DUBLIN CITY UNIVERSITY

Ollscoil Chathair Bhaile Átha Cliath

Simulation of Fluid Flow System in Process Industries

By

Nasser E. Khamkham

MEng. *2000*

Simulation of Fluid Flow System in Process Industries

By

Nasser E. Khamakham, B.Sc. (Eng.)

This thesis is submitted to Dublin City University as the fulfilment of the requirement for the award of Degree of

Master of Engineering

Supervisor: Professor M.S.J.Hashmi

Dublin City University

School of Mechanical &Manufacturing Engineering

2000

Dedication

This thesis is dedicated to my mother and father who always wished me to be a successful engineer, thanks a lot.

Declaration

This is to certify that the material presented in this thesis is entirely my own work, except where specific references have been made to the works of others, and no part of this work has been submitted in support of an application for another degree or qualification to this or any other establishment.

Signed : [illegible] I.D.95971637

Nasser E. Khamkham

September 2000

Acknowledgements

Many individuals have come to my assistance during the present work. I offer many thanks to all, and in particular I would like to acknowledge the contributions of:

Prof. M.S.J.Hashmi, my academic supervisor and Head of School of mechanical & manufacturing engineering at Dublin City University for his continued support, guidance, encouragement, and his excellent supervision throughout the course of this project.

The author would like to express his appreciation to Mr Liam Domican and Ms.Michelle Considine for their help and kindness

I would like also to thank all friends. Special thanks to Mohammed Alhemry, Mokhtar Alazharey, Hussam Elsheik,Dr Fuzi Braeiak,Abdulbasset Abuazza, Nurri Al zarroug,Mousbah Alsaket, Jamal Ibrahim,Yosef abugallia ,Ehmida Ajena, ,Dr Nuri Lekhboli, Ayad eldarhoby,Dr. Abdulrahman, Khalid Bakkar and many otherer ,whose names I forgot to mention.

Thanks to all my fellow students for their assistance, special thanks to Brian O'sullivan,Josef Stoke, Iqbal mohammed,Wong,Tang,Padu,and Shekar.

I would like to convey my sincere thanks to my family, especially my brothers,Mouammer, Dr Mokhtar , and Ibrahim for there kindness and encouragement. Thanks are due to all my brothers and sisters for their support and inspiration.
Special thanks to Mr.Mousa Suliman, Mr Emhmmed Ajaj for their support and help.

My Fiancée , Soad, whom the present work, is dedicated, for her encouragement and so much more. Thanks a lot.

Simulation of Fluid Flow System in Process Industries

By

Nasser E. Khamakham, B.Sc. (Eng.)

Abstract

A comprehensive and integrated suite of computer software has been developed to simulate the steady, one-dimensional, incompressible fluid flow in pipeline networks. The computer program accommodates Newtonian liquids, but does not generally apply to gas flow unless the assumption of constant density is acceptable. The computer program is written in C language, to solve the basic pipe system equations using the linear theory method.

This computer program is written to analyse steady state flows and pressures for pipe distribution system. The program is written to accommodate any piping configuration and various hydraulic components such as pumps, valves or any component, which produces significant head loss.

Computations can be carried out using English units of CFS, GPM or Standard International (SI) units.

In this project, the linear theory method is employed for solving the set of equations describing the pipe network. In addition, other methods for solving the pipe network have been described briefly.

The simulation software has been successfully applied to solve a number of networks. Moreover, the results of the simulation were satisfactory.

Table of contents

CHAPTER 1

Introduction and Justification

1.1 Introduction

In this chapter, the justification and the aim of study are identified; the method of approach adopted in achieving the set objective is outlined.

Finally, a summary of the content of the different chapters is provided under the heading “layout of Thesis”.

1.2 Importance of the simulation of pipe networks

Analysis and design of pipe networks create a relatively complex problem, particularly if the network consists of a large number of pipes as it frequently occurs in chemical or refinery complexes, natural gas pipe networks, or in the water distribution system of large metropolitan areas.

In Refinery complexes or water distribution pipeline networks, the steady-state analysis is a small but vital component of assessing the adequacy of a network.

Such an analysis is needed each time changing patterns of consumption or delivery are significant or add-on features, such as supplying new subdivision, addition of booster pumps or storage tanks, change the system.

In addition to steady state analysis, studies dealing with unsteady flows or transient problems, operation and control, acquisition of supply, optimisation of network performance against cost, should be given consideration.

The steady-state problem is considered solved when the flow rate in each pipe is determined under some specified patterns of supply and consumption.

The supply may be from reservoirs, storage tanks and /or pumps or specified as inflow or outflow at some point in the network.

From the known flow rates, the pressure or head losses throughout the system can be computed. Alternatively, the solution may be initially for the heads at each junction or node of the network and these can be used to compute the flow rates in each pipe in the network.

However, before the preparation of such a model is embarked upon, the objective of the study should be decided. The model may be required for new design, leakage control, pump scheduling, rehabilitation planning or general operational use. These objectives will determine the type of model, the level of detail necessary and the amount of resource and time scale of the project.

1.3 The C programming language

The programming language was developed in the early 1970s by **Dennis Ritchie,** system software engineering at **AT&T Bell Laboratories**. C evolved from a language named **B** that was developed by **Ken Thomson.**
The popularity of the C programming language has increased steadily since its creation. This has been partly due to the increase in popularity of the Unix operating system and the close association between Unix and C. C is the native programming language under Unix. A large part of the Unix operating system is written in C, and most of the software running under Unix is written in C.
However, C's success is primarily due to the fact that although it is a simple and elegant language, it is also a very powerful and efficient language. The C programming language has many features that give it an advantage over other procedural languages such as **FORTRAN, BASIC, and Pascal**. Some of these features include flexibility, efficiency, portability, and speed [1,2,3,4].

This is evident from the fact that C is being used extensively for developing a wide variety of applications. C is also a very efficient language. C programs tend to be compact and run faster than programs developed using other languages. Also, since C is a small language (C has only about 40 keywords), it encourages concise code.

The C language does have a few disadvantages. Its compact nature makes it possible to write programs that may be difficult to understand. Also, since the language imposes few constraints it is possible for inexperienced programmers to write C programs that contain many errors.

Creating a C program

The translation of programs written in a high-level language such as C into machine code is accomplished by means of a special computer program called compiler. The compiler analyzes a program written in a language such as C and translates it into a form that is suitable for execution on the particular computer system. There are several steps that one has to perform to create a C programs:

1-Use the text editor to write your program (source code) in C.

2-Compile your program using a c compiler.

3- Link your program with library functions using linker.

4- Execute and test your program.

Compiling and linking the program

The C compiler checks the source code for errors, and the linker takes the object file produced by the compiler and combines this with other objects files and library functions to produce an executable program. The exact command for linking the program depends on the compiler. On many systems, the compile and link steps are integrated into one step, which means that the compiler automatically calls the linker if there are no errors in the program. For example, the Microsoft C compiler, and the Borland C compiler.

Compiler hardware and software

The compiler used here is the Microsoft visual C++ compiler. The Microsoft visual C++ compiler package provides a comprehensive, up-to-date production level development environment for developing all windows application [1,2].

The Microsoft visual C++ v (5) compiler can be installed on Intel based PCs with Pentium processor or later version, running Windows 95 or Windows NT 3.5 or later version. The minimum and recommended requirements for running are shown in the table below [2].

Component	Minimum	Recommended
Processor	P5 66MHz	Fastest Processor available
RAM	16MB	64 MB
Hard disk	500MB	1 GB

1.4 Aim of study

The analysis of flow distribution networks has received considerable attention recently. Thanks to the wide availability of personal computers, engineering now has a variety of excellent, low-cost piping designs and analysis software tools from which to choose. So the computer costs associated with the analysis of a large network, consisting of several hundred pipes, is insignificant compared to the cost of professional time involved in assembling data, interpreting the results of the analysis, and proposing design alternatives.

This project aims to describe the facilities offered by a C program developed for the analysis of networks of pipes, pumps, bends, valves, and reservoir, and presents details of the method of analysis used and some of the associated problems.

The objective of the present study can be summarised as follow:

1-To develop a computer program written in C language to simulate the steady state, one-dimensional fluid flow in complex pipe networks.

2-To obtain the steady state solution for pipe networks from the data, which define the geometry or interconnection of the network, the characteristics of the pipes, pumps and other units.

3- To provide easier features and steps for the input data of the program to make it more user friendly.

4- To make the software more flexible in handling the various features of pipe systems, such as pumps, tanks, and pressure reducing valves.

1.5 Method of approach

A network of pipes and hydraulic elements (valves, pumps, and reservoirs) is considered solved when the heads and consumption at all nodes in the network are known. Obtaining the solution, as presented in this thesis, consisting of finding the values of the specified unknowns which satisfy the following physical laws of the network: (1) preservation of mass continuity at each node; and (2) that for each element there is a known relationship between discharge and energy gradients.
In addition, a relationship between flow rates, or velocity, and head loss or pressure drop is needed.
The continuity relationships are linear algebraic equations while the relationships describing the conservation of energy around a closed loop are generally non-linear algebraic equations, no method for the direct simultaneous solution of these equations is known, and an iterative method must be employed. One of the most widely used methods is the linear theory method, which was presented by Wood and Charles[5]. In this thesis, the linear theory method will be described and used in solving the system of equations which considers the flow rates unknown (i.e. the Q-equation). This method has several distinct advantages over other methods such as Newton-Raphson [6] or Hardy-Cross [7] methods described in the thesis. First, it does not require an initialisation, and secondly it always converges in relatively few iterations.
Figure 1.1 shows the solution approach, which has been applied to solve the sets of equations describing the pipe network.

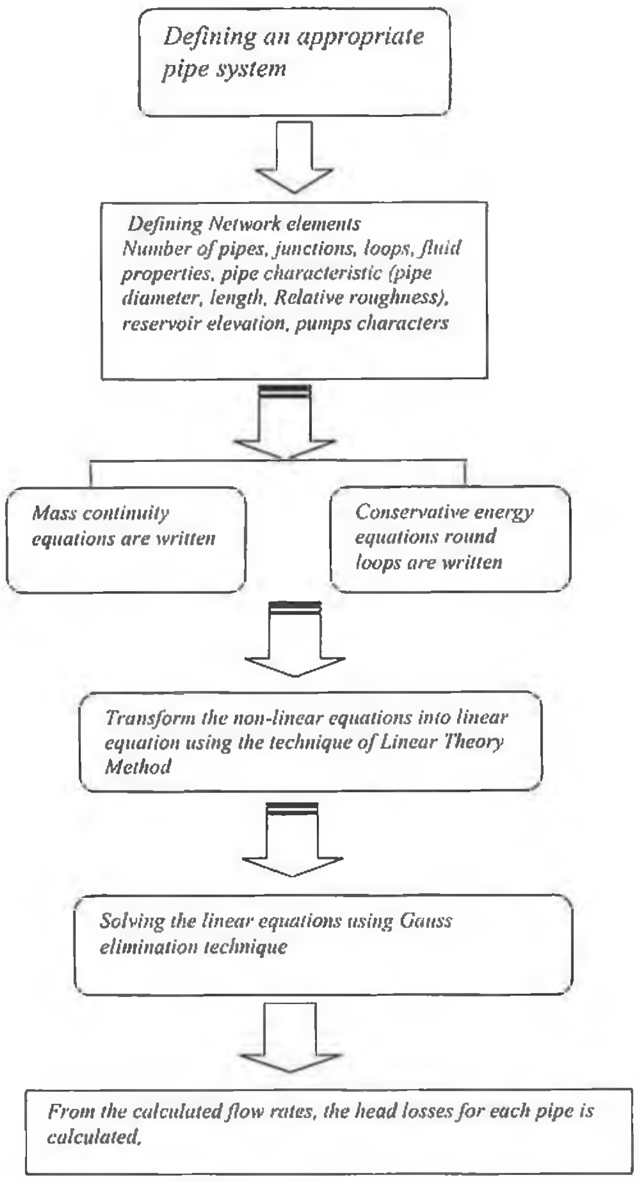

Defining an appropriate pipe system
Defining Network elements
Number of pipes, junctions, loops, fluid properties, pipe characteristic (pipe diameter, length, Relative roughness), reservoir elevation, pumps characters
Mass continuity equations are written
Conservative energy equations round loops are written
Transform the non-linear equations into linear equation using the technique of Linear Theory Method
Solving the linear equations using Gauss elimination technique
From the calculated flow rates, the head losses for each pipe is calculated.

Figure 1.1. Method of approach for pipe networks simulation

1.6 Layout of Thesis

This thesis is divided into six chapters. Following this introductory chapter, chapter 2 gives the platform for understanding the pipe network analysis, including the basic principle of fluid mechanics, and also provides a better way of defining the pipe network elements. The historical developments of the simulation of pipe networks are reviewed and discussed, and the relevant literatures are given. Chapter 3 gives full coverage of the linear theory method, which will be used to solve the pipe network problems. Chapter 4 describes and discusses in details the development of the program, which is based on the linear theory method. Also in this chapter, several examples were presented and tested for the simulation. The results of the simulation for several examples are presented and discussed in Chapter 5. Finally, conclusions based on the present work are presented in Chapter 6.

CHAPTER 2

Pipe Network Analysis Literature Review

2-1 Fundamental Of Fluid Mechanics

The aim of this section is to build on the understanding of the basic principles of fluid mechanics so that one can apply these principles to the solution of pipe network problems.

2.1.1 Fluid properties

Density: The mass per unit volume is referred to as the density of the fluid and is denoted by the Greek letter (ρ). It is independent of gravitational force, but does depend on temperature or pressure. For liquid, this dependence is very small and can sometimes be ignored. The dimensions of density are mass per length cubed.

The English system of units (abbreviated ES) uses the slug for the unit of mass, and feet for the unit of length.

$$\frac{Slug}{ft^3} \quad \text{Or} \quad \frac{lb-\sec^2}{ft^4}$$

In the International system of units (abbreviated SI) which is an outgrowth of the metric system, the mass is measured in the unit of the gram, gr. (or kg) and the force is measured in Newton, N, and the length in meters,

$$\frac{kg}{m^3} \text{ Or } \frac{N-\sec^2}{m^4}$$

Specific weight*:* The specific weight is the weight of fluid per unit volume and is denoted by the Greek letter (γ)

The specific weight has dimension of force per unit volume .

Its unit in the ES is

$$\frac{lb}{ft^3}$$

and in the SI is

$$\frac{kg}{m^2-\sec^2}$$

and the specific weight is related to the fluid density by the acceleration of gravity

$$\gamma = g\,\rho$$

Viscosity: This fluid property has meaning when the fluid is in motion. It is a measure of the fluid's resistance to shear stresses. The viscosity is given the symbol μ and is defined as the ratio of the shearing stress τ to the rate of change in viscosity

$$\tau = \mu\,\frac{dv}{dy} \qquad (2.1)$$

In which the $\frac{dv}{dy}$ is the derivative of the velocity with the respect to the distance and called the velocity gradient

$$\nu = \mu / \rho$$

2.1.2 Basic Equations

Solution to the most fluid flow problems generally involves the application of one or more of the three basic equations: Continuity, Momentum, and Energy. These three basic tools are developed from the law of conservation of mass, Newton's second law of motion, and the first law of thermodynamics.

2.1.2.1 Continuity equation

The simplest form of this equation is for one-dimensional incompressible steady flow in a closed conduit.

$$AV = Q$$

in which A is the cross sectional area of the pipe, V is the average velocity of the flow through the section, and Q is the volumetric flow rate.

In dealing with junctions of two or more pipes the continuity principle states that the mass flow rate into the junction must equal the mass flow out of the junction. Mathematically this principle is

$$\sum Q_i = 0 \qquad (2.2)$$

This equation will play an important role in analysing networks of pipe [8,9].

2.1.2.2 Energy equation

The first law of thermodynamics states that the change of internal energy of a system is equal to the sum of the energy added to the fluid and the work done by the fluid. A general form of the energy equation for incompressible pipe flow (assuming a uniform velocity profile) is

$$\frac{p_1}{\rho} + gz_1 + \frac{V_1^2}{2} = \frac{p_2}{\rho} + gz_2 + \frac{V_2^2}{2} - W_p + W_t + W_f \qquad (2.3)$$

The unit of each term is energy per unit mass. The first two terms on both sides of the equation are potential energy, the third term is the kinetic energy, W_p is pump energy added to the system, W_t is turbine energy removed from the system, and W_f represents friction and other minor losses.

Equation (2.3) is restricted to steady flow and ignores nuclear, electrical, magnetic and surface tension energy.

An alternate form of the energy equation is obtained by dividing equation (2.3) by gravity .The units are energy per unit weight of liquid: $\frac{ft.lb}{lb}$ or $\frac{N-m}{N}$, which reduce to *ft* or *m*, respectively, after simplification, the form of the equation is

$$\frac{p_1}{\gamma} + z_1 + \frac{V_1^2}{2g} = \frac{p_2}{\gamma} + z_2 + \frac{V_2^2}{2g} - H_p + H_t + H_f \qquad (2.4)$$

In hydraulic engineering practice equation (2.4) is used more widely than equation (2.3) and is known as the Bernoulli equation [8,9,10].

2.1.3 Friction Head Losses

Application of the energy equation requires an accurate estimate of the energy losses caused by shear stress between the fluid and the boundary.

Equation (2.1) identifies that the shear stress is a function of viscosity and the velocity gradient near the boundary .The velocity gradient is controlled by the velocity, the boundary roughness, and thickness of the boundary layer [9].

The most significant problem with pipeline design is to obtain a reliable value of the shear stress or pipe friction factor for fully developed flow or the loss coefficients for local losses.

From an engineering point of view, it is not practical to work in terms of wall shear stress since it requires detailed information on the velocity gradient. The velocity gradient does not vary with distance in developed flow, but it is a function of velocity and viscosity for fully developed flow.

It is easier to work in terms of the average shear stress or friction loss over a length of pipe. The friction loss between two points in a pipe is equal to the decrease in the total head. Dimensional analysis can be used to provide a functional relationship between the friction loss, the important fluid properties and flow parameters.

There are several equations that are often used to evaluate the friction head loss.

The most fundamentally sound method for computing such head losses is by means of the Darcy-Weisbach equation.

a) Darcy-Weisbach equation

The Darcy –Weisbach equation is given by

$$h_f = \frac{\Delta p}{\gamma} = f\frac{LV^2}{2gD} \tag{2.5}$$

where D is the pipe diameter, L is the length of pipe, V is the average velocity of flow, g is the acceleration of gravity and f is a dimensional friction factor [9,11,12].

The friction factor f has been evaluated experimentally for numerous pipes. Such tests have shown f to be a function of pipe diameter, roughness, and Reynolds number *Re.*

Since roughness may vary with time due to build-up of solid deposits or organic growths, f is also time dependent. Manufacturing tolerance also causes variation in the pipe diameter and surface roughness. The point that is being made is that it is not possible to know the friction factor of any pipe precisely.

A designer is required to use good engineering judgement in selecting a design value for f so that proper allowance is made for these factors.

The functional relationship of f with roughness, diameter *d*, and *Re* has been investigated quite thoroughly [9,12,13]. The pioneering work was done by Nikuradse [12,13] and Colebrook [13,14]. Their work is the basis of the Moody chart [9,13].

Nikuradse [13] measured head loss, or pressure drops, caused by bonding uniform sand particles of various sizes, e, on the interior walls of different pipes. When his test results are plotted on log-log graph paper with the Reynolds number,

$$R_e = \frac{VD}{\nu} \qquad (2.6)$$

Plotted as abscissa and the friction factor f as the ordinate then data from different values define the separate lines shown on Figure 2.1 [9,15].

Equation (2.5) is the basic equation from which the frictional pressure drop may be calculated. It is valid for all types of fluid and for both laminar and turbulent flow.

Friction factor for laminar flow

For laminar flow for which the well understood law of fluid shear, it is possible to provide a simple straightforward theoretical derivation of the Darcy-Weisbach equation, or more specifically derive the relationship

$$f = \frac{64}{R_e} \qquad \text{For } R_e < 2100 \qquad (2.7)$$

Friction factor for turbulent flow

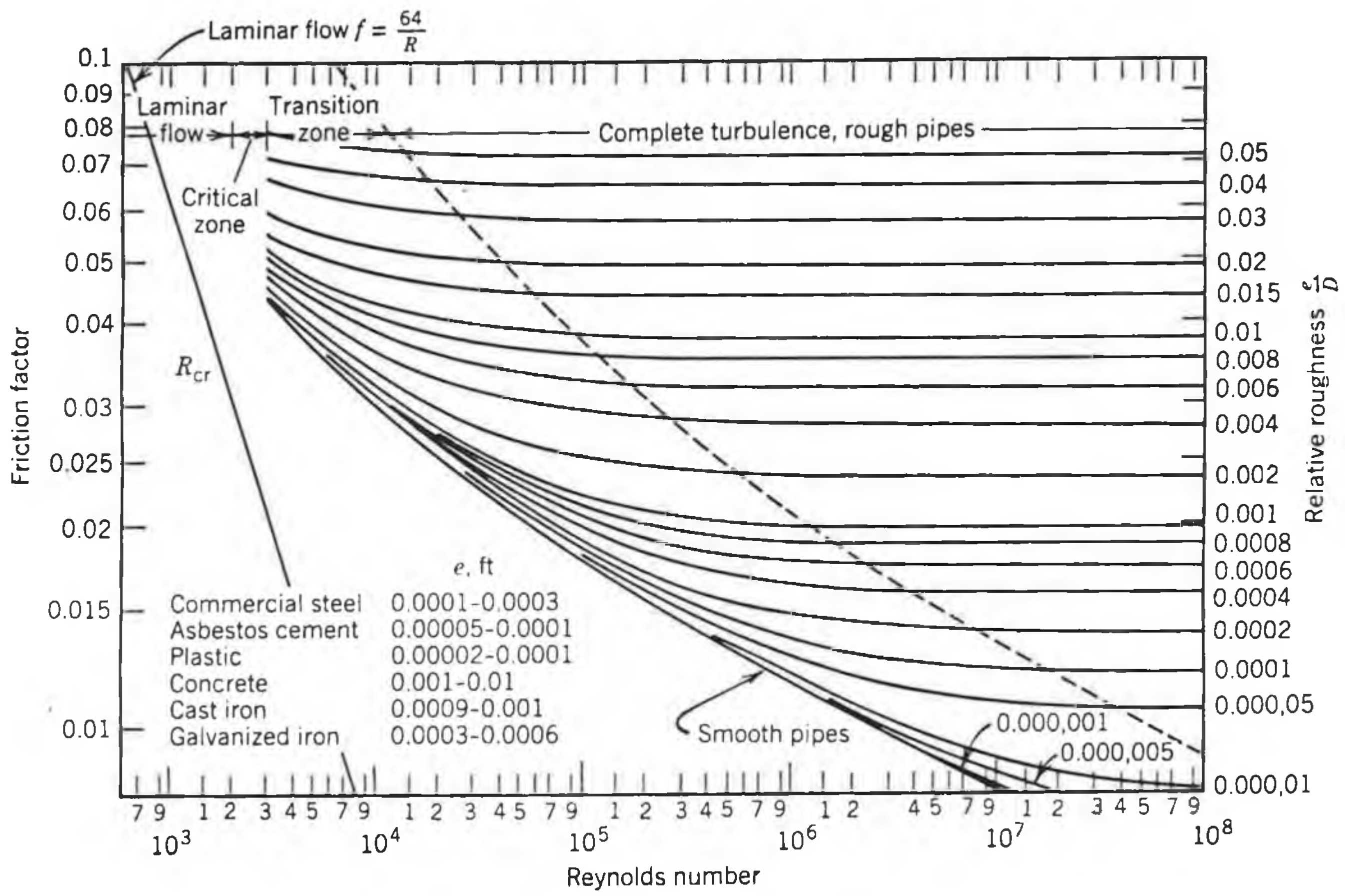
Equations relating f to R_e and $\frac{\in}{D}$ for turbulent flow (i.e. flow with R_e>2100) are summarised in Table 2.1 [12].

The pipes used by Nikuadse were artificially roughened with uniform roughness and, therefore, can not be applied directly to commercial pipes containing turbulent flows. Tests by others, notably Colbrook[14],demonstrated that flows in

Table 2.1. Summary of friction factor equation for Darcy-Weisbach equation[12].

Type of flow	*Equation giving f*	*Range*
Laminar	$f = \frac{64}{R_e}$	$R_e < 2100$
Hydraulically smooth *Turbulent smooth*	$f = \frac{0.316}{\mathrm{Re}^{0.25}}$ $\frac{1}{\sqrt{f}} = 2\log_{10}(\mathrm{Re}\sqrt{f}) - 0.8$	$4000 < R_e < 10^5$ $R_e > 4000$ and $\frac{e}{D} \to 0$
Transition between hydraulically smooth *And wholly rough*	$\frac{1}{\sqrt{f}} = 1.14 - 2\log_{10}(\frac{e}{D} + \frac{9.35}{\mathrm{Re}\sqrt{f}})$	$R_e > 4000$
Wholly rough	$\frac{1}{\sqrt{f}} = 1.14 - 2\log_{10}(\frac{e}{D})$	$R_e > 4000$

Figure 2.1 The Moody diagram [9].



Commercial pipes also become independent of Reynolds number, R_e, at a large R_e and large wall roughness. Consequently, it is possible to compute the equivalent relative roughness $\frac{e}{D}$ for commercial pipes from the experimental equation Nikuradse determined as valid for his wholly rough pipes. The equation [9,12,13] for wholly rough pipes is

$$\frac{1}{\sqrt{f}} = 1.14 - 2\log_{10}(\frac{e}{D}) \tag{2.8}$$

From these values of $\frac{e}{D}$, the equivalent sand grain size e for commercial pipes have been determined and summarised in Table2.2[16,17]

Table 2.2. Values of equivalent roughness ***e*** for commercial pipes [16,17].

Material	*e, mm*	*e, in*
Riveted steel	0.9 –9.0	0.035 – 0.35
Concrete	0.30 –3.0	0.012 – 0.12
Cast Iron	0.26	0.010
Galvanized Iron	0.15	0.006
Asphalted Cast Iron	0.12	0.0048
Commercial or Welded Steel	0.045	0.0018
PVC, Drawn Tubing, Glass	0.0015	0.00006

For “hydraulically smooth” surface the equation is

$$\frac{1}{\sqrt{f}} = 2\log_{10}(\mathrm{Re}\sqrt{f}) - 0.8 \qquad (2.9)$$

The friction factor f appears on both sides of equation (2.9) and consequently it can not be solved explicitly for f with R_e known, but must be solved by trail and error or some iterative scheme. An equation proposed by Blasius [12], which can be solved explicitly for f which apply to smooth pipes but only for flows with R_e less than 10^5, is

$$f = \frac{0.316}{\mathrm{Re}^{0.25}} \qquad (2.10)$$

Equation (2.9) applies to smooth pipe over the entire range of R_e>4000,whereas equation (2.10) is an approximation to equation (2.9) and is limited to the range $4000<Re<10^5$.

For the transition zone between smooth and wholly rough flow, Colebrook and White[12,14] give the following equation,

$$\frac{1}{\sqrt{f}} = 1.14 - 2\log_{10}\left(\frac{e}{D} + \frac{9.35}{\mathrm{Re}\sqrt{f}}\right) \qquad (2.11)$$

Equation (2.11) gives nearly the same values for f as equation (2.9) for small values of $\frac{e}{D}$ and values of f nearly equal to those of equation (2.8) for every large values of R_e. Consequently, equation (2.11) may be used to compute f for all turbulent flows [12].

Particularly for hand computations, it is convenient to summarise the equations in Table 2.1 in a graph. This graph given as Figure 2.1 is known as the Moody diagram [12,15]. It is used to eliminate the trial and error solutions.

(b) Other empirical formulas

The Darcy-Weisbach equation is commonly used for determining head losses or pressure drops in closed conduct flow because it is the most exact [13,17]. This is because the variation of f with pipe roughness and Reynolds number is properly accounted for when the Moody chart is used. The two other equations in use are

1) Hazen-Williams equation

In ES units $$Q = 1.318 C_{HW} A R^{0.63} S^{0.54} \tag{2.12}$$

In IS units $$Q = 0.849 C_{HW} A R^{0.63} S^{0.54} \tag{2.13}$$

In which C_{HW} is the Hazen-Williams roughness coefficient, S is the slope of the energy line and equals to $\frac{h_f}{L}$, R is the hydraulic radius defined as the cross-sectional area divided by the wetted perimeter, P and for pipes equals D/4.

Suggested values of Hazen-Williams C_{HW} are listed in table 2.3 [12,17]. Value of C_{HW} range from 140 for a new pipe in excellent condition to less than 100 for old pipe in poor condition, typical values would be between 120 and 130 [17].

If the head loss is desired with Q known the Hazen-Williams equation for a pipe can be written as

(ES units) $$h_f = \frac{4.73L}{C_{HW}^{1.852} D^{4.87}} Q^{1.852} \tag{2.14}$$

With D and L in feet

(SI units) $$h_f = \frac{8.52 \times 10^5 L}{C_{HW}^{1.852} D^{4.87}} Q^{1.852} \tag{2.15}$$

Table 2.3 Values of Hazen-Williams coefficient C_{HW} [12,17].

Character of Pipe	Hazen-Williams Coefficients Of Roughness C_{HW}
PVC	150
Cement-lined Ductile Iron	140
New Cast Iron,Welded Steel	130
Old Cast Iron	100
Badly corroded Cast Iron	80

2) Manning equation

ES $$Q = \frac{1.49}{n} AR^{2/3} S^{1/2} \tag{2.16}$$

SI $$Q = \frac{1}{n} AR^{2/3} S^{1/2} \tag{2.17}$$

Manning 's n values are listed in Table2.4 [9,12].

Table 2.4 Manning's coefficient n [9,12].

Pipe Material	N
PVC	0.009
Cement-lined Ductile Iron	0.012
New Cast Iron,Welded Steel	0.014
Old Cast Iron	0.020
Badly corroded Cast Iron	0.035

2.1.4 Exponential Formula

In analysing the flow distribution in large pipe networks, it is advantageous to express the head loss in each pipe of the network by an exponential formula of the form

$$h_f = KQ^n \tag{2.18}$$

Value for k and n can be obtained directly from the previous equations given for the Hazen-Williams or Manning equation .To find k and n in Darcy-Weisbach equation it should be noted that f can be approximated over a limited range by an equation of the form

$$f = \frac{a}{Q^b} \tag{2.19}$$

Substituting this equation in the Darcy equation and grouping them gives

$$n = 2 - b \tag{2.20}$$

and

$$k = \frac{aL}{2gDA^2} \tag{2.21}$$

Consequently determination of n and k in the exponential formula requires finding values of *a and b* for the range of flow rates to be encountered. If the range is too large n and k may be considered variables [12,16,19].

2.1.5 Minor Losses

Minor losses is a term referring to losses that occurs at a pipe entrance, elbow, orifice, valve, etc. These devices alter the flow pattern in the pipe creating additional turbulence, which results in head loss in excess of the normal frictional losses in the pipe. These additional head losses are termed minor losses .If the pipelines are relatively long these losses are truly minor and can be neglected. In short pipelines they may represent the major losses in the system, or if the device causes a large loss, such as a partly closed valves, its presence has dominant influence on the flow rate. Judgement must be used in deciding how important the minor losses are and, therefor, how much effort should be expended in evaluating the various loss coefficients [8,9,11,17,20].

