,qterms,ipbr,u,d,tlink,pform,integer c
pass,printit,nf,nn,npump,nfluid,string(20) linmeth)
!
externalroutinespec ilnpump(longreal p1,p2,c1,c2,den,c
integer pform,n,longrealname q,a,b)
externalroutinespec idenst(longrealarrayname cd,longrealname c
dens,longreal press,temp,integer nfluid)
externalroutinespec ivisco(longrealarrayname cv,longrealname c
visc,longreal press,temp,integer nfluid)
! calculate flows in branches once pressures are known
!
! in.. p() and pointers u(),d(), kv(),k(),den(),
! denav(),l(),da(),rk(),ft()
! and scalars pass, nf, printit
!
! out... f() and fo() when pass=0
!
integerarray iflow(1:10)
integer i,j,ii,jj,ifctr,lkflo,lkrflo,uu,vv,yy,zz,nflo,qflo
longreal flow,dp,s,ff,fp,re,pi,pav,kreal
!
owninteger op=0
op=printit
pi=3.14159
!
if op=1 then printstring("
node      pressure
")
if op=1 then start
for i=1,1,nn cycle
!
write(i,5)
print(p(i),4,4)
newline
repeat
finish
if op=1 then printstring("
Branch    from    to        flow            k               kb

")
ifctr=0
iflow(i)=0 for i=1,1,10
qflo=0
for i=1,1,nf cycle
!
      pav=0.5*(p(u(i))+p(d(i)))
!     get average density in the pipe
      idenst(cd,denav(i),pav,temp(i),nfluid)
!     get viscosity in the pipe
      ivisco(cv,vis(i),pav,temp(i),nfluid)
!
dp=(p(u(i))+9.81*denav(i)*ht(u(i)))-(p(d(i))+9.81*denav(i)*ht(d(i)))
if dp>=0 then s=1 else s=-1
dp=mod(dp)
!
!pipe data or valve const only ?
!
```

```
if tlink(i)=-99 then start
!
!flow is specified in terms of flow in another link
!
  for ii=1,1,10 cycle
    if qtctr(ii)=i then start
      ifctr=ifctr+1
      iflow(ifctr)=ii
    finish
  repeat
finish
!
if tlink(i)=1 thenstart
!pipe data supplied
!
!is it first time round ?
      if pass=0 thenstart
!     calculate laminar flow in each pipe
flow=(pi/(2*vis(i)))*denav(i)*da(i)**4/((l(i)+50*ft(i)*da(i))*64)*dp*s
      finish elsestart
!     calculate Reynolds number
      re=mod(f(i))*4/(pi*da(i)*vis(i))
!     laminar or turbulent flow ?
         if re<2500 thenstart
!        get laminar flow
flow=(pi/(2*vis(i)))*denav(i)*da(i)**4/((l(i)+50*ft(i)*da(i))*64)*dp*s
         finish elsestart
!        get flow
flow=kv(i)*sqrt(dp)*s
!if linmeth="newton" then flow=k(i)*dp*s + kb(i)
!if linmeth="hutchison" then flow=k(i)*dp*s
         finish
      finish
finish else if tlink(i)=0 thenstart
! only value for kv supplied
if pass=0 then flow=kv(i)*s*sqrt(dp)
if linmeth="hutchison" and pass=1 then flow=k(i)*dp*s + kb(i)
if linmeth="newton" and pass=1 then flow=k(i)*dp*s + kb(i)
      finish else if tlink(i)=-1 then start
      if pass=0 then flow=kv(i)*s*dp else c
      flow=k(i)*s*dp;{linear kv supplied}
      finish else if tlink(i)=2 then start
!     get the pump number corresponding to this link number
      for j=1,1,npump cycle
      if ipbr(j)=i then start
      ilnpump(p(u(i)),p(d(i)),c1(j),c2(j),denav(i),c
      pform(j),pass,flow,k(i),kb(i))
      finish
      repeat
      finish
if pass=0 then start
  if tlink(i)=-99 then flow=0
  if tlink(i)#-99 then fo(i)=mod(flow) else fo(i)=0
finish else start
  if linmeth="hutchison" then c
fo(i)=0.5*mod(fo(i)+mod(flow)) else fo(i)=flow
finish
f(i)=flow
repeat
!examine flows in links where tlink = -99 (signifies flow is
```

```
!specified in terms of other network flows)
!
if ifctr > 0 then start
  for ii=1,1,ifctr cycle
!  set flag to indicate that recursive flow definition has not (yet) occu
!
    nflo=1
    lkflo=iflow(ii)
    qflo=qtctr(lkflo)
! initialise value of flow in this link
    f(qflo)=0
    for zz=1,1,5 cycle
      lkrflo=0; yy=0; vv=0
      if qterms(lkflo,zz)>0 then lkrflo=qterms(lkflo,zz) and yy=zz
! is related flow expressed in terms of other flows ?
      if lkrflo>0 then start
        for uu=1,1,10 cycle
          if lkrflo=qtctr(uu) then vv=uu and nflo=0
        repeat
        if vv>0 then start
          for uu=1,1,5 cycle
            if qterms(vv,uu)>0 then start
              f(qflo)=f(qflo)+qrterms(lkflo,yy)*qrterms(vv,uu)*f(qterms(v
            finish
          repeat
        finish else start
          f(qflo)=-(1)*qrterms(lkflo,yy)*f(lkrflo)
        finish
      finish
    repeat
  repeat
finish
!
for i=1,1,nf cycle
  if op=1 then start
    write(i,5); write(u(i),7); write(d(i),5)
    print(f(i),8,9)
    print(k(i),5,9) and print(kb(i),6,6)
    newlines(1)
  finish
repeat
!
end
endoffile
```