The head loss h_f caused by a minor loss is proportional to the velocity head.

$$h_L = K_L \frac{Q^2}{2gA^2} \tag{2.22}$$

The loss coefficient K_L is analogous to $\frac{fL}{D}$. In fact, some prefer to express loss coefficient as an equivalent pipe length [12,21,22]:

$$\frac{L}{D}=\frac{K_L}{f} \tag{2.23}$$

It simply represents the length of pipe that produces the same head loss as the minor loss. This is a convenient means of including minor losses in the Hazen-Williams and Manning equations.

For use with the Darcy equation, K_L is used rather than equivalent length [21,22].

The total head loss terms in the energy equation can be written as

$$h_f=(\sum\frac{fL}{2gDA^2}+\sum\frac{K_L}{2gDA^2})Q^2=CQ^2 \tag{2.24}$$

The summation term represents the numerical sum of all minor loss coefficients. If the minor loss is different in diameter than the pipe, the proper area in equation (2.24) must be used [12,13].

Typical value of loss coefficient for various minor losses are summarised in Table2.5

Table 2.5 *Minor Loss Coefficients* [12,17]

Item	K_L	
	typical value	typical range
Bends		
Short radius,r/d=1		
90		0.24
45		0.10
Valves		
Check valve	0.80	0.5 to 1.50
Full open gate	0.15	0.1 to 0.3
Full open butterfly	0.20	0.2 to 0.6
Full open globe	4.0	3 to 10

2-2 Incompressible steady flow in pipe networks

2.2.1 Introduction

Analysis and design of pipe networks create relatively complex problems, particularly if the network consists of a large number of pipes as frequently occurs in the water distribution system of a large metropolitan areas, or natural gas pipe networks.

Professional judgement is involved in deciding which pipe should be included in a single analysis. Obviously it is not practical to include all pipes which deliver to all sections of the network, even though they are connected to the total delivery system.

Often only those main trunk lines that carry the fluid between separate sections of the area are included and if necessary analyses of the networks within these sections may be included.

In a water distribution or in any chemical complex pipelines network system, the steady-state analysis is a small but vital component of assessing the adequacy of a network.

Such an analysis is needed each time changing patterns of consumption or delivery are significant or add-on features, such as supplying new subdivision, addition of booster pumps, or storage tanks change the system.

The steady-state problem is considered solved when the flow rate in each pipe is determined under some specified patterns of supply and consumption.

The supply may be from reservoirs, storage tanks and /or pumps or specified as inflow or outflow at some point in the network.

From the known flow rates the pressure or head losses throughout the system can be computed. Alternatively, the solution may be initially for the heads at each junction or node of the network and these can be used to compute the flow rates in each pipe in the network.

2.2.2 Basic relations between network elements

The two basic principles, upon which all network analysis is developed, are (1) the conservation of mass or continuity principle, and (2) the work-energy principle, including the Darcy-Weisbach or Hazen-Williams equation to define the relation between the head loss and the discharge in a pipe[13,17]. The equations that are developed from the continuity principle will be called junction continuity equations, and those that are based on the work-energy principle will be called Energy Loop Equations. The number of these equations that constitutes a non-redundant system of equations is related directly to fundamental relations between the number of pipes, number of junctions and number of independent loops that occur in a branched and looped pipe networks[12,13,17]. In defining these relations *NP* will denote the number of pipes in the network. *NJ* will denote number of junctions in the network, and *NL* will denote the number of loops around which independent equations can be written. In defining junctions, a supply source will not be numbered as a junction. A supply source is a point where the elevation of the energy line, or hydraulic grade line, is established. Figure 2.2 shows a sample of the geometry of simple pipe network.

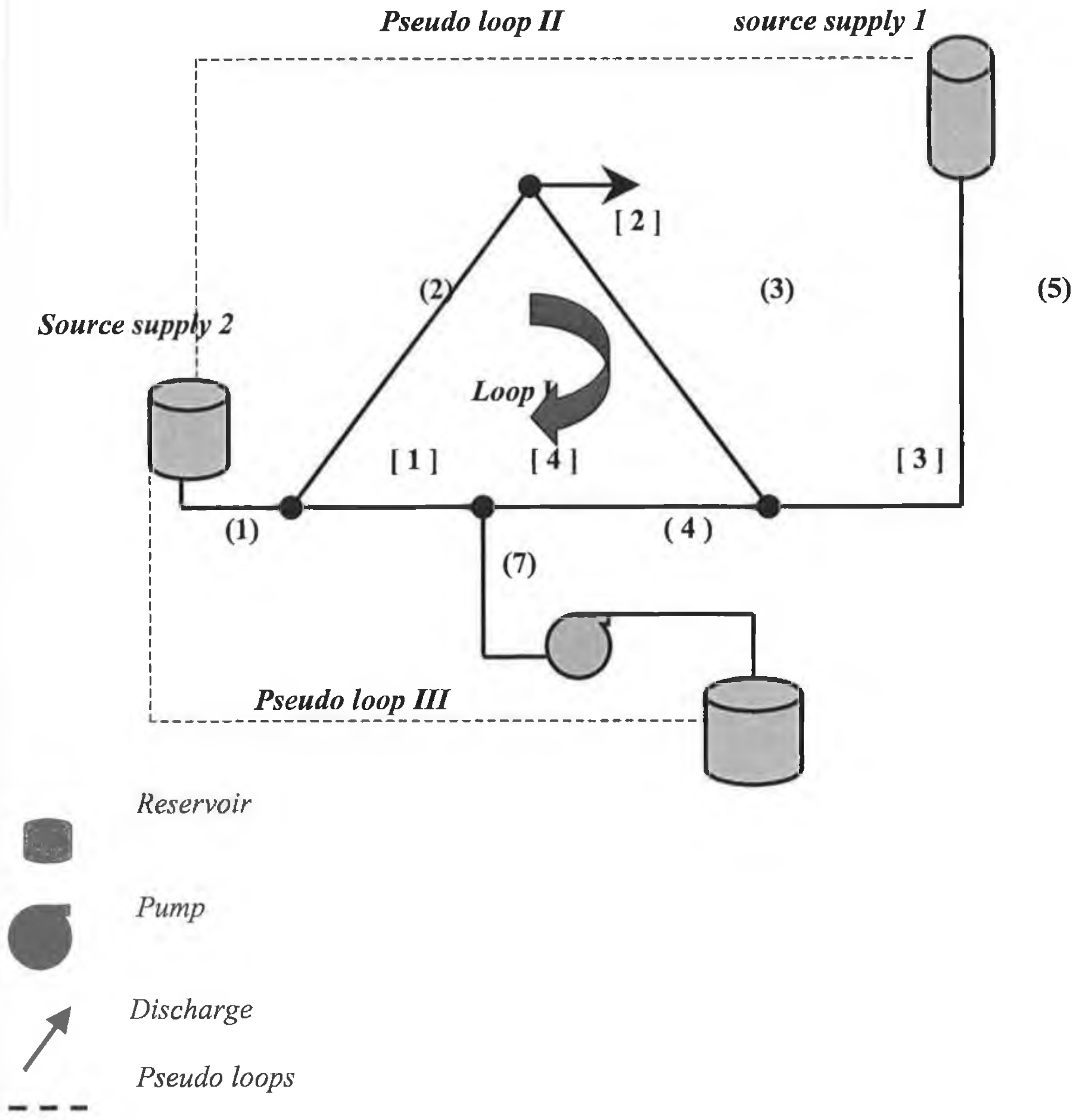


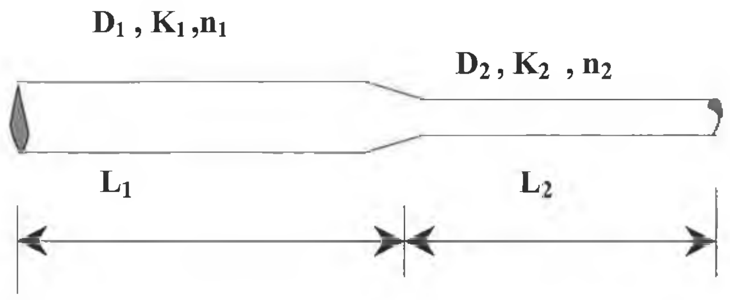
Figure 2.2. A small one real loop network and two pseudo loops connecting the supply sources

2.2.3 Reducing complexity of pipe networks

In general, pipe networks may include pipes in series, parallel pipes, and branching pipes. In addition, elbows, valves, meters, and other devices which cause local disturbances and minor losses may exist in pipes. All of the above should be combined or converted to an "equivalent pipe" in defining the network to be analyzed. The concept of equivalence is useful in simplifying networks. Method for defining an equivalent pipe for each of the above mentioned occurrences are as follows [13].

2.2.3.1 Series pipes

The method for reducing two or more pipes of different sizes in series will be explained by reference to the diagram below.



The same flow must pass through each pipe in series. An equivalent pipe is a pipe which will carry this flow rate and produce the same head loss as two or more pipes, or

$$h_{f_e} = \sum h_{f_i} \tag{2.25}$$

Expressing the individual head losses by the exponential formula gives,

$$K_e Q^{ne} = k_1 Q^{n_1} + k_2 Q^{n_2} + \dots = \sum K_i Q^{n_i} \tag{2.26}$$

For network analysis k and n are needed to define the equivalent pipe's hydraulic properties. If the Hazen-William equation is used, all exponents $n=1.85$, and consequently

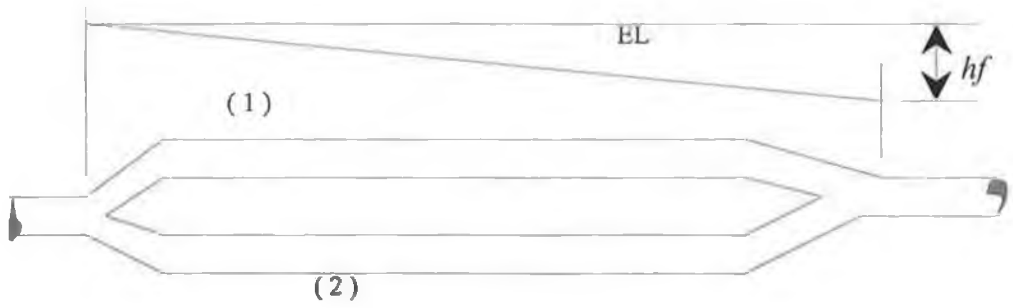
$$K_e = k_1 + k_2 + \dots = \sum K_i \tag{2.27}$$

or the coefficient k for the equivalent pipe equals the sum of k of the individual pipes in series. If the Darcy-Weisbach equation is used, the exponents n in equation (2.26) will not necessarily be equal, but generally these exponents are near enough equal to that n_e for the equivalent pipe can be taken as the average of these exponents and equation (2.27) may be used to compute K_e [13,17].

2.2.3.2 *parallel pipes*

An equivalent pipe can also be used to replace two or more pipes in parallel. The head loss in each pipe between junctions where parallel pipes part and join again must be equal, or

$$h_f = h_{f_1} = h_{f_2} = \dots \tag{2.28}$$



The total flow rate will equal the sum of the individual flow rates or

$$Q = Q_1 + Q_2 + \dots\dots = \sum Q_i \tag{2.29}$$

Solving the exponential formula $h_f = KQ^n$ for Q and substituting into equation (2.29) gives

$$\left(\frac{h_f}{K_e}\right)^{\frac{1}{n_e}} = \left(\frac{h_f}{k_1}\right)^{\frac{1}{n_1}} + \left(\frac{h_f}{k_2}\right)^{\frac{1}{n_2}} + \dots = \sum \left(\frac{h_f}{K_i}\right)^{\frac{1}{n_i}} \tag{2.30}$$

If the exponents are equal as will be the case in using the Hazen-Williams equation the head loss h_f may be eliminated from equation(2.30) giving

$$\left(\frac{1}{K_e}\right)^{\frac{1}{n}} = \left(\frac{1}{k_1}\right)^{\frac{1}{n}} + \left(\frac{1}{k_2}\right)^{\frac{1}{n}} + \dots = \sum \left(\frac{1}{K_i}\right)^{\frac{1}{n}} \tag{2.31}$$

When the *Darcy-Weisbach* equation is used for the analysis, it is common practice to assume n is equal for all pipes and use equation (2.31) to compute the K_e for the equivalent pipe [12,13,17].

2.2.3.3 branching system

In a branching system a number of pipes are connected to the main to form the topology of a tree. Assuming that the flow is from the main into the smaller laterals it is possible to calculate the flow rate in any pipe as the sum of the downstream consumption's or demands. If the laterals supply fluid to the main, as in a manifold, the same might be done.

In either case, by proceeding from the outermost branches towards the main or " root of the tree" the flow rate can be calculated and from the flow rate in each pipe the head loss can be determined using the Darcy-Weisbach or Hazen-Williams equation.

In analyzing a pipe network containing a branching system, only the main is included with the total flow rate specified by summing those from the smaller pipes. Upon completing the analysis the pressure head in the main will be known. By subtracting individual head losses from this known head, the heads at any point throughout the branching system can be determined [12].

2.2.3.4 minor losses

Valves and fittings in the piping system cause a minor loss, which is not insignificant in comparison to the friction loss in a pipe.

The easiest way to calculate these losses is to use the *equivalent length* method to estimate the effect of a valve or fitting by treating it as if it were an additional length of pipe. The equivalent pipe is formed by adding a length ΔL to the actual pipe length such that the friction head loss in the added length of pipe equals the minor loss[12,21].

For use with the Darcy equation

$$\Delta L = \frac{K_L D}{f} \tag{2.32}$$

In the exponential formula , the *K coefficient* for the equivalent pipe is

$$K_e = \frac{a(L + \sum \Delta L)}{2gDA^2} \tag{2.33}$$

Or might be given such as in Table [2.6] which lists some common devices and their equivalent length values, which, are given as the length –to-diameter

(L_e / D) ratios so they can be used directly in the modification of the Darcy equation

Table 2.6. Values of equivalent length for some device [13].

Device	Equivalent Length (L_e/D)
Check valve	150
90° standard elbow	30
45° standard elbow	16
closed return bend	50
standard tee-run	20
standard tee-branch	60

When using the Hazen-William formula ΔL can be computed from

$$(ES \text{ units}) \qquad \Delta L = 0.00532 K_L Q^{0.148} C_{HW}^{1.852} D^{0.8703} \tag{2.35}$$

$$(SI \text{ units}) \qquad \Delta L = 0.00773 K_L Q^{0.148} C_{HW}^{1.852} D^{0.8703}$$

and the K in the exponential formula is

$$(ES \text{ units}) \qquad K_e = \frac{4.73(L + \Delta L)}{C_{HW}^{1.852} D^{4.87}}$$

$$(SI \text{ units}) \qquad K_e = \frac{10.7(L + \Delta L)}{C_{HW}^{1.852} D^{4.87}}$$

2-3 System of equations used for solving pipe networks

Three different systems of equations can be developed for the solution of network analysis problems. These systems of equations are named after the variables that are regarded as the principal unknowns in that solution method. These systems of equations are called the *Q-equations* (where the discharges in the pipes of the network are the principal unknowns), the *H-equations* (where the HGL-elevation, also simply called the heads *H*, at the nodes are the principle unknowns), and the *ΔQ-equations* (when corrective discharges, *ΔQ*, are the principal unknowns). Each of these three systems of equations will be studied separately.

2.3.1 Flow rates as unknowns (Q-equations)

The analysis of flow throughout networks of pipes is based on the two fundamental laws of fluid mechanics: continuity and conservation of energy.

In addition, a relationship between flow rates, or velocity and head losses or pressure drop is needed.

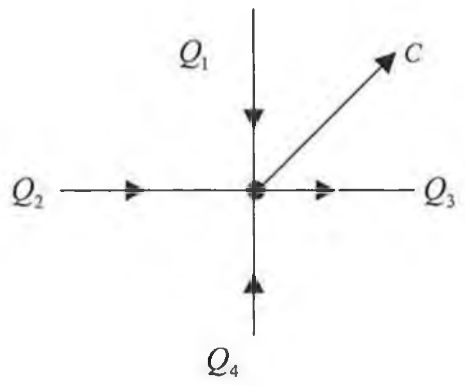
To satisfy continuity, the mass, weight, or volumetric flow rate into a junction must equal the mass, weight, or volumetric flow rate out of a junction.

For each junction a continuity relationship is written as:

$$\sum (Q_i)_{in} - \sum (Q_i)_{out} = C \qquad (2.36)$$

In which C is the external flow at the junction (commonly called consumption or demand). C is positive if flow is into the junction and negative if it is out from the junction .

Consider four pipes meeting at a junction as shown in the sketch below.



For applying the continuity equation to this example

$$Q_3 + Q_1 + Q_2 + Q_4 = -C \qquad (2.37)$$

If a pipe network contain *NJ* junction, (also called nodes) and all external flow are known then *NJ-1* independent continuity equation in the form of equation (2.36) can be written.

The last, or the NJ^{th} continuity equation, is not independent; that is, it can be obtained from some combination of the first $NJ-1$ equations. Note in passing that each of these continuity equations is linear, i.e., Q appear only to the first power [12,13].

In addition to the continuity equations, which must be satisfied, the work-energy principle provides additional equations, which must be satisfied.

These additional equations are obtained by summing head losses along both real and pseudo loops to produce independent equations.

Mathematically, the energy principle gives

$$\sum_{l}^{I} h_{fi} = 0$$

$$\ldots \qquad (2.38)$$

$$\sum_{l}^{NL} h_{fi} = 0$$

In which h_f represents the head loss in a pipe contained in that loop and is a function of the discharge Q. And *NL* represents the number of non-overlapping loops (also referred to the real loops) in the network, and the summation on small l is over the pipes in the loops $\mathrm{I}, \mathrm{II}, \ldots, NL$ by use of the exponential formula $h_f = KQ^n$.

However, the head loss in pipe (i) is best represented by a relationship

$$\sum_{\ell}^{L} K_l Q_l^{n_L} = 0 \qquad (2.39)$$

A pipe network consisting of *NJ* junction and *NL* real loops and *NP* pipes will satisfy the equation

$$NP = (NJ - 1) + NL \qquad (2.40)$$

(if all of the external flows are not known ,then all *NJ* junction equations are independent and available for use).

The (*NJ*-1) continuity equations are linear and the *NL* (head losses) equations are non-linear .

Systematic methods, which will utilize computer, are needed for solving this system of simultaneous equations [12,13,18]

2.3.2 Heads at junctions as unknowns (H-equations)

If the elevation of the energy line or hydraulic grade line throughout a network is initially regarded as the primary set of unknown variables, then one develops and solves a system of *H-equations*. One *H-equation* is written at each junction. Since looped pipe networks have fewer junctions than pipes, there will be fewer *H-equation* than *Q-equations*. Every equation in this smaller set is non-linear, however, whereas the junction continuity equations are linear in the system of *Q-equations* .

To obtain the system of equations, which contain the heads at the junctions of the network as unknowns, the *NJ-1* independent continuity equations are written as before. Thereafter the relationship between the flow rate and head loss is substituted into the continuity equations. In writing these equations, one begins by solving the exponential equation for the discharge in the form

$$Q_{ij} = \left(\frac{h_{f_{ij}}}{K_{ij}}\right)^{\frac{1}{n_{ij}}} = \left(\frac{H_i - H_j}{K_{ij}}\right)^{\frac{1}{n_{ij}}} \qquad (2.41)$$

Here the frictional head loss has been replaced by the difference in *HGL(hydraulic grade line)* values between the upstream and downstream nodes. In addition, in this equation a double subscript notation; has been introduced; the first subscript defines the upstream node of the pipe, and the second defines the downstream node. Thus Q_{ij} and K_{ij} denote the discharge and loss coefficient for the pipe from node i to node j.

Substituting equation (2.41) into the junction continuity equations, equation (2.36), yields

$$[\sum \left(\frac{H_i - H_j}{K_{ij}}\right)^{\frac{1}{n_{ij}}}]_{out} - [\sum \left(\frac{H_i - H_j}{K_{ij}}\right)^{\frac{1}{n_{ij}}}]_{in} = C \qquad (2.42)$$

Upon writing an equation of the form of equation (2.42) at *NJ-1*junctions, a system of *NJ-1* nonlinear equations is produced [12,13,23].

2.3.3 Corrective flow rates around loops of network considered unknowns (ΔQ-equations)

Since the number of junctions minus 1 *(i.e.NJ-1)* will be less in number than the number of pipes in a network by the number of loops *NL* in the network, the last set of *H-equations* will generally be less in number than the system of *Q-equations*. This reduction in number of equations is not necessarily an advantage since all of the equations are non-linear, whereas in the system of *Q-equations* only the *NL* energy equations were non-linear. A system that generally consists of even fewer equations can be written for solving a pipe network, however. These equations consider a corrective flow rate in each loop *or* *Q 's* as the unknowns.

These corrective discharges will be determined from the energy equations that are written for *NL* loops in the network, and thus *NL* of these corrective discharge equations must be developed. To obtain these equations, we replace the discharge in each pipe of the network by an initial discharge, denoted by Q_{oi}, plus the sum of all of the initially unknown corrective discharges that circulate through pipe *i*, or

$$Q_i = Q_{oi} + \sum \Delta Q_k \qquad (2.43)$$

In equation (2.43) the summation includes all of the corrective discharges passing through pipe i , the initial discharges Q_{oi} must satisfy all of the junctions continuity equations. It is not difficult to establish the initial discharge in each pipe so that the junction's continuity equations are satisfied. However, these initial discharges usually will not satisfy the energy equations that are written around the loops of the network [12].

To establish *NL* energy loop equations around the *NL* loops of the network, in which each discharge plus the sum of corrective loop discharges, $\sum \Delta Q_k$ is used as the discharge. The junction continuity equations are satisfied by the initial discharge Q_{oi} and are not a part of the system of equations. The corrective discharges can be chosen as positive if they circulate around a loop in either the clockwise or counter clockwise direction. It is necessary to be consistent within any one loop, but the sign convention may change from loop to loop, if desired. A corrective discharge adds to the flow Q_{oi} in pipe i if it is in the same direction as the pipe flow, and it subtracts from the initial discharge if it is in the opposite direction.

To summarise how the ΔQ-equations are obtained, replace the Q's in the energy loop equations, equation.(2.38) and equation (2.39) by

$$Q_i = Q_{oi} \pm \sum \Delta Q_k$$

Using the notation ℓ for the *NL* energy equations around the basic loops can be written as,

$$\sum_{i}^{I} K_i (Q_{oi} + \overrightarrow{\Delta Q_1})^{n_i} = 0 \quad \text{(Head loss around loop I)}$$

$$\sum_{i}^{II} K_i (Q_{oi} + \overrightarrow{\Delta Q_2})^{n_i} = 0 \quad \text{(head loss around loop II)}$$

$$\sum_{i}^{\ell} K_i (Q_{oi} + \overline{\Delta Q_\ell})^{n_i} = 0 \qquad \text{(Head loss around loop } \ell\text{)} \qquad (2.44)$$

In which each summation includes only those pipes in the loop designated by the Roman numeral *I, II*... ℓ, and $\overline{\Delta Q_\ell}$ always includes ΔQ_ℓ and also any other $\Delta Q's$ flowing through the pipe for which the terms applies [12].

2.4 Methods of solution (Review of previous work)

2.4.1 Introduction

Pipe networks may include serial pipes, parallel pipes and branching pipes, in addition to elbows, valves, meters and other devices that cause local disturbances and minor losses. There are several calculation techniques available to analyse flow rates and pressures or head losses throughout the pipe system.

One of the first and oldest method and the most widely used method of analysis is the Hardy Cross technique [7]. This method makes corrections to initial assumed values by using a first order expansion of the energy equation in terms of a correction factor for the flow rate in each loop. The process is, of course, repetitive and is dependent on the accuracy of the initial guess, which must be reasonably good if an answer is to be obtained rapidly. This method is well suited for solution by hand

Usually the Hardy Cross method is used to determine heads and flows in pipe network.

Essentially the usual Hardy Cross method consists of "guessing " the flow rate Q

In each pipe and then systematically revising these flow rates based on the fact that algebraic sum of all head losses in each loop should be zero. The computed sum of the head losses around a loop based on the assumed flow rates will ordinarily not be zero. The deviation from zero may be used to calculate a correction. When the correction applied to the assumed flow rates, a better approximation of the true flow is obtained. The correction process is carried out over the network until it is believed that the flow rates are close enough to the real values for the purposes at hand [7,19,23,27].

Numerous computer programs based on the Hardy-Cross procedure have been developed [19,24,25,26]

In certain cases, it has been found that the Hardy Cross method converges very slowly or not at all. This has led McCormick and Bellamy [28] and McCormick [29] to suggest special measures to improve convergence.

A second method which is being applied successfully to hydraulic network analysis utilizes the Newton-Raphson method to formulate a set of simultaneous linear equations which can be solved for flow corrections for each loop in the network. This method is described by Martin and Peters [6,30] for studies of hydraulic networks. The method has been extended by Shamir and Howard [31] to include various hydraulic components in the network such as pumps and valves in place of pipes between two joints. Martin and Peters[6] and Shamir and Howard [31] developed a method of solving for unknown flow resistance with known heads. Epp and Fowler [32] have described a technique using Newton's method to solve a system of simultaneous equations along with information on how to reduce the number of equations required and the input data needed.

Because this method adjusts the flow rate in all the loops simultaneously, convergence using the Newton-Raphson approach is much quicker than that obtained using the Hardy Cross analysis[32,33]. This is especially important when analyzing networks involving large numbers of pipes. However, both methods of analysis require an initial guess for flow distributions, and very bad estimates of these values can lead to slow convergence or, in some cases, a situation where the successive trails do not converge and the solution can not be found.

Other analytical methods have been proposed for hydraulic network problems but have not gained wide acceptance. For example, Warga [34] applied Duffin's [35] work on non-linear networks to hydraulics networks

Direct electrical analogies are also used for hydraulic network analysis, most popular is the fluid network analyser developed by McIllroy [36]. This and other available direct analogue devices are described in a paper by McPerson [37].

Other analytical methods and most widely used to solve hydraulic network problems has been developed in recent years.
The Linear Theory Method described by Wood and Charles [5] is the method used to solve for the pipe flow and can also be regarded as an application of the Newton-Raphson technique in the sub-domain of loops. The number of independent continuity and energy equations equals the number of pipe sections for all network configurations. The resulting equation set is non-linear and is expressed in terms of the unknown flow rates in the pipe sections. The solution is obtained by applying the Newton-Raphson procedure to linearize non-linear terms and solving the resulting system of linear simultaneous equations. But it requires the solution of a large system of equation (number of loops and number of nodes) although reducing the risk of the failures. Wood and Rayes [38], Ormsbee and Wood [39,40], Boulos and Wood [41], Issac and Mills [42] and Nelson [43] developed this method to improve the convergence of the solution. The method has been extended by Jepson and Travallaee

[18] , Jepson and Davis[44] to include various hydraulic components in the network such as pumps and pressure reducing valves.

Many computer programs based on this method are described by Wood [17], Jepson [12], and Larock, Jepson and Watters [13] have described methods for improving the efficiency of solution of some of these methods when applied to a large networks.

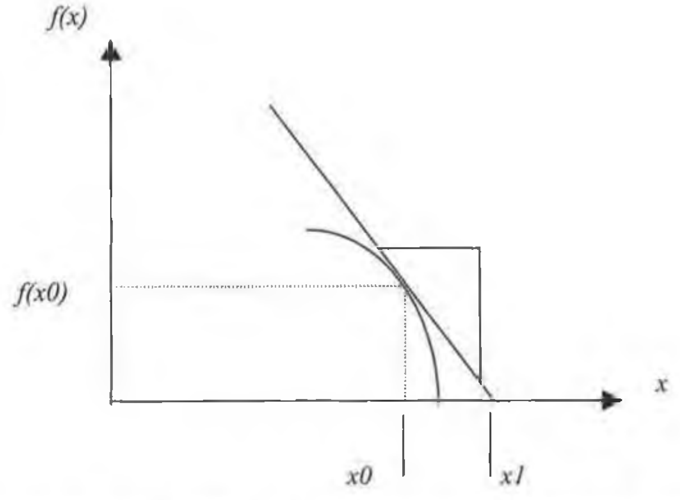
As a first step in developing a computer program using C language, various methods of analysis have therefore been reviewed. Some of these are presented herein.

2.4.2 Newton-Raphson method

The Newton-Raphson method uses an iterative process to approach one root of a function[1,3].

In using the Newton-Raphson method the equation containing the unknown (which will be called x when describing the method in general), is expressed as a function which equals zero when the correct solution is substituted into the equation or $f(x)=0$. The Newton-Raphson method computes progressively better estimate of the unknown x by the formula,

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})} \qquad \text{-----------------------------------} \quad (2.45)$$



Newton's method for finding the root of an equation.[12]

The specific root that the process locates depends on the initial, arbitrarily chosen x-value.

$$\frac{f(x_n)}{f'(x_n)} = \Delta x \qquad (2.46)$$

Here, (x_n) is the current known x-value, $f(x_n)$ represents the value of the function at x_n, and $f'(x_n)$ is the derivative (slope) at x_n. x_{n+1} Represents the next x-value that you are trying to find. Essentially, $f'(x)$, the derivative represents $\frac{f(x)}{dx}$, $(dx = \Delta x)$.

Therefore, the term $\frac{f(x)}{f'(x)}$ represents a value of Δx.

The more iterations that are run, the closer Δx will be to zero (0).

The Newton-Raphson method may be used to solve any of the three sets of equations describing flow in pipe networks which are discussed in next section.

The equations considering:

1-The flow rates in each pipe unknown

2-The head at each junction unknown

3-The corrective flow rate around each loop unknown.

The Newton-Raphson method requires an initial guess to the solution.

The iterative Newton-Raphson formula for a system of equations is ,

$$\overline{x_{n+1}} = \overline{x_n} - D^{-1}\overline{F}(x_n) \qquad (2.47)$$

The unknown vector $\overline{x}$ and $\overline{F}$ f replace the single variables x and function F and the inverse of the Jacobian D^{-1}, replace $\frac{1}{\frac{df}{dx}}$ in the Newton-Raphson formula for solving a single equation.

If solving the equation with the heads as the unknown $(i.e.the H - equations)$ the vector $\overline{x}$ becomes the vector $\overline{H}$.

If solving the equations containing the corrective loop flow rates $(i.e.the \Delta Q - equations)$ $\overline{x}$ becomes $\overline{\Delta Q}$.

The individual elements for vector $\overline{H}$ and vector $\overline{\Delta Q}$ are

$$\overline{H} = \begin{vmatrix} H1 \\ H2 \\ \\ \\ Hn \end{vmatrix} \qquad \text{Or} \qquad \overline{\Delta Q} = \begin{vmatrix} \Delta Q1 \\ \Delta Q2 \\ \\ \\ \Delta Qn \end{vmatrix} \text{ in which } n \ (1,2,3,4\ldots n)$$

The Jacobian matrix D consists of derivative elements, individual rows of which are derivatives of that particular functional equation with respect to the variables making up the column heading [1,3,12,45,46]. For the head equation the Jacobian is,

$$D = \begin{bmatrix} \frac{\partial F_1}{\partial H_1} & \frac{\partial F_1}{\partial H_2} & \cdots\cdots & \frac{\partial F_1}{\partial H_j} \\ \frac{\partial F_2}{\partial H_1} & \frac{\partial F_2}{\partial H_2} & \cdots\cdots & \frac{\partial F_2}{\partial H_j} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial F_j}{\partial H_1} & \frac{\partial F_j}{\partial H_2} & \cdots\cdots & \frac{\partial F_j}{\partial H_j} \end{bmatrix}$$

In which the row and column corresponding to the known head are omitted

The last term $D^{-1}F$ in equation (2.47) contains the inverse of D, since division by matrix is undefined. However in application of the Newton-Raphson method the inverse is never obtained and premultiplied by $\overline{F}$ as equation(2.47) implies. Rather the solution vector $\overline{z}$ of the linear system $D\overline{z} = \overline{F}$ is subtracted from the previous iterative vector of unknowns.

Selecting the $H - equations$ in the following notation, the Newton-Raphson iterative formula in practice becomes

$$\overline{H}_{(n+1)} = \overline{H}_{(n)} - \overline{z}_{(n)} \qquad (2.48)$$

The equivalence of equation (2.48) and equation (2.47) is evident since $\overline{z} = D^{-1}\overline{F}$.

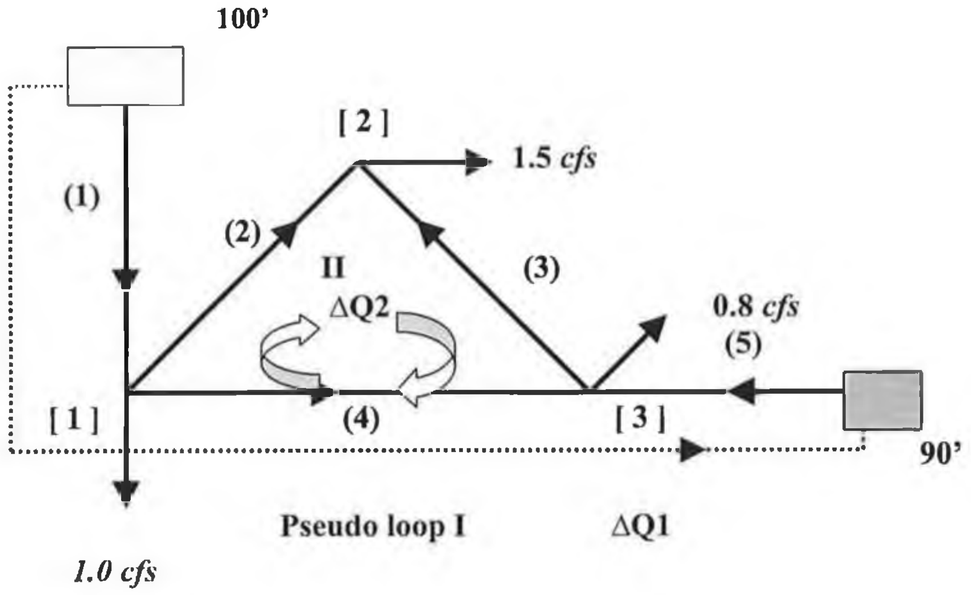
Since fewer computations are needed to solve the linear system $D\bar{z} = \bar{F}$ than to find the inverse D^{-1} obviously equation (2.48) is the form of the Newton method used in practice.

The Newton-Raphson method, therefore, obtains the solution to a system of non-linear equations by iteratively solving a system of equations.

The Newton-Raphson will now be applied in turn to the solution of the H-equation and the Q-*equations*.

2.4.2.1 Solving the H-equations by Newton-Raphson method

The Newton-Raphson method will be illustrated by using it to solve the H-equations for the simple one loop network shown in Figure 2.3 [13].



Pipe	K	*n*
1	7.59	1.936
2	9.63	1.901
3	48.6	1.882
4	39.7	1.768
5	16.50	1.935

Figure 2.3 A 5-pipe, 3-junction network

Simplify the problem the Hazen-Williams equation will be used so that k and n in the exponential formula are constant. Since there are 3 junctions and the heads are unknown and to be determined. one must construct three H-equations.

They are

$$F_1 = \left(\frac{100-H_1}{K_1}\right)^{\frac{1}{n_1}} - \left(\frac{H_1-H_2}{K_2}\right)^{\frac{1}{n_2}} - \left(\frac{H_1-H_3}{K_4}\right) - 1.0 = 0$$

$$F_2 = -\left(\frac{H_1-H_2}{K_2}\right)^{\frac{1}{n_2}} + \left(\frac{H_3-H_2}{K_3}\right)^{\frac{1}{n_3}} - 1.50 = 0$$

$$F_3 = \left(\frac{H_1-H_3}{K_4}\right)^{\frac{1}{n_4}} - \left(\frac{H_3-H_2}{K_3}\right)^{\frac{1}{n_3}} + \left(\frac{90-H_3}{K_5}\right)^{\frac{1}{n_5}} - 0.8 = 0$$

Using $D\bar{z} = \bar{F}$ and $\overline{H}_{(n+1)} = \overline{H}_{(n)} - \bar{z}_{(n)}$ to implement the Newton method.

The Jacobian matrix

$$D = \begin{bmatrix} \frac{\partial F_1}{\partial H_1} & \frac{\partial F_1}{\partial H_2} & \frac{\partial F_1}{\partial H_3} \\ \frac{\partial F_2}{\partial H_1} & \frac{\partial F_2}{\partial H_2} & \frac{\partial F_2}{\partial H_3} \\ \frac{\partial F_3}{\partial H_1} & \frac{\partial F_3}{\partial H_2} & \frac{\partial F_3}{\partial H_3} \end{bmatrix}$$

Using $D\bar{z} = \bar{F}$ and $\overline{H}_{(n+1)} = \overline{H}_{(n)} - \bar{z}_{(n)}$ to implement the Newton method

With an initial estimate of the nodal heads as

$$\{H\}^0 = \begin{bmatrix} H_1 \\ H_2 \\ H_3 \end{bmatrix} = \begin{bmatrix} 93 \\ 85 \\ 88 \end{bmatrix}$$

The solution to

$$\begin{bmatrix} -0.166 & 0.060 & 0.035 \\ 0.060 & -.100 & 0.040 \\ 0.035 & 0.040 & -0.162 \end{bmatrix} \begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \end{bmatrix} = \begin{bmatrix} -1.258 \\ -0.365 \\ -0.382 \end{bmatrix} \text{ is } [Z] = \begin{bmatrix} 15.953 \\ 17.241 \\ 10.088 \end{bmatrix} \quad [H]^{(1)} = \begin{bmatrix} 77.047 \\ 67.759 \\ 77.912 \end{bmatrix}$$

After completing 5 iterations the solution become

$$\begin{bmatrix} -0.210 & 0.052 & 0.124 \\ 0.052 & -.075 & 0.023 \\ 0.124 & 0.023 & -0.174 \end{bmatrix} \begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \end{bmatrix} = \begin{bmatrix} 0.002 \\ -0.001 \\ 0.002 \end{bmatrix} [Z] = \begin{bmatrix} -0.005 \\ -0.001 \\ 0.006 \end{bmatrix}$$

The heads are

$$[H]^{(5)} = \begin{bmatrix} 67.517 \\ 56.793 \\ 67.236 \end{bmatrix} = \begin{bmatrix} H_1 \\ H_2 \\ H_3 \end{bmatrix}$$

Depending on the desired accuracy of this solution, the process might have been terminated three or four iterations. Using these heads the flow rates can be computed.

2.4.2.2 Solving the ΔQ-equations by Newton-Raphson method

In applying the Newton-Raphson method to solve the system of equations, which consider the corrective flow rates in each loop as the unknown $(\Delta Q - equations)$, the same procedure is followed except the unknown vector in equation(2.45) is ΔQ and the Jacobian is

$$\begin{bmatrix} \frac{\partial F_1}{\partial \Delta Q_1} & \frac{\partial F_1}{\partial \Delta Q_2} & \cdots & \frac{\partial F_1}{\partial \Delta Q_L} \\ \frac{\partial F_2}{\partial \Delta Q_1} & \frac{\partial F_2}{\partial \Delta Q_2} & \cdots & \frac{\partial F_2}{\partial \Delta Q_L} \\ .. & .. & .... & .... \\ \frac{\partial F_L}{\partial \Delta Q_1} & \frac{\partial F_L}{\partial \Delta Q_2} & \cdots & \frac{\partial F_L}{\partial \Delta Q_L} \end{bmatrix}$$

With $\bar{z}$ defined as the solution to $D^{(n)}\bar{z}^{(n)} = \bar{F}^{(n)}$ as previous where F now becomes the equation evaluated from the n^{th} iterative values of ΔQ^n, the Newton-Raphson method becomes

$$\Delta Q_{(n+1)} = \Delta Q_n - \bar{z}$$

The Newton-Raphson method will be illustrated in detail by using it to solve the *ΔQ-equations* for the same network shown in figure2.3.

Since there are two loops network, one for the real loop and the other for the pseudo loop as indicated in figure 2.3, there are two corrective flow rates, ΔQ_1 and ΔQ_2 which are unknown, only one real loop energy equation and one pseudo loop equation is needed. Writing the energy equations around these loops, gives the following two simultaneous equations to solve for these two unknowns[13].

The *ΔQ-equations* are

The energy equation a round the pseudo loop is

$$F_1 = K_1(Q_{01} + \Delta Q_1)^{n_1} + K_4(Q_{04} - \Delta Q_2 + \Delta Q_1)^{n_4} - K_5(Q_{05} - \Delta Q_1)^{n_5} - \Delta H = 0$$

in which

$\Delta H = H_2 - H_1$, The difference between the elevation of the two reservoirs, which are, connected the pseudo loop. The energy equation around the pseudo loop becomes

$$F_1 = K_1(Q_{01} + \Delta Q_1)^{n_1} + K_4(Q_{04} - \Delta Q_2 + \Delta Q_1)^{n_4} - K_5(Q_{05} - \Delta Q_1)^{n_5} - 10 = 0$$

The energy equation around the real loop is

$$F_2 = K_2(Q_{02} + \Delta Q_2)^{n_2} + K_3(Q_{03} - \Delta Q_2)^{n_3} - K_4(Q_{04} - \Delta Q_2 + \Delta Q_1)^{n_4} = 0$$

The equations for the Newton method are $[D]\{Z\} = \{F\}$ and

$\{\Delta Q\}^{(n+1)} = \{\Delta Q\}^{(n)} - \{Z\}$ in which the Jacobian and vector of initial discharges are

$$[D] = \begin{bmatrix} \dfrac{\partial F_1}{\partial \Delta Q_1} & \dfrac{\partial F_1}{\partial \Delta Q_2} \\ \dfrac{\partial F_2}{\partial \Delta Q_1} & \dfrac{\partial F_2}{\partial \Delta Q_2} \end{bmatrix} \qquad \{Q_0\}^{(0)} = \begin{Bmatrix} 2.0 \\ 0.9 \\ 0.6 \\ 0.1 \\ 1.3 \end{Bmatrix}$$

Starting the Newton iteration with $\Delta Q_1 = \Delta Q_2 = 0$, then

$$\begin{bmatrix} 80.892 & -11.975 \\ -11.975 & 86.913 \end{bmatrix} \begin{Bmatrix} Z_1 \\ Z_2 \end{Bmatrix} = \begin{Bmatrix} -7.694 \\ -11.378 \end{Bmatrix} \quad \{Z\} = \begin{Bmatrix} -0.117 \\ -0.147 \end{Bmatrix} \quad \{\Delta Q\}^{(1)} = \begin{Bmatrix} 0.117 \\ 0.147 \end{Bmatrix}$$

After another two iterations

$$\begin{bmatrix} 76.103 & -9.091 \\ -9.091 & 73.662 \end{bmatrix} \begin{Bmatrix} Z_1 \\ Z_2 \end{Bmatrix} = \begin{Bmatrix} -0.068 \\ -0.799 \end{Bmatrix} \quad \{Z\} = \begin{Bmatrix} -0.002 \\ -0.011 \end{Bmatrix} \quad \{\Delta Q\}^{(2)} = \begin{Bmatrix} 0.119 \\ 0.158 \end{Bmatrix}$$

$$\begin{bmatrix} 75.163 & -8.188 \\ -8.188 & 71.954 \end{bmatrix} \begin{Bmatrix} Z_1 \\ Z_2 \end{Bmatrix} = \begin{Bmatrix} 0.004 \\ -0.009 \end{Bmatrix} \quad \{Z\} = \begin{Bmatrix} 0.000 \\ 0.000 \end{Bmatrix} \quad \{\Delta Q\}^{(3)} = \begin{Bmatrix} 0.119 \\ 0.158 \end{Bmatrix}$$

From these results we can compute the flow rates in each pipe by adding these corrective flow rates to the initially assumed values, which satisfy the junction continuity equations.

These results are:

$$Q_1 = Q_{01} + \Delta Q_1 = 2.119$$

$$Q_2 = Q_{02} + \Delta Q_2 = 1.058$$

$$Q_3 = Q_{03} + \Delta Q_2 = 0.442$$

$$Q_4 = Q_{04} + \Delta Q_1 - \Delta Q_2 = 0.061$$

$$Q_5 = Q_{05} - \Delta Q_1 = 1.181$$

The Newton-Raphson method, therefore obtains the solution to a system of non-linear equations by iteratively solving a system of linear equations. The Newton-Raphson method does require a reasonably accurate initialisation or it may not converge [3,30,32].

2.4.3 Hardy-Cross Method

Usually the Hardy-Cross method is used to determine heads and flows in a pipe network, and it is considered as one of the oldest used method to analysing pipe networks [7,12,23,24], a description of which can be found in most hydraulics or fluid mechanics textbooks.

This method can be applied to solve the system of heads equations (i.e. heads as unknown), or the system of corrective loop flow equations (i.e. the $\Delta Q - equations$).

The Hardy-Cross method is an adaptation of the Newton-Raphson method, which solves one equation at a time before proceeding to the next equation during each iteration instead of solving all equations simultaneously. In doing this all other ΔQ's except the ΔQ_L of the loop l for which the equation is written are assumed temporarily known. Based on this assumption, the Newton-Raphson method can be used to solve the single equation $F_l = 0$ or for ΔQ_L, or

$$\Delta Q_l^{(m+1)} = \Delta Q_l^m - \frac{F_l^{(m)}}{\frac{dF_l^{(m)}}{d(\Delta Q_l)}} \text{-----------------------}(2.49)$$

It is common in the Hardy-Cross method to apply one iterative correction to each equation before proceeding to the next equation.

After applying one iterative correction to all equations the process is repeated until convergence is achieved. Furthermore, it is common to adjust the initially assumed flow rate in all pipes in the loops of that equation immediately upon computing each ΔQ.

Consequently each equation $F_l = 0$ is evaluated with all ΔQ's equal to zero, and furthermore the previous $\Delta Q_l^{(m)}$ consequently equation (2.49) reduces to,

$$\Delta Q = -\frac{F_l}{\frac{dF_l}{d(\Delta Q_l)}} \qquad (2.50)$$

The superscript denoting iteration numbers in equation(2.49) are deleted in equation(2.50) because only one ΔQ appears. The equation $F_l = 0$ for l loop is the head loss equation around the loop or

$$F_l = \sum K_i Q_i^{n_i} \tag{2.51}$$

The derivative of F_l is,

$$\frac{dF_l}{d(\Delta Q_l)} = \sum \left| n_i K_i Q_i^{n_i - 1} \right| \tag{2.52}$$

Substituting equation (2.51) and equation (2.52) into equation (2.50) gives the following equation by which the Hardy-Cross method computes a corrective a flow rate ΔQ_L for each loop of the network.

$$\Delta Q = -\frac{\sum K_i Q_i^{n_i}}{\sum \left| n_i K_i Q_i^{n_i - 1} \right|} \tag{2.53}$$

If the Hazen-Williams equation is used to define K and n in the exponential formula, then equation (2.53) simplifies to

$$\Delta Q = -\frac{\sum (CHW)_i Q_i^{1.852}}{1.852 \sum \left| (CHW)_i Q_i^{0.852} \right|}$$

The Hardy-Cross method is outlined in the following procedure:

Step 1. Assume an initial flow rate for each pipe such that all junction continuity equations are satisfied.

Step 2. Compute the sum of head loss around a loop of the network. Care must be taken to maintain proper signs. This step computes the numerator of equation (2.53)

Step 3. Compute the denominator of equation (2.53) by accumulating the absolute values of $n_i K_i Q_i^{n_i-1}$ around the same loop.

Step 4. Compute ΔQ by dividing the result from step 2 by the result from step 3

Step 5. Repeat steps 2 through 4 for each loop in the network.

Step 6. Repeat the iterative procedure (steps2 through 5) until all ΔQ's computed are sufficiently small to be neglected. [8,12]

The calculation method is illustrated by the following example problem:

A pipe network can be divided into two flow loops shown in figure 2.4.

Initial guesses of the flow rates in each pipe are selected, obeying continuity at the junction of each section.

Head losses are computed in the clockwise direction. Table 2.7. Tabulates values for the iterative procedure.

For the first iteration the difference for loop (I) is

$$\Delta Q_I = -\frac{1197}{1.852(1320.7)} = -0.49 csf$$

For loop (II),

$$\Delta Q_{II} = -\frac{-5849}{1.852(1593)} = 1.98csf$$

These differences are unacceptable. Adding the ΔQ's to the initial flow rates provides estimates for the second iteration.

For the second iteration,

$$\Delta Q_{I} = -\frac{-143}{1.852(1317)} = 0.06csf$$

$$\Delta Q_{II} = -\frac{-1740}{1.852(1242)} = 0.76csf$$

ΔQ_I Is acceptable; ΔQ_{II} is not acceptable. A third trial is needed for loop (II), so is added to the second iteration flow rates. Finally, for the third iteration convergence is achieved:

$$\Delta Q_{II} = -\frac{-100.6}{1.852(1087)} = 0.05csf$$

This procedure is also widely used with Darcy-Weisbach equation to determine the head losses; however, it is slightly more cumbersome via hand calculations.

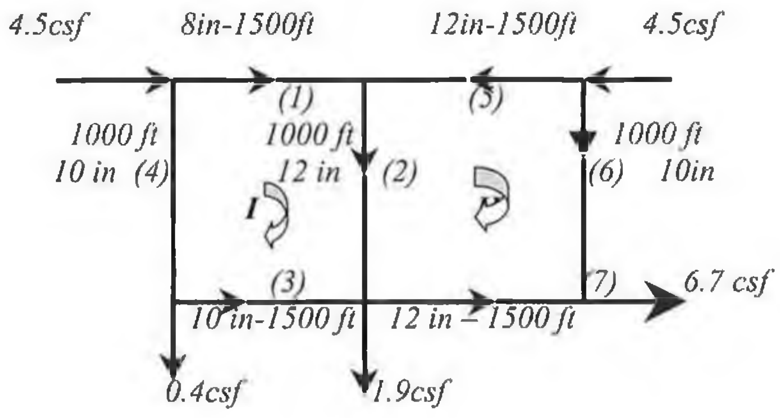


figure 2.4 two pipe networks[12]

Table 2.7. Calculation for the example problem

Iteration	***pipe***	***Q(cfs)***	$KQ_i^{1.852}$	$\lvert KQ_i^{0.852}\rvert$	***pipe no***	***Q (cfs)***	$KQ_i^{1.852}$	$\lvert KQ_i^{0.852}\rvert$
		Loop I					loop II	
1	1	1.50	275.5	183.6	5	-3.5	-1323	378
	2	5.0	2561.1	512.2	6	1.0	130	130
	3	-2.6	-645.4	293.4	7	-5.7	-3265	573
	4	-3.0	-994.4	331.5	8	-5.0	-2561	512
	totals		***1197***	***1320.7***			***-5849***	***1593***
2	1	1.0	130	130	5	-1.5	--275	184
	2	4.5	2107	468	6	3.0	995	331
	3	-3.1	-1057	341	7	-3.7	-1466	396
	4	-3.5	-1323	378	8	-3.0	-994	331
	totals		***-143***	***1317***			***-1740***	***1242***
3	1				5	-0.74	-74.4	100.6
	2				6	1.0	1510	402
	3				7	-5.7	-958	326
	4				8	-5.0	-579	258
	total						**-101**	**1087**

CHAPTER 3

Simulation of pipe networks using linear Theory method

3.1 introduction

The solution to a problem of steady-flow distribution in a pipe network is obtained when the flow satisfies both the continuity equation at each junction and the energy equation in each pipe.

In this project the linear theory method will be described and used in solving the system of equation, which considers the flow rates known.

This system of equation is easy to use if all external flows to the system are known.

The linear theory method will be described first for solving the system of *Q-equation*, thereafter it will be extended to networks containing pumps and reservoirs. For such networks, not all-external flows are known, and must be obtained as part of the solution.

In a network of *NP* pipes and *NJ* junction and *NL* loops, it has been shown that the following identity holds [12,13]:

$$NP = (NJ + NL) - 1 \text{..........................} (3.1)$$

This is true for networks with all closed loops, for open tree –type networks, or combination of both type set is possible to write $NJ - 1$ linear continuity equations

(The additional equation is redundant) for all but one of the junctions in the network stating that the discharge into the junction equals the discharge out of the junction

$$Q_{in} = Q_{out} \text{.................................}(3.2)$$

In which $Q =$ the volumetric discharge. In addition there are energy equations (one for each loop) of the form:

$$\sum h_f = 0 \text{..............................}(3.3)$$

In which h_f represents the head loss in a pipe contained in that loop and is a function of the discharge, Q.

From equation (3.2) and *equation.* (3.3) n simultaneous equations are obtained in terms of the discharge in each pipe. Theoretically, the equations could be solved for the discharge. However, the head loss in pipe *i* is best represented by a relationship

$$\sum h_{f_i} = \sum K_i Q_i^n \qquad (3.4)$$

In which K_i a pipeline constant which is normally a function of line length, diameter and type of pipe material, and n an empirical head loss exponent usually ranging between 1.8 and 2.0 for turbulent flow [5,12,17,18]. This relationship makes each of the *NL* loop equations non-linear and no method is known for the direct solution of this set of simultaneous equations.

3.2 Feature of linear theory method

Linear theory transforms the *NL* non-linear loop equations into linear equations by approximating the head loss in each pipe by,

$$h_{f_i} = \left[K_i Q_{io}^{n-1}\right] Q_i = K_i' \, Q_i \qquad (3.5)$$

In which Q_{io} = the approximate discharge in line *i*. When Q_{io} approaches the actual discharge, Q_i, equation (3.5) becomes an exact expression of the head loss. Using values for approximate discharge to compute the modified pipeline constant K_i' the loop equation can be expressed as linear equations which when combined with the continuity equations yield n linear simultaneous network equations which can be readily solved for the discharge in each line [5,12,18,38].

However, this is an approximate solution as approximate values for the discharge are used to linearise the head loss terms. The computed values for discharge can then be used to compute a new values for the modified pipeline constant, K_i' which are used to obtain a new set of n simultaneous equations which can be solved for improved values for line discharge. This process can be continued until the discharges obtained from two successive sets of calculations do not differ significantly.

It has been found that there are several features of this method, which make it attractive for hydraulic network calculations.

3.2.1 Calculation of Initial flow rates

The method requires an estimate of flow rate in each line .The converge of the solution is greatly affected by the accuracy of the initial estimates and very poor estimates can lead to a situation where the solution will not converge. For the linear theory method it is not necessary to estimate initial flow rates, instead, reasonably accurate initial flow rates can be easily calculated. This is done by assuming that the modified pipeline constant is independent of flow rate [5,12,13,38] and, as a first approximation, is given by

$$K_i' = K_i Q_{io}^{n-1} \qquad (3.6)$$

That is the coefficient K' is defined for each pipe as the product of K_i multiplied by Q_{io}^{n-1}, an estimate of the flow rate in that pipe. Combining these artificial linear equations with the junction continuity equations provide a system of n linear equations which can be solved by linear algebra.

The solution will not necessarily be correct because the Q_{io} 's will probably not have been estimated equal to the Q_i 's produced by the solution. By repeating the process ,after improving the estimate of Q_i ,eventually the Q_{io} 's will equal the Q_i 's after few iteration.

3.2.2 Converge of solution

In applying the linear theory method it is not necessary to supply an initial guess ,as perhaps implied .Instead for the first iteration each K_i' is set equal to K_i, which is equivalent to setting all of the flow rates Q_{io} equal to unity .Wood [5] in developing the linear theory method observed that successive trials always gave a result which converged to the correct solution but the result of successive trial tended to oscillate about the final result .He also noted that the average of two successive trials gave a result very close to the final value of the flow rate. Therefore, the average values of the two prior sets of calculations for flow rate were used to compute the best value of the discharge for the trial and the modified pipe line constant, K_i', which was used for the next trial [5,12,13].

This is expressed as

$$Q_{io} = \frac{Q_{i-1} + Q_{i-2}}{2} \qquad (3.7)$$

In which Q_{i-1} = the flow rate obtained from the previous trial for the line I and

Q_{i-2} = The flow rate obtained for the trial previous to that

The first step is to obtain *K* and *n* for the exponential formula for a range of flow rate to be realistic.

In implementing the method in the computer program the first values used for *K* and *n* may be obtained from the Hazen-William equation (2.14 and 2.15)[12,13].

To illustrate the Linear Theory Method, consider the small 5-pipe network shown in figure 3.1. Since no supply sources are shown for this network, only *NJ-1* junction continuity equations are available.

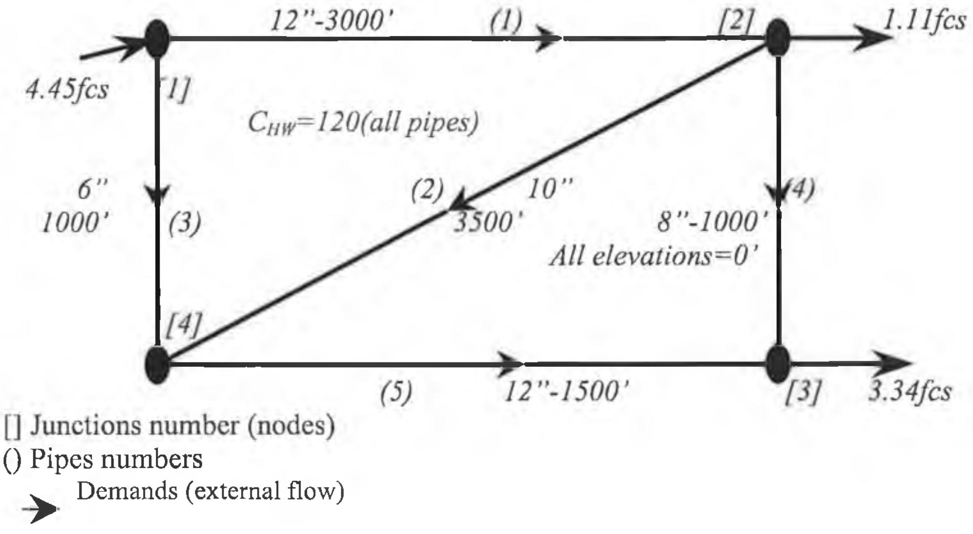


Figure 3.1 small network [12]

Writing these continuity equations for nodes (junctions) 1,2, and 3 leads to

$Q_1 + Q_3 = 4.45$

$-Q_1 + Q_2 + Q_4 = -1.11$

$-Q_4 - Q_5 = -3.34$

The continuity equation at the junction 4 is $-Q_3 - Q_2 + Q_5 = 0$. However, this equation is not independent of the other three junction equations since it is, except for the sign, the sum of these three equations. Now using the Hazen-William equation to define the head loss in each pipe and in expressing these head losses, the exponential equation will be used. From equation (2.14) the K coefficients for the exponential formula are the following

K_1=*2.018, K=5.722,K=19.674,K=4.847,K=1.009*

The energy equations around the two real loops may be written as

$$K_1Q_1^{1.852}+K_2Q_2^{1.852}-K_3Q_3^{1.852}=0$$

$$K_4Q_4^{1.852}-K_5Q_5^{1.852}-K_2Q_2^{1.852}=0$$

These two energy equations are obtained by starting at junctions 1 and 2, respectively, and moving around the respective loops *I* and *II* in a clockwise direction. If the assumed direction of flow opposes this traverse, a minus sign precedes the head loss term for that pipe. These simultaneous equations, such as those above, are called *Q-equations* because it is the *Q's*, that are the set of primary variables.

To solve the system of these simultaneous equations Gauss elimination technique is used, which is one of most widely used methods to solve simultaneous equations, and it will be discussed as part of the simulation (see appendix A).

After the *Q's* are found, the head losses in each pipe can be determined. From a known head or pressure at one junction it is then a routine computation to determine the heads and pressures at each junction throughout the network.

In some problems the external flows may not be known as was assumed in the above example. Rather, the supply of fluids may be from reservoirs and/or pumps. The amount of flow from these individual sources will not only depend on the demands, but also will depend upon the head losses throughout the system. Method for incorporating pumps and reservoirs into a network analysis using Linear Theory Method will be illustrated in the next section.

3.3 Including Pumps and Reservoirs into Linear Theory Method

All applications of the linear theory method described previously were for networks in which the external flows were assumed known. In practice this may not be the case. Rather the amount of flow being supplied from different reservoirs or pumps will depend upon the heads and flows throughout the network. Consequently the utility of the linear theory method can be enhanced by extending it to handle supply source from reservoirs or pumps, and allow booster pumps to exist within pipelines.

Each pump (not a booster pump) and each reservoir from which flow enters or leaves the system introduces an additional unknown that must be solved for in solving the network. Since pumps and reservoirs must be connected to the network by a pipe through which they supply the flow, it is natural to let the flow rates in these connective pipes be the additional unknown. However, elevation of reservoirs, and the elevations of reservoirs from which pumps obtain the fluid plus the pump characteristics (i.e. h_p versus Q_p) are known for pumps. Therefore equations are needed which relate these known to the connective pipe flow rates.

If one reservoir and one pump supply the flow to the network, as shown in the sketch below, such that flows in two connective pipes becomes additional unknowns, one additional equation is necessary beyond the NJ available continuity equations and the NL available energy equations.

A convenient means for obtaining this additional equation is by defining a so called

Pseudo loop, which connects the two reservoirs by "no flow", pipe as illustrated below. Note those two pumps and / or reservoirs must be present or the network reduces to one for which all external flows are known.

Consequently such pseudo loops can be always be defined, because at least two reservoirs and / or pumps must exist in the network if all external flow rates are not known. The additional needed equation (or equations if more than two pumps and /or reservoirs are present), comes from the energy equation around this pseudo loop. Around pseudo loops the sum of the head loss is not equal to zero but must account for the difference in reservoirs elevations or head produced by the pump or pumps [12,18].