V. Data Sets, Results and Diagrams for EQPARSE and DYNET problems.

FIG 5.1

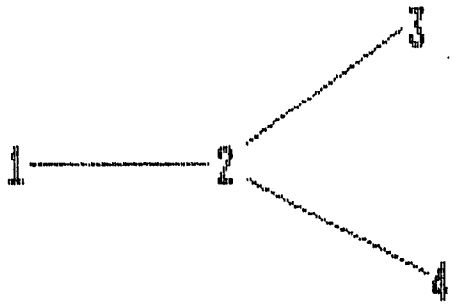


FIG 5.2

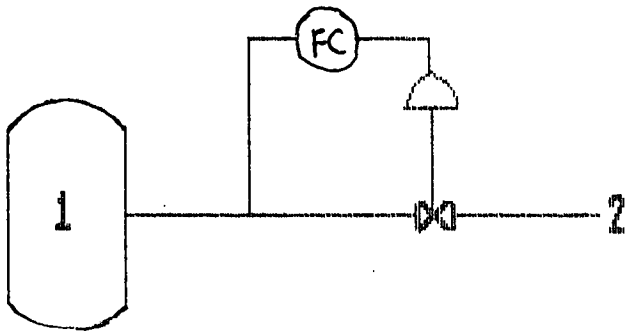
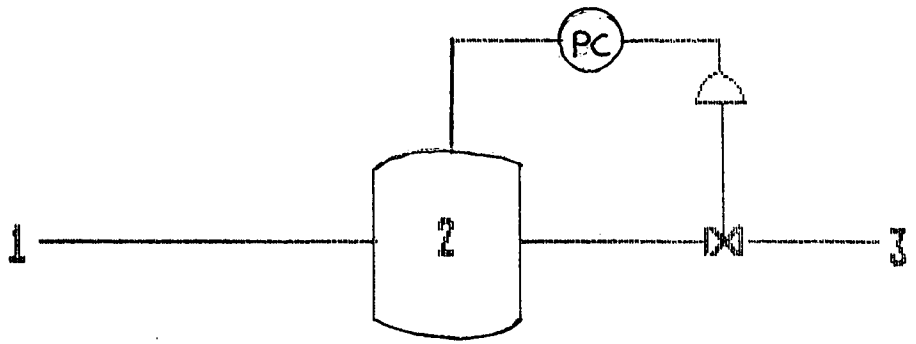


FIG 5.3



Network 5.1

```
1   2   -99   200
2   3   -99   200
2   4   -99   200
-1
-1
0
10   200   1000   1
20   300   1000   1
30   400   1000   1
1
0
P(1)=10
P(1)=P(3)+1
0.0003(P(1)-P(2))=Q(1,2)
0.0003(P(2)-P(3))=Q(2,3)
0.0003(P(2)-P(4))=Q(2,4)
Q(2,4)=15
E
```

Network 5.2

```
1   2   1   12   30   0.003   2   200
2   3   1   12   30   0.004   1   200
2   4 -99                        200
-1
-1
0
10   200   1000   1
20   300   1000   1
30   400   1000   1
1
0
P(1)=3
P(1)=P(3)+1
0.0003(P(2)-P(4))=Q(2,4)
Q(2,4)=0.009
E
```

Results for Network 5.1

Branch	from	to	P in (bar)	P out (bar)	Flow (kg/s)
1	1	2	10.0	9.25	22.5
2	2	3	9.25	9.0	7.5
3	2	4	9.25	8.75	15.0

Results for Network 5.2

Branch	from	to	P in (bar)	P out (bar)	Flow (kg/s)
1	1	2	3.0	2.4938	1.9658
2	2	3	2.4938	2.0	1.9568
3	2	4	2.4938	2.4935	0.009

Network 5.3

```
1   2   -1   0.5E-5   0.1E-6   -1   5   25
-1
1   1   3   20   0   20   25
1   2   1    0   0    1   30
-1
0
10    27   11      0.018
10   127  8.33     0.018
20   377   10      0.018
-1
E
```

Network 5.4

```
1   2   0   0.005     0        0      0   30