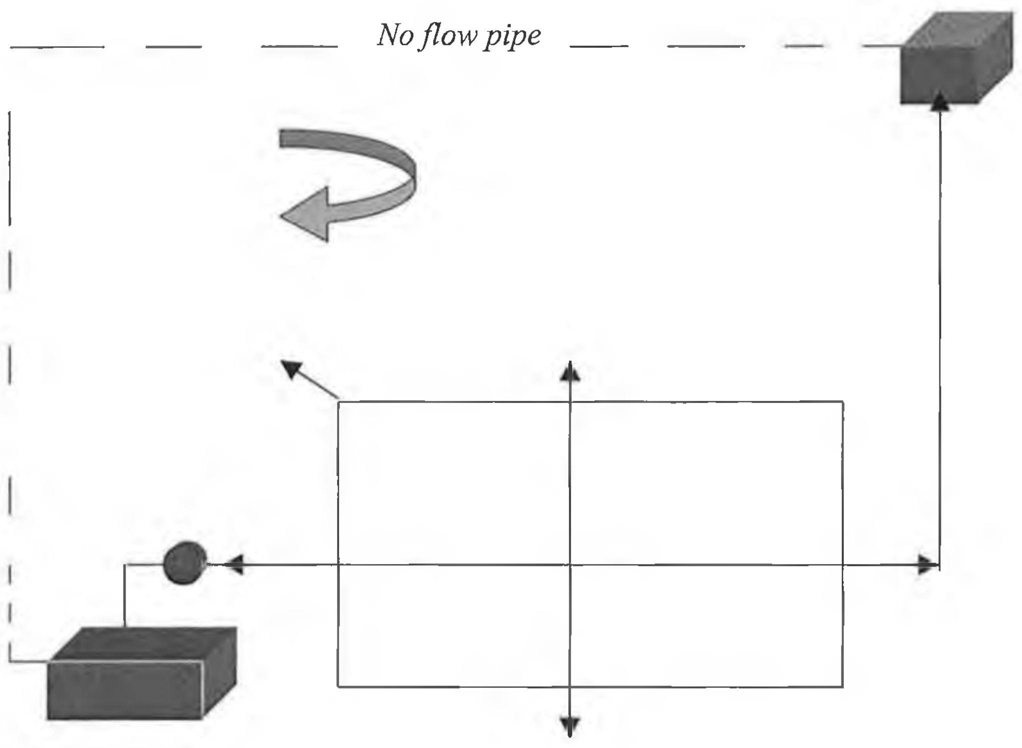


Figure 3.2 small pipe network including pumps

Around the pseudo loop in the diagram above the energy equation is

$$\sum_i K_i Q_i^{ni} \pm \sum_i h_p = \Delta H \text{..................................}(3.8)$$

which represents the energy equation around pseudo loop containing two reservoirs and/or source pumps.

A number of alternative methods might be used to quantify the head hp produced by the pump. The method used herein [12,18] will approximate hp over its working range by quadratic equation of the form

$$h_P = AQ^2 + BQ + H_o \text{..}(3.9)$$

in which A , B , and H_o are constant obtained by fitting the pump characteristic curves When equation (3.9) is substituted into equation (3.8) a non-linear equation results which contains only flow rates in pipes of the network as the unknowns .In this form the linear theory method does not gives rapid convergence as it does when pumps or reservoirs are present. The system of equation will therefore be modified [18] to allow the linear theory method to converge rapidly.

To improve this situation a transformation of variables is needed so that unknown which replace h_P in equation (3.8) has an exponent value close to the rest of the $N's$ [18].

The transformation is

$$G = Q + \frac{B}{2A} \qquad (3.10)$$

in which G is the new transformation variable

This transformation replaced h_p by

$$h_p = AG^2 + h_o \qquad (3.11)$$

in which

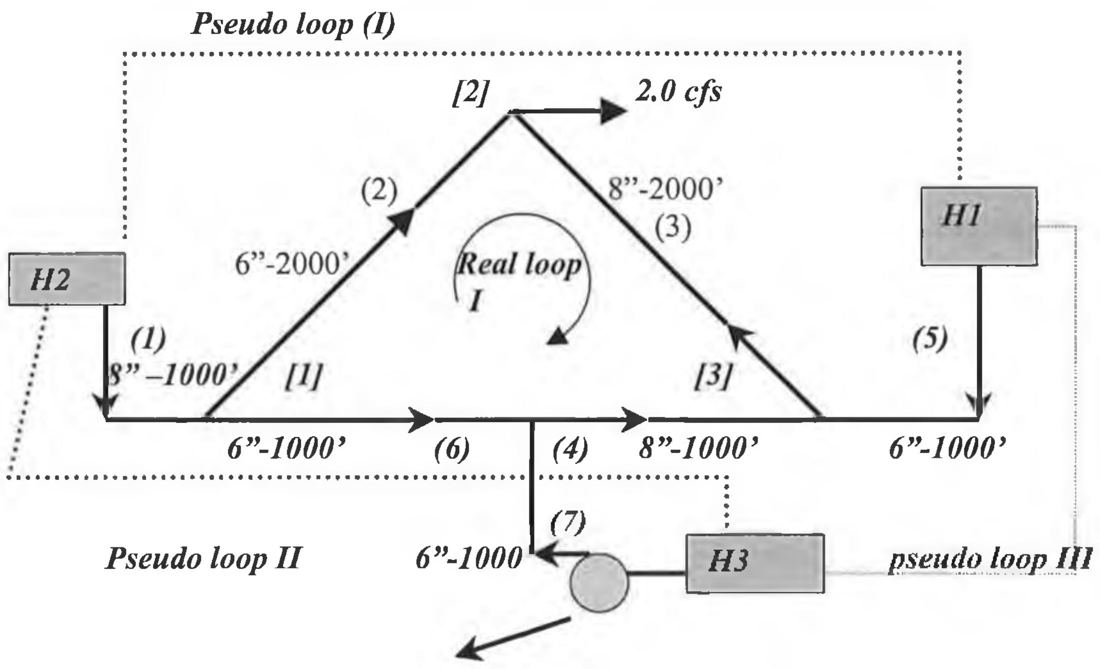
$$h_o = H_o - \frac{B^2}{4A} \qquad (3.12)$$

Obviously the exponent of G is close to the value of a typical n in the exponential formula [12,18]. So the modified equation is

$$\sum K_i Q_i^{ni} \pm \sum AG^2 = \Delta H \pm \sum h_o \text{ ------------------------------}(3.13)$$

Each term in equation (3.13) is similar to typical terms in the energy equation written around real loops. The only problem now is that for each pump an additional unknown G has been introduced. However, the transformation equation, equation (3.13) , is a linear equation which relates G to the flow rate in the line containing the pump to the network. By adding these additional equations to the system as many equations are produced as unknowns exist and a solution can be obtained.

To illustrate that, consider the six-pipe network, one loop network supplied by one pump and two reservoirs as shown in Figure 3.3 [12].



$$h_P = -10.33Q_P^2 + 2.823Q_P + 22.29$$

Figure 3.3. Small 6-pipes network [18]

Since there are six pipes in this network there will be six unknown flow rates, plus one additional unknown, i.e. the G of equation for the pump which supplies the flow from the reservoir 3.

Four junction continuity equations are available. They are the following:

$-Q_1 + Q_2 + Q_6 = 0$ (3.14a)(1)

$-Q_2 - Q_3 \quad = -2.0$ (3.14b)(2)

$Q_3 - Q_4 - Q_5 = 0$ (3.14c)(3)

$Q_4 - Q_6 - Q_7 = 0$ (3.14d)(4)

The number of energy loop equations is *NL=NP-NJ=7-3=4* (one for the real loop and the other three for the pseudo loops connection reservoirs with no pipe flow. But there are 7 unknowns, so one of the pseudo loop will be ignored since only two pseudo loops required.

The number of pseudo loop equation generally available equals the sum of the reservoirs and source pumps minus one or

Ls=Nr+NPS-1 (3.15)

A possibility is one pseudo loop connecting the reservoirs supplying pipes 1 and 5 through pipes 1,6,4,and 5; and the other connects the pump reservoir and the reservoir supplying pipe 1 through pipes 1,6, and 7.

The three energy equations are

$K_2Q_2^{n_2} - K_3Q_3^{n_3} - K_4Q_4^{n_4} - K_6Q_6^{n_6} = 0$ *(real loop)* (3.16 a)

$$K_1Q_1^{n_1} + K_6Q_6^{n_6} - K_7Q_7^{n_7} + h_{P1} = \Delta H = H_2 - H_3 = 100\text{-}95 \quad \textit{(pseudo loop 4 II)} \quad (3.16\text{ b})$$

$$K_1Q_1^{n_1} + K_6Q_6^{n_6} + K_4Q_4^{n_4} - K_5Q_5^{n_5} = \Delta H = H_2 - H_1 = 100\text{-}105 \quad \textit{(pseudo loop I)} \quad (3.16\text{c})$$

The head produced by the pump can be defined by a second-order polynomial passing through three points of the pump curve as described earlier, so

$$h_{P1} = AQ^2 + BQ + H_o$$

Or

$$h_{P1} = AG^2 + h_o$$

Equation (3.16b) become

$$K_1Q_1^{n_1} + K_6Q_6^{n_6} - K_7Q_7^{n_7} + AG_1 = \Delta H - h_o = 100 - 95 - h_o$$

the additional unknown is G , by applying the equation (3.10) , the transformation equation can be written as follow

$$-Q_7 + G_1 = \frac{B}{2A} \qquad (\textit{Transformation equation}) \qquad (3.17)$$

there are now eight independent equations that contain the seven unknowns discharges $Q_1, Q_2, \ldots Q_7$. Plus the additional unknown G.

In applying the linear theory method, the three nonlinear energy equations are linearized as described previously, and the resulting system solved.

In summary then, if pumps or reservoirs exist in a pipe network, a solution by the linear theory method accomplished as follow:

1- *NJ* linear junction continuity equations are written

2- *NL* non-linear energy equations are written around the real loops of the system

3- Additional pseudo loops are defined which connect supply reservoirs or reservoirs from which pumps obtain their supply by no flow pipes.

Energy equations are written around these pseudo loops. These energy equations contain a new unknown G_i for each pump in the network. The number of these pseudo loops must equal the difference between the number of unknown flow rate, i.e. *NP* and *(NJ+NL)*.

4- as many additional linear equations of the form $G = Q + \frac{B}{2A}$ are written as pumps exist

5- The non-linear energy equations are linearized by defining coefficients K'' of the Q's equations which are obtained by $K'' = KQ_M{}^{n-1}$ and coefficient K''_G for the G unknown are obtained by $K''_G = AG$

6-the resulting system is solved iteratively, adjusting the coefficients as described earlier to reflect the average of the flow from the past two solutions until convergence occurs [12,18,44].

3.4 Including Pressure Reducing Valves into Linear Theory Method

A pressure reducing valves (denoted **PRV**) is designed to maintain a constant pressure downstream from it regardless of how large the upstream pressure is.

The exceptions to this occurrence are: (1) if the upstream pressure becomes less than the valve setting, and (2) if the downstream pressure exceeded the pressure setting of the valves so that if the PRV were not present the flow would be in the opposite direction of the valve. If the first condition occurs, the valve has no effect on flow condition. The PRV acts as a check valve, preventing reverse flow if the second condition occurs by preventing reverse flow, the PRV allows the pressure immediately downstream from the valve to exceed its pressure setting.

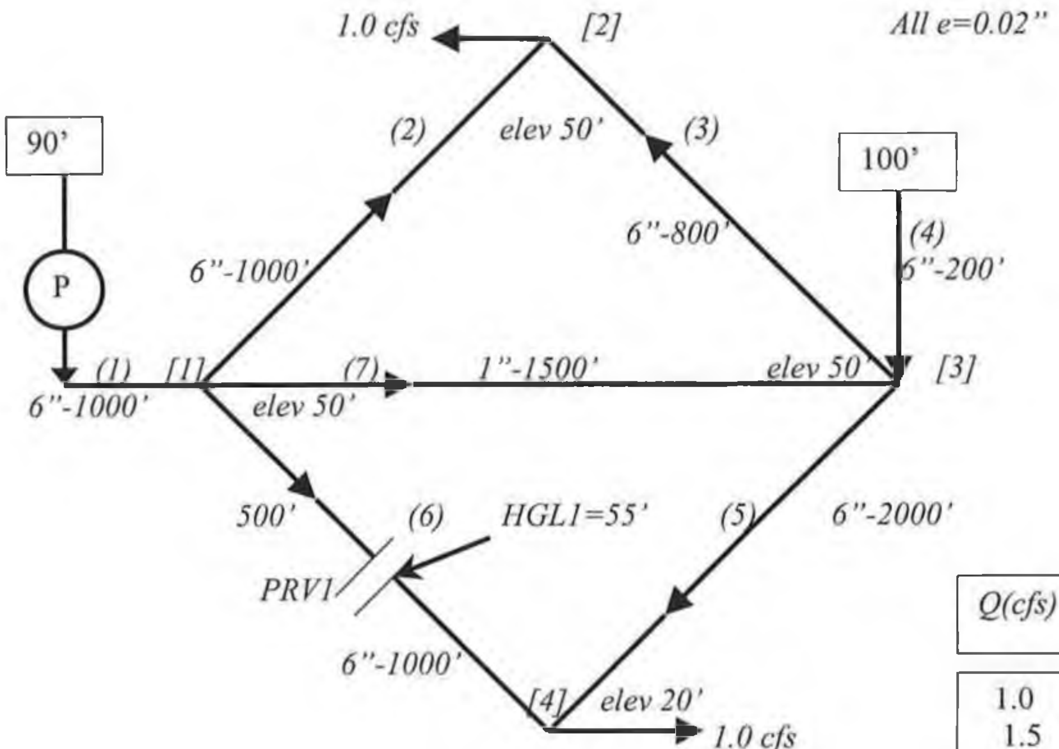
PRV are used to reduce in portions of a pipe distribution system if the pressure would otherwise be excessive, and they may also be used to control from which source of supply the flow comes under various demand levels. In the latter applications the PRV acts as a check valve until the pressure is reduced to critical levels by large demands at which time additional sources of supply are drawn upon [8,11,12,44].

The analysis of a pipe network containing one or more PRV 's must be capable of determining which of these conditions exist. Methods for accomplishing this, which are consistent with the linear theory method, are discussed in this section.

The procedure for including PRV into linear theory method can be summarized as follow:(1) write the junction continuity equations in the usual manner, ignoring the PRV's; (2) replace each PRV with an artificial reservoir which has a fluid surface elevation equal to the HGL-elevation that is the sum of the pressure head set on the PRV and its elevation in the pipeline; finally (3) write the energy equations around the loops of this modified network [12,13,44]. The resulting equations describe the normal mode of operation.

To illustrate that consider the seven-pipe network in figure 4.6[44], in which a PRV exist in pipe 6 , located 500 ft. downstream from junction 1, the upstream end of this pipe.

Since a PRV is a directional device , one must always identify the upstream and downstream ends of the pipe containing it.



Q(cfs)	Head(ft)
1.0	60
1.5	55
1.6	48

Figure 3.4 A seven-pipe network [8]

The system of *Q-equation* for this network consists of four junction continuity equations and three energy equations obtained from loops. The junction continuity equations are identical to those that would be written if the PRV were not present and they are:

$-Q_1 + Q_2 + Q_6 + Q_7 = 0$ (3.18a)

$-Q_2 - Q_3 = -1.0$ (3.18b)

$Q_3 - Q_4 + Q_5 - Q_7 = 0$ (3.18c)

$-Q_5 - Q_6 = -1.0$ (3.18d)

In forming the loops for the energy equations, we modify the network so the upstream portion of the pipe containing the PRV is removed and the PRV is replaced by a reservoir with a fluid surface elevation equal to the HGL of the pressure setting of the PRV as shown below.

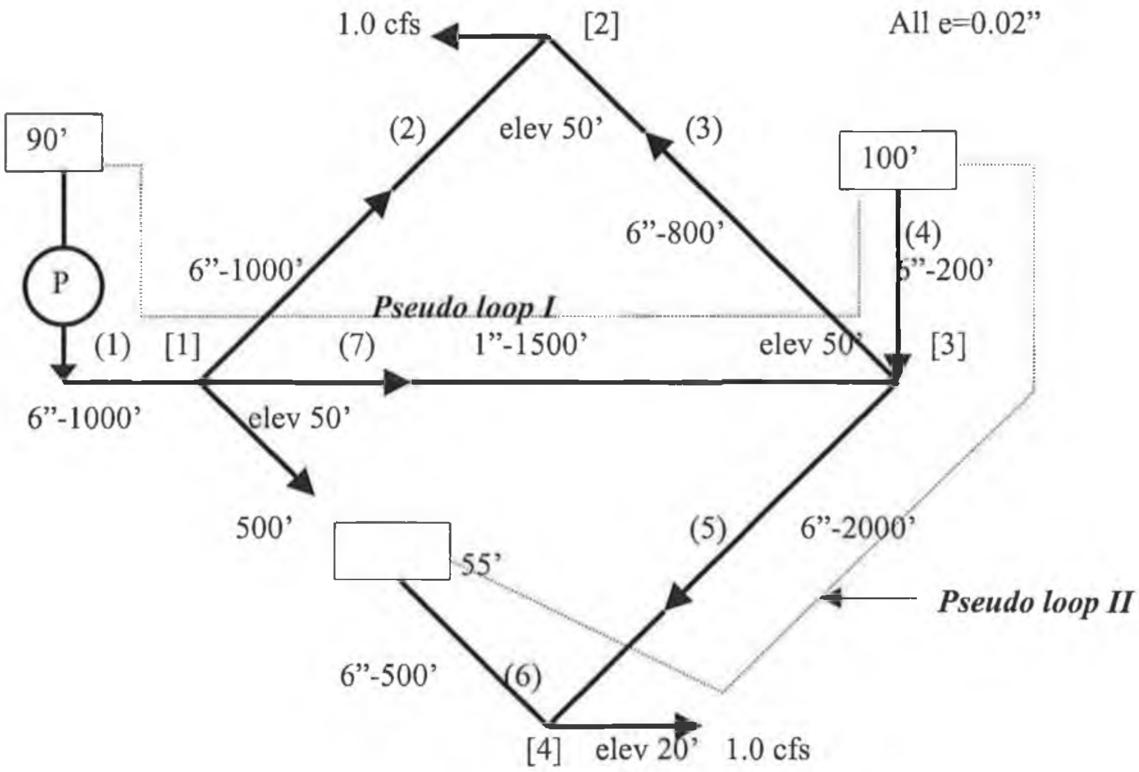


Figure 3.5 A modified network [8]

Of the three loops that exist in this modified network, only one is a real loop, which traverses pipes 2, 3, and 7. Two pseudo loops also exist. One pseudo loop connects the two original supply sources. This loop can start at the reservoir and the end at the source pump so it includes pipes 4, 7, and 1. The second loop must extend from the artificial reservoir created by the PRV to one of the other supply sources. The shortest path for this second pseudo loop will traverse pipes 4,5,and 6. Note that with pipe 6 disconnected from junction 1, only one real loop is available whereas two independent real loops existed before. The real loop, which is lost through the disconnection, is compensated for by the additional pseudo loop from the artificial reservoir created by the PRV.

A modified loss coefficient K' will be used to denote this change in the exponential formula. The new coefficient K' equals the K for the pipe containing the PRV, multiplied by the ratio of the pipe length from the PRV to the pipe's downstream end divided by the total pipe length [12,13,44], or

$$K' = K(L_d/L) \qquad (3.19)$$

Or

$$K_6' = K_6(500/1000) = 0.5K_6 \qquad (3.20)$$

The energy equations are

$$K_2Q_2^{n_2} - K_3Q_3^{n_3} - K_7Q_7^{n_7} = 0 \qquad \textit{(Real loop)} \qquad (3.21a5)$$

$$K_4Q_4^{n_4} - K_7Q_7^{n_7} - K_1Q_1^{n_1} + h_{P1} = 100 - 90 \qquad (3.21b)$$

the head produce by the pump can be defined by a second-order polynomial passing through three points of the pump curve, or

$h_{P1} = AQ^2 + BQ + H_o$

So equation (3.21b) becomes

$K_4 Q_4^{n_4} - K_7 Q_7^{n_7} - K_1 Q_1^{n_1} + AG_1^2 = 100 - 90 - h_{o1}$ *(Pseudo loop I)*

$K_4 Q_4^{n_4} + K_5 Q_5^{n_5} - K_6' Q_6^{n_6} = 100 - 55$ *(Pseudo loop II)* *(4.21c7)*

the number of equations which are available always equals the number of unknown flow rates Q. using this scheme, eight equations(eight are used instead of seven because one equation is added by the pump transformation as described earlier) needed for a solution by the Linear Theory Method.

The transformation equation is

$G_1 - Q_1 = (B/2A)$ (3.21d)

upon solving these eight equations by the linear theory method, using the procedure described previously, the following solution results :

PIPE OUTPUT

PIPE NO	NODES FROM	TO	LENGTH Feet	DIAMETER Feet	FLOW RATES (Q) Feet3/s	HEAD LOSS Feet
1	0	1	1000.00	0.50	1.108988	27.220783
2	1	2	1000.00	0.50	1.068526	25.289313
3	3	2	800.00	0.50	-0.068526	-0.101441
4	0	3	200.00	0.50	0.891012	3.530955
5	3	4	2000.00	0.50	0.966706	41.486069
6	1	4	500.00	0.50	0.033294	0.017024
7	1	3	1500.00	0.08	0.007167	25.390755

CHAPTER 4

Analysis and program Development

4.1 Introduction

The computer program described in this chapter is written to analyse steady state flows and pressures for pipe distribution systems. The program can be applied to other liquids, but does not generally apply to gas flow unless the assumption of constant density is acceptable.

The program is written to accommodate any piping configuration and various hydraulic components such as pumps, valves (including check valves), any component which produce significant head loss (such as meters, bends, etc.) and pressure regulating valves. Computations can be carried out using English units of CFS, GPM, or Standard International (SI) units.

4.2 Program Analysis

This program is written to compute the flow rate in each pipe in the network and therefore to compute the head loss in each pipe. To simplify the problem it is necessary to describe the features of the piping system using data, which assigns numerical values to the pertinent system characteristics. Part of this data refers to the physical characteristics of the pipe system components and the rest to pressure and flow requirements imposed on the system.

Before analysis, a network must be defined in terms of units such as pipes and pumps that the program allows for.

4.2.1 Representation of Networks

This section includes a general description of pipe system configuration and pipe system parameters which require data input.

☑ **Pipe system geometry**

✓ ***Pipe section***

The principal element in the pipe system is the pipe section which of constant diameter sections that can contain fittings such as bends, valves and pumps .

The end points of a pipe section are called nodes and are classified either as junction node or fixed grade node.

✓ ***Junction node***

A node where two or three pipes meet or where flow is input in or removed from the system.

If a pipe diameter change occurs at a component such as a valve or a pump, this point is a junction node.

✓ ***Real loop***

A closed pipe circuit with no closed pipe circuits contained within it.

✓ ***Pseudo loop***

Which connects one reservoir to another one or to a source pump

Consequently such pseudo loops can always be defined, because at least two reservoirs and /or pumps must exist in a network if all external flow rates are not known.

If the junctions, real loops, and the pseudo loops are identified as described above the following holds for all pipe system: $NP = NJ + NL$

NP = number of pipes

NJ = number of junction

NL = number of loops

in which

$NL = NL_{Real} + NPS$

Where

NL_{Real} = no of real loops

NPS = no of pseudo loops

$NPS = N_{reservoir} + N_{pumps} - 1$

$N_{reservouir}$ = no of the reservoirs

N_{pumps} = no of source pumps

☑ Pipe System Components

Data regarding the physical characteristics of the components in the pipe system must be obtained prior to making compute analysis.

A general description of the components, which are incorporated into the program and the necessary data, follows:

- ***Pipe section***

The total length, inside diameter and roughness of each pipe section must be input as data. The designation of pipe roughness depends on the head loss equation used.

There are two major methods to compute the line losses:

a) *Hazen-William equation*:

Because most users are primarily interested in water distribution, the Hazen William equation, which was developed primarily for this purpose is normally used to compute line losses.

The Hazen William equation is:

(ES units)

$$h_f = \frac{4.73L}{C_{HW}^{1.852} D^{4.87}} Q^{1.852}$$

If this expression is to be employed, the roughness coefficient for this expression must be input as data for each pipe .

This coefficient depends on the type of the condition of each pipe.

b) Darcy-Weisbach equation:

This expression can be applied to systems transporting water and other liquids other than water. If this option is employed, the roughness for each pipe section corresponding to the Darcy-Weisbach expression must be input as data as well as the kinematics viscosity of the liquid for the system.

The Darcy –Weisbach equation is,

$$h_f = f\frac{LV^2}{2gD}$$

$$\frac{1}{\sqrt{f}} = 1.14 - 2\log_{10}\left(\frac{e}{D} + \frac{9.35}{R_e\sqrt{f}}\right)$$

where

$f = friction \quad factor$

$R_e = \mathrm{Re}\,ynolds \quad number$

$e = roughness(ft)$

- ***PUMPS:***

A pump can be included in any line of the pipe system .The characteristics of pumps can be described as follow.

☑ Points of operating data.

Pump characteristics can usually be fitted approximately with a parabolic equation An exponential curve can be fit to this data to obtain characteristic curve describing the pump operation of the form:

$$h_p = AQ^2 + BQ + H_o$$

in which A, B, and Ho are constants for a given pump and might be determined by fitting h_p to the three points taken from a pump characteristics curve using Least square method [1,3] (see appendix A).

- *Minor loss components*

The head loss h_L caused by a minor loss is proportional to the velocity head.

$$h_L = \frac{K_L}{2gA^2} Q^2$$

The loss coefficient K_L is analogous to fL/D. In fact, some prefer to express loss coefficient as an equivalent pipe length:

4.2.2 Algorithm for the solution of the linear theory method

Pipe network equations for steady state analysis have been commonly expressed in two ways. Equation which express mass continuity and energy conservation in terms of the discharge in each pipe section have been referred to as real loop and pseudo loop equations. In terms of the unknown discharge in each pipe, a number of mass continuity and energy equations can be written equalling the number of pipes in the system. For each junction node a continuity relationship equating the flow into the junction to the flow is written as:

$$\sum (Q_i)_{in} - \sum (Q_i)_{out} = C$$

in which C is the external flow at the junction (commonly called consumption or demand). C is positive if flow is into the junction and negative if it is out from the junction .

For each primary loop the energy conservation equation can be written for pipe sections in the loop as follows:

$$\sum_{l}^{L} K_l Q_l^{n_L} = 0$$

where K is a pipeline constant which is a function of line length (L), diameter (D), and roughness e, or friction factor (f), and n is an exponent. The values of K and n depend on the energy loss expression used for the analysis.

For the Hazen-William equation

$$K = \frac{4.73L}{C_{HW}^{1.852} D^{4.87}}$$

the exponent $n=1.85$.

For the Darcy-Weisbach equation

$$K = \frac{fL}{2gDA^2}$$

and the exponent n is in the range (1.85 to 2.0)

The minor loss in pipe section (h_L)is given as

$$h_L = K_{Lm} \frac{Q^2}{2gA^2}$$

where K_{lm} is a function of the sum of the minor loss coefficients for the fitting in the pipe section.

That is true for loops that do not include pumps, if there are pumps in the loop then the energy equation states that the sum of the energy losses around the loop equals the energy put into the liquid by a pump, or

$$h_f = (\sum \frac{fL}{2gDA^2} + \sum \frac{K_L}{2gDA^2})Q^2 \pm h_p$$

or

$$h_f = (\sum \frac{fL}{2gDA^2} + \sum \frac{K_L}{2gDA^2})Q^2 \pm (AQ^2 + BQ + H_o)$$

The linear method is based on a simultaneous solution of the basic equations for the pipe system. Since the energy equations for the loop equation s are non-linear these equations are first linearized . This is done by approximating the head in each pipe as

$$h_{f_i} = \left[K_i Q_{io}^{n-1}\right] Q_i = K_i' Q_i$$

The first step is to obtain K and n for the exponential formula , that is done by computing the first value of K and n from the Hazen-William equation.

For the first iteration each K' is set to equal to K_i, which is equivalent to setting all flow rates Q_{io} equal to unity [5,12,13,18,44].

Combining these artificial linear loop equations with the continuity equations provide a system of n linear equations which can be solved by linear algebra.

The solution will not necessarily be correct because of Q_{io} 's will probably not have been estimated equal to the Q_i's produced by the solution. By repeating the process, after improving the estimates to Q_i,eventually the Q_{io}'s will equal the Q_i's, after the iteration the correct solution has been obtained.

4.3 Computer program

The computer program is written in C language, to solve the basic pipe system equations using the linear theory method .

Basically the program reads input data defining parameter values for each pipe and pressure and flow specifications.

Several items should be noted about this program:

- ✓ different type of liquids are applied to the program
- ✓ the basic equation used for the simulation is Darcy-Weisback equation unless otherwise specified
- ✓ Unconditional number of trials allowed in the program and it is often depends on the accuracy specification.
- ✓ The calculation continues until a relative accuracy of 0.0000001 is attained unless otherwise specified.

The basic system equations are solved using spare matrix.

A complete listing of the C program including all functions (subroutines) is presented in appendix B

4.3.1 Program algorithm

The major tasks performed by the program are the following

1 Determine type of operation to be performed and entering file name, there is a list of choice to choose such as display the existing file or exit the program.

2 Read the input data that defines the network.

3 Develop from this information the system of *Q-equations*, i.e., the junction continuity equations and the energy equation around pseudo and real loops of the network.

4 Make use of the linear theory method to transform the non-linear energy equation into linear equations as described earlier. Set up the arrays and solve the simultaneous linear equations using Gauss elimination method.

5 Obtains the head loss at each pipe after the pipes flow rates have been found.

6 Write the solution results in tables that can be readily understood.

4.3.2 Program Variables

The data requirements for the program are as follows:

Input variables

Detailed instructions on the preparation of input data are as follow:

❑ System input data requirement

The input data for the system to solve the network equations are as follow

a) the type of equation will calculate the head loos friction(Type)

 i. if the Darcy-Weisback equation is used Type----1

 ii. if the Hazen-Williams equation is used Type----2

b) after the programmer indicates the type of equation to calculate the head friction losses, the program require the network elements to be read in. The network elements variables are as follow

1) The number of pipes in the network, *NP* (int *NP*) which indicates how many pipe in the network.

2) The number of junctions in the network, *NJ* (int *NJ*) which indicate how many junctions in the network.

3) the number of real loops in network, *NL* (int *NL*).If the network is branching network .i.e. there is no closed loops in the network the user must read in the *NL* equal to 0

4) If the network contains pumps, the program requires to read in how many pumps in the network, *NPUMP* (int *NPUMP*).

5) If there are tanks or reservoir in the network, Number of reservoirs in network, *NoR* (int *NoR*) must be read in.

6) if there are pressure reducing valves in the network, the data to be read in is to indicates how many PRV in network , otherwise , the PRV is 0

7) If the data input for the program is available in the SI, Or English unit. The program is capable to use both of the unit system.

Table 4.3.1 shows type of units implemented on the program.