2   3   0   0.005   0.5E-8     2    -12   30
-1
1   1   1   0   0   20   30
1   2   3  20   0    0   30
1   3   1   0   0    1   30
-1
0
29.98   420    12.95    0.0188
25.84   460    10.27    0.0198
30.81   540    11.34    0.018
-1
E
```

Network 5.5

```
1   2   -1   0.001    1E-7    4    7   25
2   3    1     0      100     0    0   25
3   4   -1   1E-6     0       0    0   25
4   2   -1   1E-6     1E-6   -3  0.6   25
4   5   -1   1E-6     0       0    0   25
-1
1   1   1   0   0   1   25
1   2   3   5   0   0   25
1   3   3   0   0   0   25
1   4   3   1   0   0   25
1   5   1   0   0   6   25
-1
1
3
2   3   1.2E6   0   8E5   1   1000   2
10   27   11   0.018
10  127  8.33  0.018
20  377   10   0.018
-1
E
```

Results for Network 5.4

Branch	from	to	P in (bar)	P out (bar)	Flow(kg/s)
1	1	2	20.00000	14.59973	3.4763
2	2	3	14.59973	1.00000	3.4763

Results for Network 5.5

Branch	from	to	P in (bar)	P out (bar)	Flow (kg/s)
1	1	2	1.00000	0.99997	0.0251
2	2	3	0.99997	11.92359	0.5673
3	3	4	11.92359	6.25054	0.5673
4	4	2	6.25054	0.99997	0.5422
5	4	5	6.25054	6.00000	0.0251

Fig 5.4 : Simple Flow Control

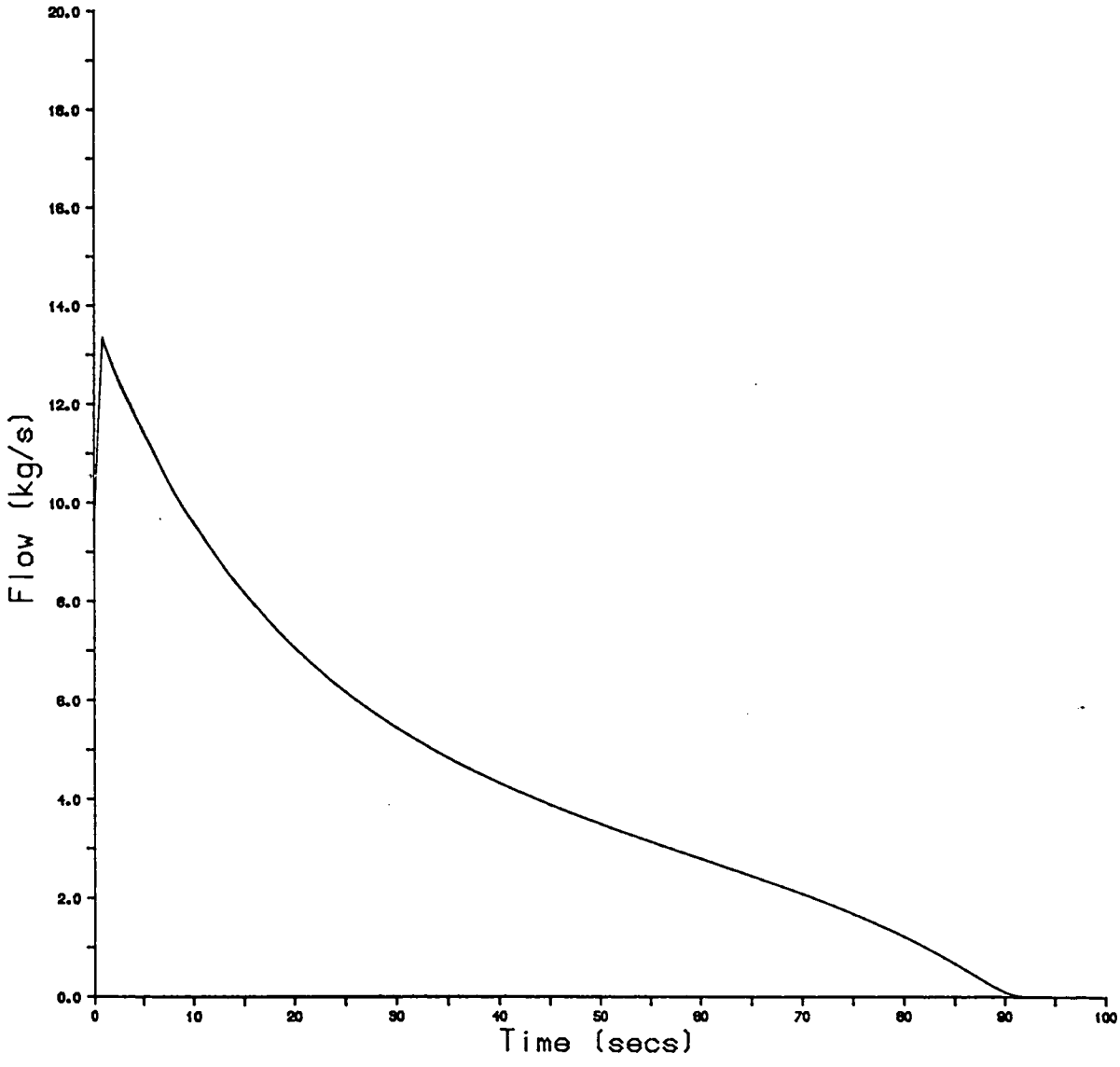


Fig 5.5 : Simple Pressure Control

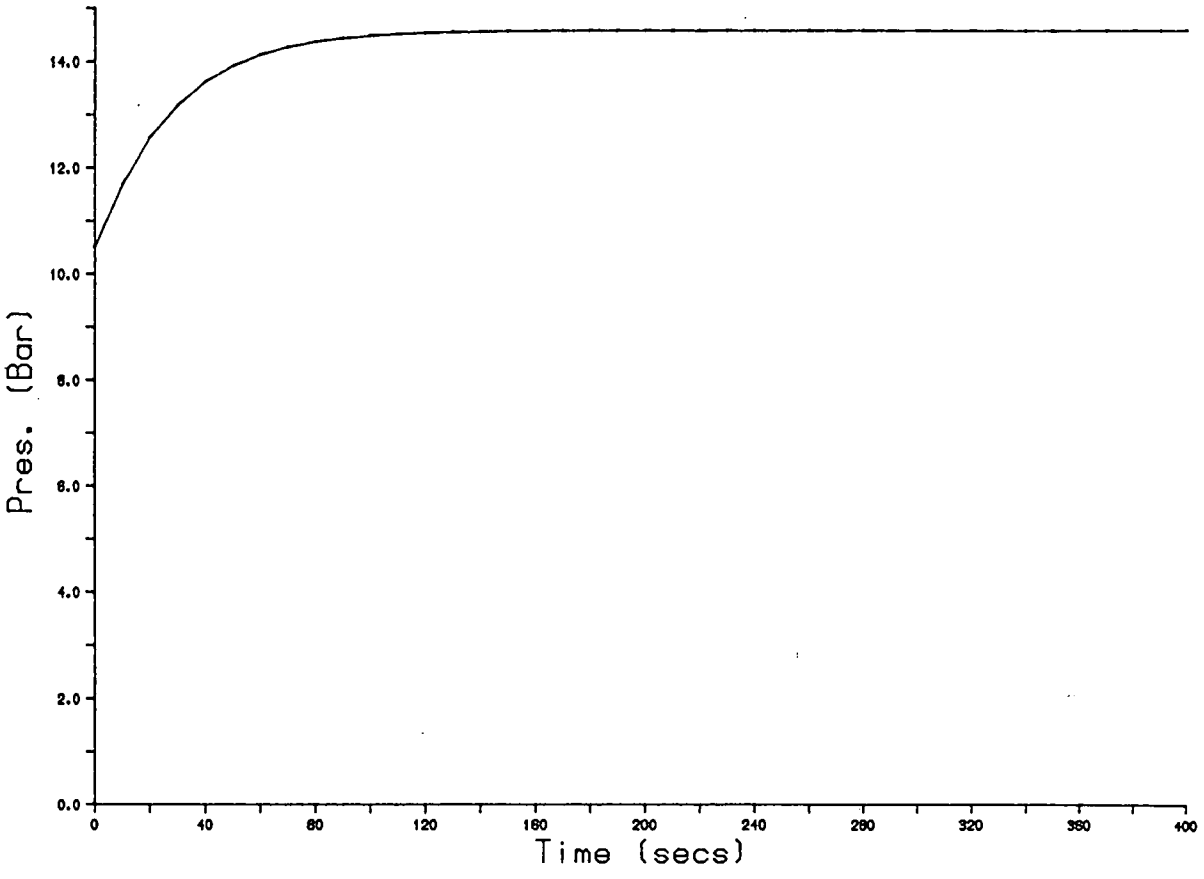


FIG 5.6

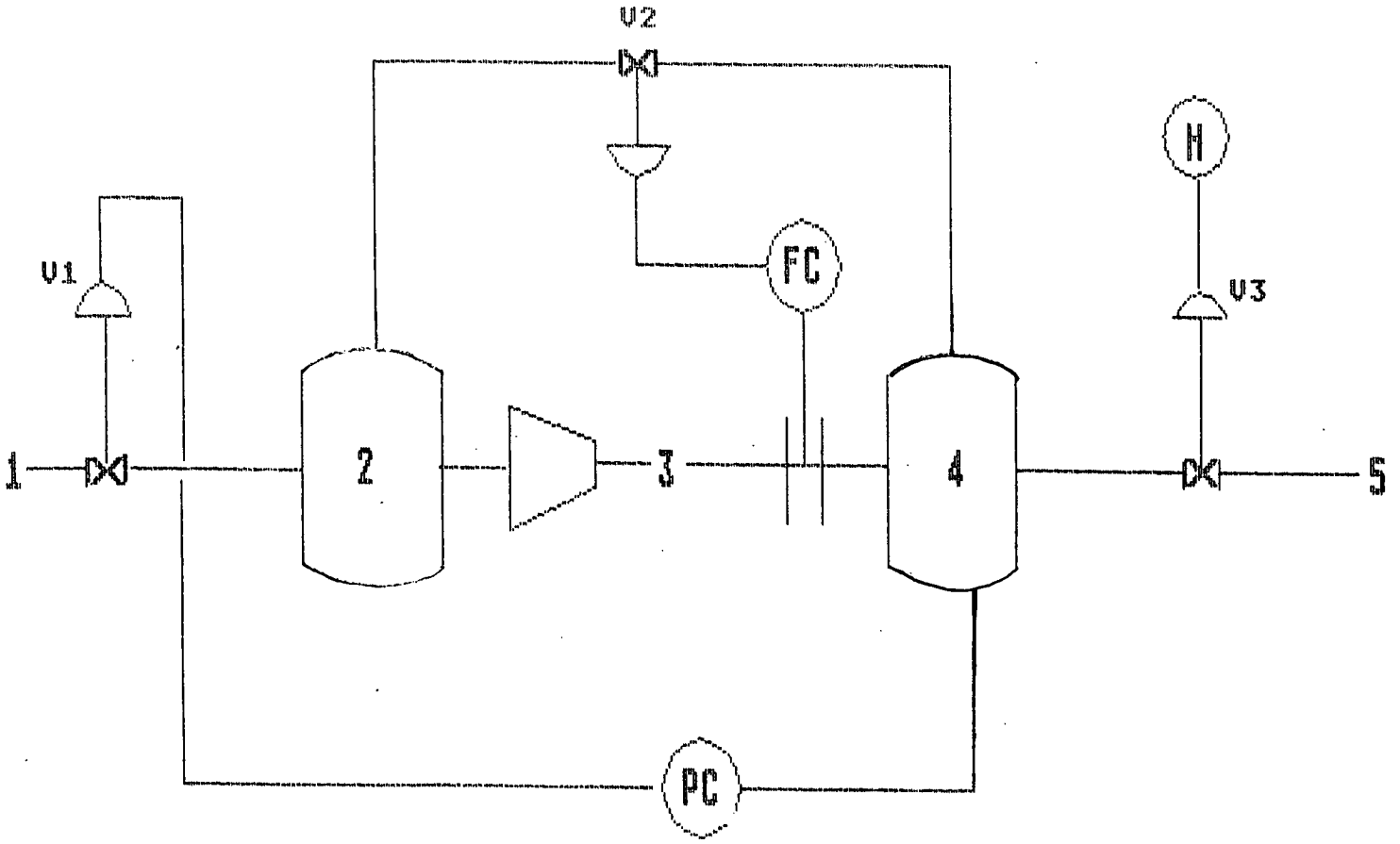