Input data denotation	Description
NUNIT (int NUNIT)	The system uses the units English unit (if D(feet),L(feet))----0 (if D(inch),L(feet))---1 SI units (if D,L in (meters))---2 (if D(centimetre),L(meter))--3

❑ Fluid characteristic data

The program requires reading in the input data to describe the physical characteristic for the fluid in the network such as the kinematics viscosity, the specific weight , and type of fluid.

❑ Pipe data

For each pipe in the network, every pipe is given pipe number, diameter, length, and roughness. The roughness may be either the equivalent sand roughness *e* (in the same units as the pipe diameter) for use in the Colebrook-White and Darcy-Weisback equations, or a Hazen-Williams CHW. If the pipe include any minor loss device, the value of minor loss coefficient is required to be read in.

❑ Junction data

For each junction in the network, each junction is given junction number. The input data required to describe every junction.

They are as follows:

1- First, if there is a demand at the junction, there are four options the read in the demand in terms of unit. For example, if the demand in cubic feet per second the option is available by indicating the demand in CFS

2- If the demand leaves the junction, a minus sign is given, and a positive sign if the opposite.

3- Number of pipes around the junction, and a list of these pipe numbers with a minus sign if the flow is from the junction. These information are used to define the junction continuity equations.

❑ *Real loop data*

For every real loop (closed loop) in the network, the number of pipes in that loop, and a list of these pipes must be entered. A negative sign must precede the pipe number if the direction around the loop opposes the assumed direction of flow in the pipe.

❑ *Pseudo loop data*

The pseudo loop is a loop with no flow pipe, which connects reservoirs. The program will compute how many pseudo loops are needed for the network.

It the responsibility of the user to decided which path to suggested connecting the reservoirs.

For every pseudo loop in the network, the number of pipes in the suggested path, and a list of these pipes must be read in. A negative sign must precede the pipe number if the direction around the loop opposes the assumed direction of flow in the pipe.

In addition, the data for the pseudo loop contains the elevation of each reservoir

❑ *Pump data*

For each pump in the network if they exist, the number of the pump , and the description of the operation data for the pump are needed.

If the pump is described by performance data, a list of the three points describing the (*Q, hp*) is needed. The program makes use of the least square method to define these data into the form of quadratic equation of the form

$$h_p = AQ^2 + BQ + H_o$$

4.3.3 Program Structure

The structure chart of the program is as shown in Figure (4.1)

The program is divided into the following functions:

1)*Main ( )* This function determines the type of operations to be performed and calls the appropriate computational functions, and displays a list of choice whether to show all the inputs or to print out the result of the program.

2)*Input ()*. Read in the network parameters such as number of pipes in the network, number of junctions, the system unit which has been used, the type of equations used to define the friction losses, and the number of loops. This function stores and writes these data to an input file, which can be very useful for updating the data for the network if that is necessary. Figure (4.2) shows the structure of this function.

3)*Output ( )*. This function reads in the data for the network from the input file. Also this function prints the result of the program after calling .This function has two sub-function. Figure (4.3) and Figure (4.4) shows the structure.

a) *Linear_Darcy ( )*. If the Darcy-Weisbach equation is used, this function will be called. This function will set up the mass continuity, loops equations as arrays, and solve the simultaneous linear equations using Gauss elimination method. This function computes the flow rates of each pipe in the network based on the use of Darcy-Weisbach equation for computing the friction loss. Figure (4.4) shows the structure of this function.

b) *Linear_Hazen_William ( )* If the Hazen-William equation is used, this function will be called. This function computes the flow rates of each pipe in the network based on the use of Hazen-William equation for computing the friction loss.

4) *Gauss_elimination ( )* This function solves the system of equations [A]{X}={B} using the Gauss elimination method with partial pivoting. This subroutine will be discussed in details in appendix (A).

5) *Newton_Raphson ( )*. This function computes the roots of an equation of the form *f(x)=0* using the Newton-Raphson method, i.e. computes the value of the friction factor *f*. This program will be discussed in details in appendix (A).

6) f( ). The main purpose of this function is to return the value of a function *f(x)* evaluated at *x*. This function is called by function *Newton_Raphson ( )* .

7) df( ). Returns the value of derivative of the function *f(x)*. This function is called as well by the function *Newton_Raphson ( )*.

8) *-poly_leastsqr ()*. If a pump exist in the network, and its characteristic is described by performance operating data, this function is invoked to fit the curve for the operating data using least square method (see appendix A).

Figure 4.1 structure for the computer program

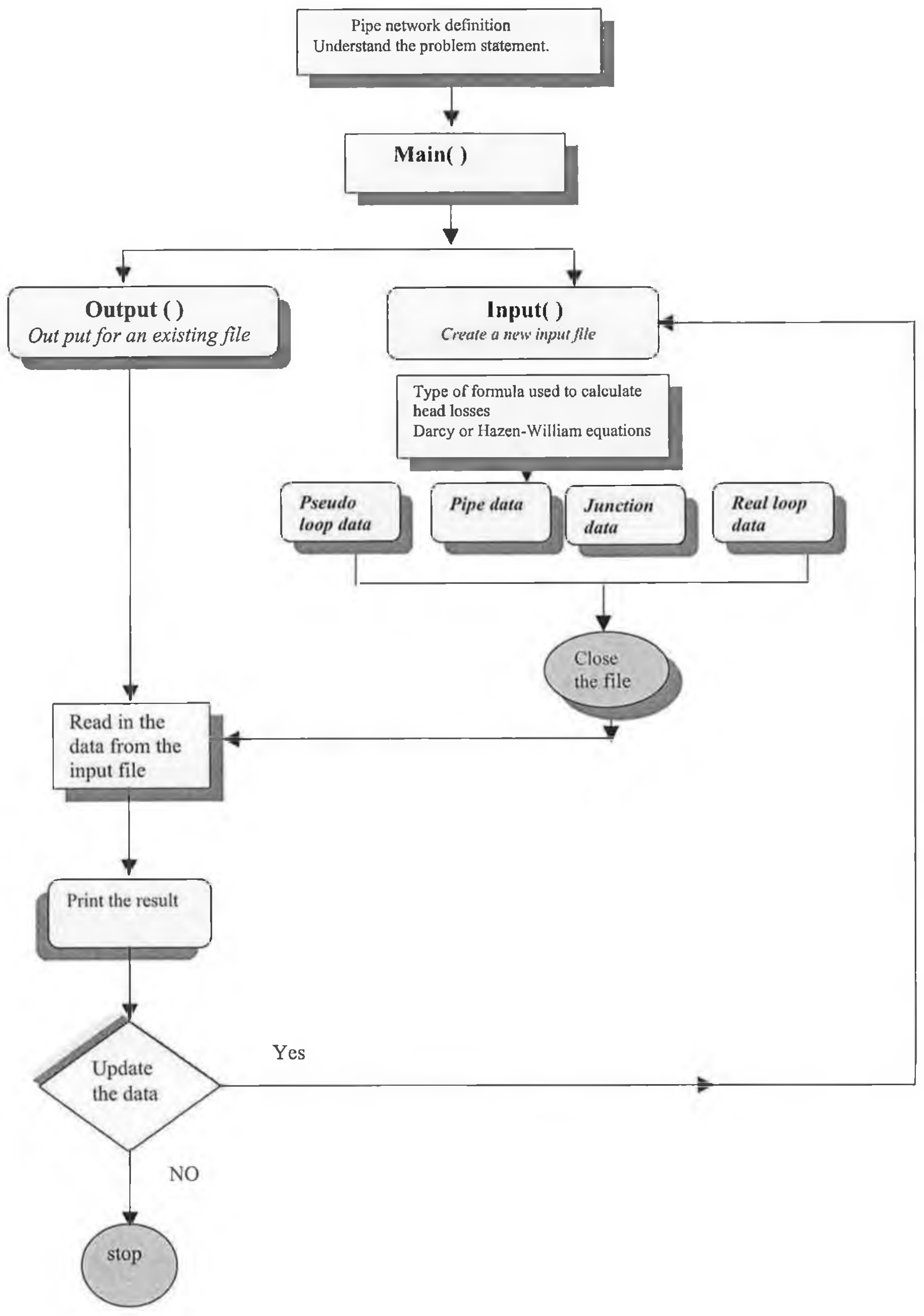


Figure 4.2 structure of function **Input ( )**

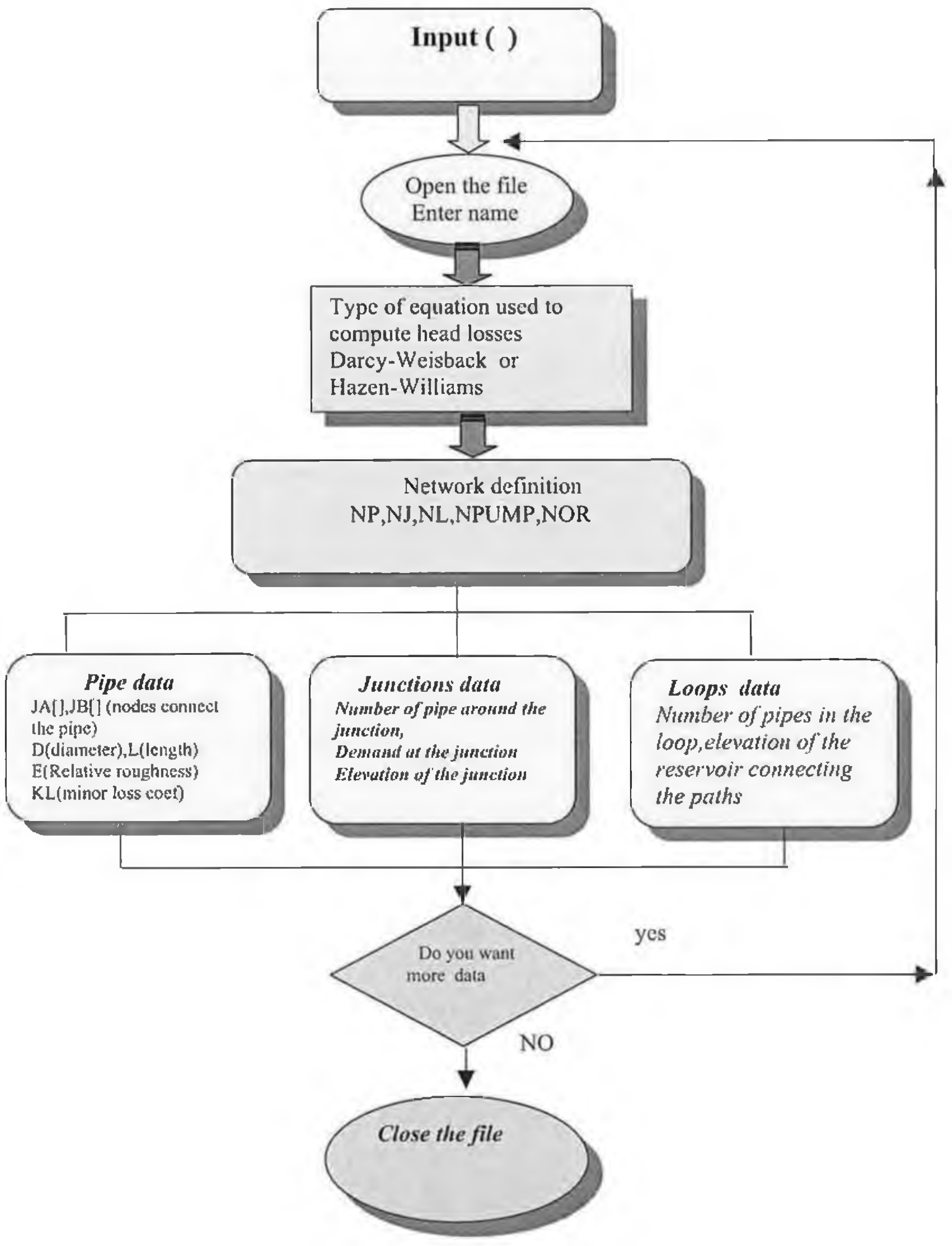

Input ( )
Open the file Enter name
Type of equation used to compute head losses Darcy-Weisback or Hazen-Williams
Network definition NP,NJ,NL,NPUMP,NOR
Pipe data
JA[],JB[] (nodes connect the pipe)
D(diameter),L(length)
E(Relative roughness)
KL(minor loss coef)
Junctions data
Number of pipe around the junction,
Demand at the junction
Elevation of the junction
Loops data
Number of pipes in the loop,elevation of the reservoir connecting the paths
Do you want more data
yes
NO
Close the file

Figure 4.3 structure of function **Output( )**

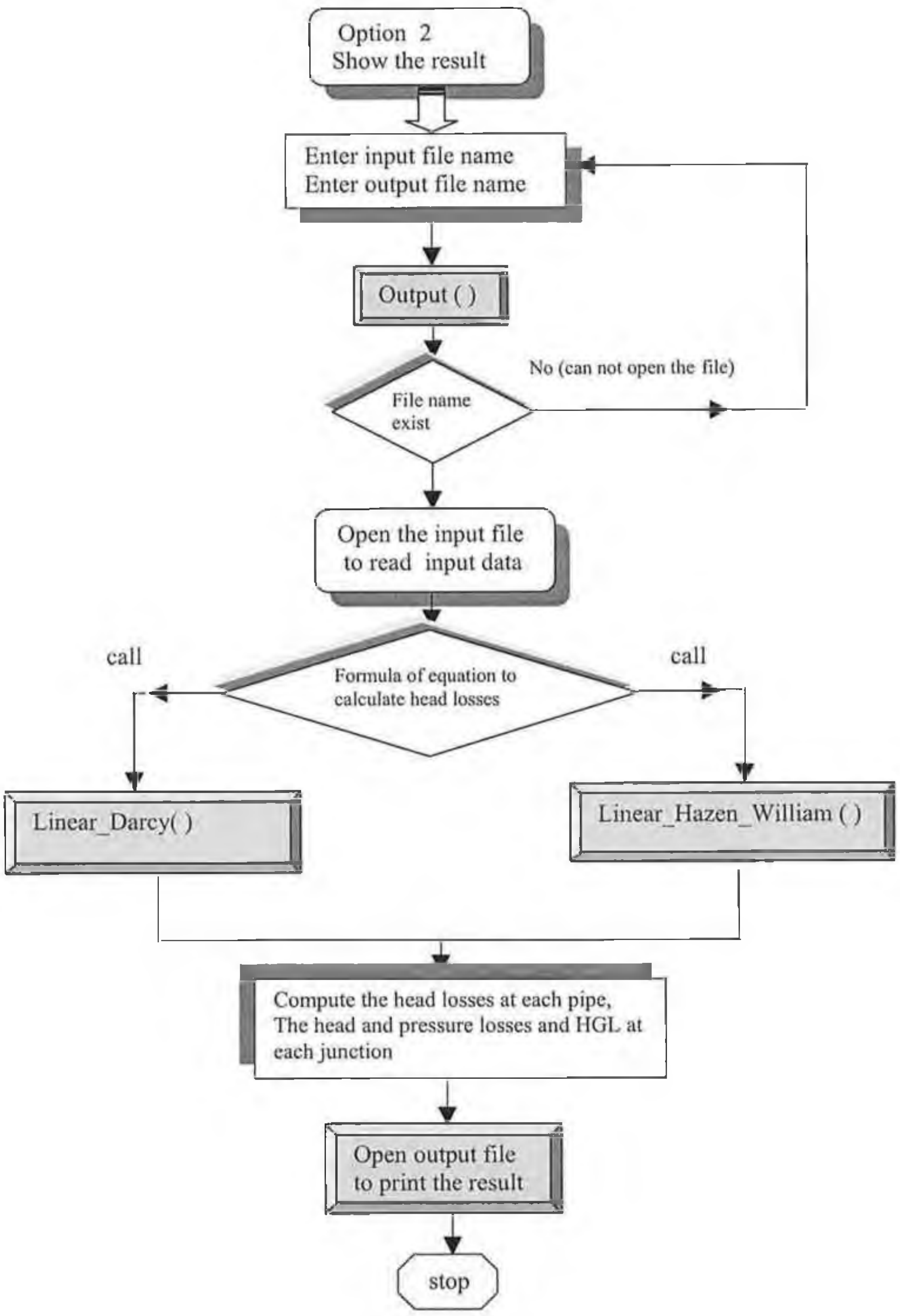
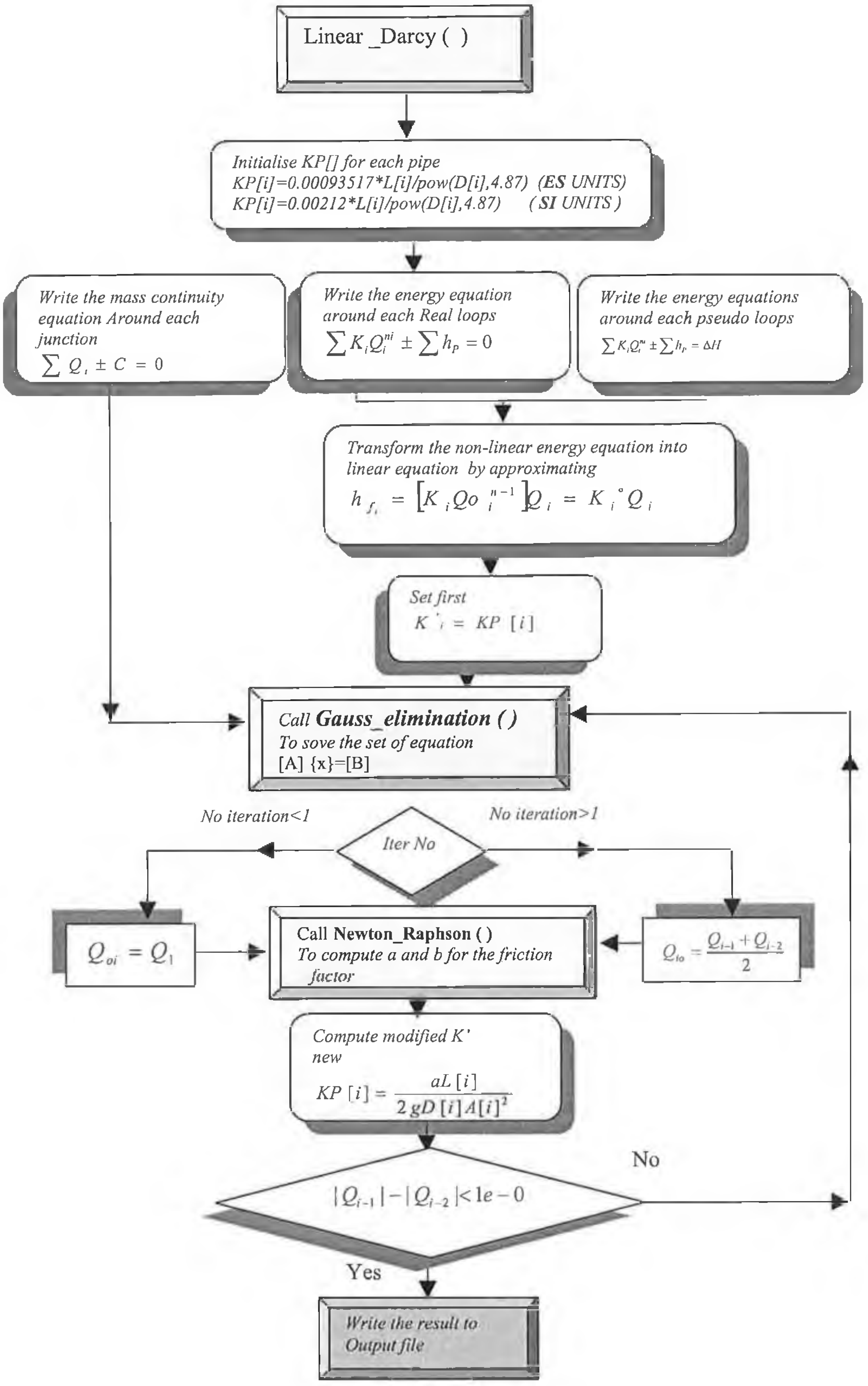

Option 2
Show the result
Enter input file name
Enter output file name
Output ( )
File name exist
No (can not open the file)
Open the input file to read input data
call
Formula of equation to calculate head losses
call
Linear_Darcy( )
Linear_Hazen_William ( )
Compute the head losses at each pipe, The head and pressure losses and HGL at each junction
Open output file to print the result
stop

Figure 4.4 structure for the function **Linear_Darcy ( )**



Linear _Darcy ( )
Initialise KP[] for each pipe
KP[i]=0.00093517*L[i]/pow(D[i],4.87) (ES UNITS)
KP[i]=0.00212*L[i]/pow(D[i],4.87) (SI UNITS)
Write the mass continuity equation Around each junction
$\sum Q_i \pm C = 0$
Write the energy equation around each Real loops
$\sum K_i Q_i^{ni} \pm \sum h_p = 0$
Write the energy equations around each pseudo loops
$\sum K_i Q_i^{ni} \pm \sum h_p = \Delta H$
Transform the non-linear energy equation into linear equation by approximating
$h_{f_i} = [K_i Qo_i^{n-1}] Q_i = K_i' Q_i$
Set first
$K'_i = KP[i]$
Call Gauss_elimination ( )
To sove the set of equation
[A] {x}=[B]
No iteration<1
No iteration>1
Iter No
$Q_{oi} = Q_1$
Call Newton_Raphson ( )
To compute a and b for the friction factor
$Q_{io} = \frac{Q_{i-1} + Q_{i-2}}{2}$
Compute modified K' new
$KP[i] = \frac{aL[i]}{2gD[i]A[i]^2}$
$|Q_{i-1}| - |Q_{i-2}| < 1e-0$
No
Yes
Write the result to Output file

4.3.4 Computer program setup

What do you need to run this program?

Software requirements: *Microsoft visual C ++5* or later

Hardware requirements: one needs to have the same specifications which are described previously in this thesis in page 4,in chapter 1.

1- Create a directory on C :\ drive called "FLOWSIMUL"

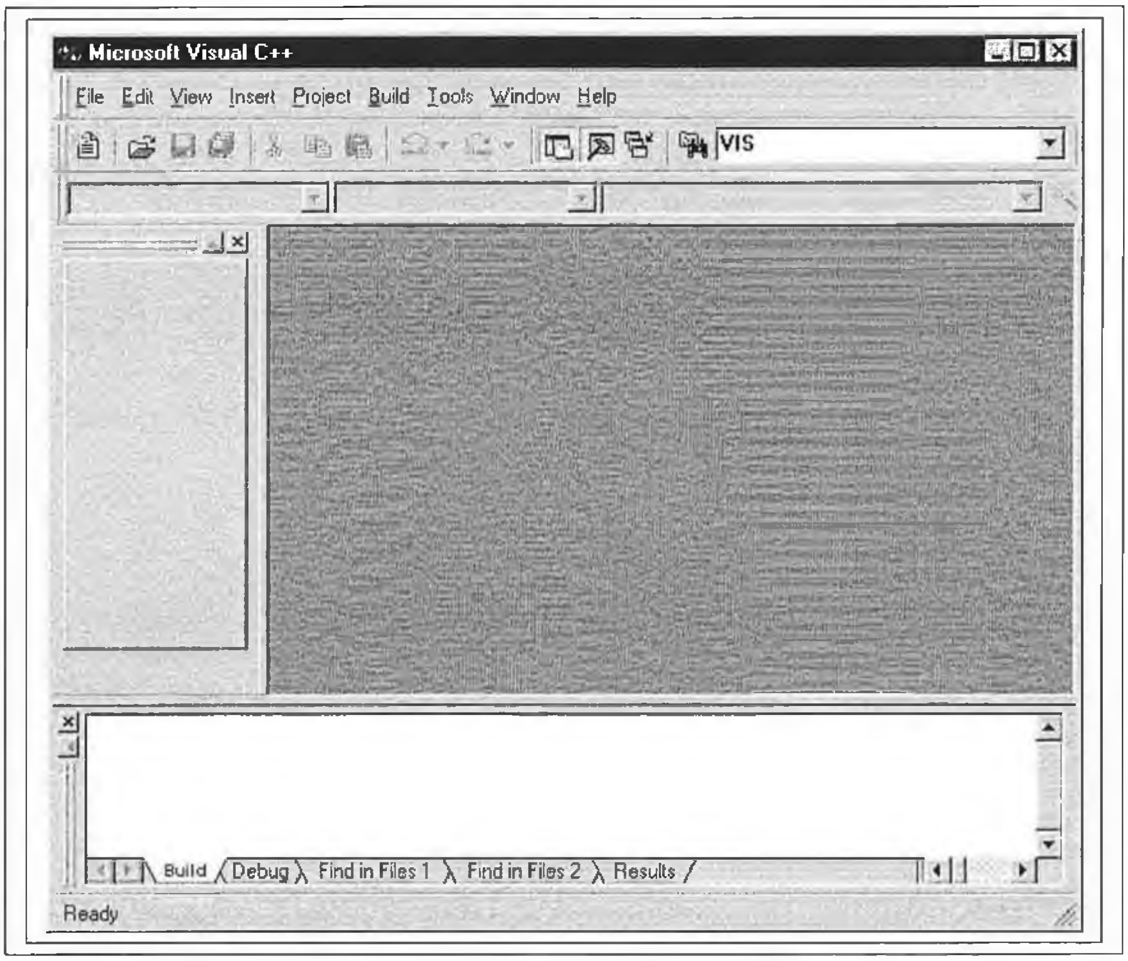
2- Copy the contents of the floppy disk into C : \ "FLOWSIMUL "

you should have now a directory structure as follows C :\ "FLOWSIMUL"

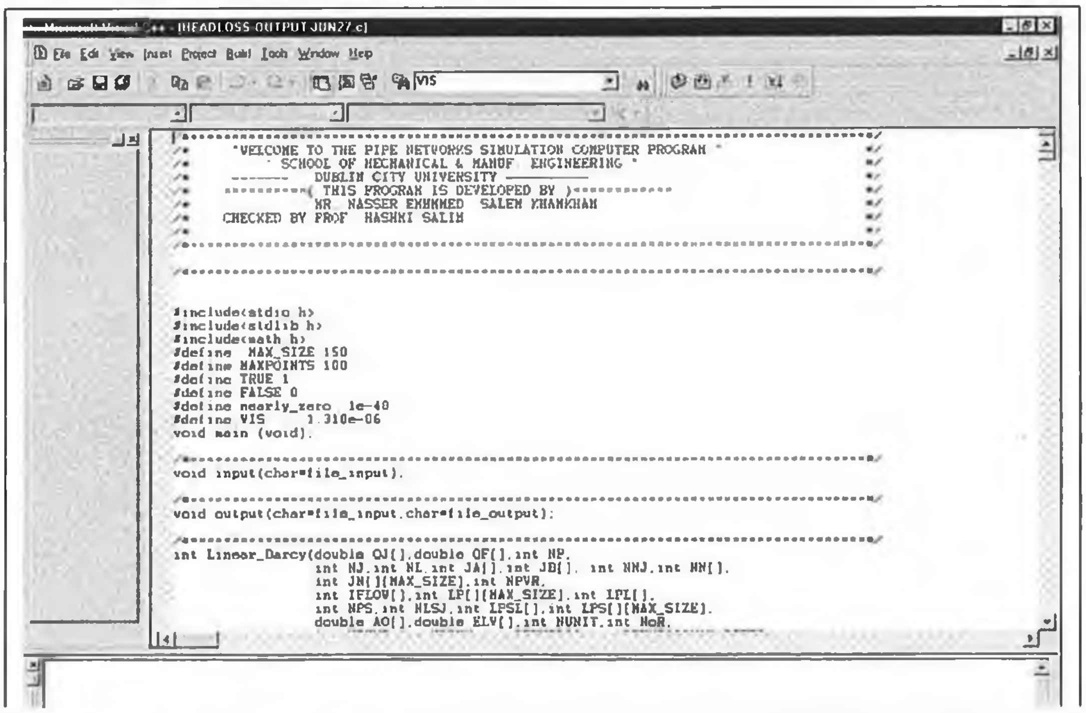
3-Start Microsoft visual C++ by clicking on the icon on the screen.

4-Figure 4.5 will be appeared then go to File/Open to open " FLOWSIMUL "

Figure 4.5



4- Open the directory “FLOWSIMUL”, and then open the file name FLOW_SIMULATION.C Figure 4.6 will appear


Microsoft Visual C++ - [HEADLOSS OUTPUT JUN27.c]
File Edit View Insert Project Build Tools Window Help
VIS

```
/**************************************************************************/
/*      "WELCOME TO THE PIPE NETWORKS SIMULATION COMPUTER PROGRAM "        */
/*            " SCHOOL OF MECHANICAL & MANUF  ENGINEERING "                */
/*      -------    DUBLIN CITY UNIVERSITY ----------                       */
/*     ==========( THIS PROGRAM IS DEVELOPED BY )============               */
/*                 MR  NASSER EMHMMED  SALEM KHANKHAN                      */
/*    CHECKED BY PROF  HASHMI SALIM                                        */
/*                                                                         */
/**************************************************************************/

/**************************************************************************/


#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define  MAX_SIZE 150
#define MAXPOINTS 100
#define TRUE 1
#define FALSE 0
#define nearly_zero  1e-40
#define VIS      1.310e-06
void main (void);

/**************************************************************************/
void input(char*file_input);

/**************************************************************************/
void output(char*file_input,char*file_output);

/**************************************************************************/
int Linear_Darcy(double QJ[],double QF[],int NP,
                 int NJ,int NL,int JA[],int JB[], int NNJ,int NN[],
                 int JN[][MAX_SIZE],int NPVR,
                 int IFLOW[],int LP[][MAX_SIZE],int IPL[],
                 int NPS,int NLSJ,int LPSL[],int LPS[][MAX_SIZE],
                 double AO[],double ELV[],int NUNIT,int NoR,
```

Figure 4.6

6-Run the program by clicking on the compile icon (F7) to check if there is any logical error

6- Execute the program by clicking on the icon execute (F8)

7- Figure (4.7) will appear on the screen

The main menu will give you 3 options

a) create new data

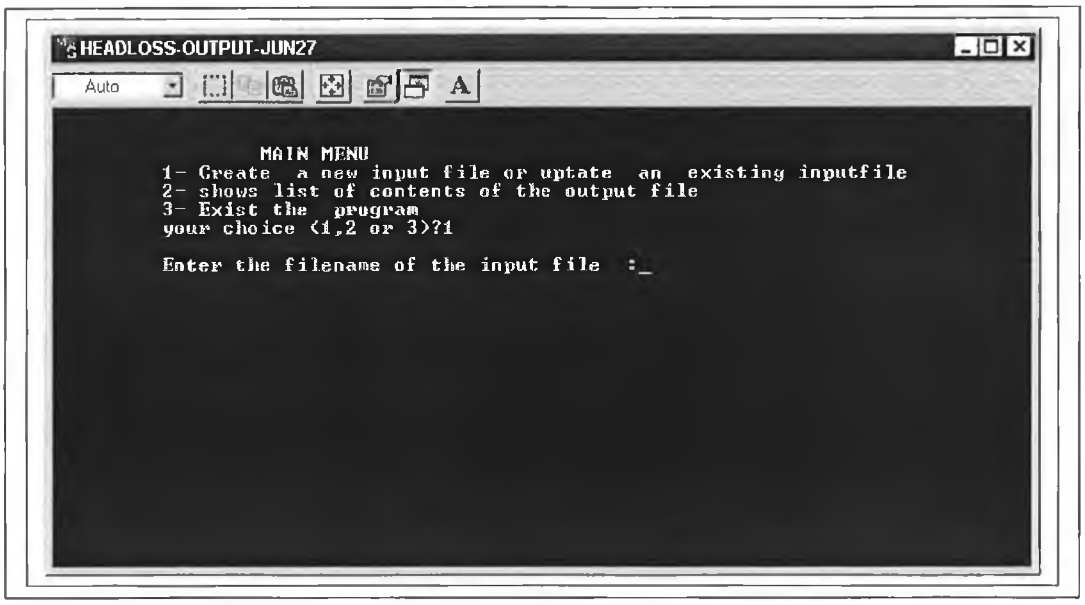
b) show the out put for an existing file

c) exit the program

9- If your choice is (a) or (b) then

a) enter name for the file input to read in all the data requirements

b) enter name for the file output to show the result



```
            MAIN MENU
1- Create  a new input file or uptate  an  existing inputfile
2- shows list of contents of the output file
3- Exist the  program
your choice (1,2 or 3)?1

Enter the filename of the input file  :_
```

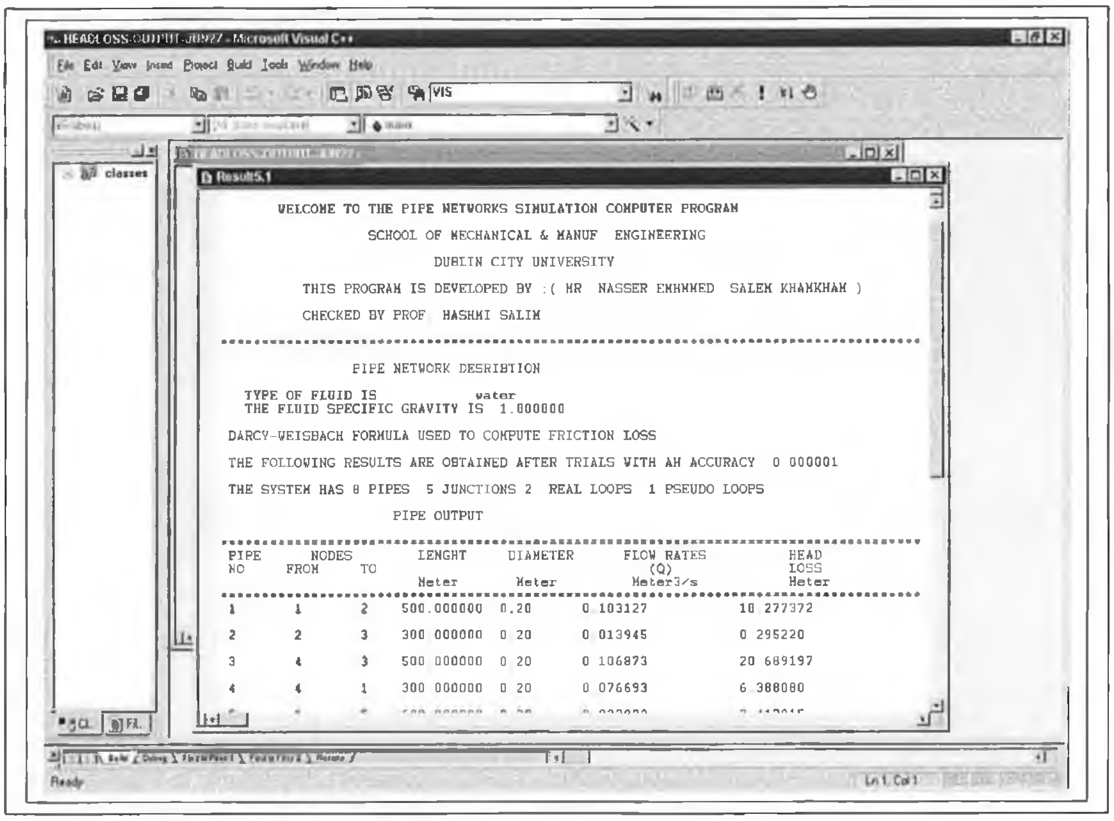
Figure 4.7

10- After entering the whole input data to the program click on the choice (b) and then enter the name for the input file and the output file

11- Go to the directory folders and click on All files

12- Enter the name of the out put file and click enter

12 –Figure(4.8) will appear for the text file of the output result



HEADLOSS OUTPUT - Microsoft Visual C++
File Edit View Insert Project Build Tools Window Help
VIS
classes
Result5.1
Ready
Ln 1, Col 1

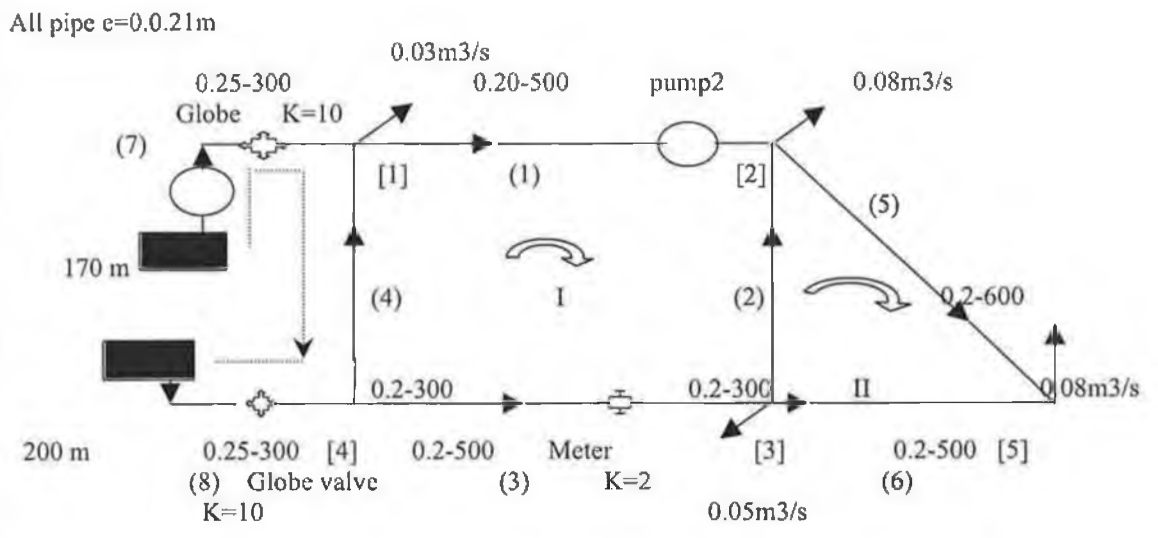
```
        WELCOME TO THE PIPE NETWORKS SIMULATION COMPUTER PROGRAM
                   SCHOOL OF MECHANICAL & MANUF  ENGINEERING
                               DUBLIN CITY UNIVERSITY
           THIS PROGRAM IS DEVELOPED BY :( MR  NASSER EMHMMED  SALEM KHAMKHAM )
           CHECKED BY PROF  HASHMI SALIM
*************************************************************************************
                 PIPE NETWORK DESRIBTION
  TYPE OF FLUID IS            water
  THE FLUID SPECIFIC GRAVITY IS  1.000000
DARCY-WEISBACH FORMULA USED TO COMPUTE FRICTION LOSS
THE FOLLOWING RESULTS ARE OBTAINED AFTER TRIALS WITH AH ACCURACY  0 000001
THE SYSTEM HAS 8 PIPES  5 JUNCTIONS 2  REAL LOOPS  1 PSEUDO LOOPS
                    PIPE OUTPUT
*************************************************************************************
PIPE      NODES         LENGHT     DIAMETER      FLOW RATES           HEAD
NO     FROM     TO                                  (Q)               LOSS
                        Meter       Meter        Meter3/s             Meter
*************************************************************************************
1        1       2     500.000000  0.20        0 103127           10 277372
2        2       3     300 000000  0 20        0 013945           0 295220
3        4       3     500 000000  0 20        0 106873           20 689197
4        4       1     300 000000  0 20        0 076693           6 388080
```

Figure 4.8

4.3.5 Testing

We can test the program quite easily by creating a data file for the input and the output for the network shown below in Figure (4.9)

The input and out put of the program is shown in Figure (4.10).



Pump characteristics

Pump no 1 / Pump no 2

$Q(m^3/\sec)$	$h_p(m)$	$Q(m^3/\sec)$	$h_p(m)$
0.025	12.00	0.060	4.0
0.040	10.50	0.090	3.8
0.055	8.00	0.120	3.5

```
                        MAIN MENU

              1- Create a new input file or update an existing input file
              2- shows list of contents of the output file
              3- Exist the program


        Your choice (1,2 or 3)?    1

              Enter the filename of the input file: example5.1

              Enter the filename of the output file: result5.1


     THE FORMULA USED TO COMPUTE HEAD LOSSES IS

     DARCY-WEISBACH------------------- [ 1 ]
     HAZEN-WILLIAMS -------------------- [ 2 ]            1

                        SYSTEM GEOMETRY

    Number of pipes in the network                       8

    Number of junctions in the network                   5

    Number of real loops in the network                  2

    Number of real loops in the network                  2

    Number of source pumps in the network                2

    Number of Reservoirs in the Network                  2

    Number of Pressure Reducing valves in the network    0

    ===============[ THE SYSTEM USES  UNITS]===============

    Pipe diameter in[ Feet     ] &  pipe length in  [ Feet    ]          (0)

    Pipe diameter in[ Inches   ] &  pipe length in [ Feet     ]          (1)

    Pipe diameter in[ Meters   ] &  pipe length in [ Meters  ]           (2)

    Pipe diameter in[ C-meter ] &  pipe length in [ Meters  ]            (3)
```

```
=========[ UNITS DEMAND AT JUNCTIONS ]==================

     There is a Demand at the junction in [gallon/minute    ]        [ 1 ]

     There is a Demand at the junction in  [cubic feet/second ]      [ 2 ]

     There is a Demand at the junction in  [cubic meter/second]      [ 3 ]          3

  ===================[   JUNCTIONS DATA    ]=================

   if  flow Leaves  the junction ----> The pipe Number is       [ + ]
   if  flow Enters  the junction-----> The pipe Number is       [ -  ]

  ========[  INPUT DATA FOR JUNCTION  [ 1]      ]================

  How many pipes round the junction                        3

    Number of the pipe at junction      -1

    Number of the pipe at junction       4

    Number of the pipe at junction       7

    The Flow rate in    [   CMS ]             0.03

   THE ELEVATION OF THE JUNCTION                   0.0

========[  INPUT DATA FOR JUNCTION  [ 2 ]      ]================

  How many pipes round the  junction                    3

     Number of the pipe at junction            1

     Number of the pipe at junction            2

     Number of the pipe at junction           -5

    The Flow rate in    [   CMS ]             0.08

     THE ELEVATION OF THE JUNCTION                   0.0

=========[ INPUT DATA FOR THE JUNCTION [3]  ]=============

How many pipes round the  junction                   3

      Number of the pipe at junction           -2

      Number of the pipe at junction            3

      Number of the pipe at junction           -6
    The Flow rate in    [   CMS ]              0.05

    THE ELEVATION OF THE JUNCTION                        0.0


=========[ INPUT DATA FOR THE JUNCTION[4]==============

How many pipes round the  junction                          3

    Number of the pipe at junction            -3

    Number of the pipe at junction            -4

    Number of the pipe at junction             8

    The Flow rate in    [   CMS ]              0.0

    THE ELEVATION OF THE JUNCTION                        0.0
=========[ INPUT DATA FOR THE JUNCTION[5]==============

How many pipes round the  junction                          2

      Number of the pipe at junction           5

      Number of the pipe at junction         6

     The Flow rate in    [   CMS ]              0.08

     THE ELEVATION OF THE JUNCTION                       0.0
====================================================================

==================    THE FLUID PROPERTIES =============

        TYPE OF FLUID                                    WATER

        THE FLUID KINEMATICS VISCOSITY          1.31E-06

        THE FLUID SPECIFIC GRAVITY                 1.000
====================================================================

====================================================================
===================[ PIPES DATA ] =================
Pipe [1]

 Node No  [1]  connects the pipe          1

 Node No  [2]  connects the pipe          2

              Pipe Diameter    0.2

              Pipe Length      500

              Relative roughness of pipe  0.00021

              Minor Lose Coefficient  0.0

Pipe [2]

Node No 1 connects the pipe      2

Node No 2 connects the pipe      3

              Pipe Diameter  0.2

              Pipe Length   300

              Relative roughness of pipe  0.00021

              Minor Lose Coefficient  0

 Pipe [3]

 Node no 1 connects the pipe      4

 Node no 2 connects the pipe     3

              Pipe Diameter  0.20

              Pipe Length   500

              Relative roughness of pipe  0.00021

              Minor Lose Coefficient  2.0

 Pipe [4]

 Node no 1 connects the pipe      4

 Node no 2 connects the pipe      1

               Pipe Diameter  0.20
```

```
                 Pipe Length   300
                 Relative roughness of pipe  0.00021
                 Minor Lose Coefficient  0.0
Pipe [5]
        Node no 1 connects the pipe      1
       Node no 2 connects the pipe      5
                  Pipe Diameter  0.20
                 Pipe Length   600
                 Relative roughness of pipe   0.00021
                 Minor Lose Coefficient  0.0
 Pipe [6]
      Node no 1 connects the pipe3
     Node no 2 connects the pipe5
                  Pipe Diameter  0.20
                  Pipe Length   500
                  Relative roughness of pipe   0.00021
                  Minor Lose Coefficient  0.0
 Pipe [7]
       Node no 1 connects the pipe0
       Node no 2 connects the pipe1
                 Pipe Diameter  0.25
                 Pipe Length   300
                 Relative roughness of pipe   0.00021
                 Minor Lose Coefficient  10
```

```
Pipe [8]
        Node no 1 connects the pipe0
        Node no 2 connects the pipe4
                  Pipe Diameter  0.25
                  Pipe Length   300
                  Relative roughness of pipe   0.00021
                 Minor Lose Coefficient  10
================ INPUT DATA FOR THE REAL LOOPS ================
The direction of the loop always is ***Clockwise****
 If the direction of the flow in the pipe Clockwise -------[+]
 If the direction of the flow in the pipe anti-Clockwise---[-]
 ============== INPUT DATA FOR THE REAL LOOP [ 1 ] ==========
 How many  pipes in the Real  loop  [ 1 ]5

 the number of the pipe in the loop1
 the number of the pipe in the loop-2
 the number of the pipe in the loop-3
 the number of the pipe in the loop4
 the number of the pipe in the loop-9
 =============== INPUT DATA FOR THE REAL LOOP [ 2 ] ====
 How many pipes in the Real loop  [ 2 ]      3

 the number of the pipe in the loop          5
 the number of the pipe in the loop          -6
 the number of the pipe in the loop          2
```

```
=============[ DATA FOR RESERVOIRS ]=============

 Enter the Reservoir elevation which connected to pipe   [7 ] 170

 Enter the Reservoir elevation which connected to pipe   [8 ] 200


==========[   INPUT DATA FOR  PUMPS        ]==========

 ==========[   INPUT DATA FOR  PUMP [ 1 ]  ]==========

Enter the pipe number contains the pump         1

 type of the operator for the pump
 type of pump data  :( [1]--> operating data
                     :( [2]--> performance operating data     2

  Forming the transformation equations for the pumps
  in the form  -Q+G=B/2A
    the sign [ - ] goes with the pipe number
    the sign [ + ] goes with the number of the pump

   the number of the pipe in the loop                        -1

   the number of the pipe in the loop                         9

 The pump is described by performance operating data degree of polynomial 2

  number of data points 3

  enter x[1];0.060

  enter y[1];4.0

 enter x[2];0.090

 enter y[2];3.80

 enter x[3];0.120

 enter y[3];3.5

 The coefficients of the best-fit polynomial are

  a(1) = 4.100000
  a(2) = 1.666667
  a(3) = -55.555556
```

```
==========[  INPUT DATA FOR  PUMP [ 2 ]  ]==========

Enter the pipe number contains the pump 7


 type of the operator for the pump
 type of pump data  :( [1]--> operating data
                    :( [2]--> performance operating data
                    :( [3]--> useful horse power        2



Forming the transformation equations for the pumps
 in the form  -Q+G=B/2A

  the sign [ - ] goes with the pipe number
  the sign [ + ] goes with the number of the pump


the number of the pipe in the loop      -7

 the number of the pipe in the loop     10

 the pump is described by performance operating data degree of polynomial 2
number of data points 3

 enter x[1];0.025

 enter y[1];12.00

 enter x[2];0.040

 enter y[2];10.50
 enter x[3];0.055

 enter y[3];8.00

 the coefficient of the best fit polynomial are
 a(1) = 12.277778
 a(2) = 44.444444
 a(3) = -2222.222222
================================================================
 The Network needs [1] pseudo loops

==========[DATA FOR PSEUDO LOOPS]======================

 ===========[  DATA FOR PSEUDO LOOP [ 1 ]  ]==================

Suggest a path to connect the two reservoir for pseudo loop [1]

 If the path contains a pump [sign the pump by Np+1] ( + )-. if the flow in .the direction of the
pipe which contains the pump in the direction of the path

How many pipes in the path                    4

  [ + ] if flow in the same direction of the energy line

  [ - ] if flow opposite the energy line direction

    the number of the pipe in the loop              7

    the number of the pipe in the loop             -4

   the number of the pipe in the loop              -8

   the number of the pipe in the loop             -10


            DO YOU WANT TO ENTER MORE  DATA- Y/N?

                        MAIN MENU

        1- Create  a new input file or update  an  existing input file
        2- shows list of contents of the output file
        3- Exist the  program

          your choice (1,2 or 3)?2

        Enter the filename of the input file  :example5.1

        Enter the filename of the output file :result5.1
        WELCOME TO THE PIPE NETWORKS SIMULATION COMPUTER PROGRAM

            SCHOOL OF MECHANICAL & MANUF. ENGINEERING

                    DUBLIN CITY UNIVERSITY

    THIS PROGRAM IS DEVELOPED BY :( MR. NASSER EMHMMED  SALEM KHAMKHAM )

       CHECKED BY PROF. M.S.J HASHMI
 ------------------------------------------------------------------------------
                    PIPE NETWORK DESCRIPTION

  TYPE OF FLUID IS          WATER

   THE FLUID KINEMATICS VISCOSITY   1.310 E -06

  THE FLUID SPECIFIC GRAVITY IS  1.000000

  DARCY-WEISBACH FORMULA USED TO COMPUTE FRICTION LOSS

  THE FOLLOWING RESULTS ARE OBTAINED  WITH ACCURACY 0.000001

 THE SYSTEM HAS 8 PIPES 5 JUNCTIONS 2 REAL LOOPS 1 PSEUDO LOOPS

                          PIPE OUTPUT

 ==============================================================================
 PIPE      NODES        LENGTH     DIAMETER        FLOW RATES      HEAD
 NO     FROM    TO                                    (Q)          LOSS
                         Meter       Meter          Meter3/s       Meter
 ------------------------------------------------------------------------------
  1        1     2      500.00       0.20           0.103127       18.277372

  2        2     3      300.00       0.20           0.013945       0.295220

  3        4     3      500.00       0.20           0.106873       20.689197

  4        4     1       300.00       0.20           0.076693       6.388080

  5        1     5       600.00       0.20           0.037072       3.417815

  6        3     5       500.00       0.20           0.042928       3.713034

  7        0     1       300.00       0.25           0.056434       1.921787

  8        0     4       300.00       0.25           0.183566       17.825108

 ==============================================================================
```

JUNCTIONS OUTPUT

JUNCTION NO	DEMAND Meter3/sec	ELEVATION Meter	HEAD LOSS Meter	PRESSURE KPa	HGL ELV Meter
1	0.030000	0.00	175.786807	1723.941213	175.786807
2	0.080000	0.00	161.190469	1580.794933	161.190469
3	0.050000	0.00	161.485694	1583.690203	161.485694
4	0.000000	0.00	182.174892	1786.589161	182.174892
5	0.080000	0.00	157.772660	1547.276477	157.772660

```
THE HEAD PRODUCED BY PUMPS ARE

The Pump in Pipe [ 1 ]        produced Head = 3.681035 [ meter ]
The Pump in Pipe [ 7 ]        produced Head = 7.708594 [ meter ]
```

CHAPTER 5

Results and Discussion

In order to illustrate the simulation of pipe networks using the linear theory method, several examples will be presented here with their results, and will be compared with other results that implemented different method of analysis.

First of all, the computer program is shown in appendix (B). Several computer programs have been written in the past few years, and every program uses a different approach for the method. What can be seen here is that the linear theory method has been used to analyze the pipe network. In addition the method used here is based on the Jeppson [12,13,18,44] approach.

Wood [17] had written a computer program based on the linear theory method, but his method of approach has a few disadvantages, although it converges very rapidly. One of these disadvantages is that the algorithm used in the simulation requires an initial guess and that might cause problems if the initial guess was not close enough to the real flow rate.

The first section of this chapter will implement the simulation on different samples of pipe networks. To use the computer program several different types of information are required.

First, the number of pipes, number of junctions, and number of loops in the network plus other specifications such as denoting type of unit(i.e. ES or SI), number of allowable iterations, viscosity of the fluid ,etc.

Second, data given the diameters, length, and wall roughness for each pipe. Third, information for establishing junction continuity equations. This information is provided by data input for each junction which contains the pipe numbers meeting at the junction. If the assumed direction of flow is into the junction this number preceded by a minus. Also if external flow occurs at the junction we must consider it as part of the junction input data. Finally, information is required for the energy equations for each loop in the network. For each loop in the network information data is provided to list the pipe numbers in that loop. a minus proceeds the pipe number if the assumed direction of flow is counterclockwise around the loop.

Example 5.1

Consider this simple 6-pipe, 5-jnuction network shown below. This network is a one-loop network.

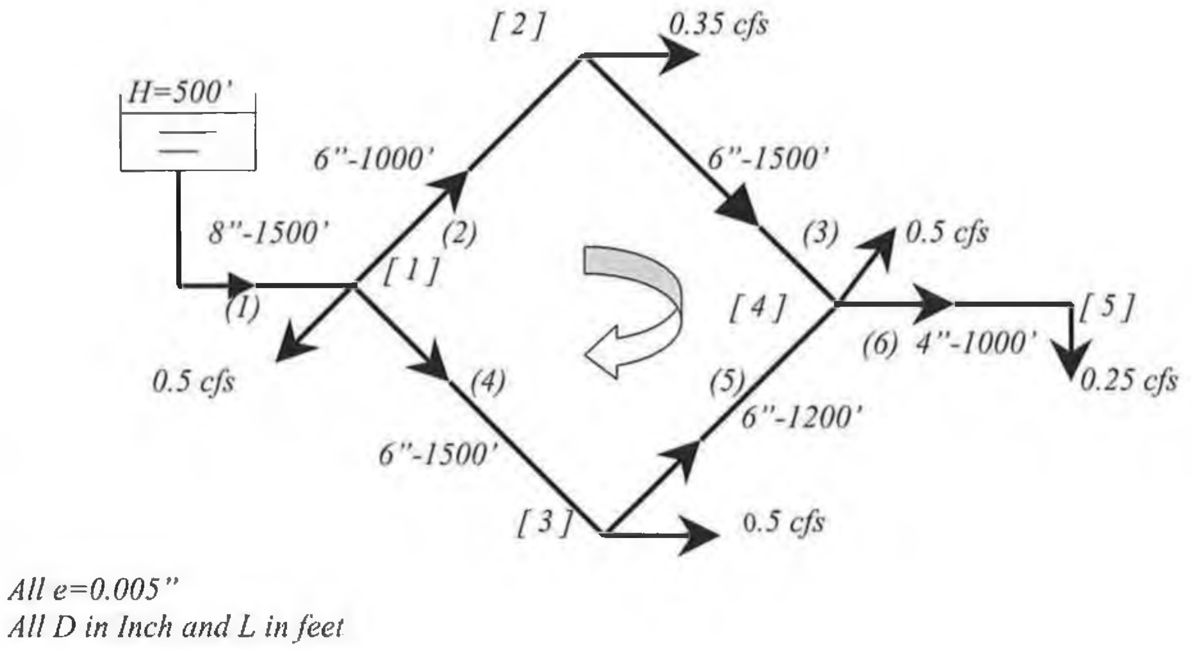


Figure 5.1.A Small 6-pipe, 5-node network [13]

For this particular network, the 6 linear equations to be solved are shown in table 5.1. The Darcy-Weisbach equation was used to define the frictional head losses.

In figure 5.1 an assumed flow direction is also shown for each pipe and the equations given in table 5.1 are based on these assumed direction. These directions are assumed arbitrarily and the solution of the equations will simply yield a negative result for the pipes where the direction is assumed incorrectly.

The first step to implement the linear theory method is to obtain the value of K and n for the exponential formula for a range of flow rates to be realistic. This is done by obtaining the initial value of K from the Hazen-Williams equations.

Initial calculated flows are obtained by letting $K' = K_i$, and the equations are solved simultaneously to obtain the results for the first trial. These result are then averaged with the initial calculated flows to give the discharges for trial 1 which are used to compute the modified pipeline constant for trial 2 . This procedure is continued until the desired accuracy is reached. Figure (5.1a) shows the results obtained for this example.

Table 5.1. Equations for example 5.1 – flow rates in cubic feet per second (ES)

Junction continuity equations	
At junction [1]	$Q_1 - Q_2 - Q_4 = 0.5$
At junction [2]	$Q_2 - Q_3 = 0.35$
At junction [3]	$Q_4 - Q_5 = 0.5$
At junction [4]	$Q_3 + Q_5 - Q_6 = 0.5$
At junction [5]	$Q_6 = 0.25$
The energy equation around the loop	
$K_2 Q_2^{n_2} + K_3 Q_3^{n_3} - K_5 Q_5^{n_5} - K_4 Q_4^{n_4} = 0.0$	

Analyzing the results from Figure (5.1a), leads to the following observations.
The relationship between the kinematic viscosity and the flow rates in pipes is established. In this example methanol was applied on the network to compute the flow and head losses at various temperatures. From Figure (5.1b) one can notice that the flow rate is increased when the kinematic viscosity of the fluid is decreased.
From Figure (5.1c) the head losses in the pipes are increased when the flow rate increases.
The relation between the head losses at junctions and the hydraulic grade line is always linear, this is the same for pressure at the junctions as shown in Figure (5.1d, 5.1e).
To see how the changing of the demand at junction affects on the calculation of the head losses at junctions, different value of demand was applied at junction 1. One can notice that the head loss at junction 1 is decreased when the demand at junction 1is decreased, since the flow rate in pipe 1 is increased. One can observe that the flow rates in the pipes 2,3,4,5,and 6 are unchanged because pipe 1 is not involved in the energy equations. If one applies the change on junction 3, because this junctions is a joint to the pipe 4 and 5, and these two pipes are taking part in forming the energy equations, thus, whatever change is applied, there must be changes to the amount of flow rates passing through these pipes. One can notice the flow rate in pipe 1 is unchanged.
Another observations can be noticed from the results. If we change the elevation of junction 1, to see the effects of the elevations of the junctions on the computations of the head losses and the HGL at junctions, from Figure (5.1g) the relation between the elevations at junction 1 and the head losses at the other junctions is always linear and the

head loss is decreased when we increase the elevations of junction 1. From Figure (5.1h) one can see that the HGL at junction 1 is unchanged ,and changed at other junctions.