Fig. 5.7 Compressor Flow

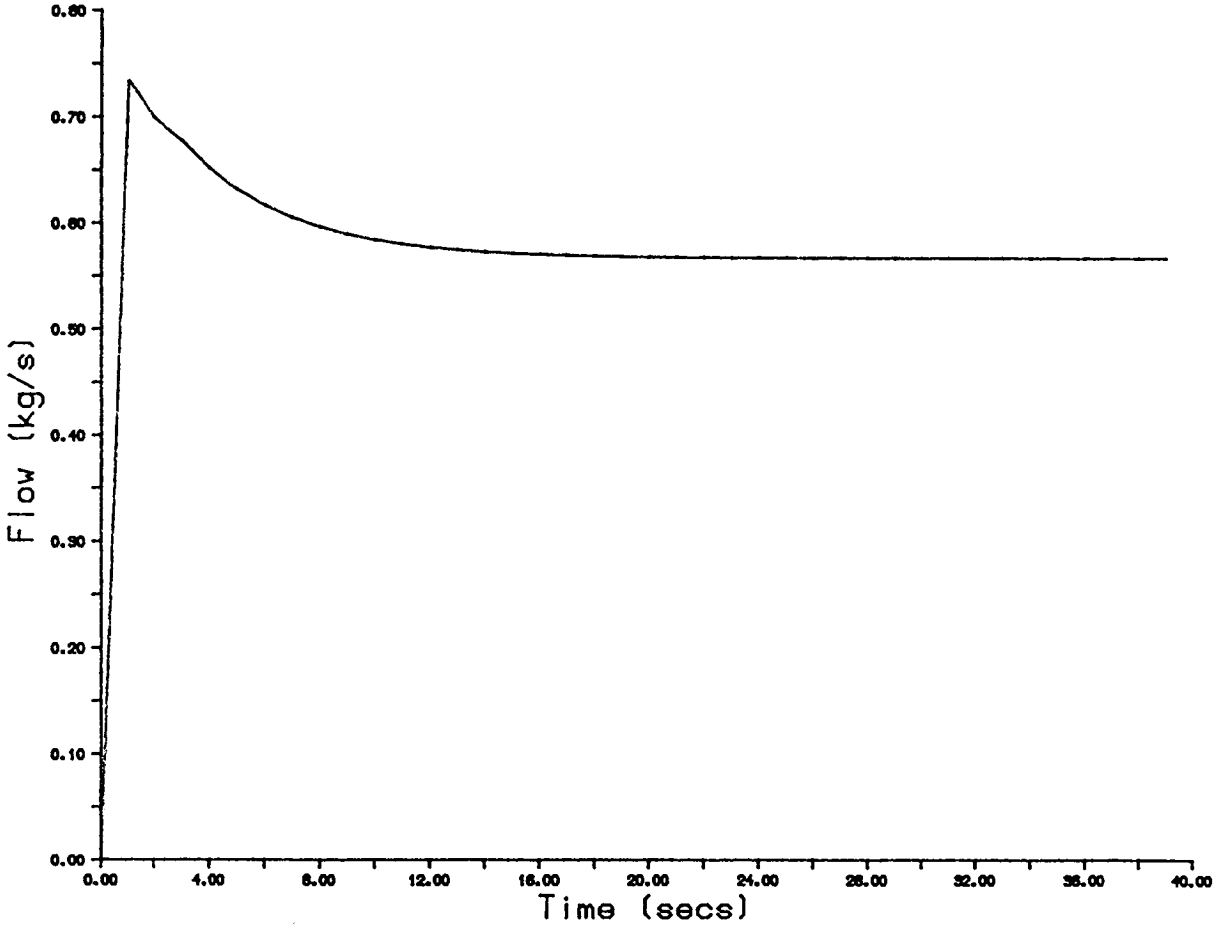
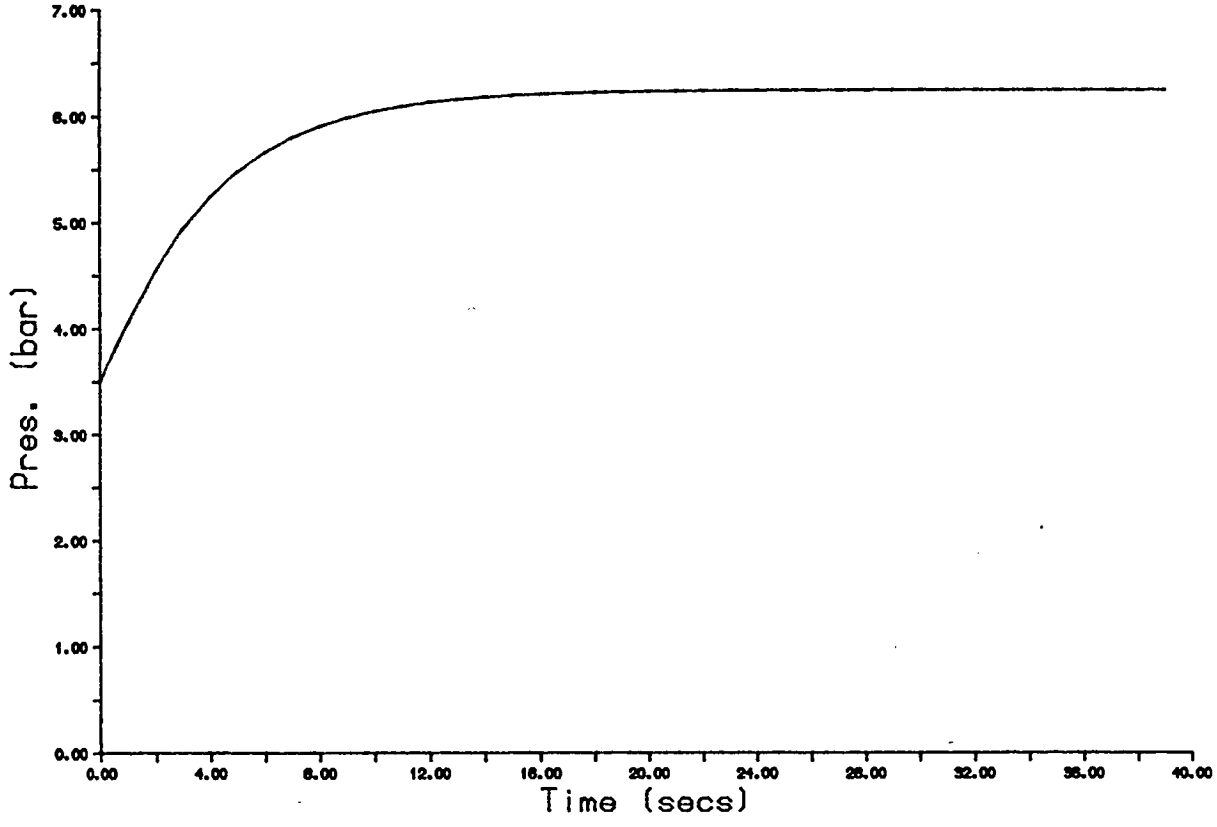


Fig. 5.8 Compressor Pres.



VI. Data Sets, Results and Diagrams for HF Kiln Network

This appendix relates to Chapter 6 and includes the following:

1. Data set for HF kiln network analysed using **FLONET**
2. **FLONET** results
3. Data set 1 for HF kiln network analysed using **EQNET**
4. Graph of max. nodal flow residual vs. iteration number for steady-state network
5. **EQNET** results for data set 1
6. Data set 2 for HF kiln network analysed using **EQNET**
7. **EQNET** results for data set 2
8. Diagram for 'dynamic' HF network
9. Data set for HF network analysed using **DYNET**
10. Graph of pressure vs time at node 29 in HF network

CASE 5

DATA AND RESULTS UNITS -

MASS FLOWRATES: KG / S

PRESSURES: BARS ABS

DENSITY: KG / CU.M

VISCOSITY: CENTIPOISE

PIPE BORE: MILLIMETRES

PIPE LENGTH AND NODE HEIGHT: METRES

TEMPERATURE: CELCIUS

MEAN FLOW VEL: M / S

PIPING DETAILS DATA -

NUMBER OF PIPES = 38

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS	FIT. LOSS COEFF.	MEAN TEMP.
1	2	2.00	2800.000	0.25000	0.000	400.0
2	3	3.00	2300.000	0.25000	0.500	400.0
3	29	2.00	2500.000	0.25000	2.000	400.0
29	4	8.00	2500.000	0.25000	2.000	530.0