Figure 5.1a. Computed flow rates for example 5.1using various types of network analysis

```
                          PIPE NETWORK DESCRIPTION

  TYPE OF FLUID IS           METHANOL
  THE FLUID SPECIFIC GRAVITY IS  0.792000
  THE FLUID DENSITY IS   49.4      Lbm/ ft3
  THE FLUID KINEMATICS VISCOSITY  7.93E-06 Ft2/sec
 DARCY-WEISBACH FORMULA USED TO COMPUTE FRICTION LOSS

 THE FOLLOWING RESULTS ARE OBTAINED  WITH  ACCURACY  0.000001
 THE SYSTEM HAS 6 PIPES  5 JUNCTIONS 1  REAL LOOPS  0 PSEUDO LOOPS

                           PIPE OUTPUT
===========================================================================
  PIPE      NODES        LENGTH    DIAMETER    FLOW RATES      HEAD
  NO     FROM    TO                                (Q)          LOSS
                          Feet        Inch        Feet3/s       Feet
===========================================================================
   1       0      1     1500.00       0.67        2.100000     23.241361

   2       1      2      1000.00      0.50        0.820223     10.845312

   3       2      4      1500.00      0.50        0.470223     5.552998

   4       1      3      1500.00      0.50        0.779777     14.745161

   5       3      4      1200.00      0.50        0.279777     1.653151

   6       4      5      1000.00      0.33        0.250000     8.657950
===========================================================================

                           JUNCTIONS OUTPUT

===========================================================================
 JUNCTION    DEMAND     ELEVATION    HEAD      PRESSURE     HGL ELV
    NO                               LOSS
             Feet3/s       Feet      Feet       Lb/in2        feet
===========================================================================
   1        0.500000     350.00    126.758639  43.503565   476.758639

   2        0.350000     350.00    115.913326  39.781454   465.913326

   3        0.500000     350.00    112.013478  38.443026   462.013478

   4        0.500000     350.00    110.360328  37.875665   460.360328

   5        0.250000     350.00    101.702377  34.904256   451.702377
===========================================================================
```

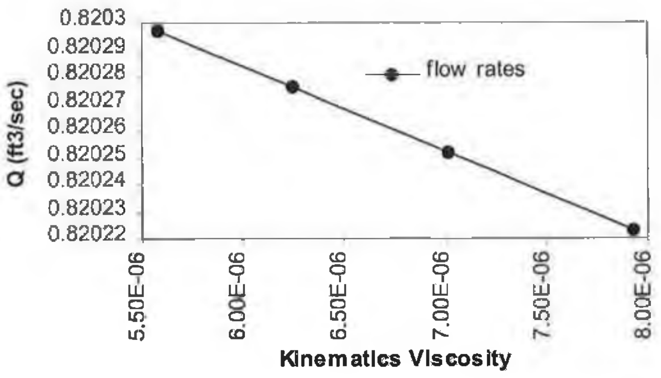


Figure 5.1B.The relation between the computed flow rate and the kinematic viscosity of the fluid

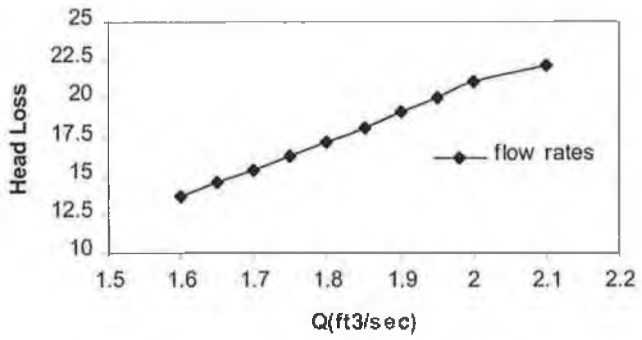


Figure 5.1C. The relation between the flow rates and the head losses in pipe 1

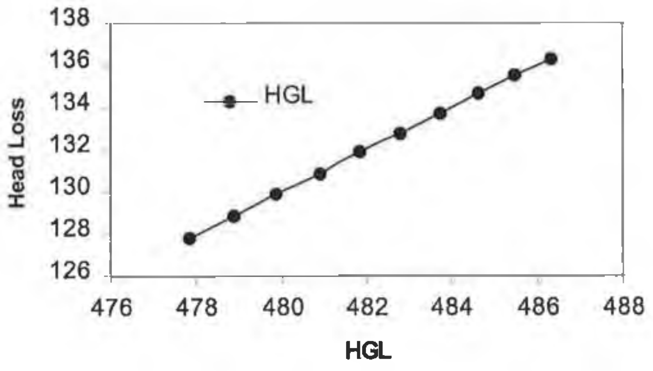


Figure 5.1D. The relation between the head losses and the Hydraulic Grad Line (HGL) at Junction 1

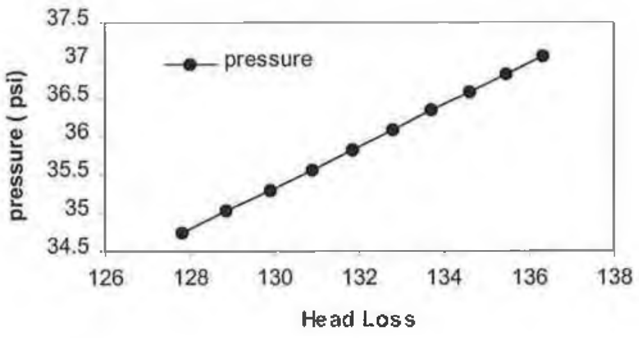


Figure 5.1E. The relationship between the head losses and the pressure at junction 1

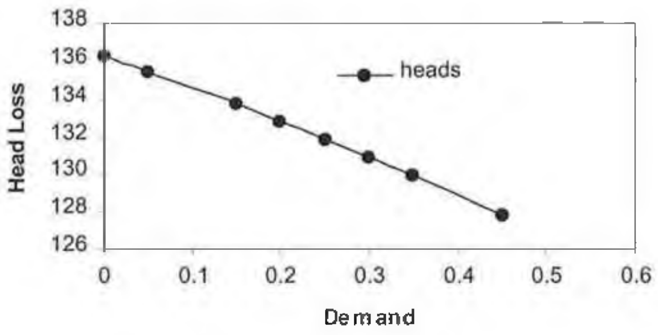


Figure 5.1F. Effects of changing the demands at junction 3 on the head losses at junction 1

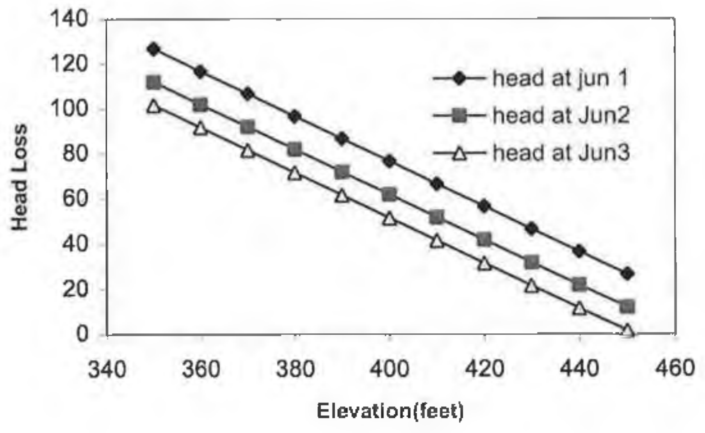


Figure 5.1G. Influence of changing the elevations of junction 1 on the value of head losses at junction 1,2 and 3.

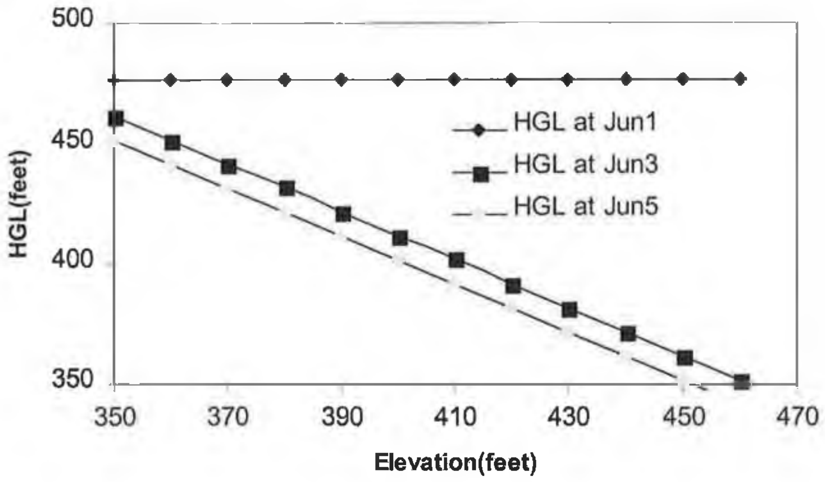


Figure 5.1H. The influence of changing of the elevation of junction 1 on the computations of hydraulic grade line of junction 1,3,and 5

Example 5.2.

This example concentrates on the implementation of solution to networks using the computer program, and how pumps are readily included. To begin this process consider first the seven-pipe network in Figure (5.2) that includes a source pump that supplies some of the system demand.

For this network there are four junction continuity equations and three loop energy equations. The Q-equations are

Junction continuity equations

At junction [1] $-Q_1 + Q_2 + Q_6 = 0$

At junction [2] $-Q_2 - Q_3 = -2.0$

At junction [3] $Q_3 - Q_4 - Q_5 = 0$

At junction [4] $Q_4 - Q_6 - Q_7 = 0$

There is one real loop in the network, so the energy equation around the real loop is

$$K_2 Q_2^{n_2} - K_3 Q_3^{n_3} - K_4 Q_4^{n_4} - K_6 Q_6^{n_6} = 0.0$$

Two pseudo loops are required. A possibility is one pseudo loop connecting the reservoirs supplying pipes 1 and 5 through pipes 1,6,4,and 5; and the other connects the pump reservoir and the reservoir supplying pipe 1 through pipes 1,6,7.

The energy equations around these two loops are

$$K_1 Q_1^{n_1} + K_6 Q_6^{n_6} + K_4 Q_4^{n_4} - K_5 Q_5^{n_5} = 100 - 105$$

$$K_1 Q_1^{n_1} + K_6 Q_6^{n_6} - K_7 Q_7^{n_7} - h_{pump} = 100 - 95$$

in solving the system of equations, the introduction of the transformation described in chapter 3. There will be an extra additional unknown. i.e. the pump head is written in the form of a quadratic equation as follow

$$h_P = -10.33Q_P^2 + 2.823Q_P + 22.29$$

This transformation replaced hp by

$$h_P = -10.33G^2 + 2.823G + 22.29$$

And adding the following linear equation to the system:

$$-Q_7 + G = \frac{B}{2A} = -0.137$$

in which G is the new transformation variable.

Should any flow rate Q becomes negative during the solution process, the computed direction of flow is in the opposite direction that assumed in writing the equations.

The output of the program is show in Figure (5.2a)

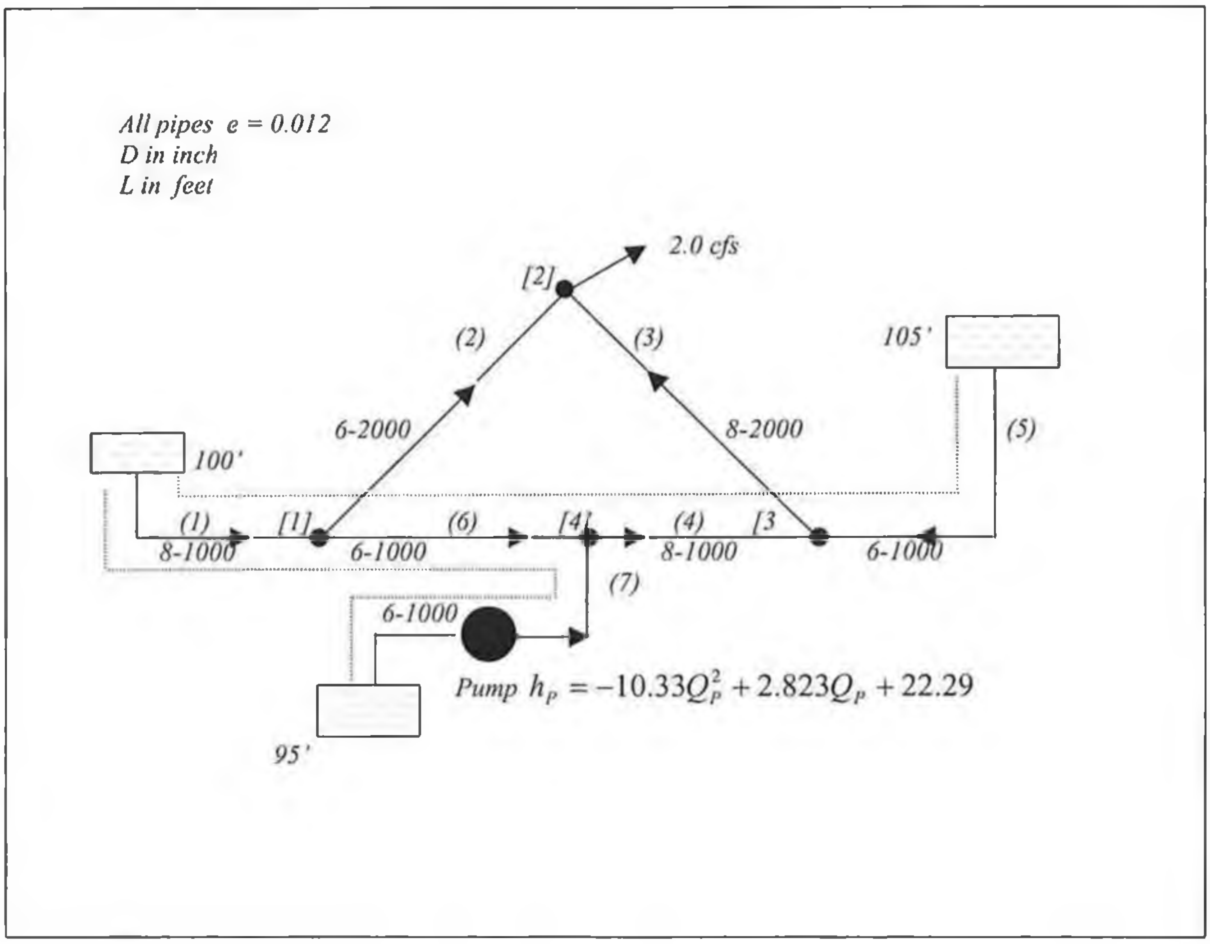
Several observations can be concluded from the output result. First the same behavior was noted regarded the relation between the flow rate and head losses as discussed in the previous example. Different changes are now made on the components of the pipe network. If one starts with the change the length of pipe 1 to see how it affects the results, from Figure (5.2b) we can notice that the flow rates in pipe 1 is decreased when the length is increased, it is the same for the relation between the head losses and the flow rates in pipe 1 . The same has been noted with the change of the diameter in pipe 1 , this shown in Figure (5.2 c,5.2d).

Form these two figures , the flow rate is increases when the diameter of the pipe is larger, and consequently the head losses are decrease when the diameter is getting smaller as shown in Figure (5.2e).

If the length of pipe 7 is changed, which contains the pump, one can see that the flow rate has the same behavior as in Figure (5.2b, 5.2c), this is shown in Figure (5.2f, 5.2g). For the head produced by the pump, the change in the length of the pipe shows that the head produced by pump is increased with the increase in length of the pipe containing the pump, as shown in Figure (5.2h, 5.2i).

If a change in the diameter is applied in pipe 7 it will be seen that the head produced by the pump is decreased when the diameter is bigger, this is applied for the head losses in that pipe as shown in Figure (5.2j).

Figure 5.2 a small 7-pipe,4-node network including pumps[12]



```
                    PIPE NETWORK DESRIBTION
 TYPE OF FLUID IS          BENZEN
 THE FLUID SPECIFIC GRAVITY IS  0.858000
 THE FLUID DENSITY IS   53.6 Lbm / Ft3
 THE FLUID VISCOSITY IS   6.31E-06 Ft2 / sec
DARCY-WEISBACH FORMULA USED TO COMPUTE FRICTION LOSS
THE FOLLOWING RESULTS ARE OBTAINED WITH ACCURACY 0.000001
THE SYSTEM HAS 7 PIPES  4 JUNCTIONS 1  REAL LOOPS  2 PSEUDO LOOPS

                       PIPE OUTPUT
=========================================================================
PIPE      NODES         LENGHT      DIAMETER       FLOW RATES     HEAD
NO     FROM     TO                                    (Q)         LOSS
                         Feet          Feet          Feet3/s         Feet
=========================================================================
1       0        1      1000.0          0.67          0.527802     1.226809

2       1        2      2000.00         0.50          0.661658     16.981763

3       3        2      2000.00         0.67          1.338342     15.244923

4       4        3      1000.00         0.67          0.697841     2.116524

5       0        3      1000.00         0.50          0.640501     7.963649

6       1        4      1000.00         0.50         -0.133856     -0.379684

7       0        4      1000.00         0.50          0.831697     13.340914
=========================================================================
                       JUNCTIONS OUTPUT
=========================================================================
JUNCTION    DEMAND      ELEVATION     HEAD       PRESSURE      HGL ELV
 NO                                   LOSS
            Feet3/s          Feet      Feet         Lb/in2       feet
=========================================================================
 1          0.000000     80.00      18.773191    6.979872     98.773191

 2         -2.000000     80.00      1.791428     0.666053     81.791428

 3          0.000000     80.00      17.036351    6.334115     97.036351

 4          0.000000     80.00      19.152873    7.121038     99.152873

  THE HEAD PRODUCED BY PUMPS ARE
=========================================================================
 The Pump in Pipe [7 ]          produced Head  =  17.493787  [ feet ]
```

Figure (5.2a) *the output for example 5.2*

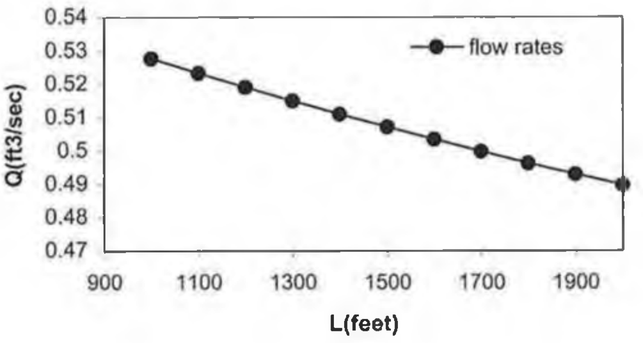


Figure 5.2B. Effects of changing the length of pipe 1 on the computations of the flow rates in pipe 1

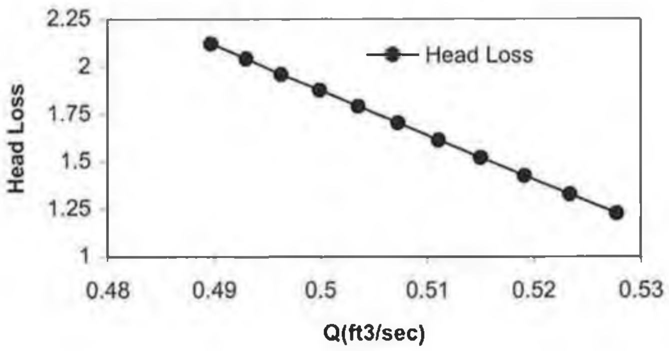


Figure 5.2C. The established relation between the flow rates and the head losses in pipe 1 at different lengths

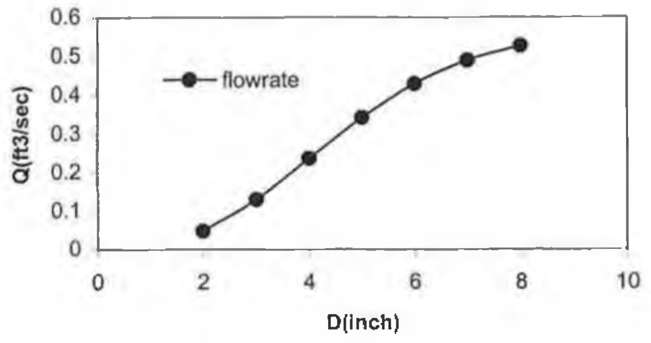


Figure 5.2D. Effects of diameter changing on the computed flow rates in pipe 1

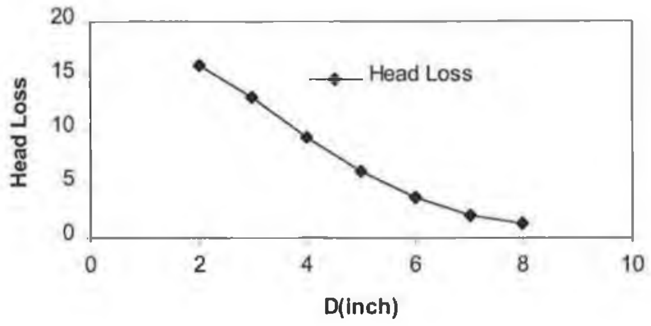


Figure 5.2E. Effects of changing the diameter of pipe 1 on the head losses in pipe 1 computations

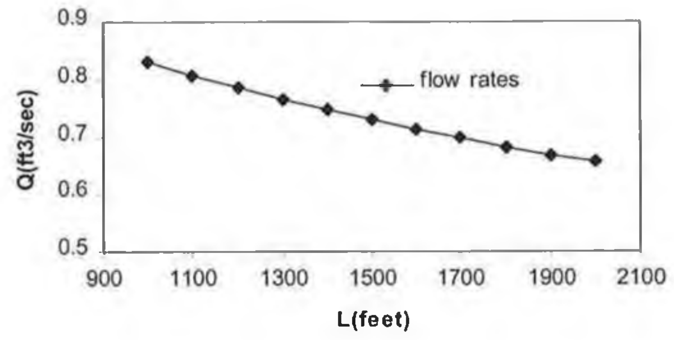


Figure 5.2F. Effects of lengths changes in the pipe containing the pump on the flow rates through the pipe

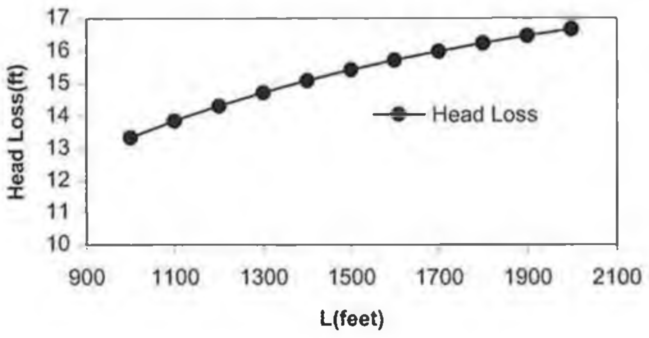


Figure 5.2G. Effects of lengths changes in the pipe containing the pump on head losses computation in that pipe

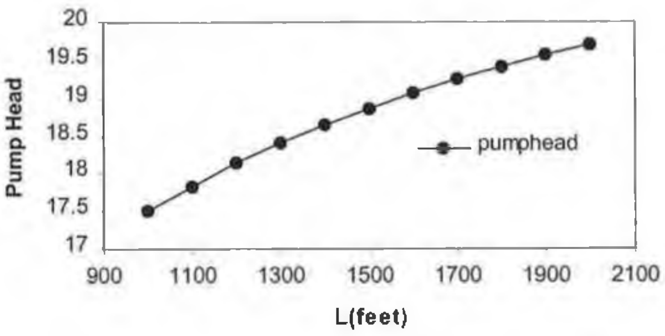


Figure 5.2H. Effects of the length changes in the pipe containing the pump on the head produced by the pump

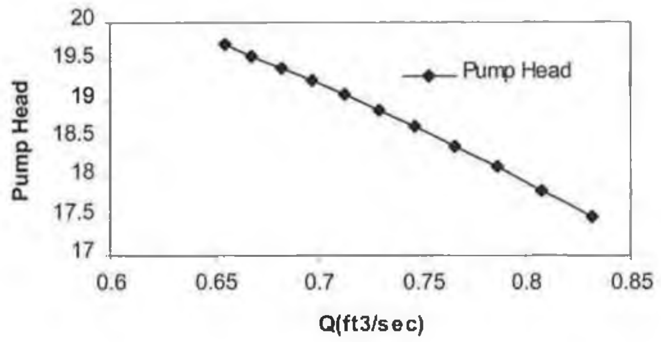


Figure 5.2 I. The established relationship between the flow rate in the pipe containing the pump the head produced by the pump in that pipe

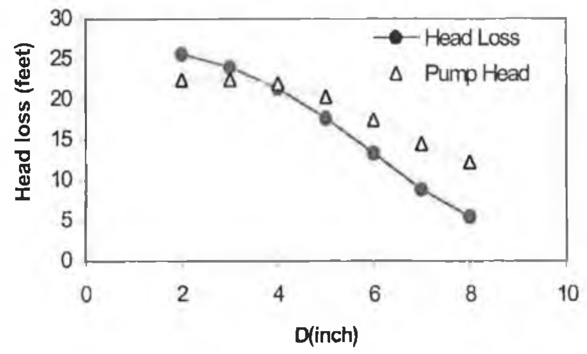


Figure 5.2J. Effects of the diameter changes in pipe 7 on head produced by the pump and the head losses in pipe 7 which contains the pump.

Example 5.3

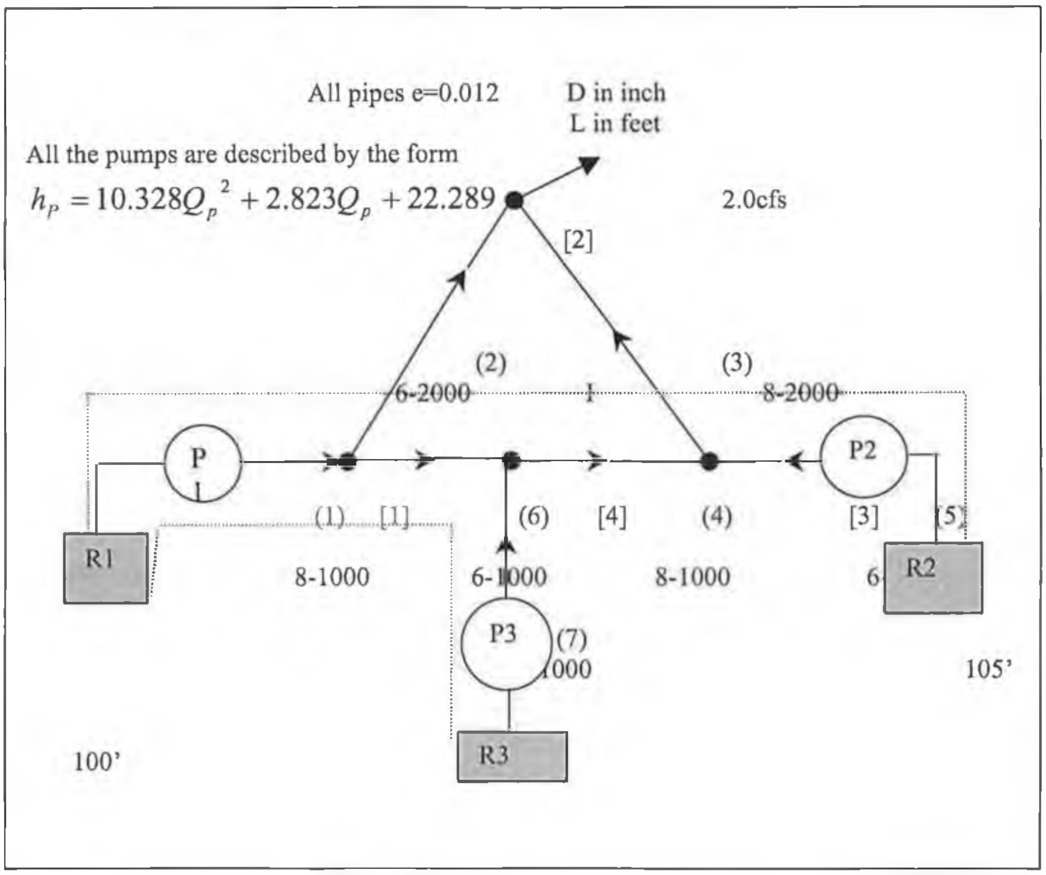
This is another example on how to include the pumps in the network analysis. Consider the seven-pipe network supplied by three identical pumps shown in Figure (5.3).Each pump supplies head according to the equation

$$h_p = 10.328Q_p^{\ 2} + 2.823Q_p + 22.289$$

since there are seven pipes in the network there will be seven unknown flow rates, plus three additional unknown, i.e. the *G's* of equation (3.10) for the three pumps which supply flow. Consequently a total of 10 simultaneous equations are needed. Four of these equations are the junction continuity equations; three are from equation (3.10) relating the

three *G's* to Q_1, Q_5 ,and Q_7; and consequently three energy equations are needed, one for the real loop and two from pseudo loops connecting pumps reservoirs with no flow pipes In applying the linear theory method, the three non-linear energy equations are linearized as described previously, and the result is shown in Figure (5.3a).

Figure 5.3 including pumps in the network analysis [18]



```
                         PIPE NETWORK DESCRIPTION

  TYPE OF FLUID IS           LUBRICATING OIL
  THE FLUID SPECIFIC GRAVITY IS  0.845000
  THE FLUID DENSITY IS  52.6 Lbm / Ft3
  THE FLUID KINEMATICS VISCOSITY IS  5.33E-06 Ft2 / sec
 DARCY-WEISBACH FORMULA USED TO COMPUTE FRICTION LOSS
 THE FOLLOWING RESULTS ARE OBTAINED WITH ACCURACY    0.000001
 THE SYSTEM HAS 7 PIPES  4 JUNCTIONS 1  REAL LOOPS  2 PSEUDO LOOPS
                           PIPE OUTPUT
=====================================================================
PIPE    NODES     LENGTH    DIAMETER    FLOW RATES       HEAD
NO   FROM   TO                              (Q)          LOSS
                   Feet       Feet        Feet3/s        Feet
=====================================================================
1     0      1   1000.00     0.67          0.842161      3.667463

2     1      2   2000.00     0.50          0.667207      19.993466

3     3      2   2000.00     0.67          1.332793      17.288454

4     4      3   1000.00     0.67          0.578775      1.839112

5     0      3   1000.00     0.50          0.754018      12.576706

6     1      4   1000.00     0.50          0.174954      0.865900

7     0      4   1000.00     0.50          0.403821      3.936704
=====================================================================
                        JUNCTIONS OUTPUT
=====================================================================
JUNCTION    DEMAND     ELEVATION     HEAD         PRESSURE     HGL ELV
 NO                                  LOSS
            Feet3/s      FEET        Feet          Lb/in2        feet
=====================================================================
 1         0.000000    0.000000    113.673977    41.623621   113.673977

 2        -2.000000    0.000000    93.680511     34.302680   93.680511

 3         0.000000    0.000000    110.968967    40.633137   110.968967

 4         0.000000    0.000000    112.808079    41.306558   112.808079

  THE HEAD PRODUCED BY PUMPS ARE
=====================================================================
 The Pump in Pipe [ 1 ]       produced Head  =  17.341440  [ feet ]

 The Pump in Pipe [ 5 ]       produced Head  =  18.545673  [ feet ]

 The Pump in Pipe [ 7 ]       produced Head  =  21.744783  [ feet ]
```

. Figure 5.3a: *the output for example 5.3*

Example 5.4

In order to illustrate how the PRV are incorporated in an analysis using the linear theory method, several examples will be illustrated.

For the network shown in Figure (5.4), the PRV exist in pipe 6, 500 feet downstream for the beginning of this pipe. The junction continuity equations are identical to those that would be written if the PRV were not present. In obtaining the second portion for the system of *Q-equation*, loops are formed after the pipes containing the PRV have been imagined from their upstream junctions and the PRV in the network is replaced by an artificial reservoir with constant head equivalent to the valve's pressure sitting. As long the operation of the PRV is normal, this artificial reservoir has an apparent constant head, Using this scheme, the eight equations needed for a solution by the linear theory method are

$$-Q_1 + Q_2 + Q_6 + Q_7 = 0$$

$$-Q_2 - Q_3 = -1.0$$

$$Q_3 - Q_4 + Q_5 - Q_7 = 0$$

$$-Q_5 - Q_6 = -1.0$$

$$K_2 Q_2^{n_2} - K_3 Q_3^{n_3} - K_7 Q_7^{n_7} = 0 \qquad \textit{(Real loop)}$$

$$K_4 Q_4^{n_4} - K_7 Q_7^{n_7} - K_1 Q_1^{n_1} + A G_1^2 = 100 - 90 - h_{o1} \qquad \textit{(Pseudo loop I)}$$

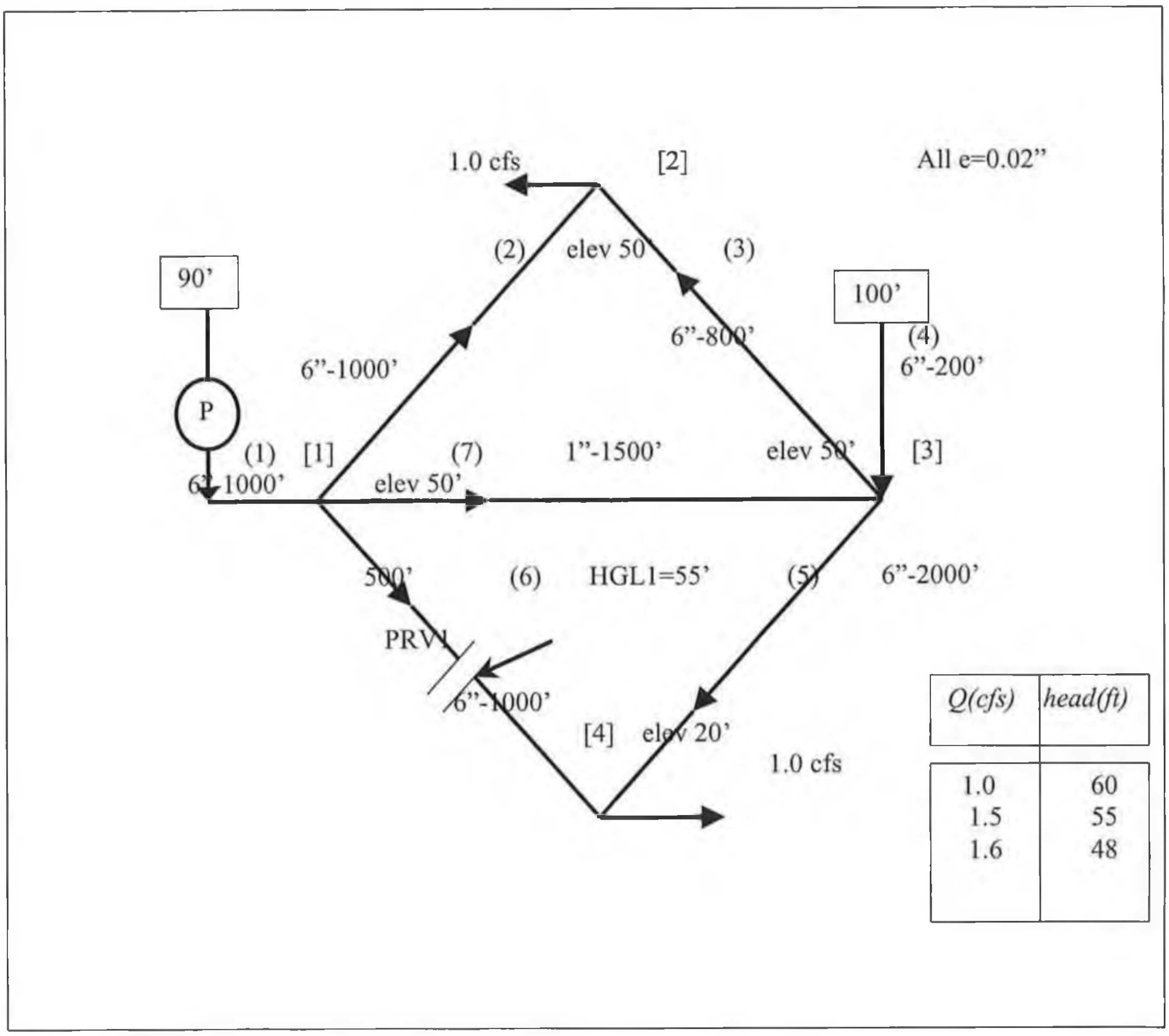
$$K_4 Q_4^{n_4} + K_5 Q_5^{n_5} - K_6' Q_6^{n_6} = 100 - 55 \qquad \textit{(Pseudo loop II)}$$

$$G_1 - Q_1 = (B/2A) \qquad \text{(pump transformation)}$$

in which is K_6' determined only for the portion of pipe 6 downstream from the PRV.