4	5	10.00	2500.000	0.25000	1.100	530.0
5	9	18.00	1000.000	0.25000	0.000	530.0
5	8	2.00	2500.000	0.25000	0.000	530.0

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS	FIT. LOSS COEFF.	MEAN TEMP.
8	7	24.00	1800.000	0.25000	0.000	530.0
7	6	10.00	1300.000	0.25000	0.000	530.0
9	13	4.00	1000.000	0.25000	3.250	530.0
13	17	5.00	1100.000	0.25000	0.000	445.0
17	21	18.00	1300.000	0.25000	6.500	360.0
8	12	4.00	1500.000	0.25000	3.250	530.0
12	16	17.00	1600.000	0.25000	0.000	455.0
16	20	4.00	1500.000	0.25000	3.250	380.0
7	11	4.00	1500.000	0.25000	3.250	530.0
11	15	17.00	1600.000	0.25000	0.000	480.0
15	19	4.00	1500.000	0.25000	3.250	430.0
6	10	18.00	1300.000	0.25000	3.250	530.0
10	14	8.00	1100.000	0.25000	0.000	485.0
14	18	4.00	1000.000	0.25000	3.250	460.0
28	31	0.00	750.000	0.00000	0.000	275.0
31	29	23.00	750.000	0.25000	6.400	275.0
2	24	14.00	750.000	0.25000	2.200	400.0
24	25	3.00	750.000	0.25000	0.000	250.0
25	26	5.00	750.000	0.25000	1.100	100.0
26	27	18.00	900.000	0.25000	1.100	100.0
11	14	1.00	300.000	0.25000	1.500	495.0
12	15	1.00	400.000	0.25000	1.500	480.0
13	16	1.00	300.000	0.25000	1.500	455.0
21	20	8.00	1300.000	0.25000	0.000	360.0
20	22	9.00	2500.000	0.25000	0.000	370.0

NODE LABELS XXXX --> XXXX		PIPE LENGTH	PIPE BORE	INSIDE WALL ROUGHNESS	FIT. LOSS COEFF.	MEAN TEMP.
18	19	19.00	1000.000	0.25000	0.000	460.0
19	22	17.00	1800.000	0.25000	0.000	440.0
22	30	10.00	2500.000	0.25000	1.100	400.0
30	23	1.00	2500.000	0.25000	1.100	400.0
23	32	0.00	2500.000	0.00000	0.000	400.0
32	1	1.00	2500.000	0.25000	0.000	400.0

SPECIFIED FLOW AND PRESSURE CONDITIONS -

NUMBER OF CONDITIONS = 2

NODE LABEL	FLOW INTO NODE	PRESSURE	NODE HEIGHT ABOVE STANDARD LEVEL
28	0.000	1.0010	0.0
27	0.000	1.0010	0.0

PUMP CHARACTERISTIC DATA -

NUMBER OF PUMPS = 2

PUMP 1 PIPE 23 TO 32

HEAD	FLOW
477.00	88.8900
500.00	0.0000

PUMP 2 PIPE 28 TO 31

HEAD	FLOW
477.00	6.7000
500.00	0.0000

FLUID PROPERTIES DATA -

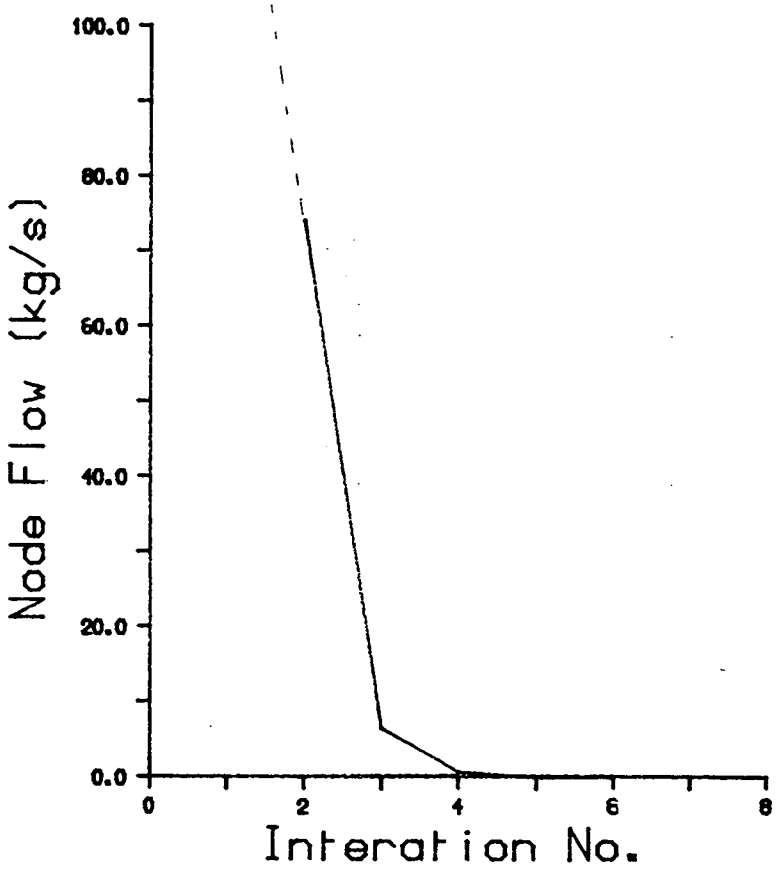
TYPE OF FLUID : GAS
RATIO OF SPECIFIC HEATS = 1.403

PRESSURE	TEMPERATURE	DENSITY	VISCOSITY
1.000	20.0	1.2050	0.18000E-01
1.010	400.0	0.52420	0.33000E-01
1.000	550.0	0.42860	0.37000E-01

Case 5 : FLONET Results for Steady-State HF3 Kiln Network

NODE LABELS XXX --> XXX		NODE PRESSURES (BAR)		FLOW (KG / S)	VELOCITY (M / S)	REYNOLDS NUMBER
1	2	1.00192	1.00188	20.1013	9.2981	337203
2	3	1.00188	1.00174	18.6729	8.6380	313241
3	29	1.00174	1.00144	18.6729	7.3127	288181
29	4	1.00144	1.00094	20.1013	9.3224	276574
4	5	1.00094	1.00060	20.1013	9.3260	276574
5	9	1.00060	1.00003	3.0515	8.8522	104965
5	8	1.00060	1.00058	17.0498	7.9116	234588
8	7	1.00058	1.00023	8.9538	8.0161	171105
7	6	1.00023	1.00017	2.7035	4.6411	71533
9	13	1.00003	0.99935	3.0515	8.8574	104965
13	17	0.99935	0.99925	3.1464	6.7854	105763
17	21	0.99925	0.99879	3.1464	4.3071	97271
8	12	1.00058	0.99968	8.0960	10.4399	185664
12	16	0.99968	0.99939	7.7167	7.9686	176730
16	20	0.99939	0.99873	7.6218	8.0738	200013
7	11	1.00023	0.99970	6.2504	8.0612	143331
11	15	0.99970	0.99950	6.3800	6.8039	142949
15	19	0.99950	0.99894	6.7593	7.6831	168938
6	10	1.00017	0.99989	2.7035	4.6418	71533
10	14	0.99989	0.99978	2.7035	6.1371	87731
14	18	0.99978	0.99933	2.5739	6.8487	93897
28	31	1.00100	1.00243	1.4284	5.0999	84439
31	29	1.00243	1.00144	1.4284	5.0989	84439
2	24	1.00188	1.00132	1.4284	6.2156	73485
24	25	1.00132	1.00126	1.4284	4.8757	87225
25	26	1.00126	1.00113	1.4284	3.4781	111542
26	27	1.00113	1.00100	1.4284	2.4156	92952
11	14	0.99970	0.99978	-0.1296	-4.0064	15294
12	15	0.99968	0.99949	0.3793	6.4719	33993
13	16	0.99935	0.99939	-0.0949	-2.7865	11586
21	20	0.99879	0.99873	3.1464	4.3081	97271
20	22	0.99873	0.99871	10.7682	4.0474	171302
18	19	0.99933	0.99894	2.5739	6.8514	93897
19	22	0.99894	0.99871	9.3331	7.4700	192600
22	30	0.99871	0.99841	20.1013	7.8941	310227
30	23	0.99841	0.99822	20.1013	7.8958	310227
23	32	0.99822	1.00193	20.1013	7.8831	310227
32	1	1.00193	1.00192	20.1013	7.8687	310227

Case 5: Kiln Network



Network 5a (HF3 network)

```
 1   2   1   2   2300   .25     0   400
 2   3   1   3   2300   .25    .5   400
 3  29   1   2   2500   .25     2   400
29   4   1   8   2500   .25     2   530
 4   5   1  10   2500   .25   1.1   530
 5   9   1  18   1000   .25     0   530
 5   8   1   2   2500   .25     0   530
 8   7   1  24   1800   .25     0   530
 7   6   1  10   1300   .25     0   530
 9  13   1   4   1000   .25  3.25   530
13  17   1   5   1100   .25     0   445
17  21   1  18   1300   .25   6.5   360
 8  12   1   4   1500   .25  3.25   530
12  16   1  17   1600   .25     0   455
16  20   1   4   1500   .25  3.25   380
 7  11   1   4   1500   .25  3.25   530
11  15   1  17   1600   .25     0   480
15  19   1   4   1500   .25  3.25   430
 6  10   1  18   1300   .25  3.25   530
10  14   1   8   1100   .25     0   485
14  18   1   4   1000   .25  3.25   460
28  31   1   0    750     0     0   275
31  29   1  23    750   .25   6.4   275
 2  24 -99                         400
24  25   1   3    750   .25     0   250
25  26   1   5    750   .25   1.1   100
26  27 -99                         100
11  14   1   1    300   .25   1.5   495
12  15   1   1    400   .25   1.5   480
13  16   1   1    300   .25   1.5   455
21  20   1   8   1300   .25     0   360
20  22   1   9   2500   .25     0   370
18  19   1  19   1000   .25     0   460
19  22   1  17   1800   .25     0   440
22  30   1  10   2500   .25   1.1   400
30  23   1   1   2500   .25   1.1   400
23  32   1   0   2500     0     0   400
32   1   1   1   2500   .25     0   400
-1
1   28   1   0   0   1.001   100
-1
2
2 2
23   32   477   88.89   500   0
28   31   477    6.7    500   0
   1   20    1.205    0.018
1.01  400    0.5242   0.033
   1  550    0.4286   0.037
0
0.03(P(2)-P(24))=Q(2,24)
0.12(P(26)-P(27))=Q(26,27)
Q(26,27)=1.5
E
```

Results for Network 5a

Branch	from	to	P in (bar)	P out (bar)	Flow (kg/s)
1	1	2	1.01095	1.01094	13.0182
2	2	3	1.01094	1.01089	11.5182
3	3	29	1.01089	1.01079	11.5182
4	29	4	1.01079	1.01056	13.0182
5	4	5	1.01056	1.01040	13.0182
6	5	9	1.01040	1.01014	1.9585
7	5	8	1.01040	1.01039	11.0600
8	8	7	1.01039	1.01023	5.7853
9	7	6	1.01023	1.01020	1.7529
10	9	13	1.01014	1.00983	1.9585
11	13	17	1.00983	1.00979	2.0282
12	17	21	1.00979	1.00963	2.0282
13	8	12	1.01039	1.00997	5.2743
14	12	16	1.00997	1.00985	5.0425
15	16	20	1.00985	1.00961	4.9729
16	7	11	1.01023	1.00999	4.0325
17	11	15	1.00999	1.00990	4.1230
18	15	19	1.00990	1.00969	4.3548
19	6	10	1.01020	1.01007	1.7529
20	10	14	1.01007	1.01002	1.7529
21	14	18	1.01002	1.00985	1.6624
22	28	31	1.00100	1.01169	1.5000
23	31	29	1.01169	1.01079	1.5000
24	2	24	1.01095	1.01044	1.5000
25	24	25	1.01044	1.01039	1.5000
26	25	26	1.01039	1.01026	1.5000
27	26	27	1.01026	1.01013	1.5000
28	11	14	1.00999	1.01002	-0.0905
29	12	15	1.00997	1.00990	0.2318
30	13	16	1.00983	1.00985	-0.0697
31	21	20	1.00963	1.00961	2.0282
32	20	22	1.00961	1.00960	7.0010
33	18	19	1.00985	1.00969	1.6624
34	19	22	1.00969	1.00960	6.0171
35	22	30	1.00960	1.00949	13.0182
36	30	23	1.00949	1.00942	13.0182
37	23	32	1.00942	1.01096	13.0182
38	32	1	1.01096	1.01095	13.0182

Network 5b

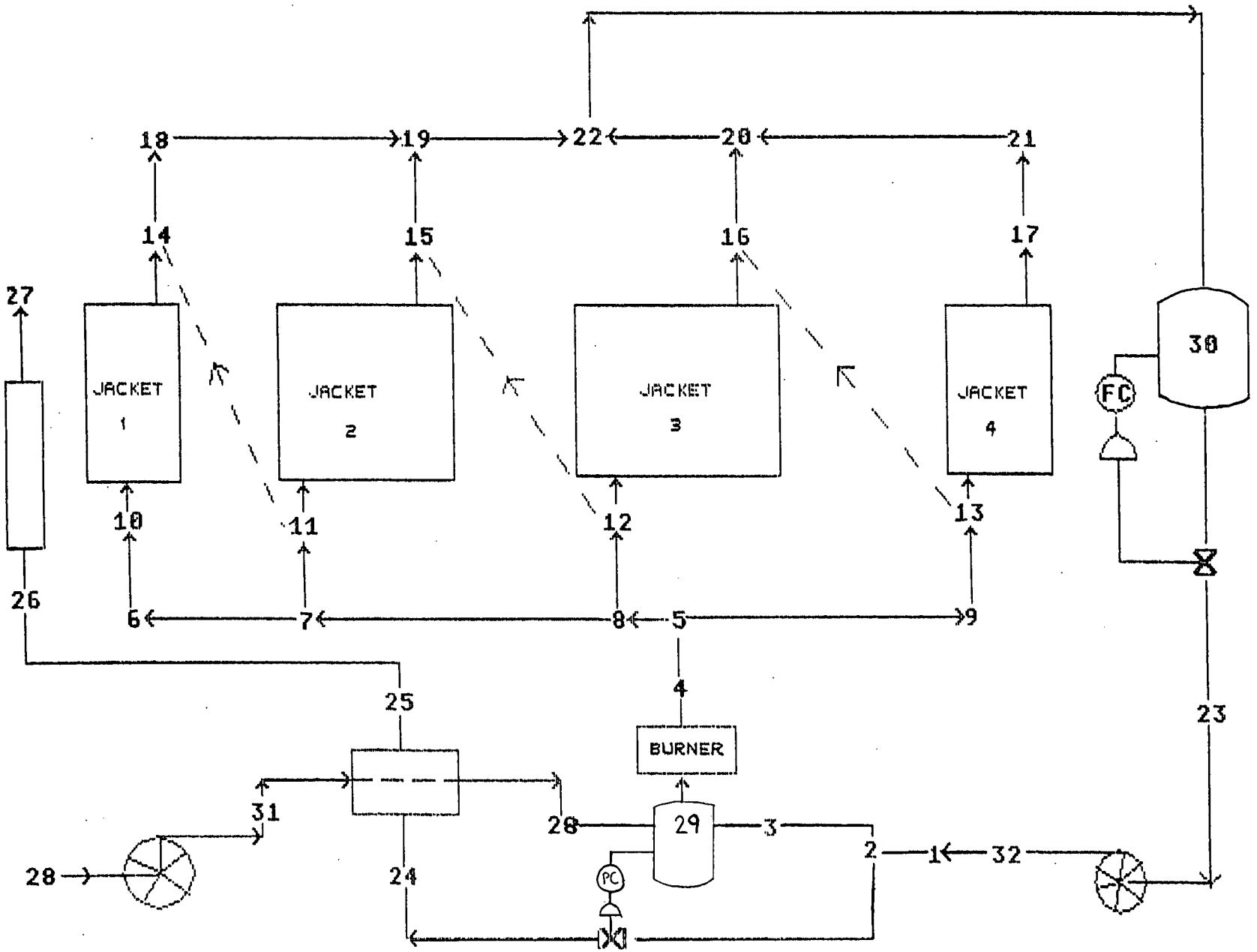
```
 1    2    1    2    2300    .25    0    400
 2    3    1    3    2300    .25   .5    400
 3   29    1    2    2500    .25    2    400
29    4    1    8    2500    .25    2    530
 4    5    1   10    2500    .25  1.1    530
 5    9    1   18    1000    .25    0    530
 5    8    1    2    2500    .25    0    530
 8    7    1   24    1800    .25    0    530
 7    6    1   10    1300    .25    0    530
 9   13  -99                              530
13   17    1    5    1100    .25    0    445
17   21    1   18    1300    .25  6.5    360
 8   12  -99                              530
12   16    1   17    1600    .25    0    455
16   20    1    4    1500    .25 3.25    380
 7   11  -99                              530
11   15    1   17    1600    .25    0    480
15   19    1    4    1500    .25 3.25    430
 6   10    1   18    1300    .25 3.25    530
10   14    1    8    1100    .25    0    485
14   18    1    4    1000    .25 3.25    460
28   31    1    0     750      0    0    275
31   29    1   23     750    .25  6.4    275
 2   24  -99                              400
24   25    1    3     750    .25    0    250
25   26    1    5     750    .25  1.1    100
26   27  -99                              100
11   14    1    1     300    .25  1.5    495
12   15    1    1     400    .25  1.5    480
13   16    1    1     300    .25  1.5    455
21   20    1    8    1300    .25    0    360
20   22    1    9    2500    .25    0    370
18   19    1   19    1000    .25    0    460
19   22    1   17    1800    .25    0    440
22   30    1   10    2500    .25  1.1    400
30   23    1    1    2500    .25  1.1    400
23   32    1    0    2500      0    0    400
32    1    1    1    2500    .25    0    400
-1
1    28    1    0    0    1.001    100
-1
2
2    2
23    32    477    88.89    500    0
28    31    477      6.7    500    0
   1    20    1.205    0.018
1.01   400   0.5242    0.033
   1   550   0.4286    0.037
0
0.03(P(2)-P(24))=Q(2,24)
0.12(P(26)-P(27))=Q(26,27)
Q(26,27)=1.5
```

```
Q(7,11)=Q(6,10)
Q(8,12)=Q(6,10)
Q(9,13)=0.8(Q(6,10)+Q(7,11)+Q(8,12))
E
```

Results for Network 5b

Branch	from	to	P in (bar)	P out (bar)	Flow (kg/s)
1	1	2	1.01095	1.01094	12.9922
2	2	3	1.01094	1.01089	11.4922
3	3	29	1.01089	1.01079	11.4922
4	29	4	1.01079	1.01056	12.9922
5	4	5	1.01056	1.01040	12.9922
6	5	9	1.01040	1.00850	5.3200
7	5	8	1.01040	1.01040	7.6726
8	8	7	1.01040	1.01027	5.1151
9	7	6	1.01027	1.01021	2.5575
10	9	13	1.00850	1.01041	6.1381
11	13	17	1.01041	1.01020	4.7982
12	17	21	1.01020	1.00929	4.7982
13	8	12	1.01040	1.00930	2.5575
14	12	16	1.00930	1.00927	2.6323
15	16	20	1.00927	1.00917	3.1538
16	7	11	1.01027	1.00935	2.5575
17	11	15	1.00935	1.00931	2.8759
18	15	19	1.00931	1.00922	2.8011
19	6	10	1.01021	1.00994	2.5575
20	10	14	1.00994	1.00983	2.5575
21	14	18	1.00983	1.00951	2.2392
22	28	31	1.00100	1.01169	1.5000
23	31	29	1.01169	1.01079	1.5000
24	2	24	1.01094	1.01044	1.5000
25	24	25	1.01044	1.01039	1.5000
26	25	26	1.01039	1.01026	1.5000
27	26	27	1.01026	1.01013	1.5000
28	11	14	1.00935	1.00983	-0.3184
29	12	15	1.00930	1.00931	-0.0748
30	13	16	1.01041	1.00927	0.5215
31	21	20	1.00929	1.00917	4.7982
32	20	22	1.00917	1.00916	7.9520
33	18	19	1.00951	1.00922	2.2392
34	19	22	1.00922	1.00916	5.0403
35	22	30	1.00916	1.00905	12.9922
36	30	23	1.00905	1.00898	12.9922
37	23	32	1.00898	1.01096	12.9922
38	32	1	1.01096	1.01095	12.9922

Fig. 6.1. HF DYNAMIC NETWORK



Network 5c (HF3 dynamic network)

```
 1   2   1   2   2300    .25     0    400
 2   3   1   3   2300    .25    .5    400
 3  29   1   2   2500    .25     2    400
29   4   1   8   2500    .25     2    530
 4   5   1  10   2500    .25   1.1    530
 5   9   1  18   1000    .25     0    530
 5   8   1   2   2500    .25     0    530
 8   7   1  24   1800    .25     0    530
 7   6   1  10   1300    .25     0    530
 9  13   1   4   1000    .25  3.25    530
13  17   1   5   1100    .25     0    445
17  21   1  18   1300    .25   6.5    360
 8  12   1   4   1500    .25  3.25    530
12  16   1  17   1600    .25     0    455
16  20   1   4   1500    .25  3.25    380
 7  11   1   4   1500    .25  3.25    530
11  15   1  17   1600    .25     0    480
15  19   1   4   1500    .25  3.25    430
 6  10   1  18   1300    .25  3.25    530
10  14   1   8   1100    .25     0    485
14  18   1   4   1000    .25  3.25    460
28  31   1   0    750      0     0    275
31  29   1  23    750    .25   6.4    275
 2  24   0 0.25 0.0001     2 -1.009   400
24  25   1   3    750    .25     0    250
25  26   1   5    750    .25   1.1    100
26  27   1  18    900    .25   1.1    100
11  14   1   1    300    .25   1.5    495
12  15   1   1    400    .25   1.5    480
13  16   1   1    300    .25   1.5    455
21  20   1   8   1300    .25     0    360
20  22   1   9   2500    .25     0    370
18  19   1  19   1000    .25     0    460
19  22   1  17   1800    .25     0    440
22  30   1  10   2500    .25   1.1    400
30  23   0   5      1    -36    12    400
23  32   1   0   2500      0     0    400
32   1   1   1   2500    .25     0    400
-1
1   27   2   0   0  -1.5      100
1   30   3 100   0     0      100
1    2   3 100   0     0      400
1   28   1   0   0  1.001     100
-1
2
2 2
23   32   477   88.89   500   0
28   31   477    6.7    500   0
   1   20    1.205     0.018
1.01  400    0.5242    0.033
   1  550    0.4286    0.037
-1
E
```

Fig. 6.2 : Pressure control at node 29 of HF network.
(Lower graph - pressure at node 29 (in Bar))