Upon solving this system of equations, the following results are shown in Figure (5.4a)

Figure 5.4 a seven-pipe network including pressure reducing valve in pipe 6 [13]



Q(cfs)	*head(ft)*
1.0	60
1.5	55
1.6	48

From the output of this example which is shown in Figure (5.4a) we could notice that the pressure upstream from the PRV equals the 121.97ft. and the downstream equals 54.98 ft. consequently , the assumption used in writing the final loop equation is correct.

Changing the assumption and setting the pressure downstream equal to 40 ft, figure (5.4a2), shows that the flow rates in pipe 6 which contains the pressure reducing valves is a negative flow rate. This assumption is incorrect, since the PRV would then have acted as check valve and allowed the elevation of the HGL downstream from the PRV to rise above 40 ft. the flow rate in pipe 6 would no longer be unknown, but equal to zero. And this causes the PRV to shut off. The computer program is capable of warning the user if the PRV is operating normally or not.

Another assumptions has been applied to this network by setting the HGL at various values and Figure (5.4b) shows the effect of setting the elevation of the artificial reservoir.

Several other observations can be established by changing the geometry of the network. Making a change to the location of the PRV , for example 400 feet down stream for the beginning of pipe 6, one shall notice that locations of the PRV have slight effect on the calculation of the flow rates through the pipe; Figure (5.4c) and (5.4d) shown that.

In figures (5.4e),(5.4f) and (5.4g) the same behavior of changing the diameter in the pipe containing the PRV was noted as discussed in example 5.2

Another important observation has been noted when applying the change of the demand on junctions, first the change is applied in junction 2. One can see that the flow rate in the pipe 6 is always positive with certain amount to be drawn from the system. If the amount on the demand is decreased to less than 0.5 *cfs* as shown in Figure (5.4h), the flow rate in pipe 6 will be negative , which means that the PRV will act as Check valve and cause the valve to shut off as discussed earlier.

If the demand is changed at junction 4 one will get similar behavior, this shown in Figure (5.4k).

```
                         PIPE NETWORK DESCRIPTION

  TYPE OF FLUID IS           BENZEN
  THE FLUID SPECIFIC GRAVITY IS  0.879000
  THE FLUID DENSITY IS         54.9 Lbm / Ft3
  THE FLUID KINEMATICS VISCOSITY IS      7.99 E --06  Ft2 / sec
 DARCY-WEISBACH FORMULA USED TO COMPUTE FRICTION LOSS
 THE FOLLOWING RESULTS ARE OBTAINED WITH  ACCURACY  0.000001
 THE SYSTEM HAS 7 PIPES  4 JUNCTIONS 1  REAL LOOPS  2 PSEUDO LOOPS

                        PIPE OUTPUT

==========================================================================
PIPE    NODES      LENGTH      DIAMETER     FLOW RATES      HEAD
NO    FROM   TO                                (Q)           LOSS
                    Feet         Feet        Feet3/s         Feet
==========================================================================
1       0     1    1000.00       0.50         1.110179      27.094731

2       1     2    1000.00       0.50         1.073766      25.357767

3       3     2     800.00       0.50        -0.073766      -0.110007

4       0     3     200.00       0.50         0.889821       3.492493

5       3     4    2000.00       0.50         0.970859      41.519839

6       1     4     500.00       0.50         0.029141       0.012332

7       1     3    1500.00       0.08         0.007272      25.467775
==========================================================================
                        JUNCTIONS OUTPUT
==========================================================================
JUNCTION    DEMAND      ELEVATION     HEAD        PRESSURE      HGL ELV
  NO                                  LOSS
            Feet3/s        Feet       Feet         Lb/in2        feet
==========================================================================
  1          0.000000     50.00     71.975281     27.415385    121.975281

  2         -1.000000     50.00     46.617514     17.756611     96.617514

  3          0.000000     50.00     46.507507     17.714709     96.507507

  4         -1.000000     20.00     34.987668     27.410687     91.962949

   THE HEAD PRODUCED BY PUMPS ARE
==========================================================================
  The Pump in Pipe [ 1 ]          produced Head  =  59.070012  [ feet ]
==========================================================================
 THE PRV[1] IN PIPE [6] IS OPERATING NORMALLY
==========================================================================
```

. Figure 5.4a : *the output result of example 5.4 (with HGL=55 feet)*

```
                    PIPE NETWORK DESCRIPTION

  TYPE OF FLUID IS          BENZENE
  THE FLUID SPECIFIC GRAVITY IS  0.879000
  THE FLUID DENSITY IS  54.9 Lbm / Ft3
  THE FLUID KINEMATICS VISCOSITY IS 7.99 E -06 Ft2 / sec
 DARCY-WEISBACH FORMULA USED TO COMPUTE FRICTION LOSS
 THE FOLLOWING RESULTS ARE OBTAINED  WITH  ACCURACY  0.000001
 THE SYSTEM HAS 7 PIPES  4 JUNCTIONS 1  REAL LOOPS  2 PSEUDO LOOPS

                       PIPE OUTPUT
=========================================================================
 PIPE    NODES       LENGTH    DIAMETER     FLOW RATES        HEAD
 NO    FROM    TO                              (Q)             LOSS
                      Feet        Feet        Feet3/s          Feet
=========================================================================
 1       0      1   1000.000      0.50        1.086106        25.939997

 2       1      2   1000.000      0.50        1.104328        26.811775

 3       3      2   800.000       0.50       -0.104328        -0.212165

 4       0      3   200.000       0.50        0.913894        3.682446

 5       3      4   2000.000      0.50        1.025719        46.307719

 6       1      4   500.000       0.50       -0.025719        -0.009835

 7       1      3   1500.000      0.08        0.007497        27.023941
=========================================================================
                       JUNCTIONS OUTPUT
=========================================================================
 JUNCTION    DEMAND     ELEVATION     HEAD      PRESSURE     HGL ELV
  NO                                  LOSS
             Feet3/s       FEET       Feet        Lb/in2        feet
=========================================================================
  1         0.000000     50.000    73.341494    27.935775   123.341494

  2        -1.000000     50.00     46.529719    17.723170    96.529719

  3         0.000000     50.00     46.317554    17.642356    96.317554

  4        -1.000000     20.00     73.351329    27.939521    93.351329

   THE HEAD PRODUCED BY PUMPS ARE
=========================================================================
  The Pump in Pipe [ 1 ]          produced Head  =  59.281491  [ feet ]
=========================================================================
              *********** (((((( WARNING ))))))****************
=========================================================================

              THE PRV[1] IN PIPE [6] IS NOT OPERATING NORMALLY
=========================================================================
```

. ***Figure 5. 4a2*** *output for example 5.4 (with HGL of the PVR=40 feet)*

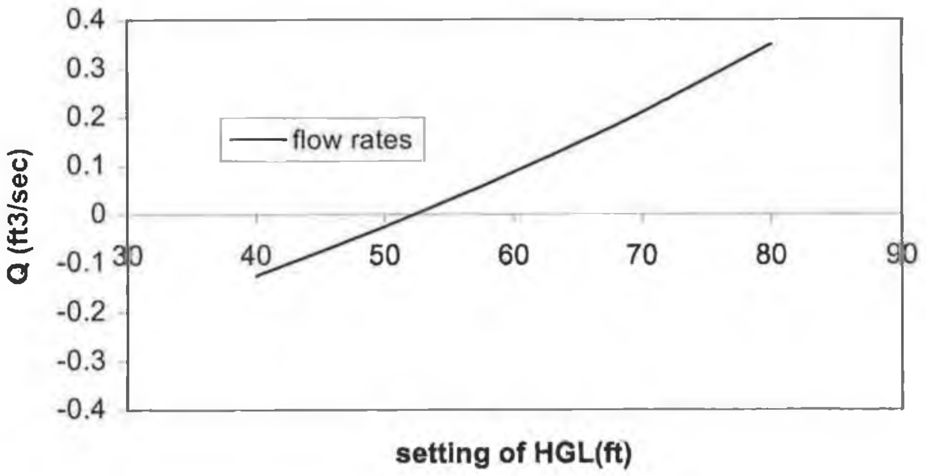


Figure 5.4B. Influence of changing the setting of the HGL of the PRV on the flow rates through the pipe 6

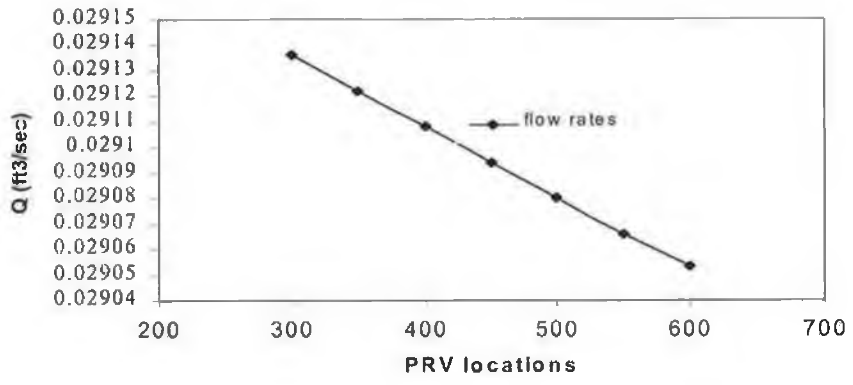


Figure 5.4C. Effects of changing the locations of PRV downstream of the beginning of pipe 6 on the flow rates calculations

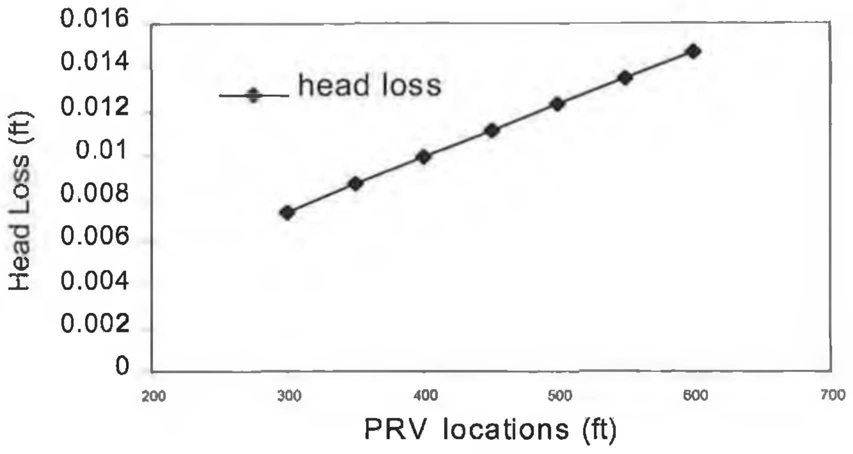


Figure 5.4D. Effects of changing the locations of PRV in pipe 6 on the head losses calculations

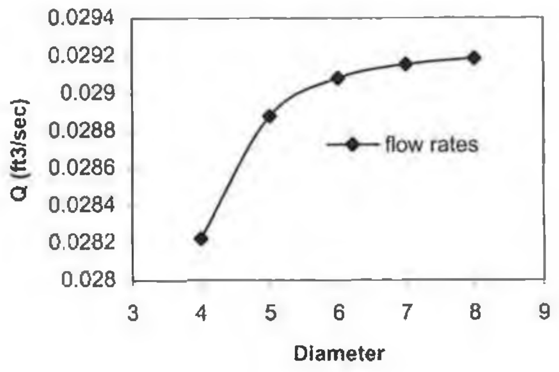


Figure 5.4E. effects of changing the diameter in pipe 6 on the flow rates

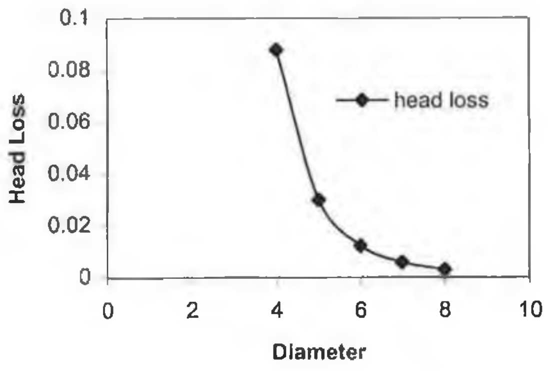


Figure 5.4F. effects of changing the diameter in pipe 6 on the head losses computations

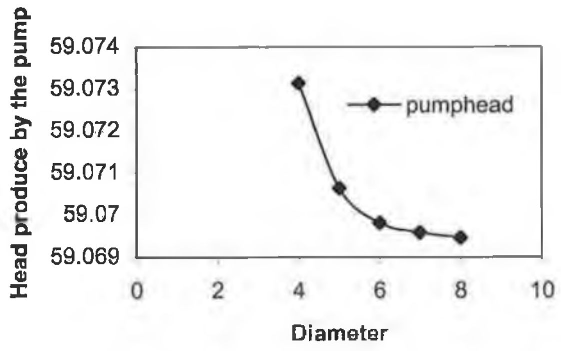


Figure 5.4G. Effects of changing the diameter in pipe 6 on the head produced by pump in pipe 1

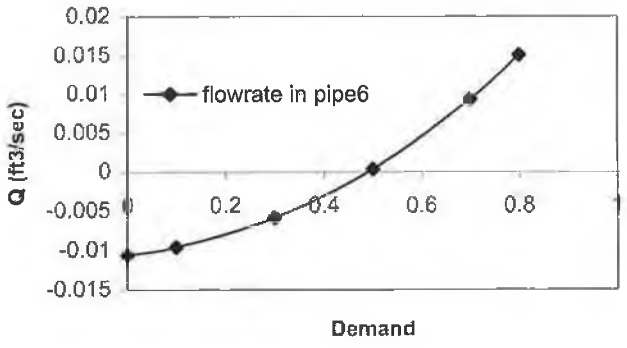


Figure 5.4H . the effect of the demand changes on the flow rate computation in pipe 6

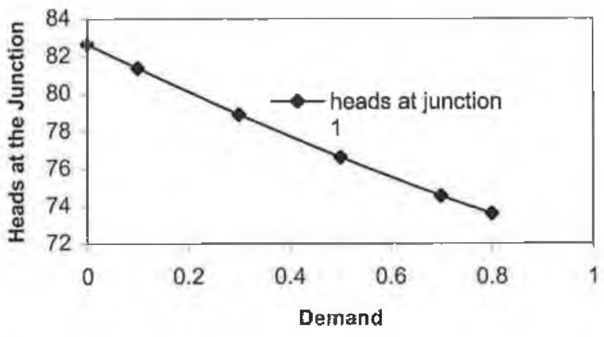


Figure 5.4 I.the effect of the demand changes at junction 2 on the head losses at junction1

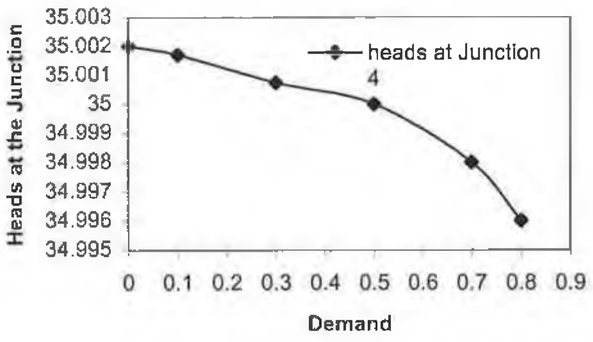


Figure 5.4J . the effect of the demand changes at junction 2 on the head losses at junction 4

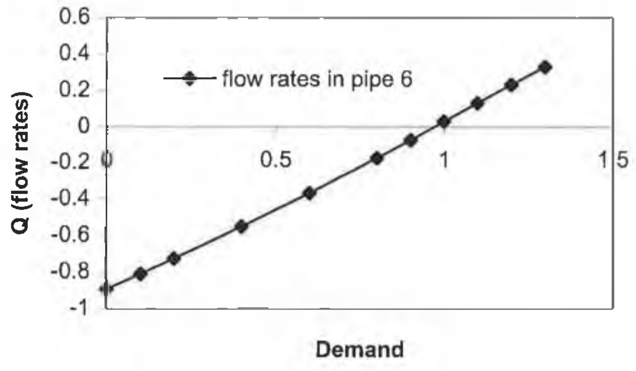


Figure 5.4K . the effect of the demand changes at junction 4 on the flow rate computation at in pipe 6

CHAPTER 6

Conclusion and Suggestion For Further Work

6.1 Conclusion

The present project deals with the steady flow analysis of incompressible, single-phase fluid flows and the computer program developed is general purpose in nature. A number of case studies have been carried out using the program.

Based on verification made using these case studies the following conclusion can be drawn.

The linear theory method of analysis of networks of pipes and pumps provides a convenient method for a simulation. In particular it enables the network to be specified in very simple terms.

The simulation calculates the flow rate in each pipe, the head loss in each pipe can be easily calculated, and the head loss and the pressure at each junction.

The simulation makes use of several numerical methods to simplify the computation such as Gauss elimination method to solve linear simultaneous equations. These computer programs are discussed in appendix (A) and can be summarised as follow

1) A computer program is written to calculate the friction head loss f when the Darcy-Weisbach is implemented to compute the head friction.
2) Polynomial least square method is implemented to represent the head produced by pumps in the computation when the pumps are described by operation data. The computer program is capable of fitting the pump's operating data from the

pump characteristic curve by approximating the head produced by pumps over its working range by a quadratic second order equation.

3) Gauss elimination with partial pivoting method has been used to solve the set of linear simultaneous equation since it is one of the most widely direct methods

However, the program has certain limitations.

- It can not take into account pumps described by horse power
- In the case of including the pressure reducing valve in the pipe networks, the program can indicate if the valves are operating normally or not, but can not rewrite the model of equations to give the precise solutions for the setting of the hydraulic grade line of the PRV.

6.2 Suggestion for further work

Further work could be carried out to link this present study to a real more comprehensive computer simulation of pipe flow network in a refinery complex. Some suggestions for further work are listed as follows:

i. This computer program can be developed to accommodate the non-Newtonian liquid or mixtures including correction for changes in properties with temperature and pressure

ii. Incorporate the linear theory method to simulate the two-phase flow pipeline networks

iii. The effect of heat loss on pressure drop can be predicted by developing the model to calculate the fluid temperature profile based on the operation condition.

iv. In addition, studies dealing with unsteady flows or transient problems, operation and control, acquisition of supply, optimisation of network performance against cost, should be given consideration.

References

References:

1- **Rojiani, Kamal B.,**" *programming in C with Numerical methods for Engineers," Prentice-Hall, Inc, New Jersey, 1st edition, 1996.*

2- **Holmes, Barry.," Through C to C++" Jones and Bartlett** *publishers,Sudbury,MA, 1st editions, 1997.*

3- **William, H. P, et al.,"** *Numerical Recipes in C" Cambridge University Press, Cambridge , 1st edition 1988.*

4- **Herbert, Schildt.,"** *C: The Complete Reference, " Mc Graw Hill, 1st edition, 1987.*

5- **Wood, D. J. & Charles, C.O.A.,"** *Hydraulic Network Analysis Using Linear Theory," Journal of the Hydraulics Journal of the Hydraulics Division, ASCE, Vol.98, Proc.Paper 9031,July 1972,pp 1157-1170.*

6- **Martin, D.W.& Peters, G., "***the application of Newton's method to network analysis by digital computer," Journal of the Institute of Water Engineers, Vol.17, 1963,pp. 115-129.*

7- **Cross, H.,** *"analysis of flow in networks of conduits or conductors," 1936. University of Illinois Bull 286.*

8- **Holland, F.A. & Bragg, R.,"** *Fluid Flow for Chemical Engineers," Edward Arnold, 2nd edition, 1995.*

9- ***Irving H. Shames.**," Mechanics of Fluids," McGraw-Hill, 3rd edition, 1992.*

10- ***Cheremisinoff, Nicolas P.,**" Fluid Flow: Pumps, Pipes and Channels," Ann Arbor Science, Ann Arbor, Michigan, 2nd edition, 1982.*

11- ***Tullis, J.Paul.**" Hydraulics of Pipelines: Pumps, Valves, Cavitation, Transients," John Wiley & sons, Inc, 1st edition, 1989.*

12- ***Jepson, R.,**" analysis of flow in pipe networks," Ann Arbor Science, Ann Arbor, Michigan, 1976.*

13- ***Bruce E. Larock, Roland W. Jeppson, & Gary Z. Watters.**" Hydraulics of Pipeline Systems," CRC Press LLC, 1st edition, 1999.*

14- ***Colebrook, C. F., "** Turbulent flow in pipes, with particular reference to the transition region between the smooth and rough pips laws," J. Inst. Civ. Eng. Lond.,1938-1939,pp133-156.*

15- ***Moody, L, F., "** Friction factors for pipe flow," trans. Am. Soc. .Mech. Eng., 1944,pp 671-684.*

16- ***Wood, D. J., "** An Explicit Friction Factor Relationship," Civil Engineering, ASCE, Vol. 36, No. 12, Dec. 1966,pp. 60-61.*

17- ***Wood, D. J., (1980). "** Users manual-A computer program for the analysis of pressure and flow in pipe distribution systems." Office of Engineering Continuing Education, University of Kentucky, KY..*

18- Jepson, R.W. & Travallaee, A., *" pumps and Reservoirs in Networks by Linear Theory," Journal of the Hydraulics Journal of the Hydraulics Division,ASCE,Vol.101,Proc.Paper 11153, Mar ., 1975,pp 576-580*

19- Hoag, L.N. & Weinberg, G., *"pipeline network analysis by electronic digital computer", Jour.AWWA, Vol. 49, pp. 517,(Jan 1957).*

20- Nogueira, A. C. *," Steady State fluid Network Analysis," J. Hydr.Eng.ASCE 1993, 119(3), pp. 431-436.*

21- Tong, A. L., et.al." *Analysis of Distribution Networks By Balancing Equivalent Pipe Length," Journal of American Water Works Association., Vol. 53,No. 2, Feb 1966, pp.192.*

22- Raman, V., & Raman, S.," *New method of Solving Distribution System Networks Based on Equivalent Pipe Length," Journal of American Water Works Association., Vol. 58,No. 5, May 1966, pp.615.*

23- Chenoweth, H.,& Crawford, C.," *pipe network analysis, " Jour.AWWA, , pp. 55-58,(Jan 1975).*

24- Marlow, T. A., et al., *" Improved Design of Fluid Networks with computers," journal of hydraulic division, ASCE, vol.. 92 , proc paper 4866,July ,1966,pp. 43-61*

25- Adams, R.W., *"Distributions analysis by electronic* computer," *Journal of the Institute of Water Engineers Vol.15, pp.415. (1961).*

26- Jacoby, S,L.S.,and Twigg,D.W.," *computer solutions to Distribution Network Problems," Boeing Research Report,Renton,Wash,.1968*

27- Bellamy, C. J.," *the analysis of networks of pipes and pumps," journal of the institute of engineers, Australia, Vol. 37,No. 4-5, Apr.-May, 1965,p.116-16, (paper No. 1975).*

28- McCormick, M., & Bellamy, C. J.," *A Computer Program for the Analysis of Pipes and Pumps," journal of institution of engineers, Australia, Vol. 98, No. 3, Mar. 1978, pp. 51-58.*

29- McCormick, G. P.," *Non-Linear programming :Theory, Algorithms and Applications," J. Wiley, New York,1983*

30- Martin, D.W.& Peters, G., *"the application of Newton's method to network analysis by digital computer," Journal of the Institute of Water Engineers, Vol.38, No. 3,Mar. 1968,pp. 51-58.*

31- Shamir, U. & Howard, D. D., *"water distribution systems analysis," Journal of the Hydraulics Division,ASCE,Vol.94,Proc.Paper 5758, Jan ., 1968,pp 219-234*

32- Epp, R. & A. Fowler, *" Efficient Code for Steady State Flows in Networks," Journal of the Hydraulics Division, ASCE, Vol.96, January 1970,pp 43-56.*

33- Chadrashekar, M., & Stewart, K. H.," *Sparsity Oriented Analysis of Large Pipe Networks," Journal of the Hydraulics Division, ASCE, Vol.101, Proc.Paper 11260,April 1975,pp 341-355.*

34- Wraga, J., " *Determination of steady state flows and currents in a network," proceeding instrument society of America,vol.9,pt.5, Paper 54-43-4,1954.*

35- Duffin, R. J.," *Nonlinear networks," bulletin of the American mathematical Society, vol. 53,1947,pp.963-971.*

36- McIlroy, M.S., " *direct reading analyzer for pipeline networks," Journal of the Hydraulics Division, ASCE, (Apr.1958).*

37- McPherson, M. B., and Radzidul, J. V.," *Water distribution design and the Mcilroy network analyzer," journal of hydraulics division, ASCE, vol. 84,April.1958, pp.1588-1-1588-15.*

38- Wood, D. J., & Rayes, A.G.," *Reliability of algorithms for pipe network analysis," Journal of the Hydraulics Division, ASCE, 107(7), 1981,pp 1145-1161.*

39- Ormsbee , L. E, & Wood, D. J.,(1986a). *"Explicit pipe Network Calibration." Water Resource. Plng.and Mgmt., ASCE, 112(2),166-18*

40- Ormsbee , L. E,& Wood, D. J.,(1986b). *"Hydraulic design algorithm for pipe Networks.," J. Hydr.Eng. .,ASCE,112(2),1195-1207.*

41- Boulos, P. F. & Wood, D. J.," *Explicit Calculation of Pipe Network Parameters," J. Hydr.Eng.ASCE 1990,116(11), 1329-1344.*

42- Isaac, L. T., & Mills, K. G.," *Linear theory method for pipe network analysis," Journal of the Hydraulics Division, ASCE, Vol.106, 1980,pp 1191-1201.*

43- **Nielsen, H.B.,**" *methods for analyzing pipe networks," Journal of the Hydraulics Division, ASCE, 115(2), 1989,pp 139-157.*

44- **Jepson, R.W. & Davis, A. L.,** *"Pressure Reducing Valves in Pipe Network Analysis," Journal of the Hydraulics Journal of the Hydraulics Division, ASCE, Vol.102, 1976,pp987-1001.*

45- **Avriel. M.,**" *non-linear programming: analysis and methods," prentice-Hall inc., Englewood Cliffs, New Jersey, 1976*

46- **Bazarra, M. S., Sherali, H. D., & Shetty, C. M.,**" *Non-Linear programming: Theory, Algorithms," 2nd edition. J. Wiley, New York,1993*

47- **Wilson, E. L., Bathe, K.J., & Doherty, W.P.,**" *direct solution of large systems of linear equations," Computer and Structures, Vol. 4, Pergamon Press, Inc., New York, N.Y., 1974,pp 363-372.*

Appendix A

Appendix A

NUMERICAL METHODS

A.1 Introduction

The goal of appendix A is to provide enough information so the reader can effectively use some subroutines (functions) that implement commonly used numerical methods. For details about the methods, readers may refer to any of a number of books on numerical analysis listed in references (1) and (3).

With versions that emphasise either Fortran, Pascal, C or Basic provides details on effectively implementing these methods in computer code. The order in which numerical methods will be described in this appendix is (1) Linear simultaneous equations,(2)Roots of non-linear equations,(3)curve fitting.

A.2. Linear simultaneous equations

Simultaneous equations occur in every branch of science and engineering. Many physical problems can be expressed in terms of simultaneous equations. In fact, one of the most basic and important problems in science and engineering is the efficient and accurate solution of systems of simultaneous equations.

A system of simultaneous equations is usually given in the form

$$\begin{array}{llll} a_{11}x_1 & + a_{12}x_2 & + \ldots.. + a_{1n}x_n & = b_1 \\ a_{21}x_1 & + a_{22}x_2 & + \ldots.. + a_{2n}x_n & = b_2 \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot \\ a_{n1}x_1 & + a_{n2}x_2 & + \ldots.. + a_{nn}x_n & = b_n \end{array}$$

where a_{ij} are known coefficients, b_i are known constants, and x_i are the unknowns for which the equations are to be solved. The unknowns $x_i^{'}s$ appear only to the first power and do not multiply each other. Hence, each equation is linear.
Using matrix notation the equations can be written as

$$\begin{bmatrix} a_{11} & a_{12} & \dots\dots & a_{1n} \\ a_{21} & a_{22} & \dots\dots & a_{2n} \\ .. & .. & .. & .. \\ a_{n1} & a_{n2} & \dots\dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ . \\ b_n \end{bmatrix}$$

or as

$$Ax = B$$

where A represents the square array of coefficients a_{ij} and is known as the coefficient matrix, x represents the n component matrix of unknowns x_i and B is the column matrix of the right-hand side constants b_i. In general there is a set of x_i values which when substituted in equations simultaneously satisfies all of them. Under certain circumstances there may be an infinite number of sets of $x_i^{'}s$ that satisfy the simultaneous equations, while under other circumstances there is no set of $x_i^{'}s$ that satisfies the system of equations.
A system of equations is either singular or non-singular. One test of singularity consists of computing the determinant of the coefficient matrix A. If the determinant of the coefficient matrix is not zero, then the system is non-singular. Sometimes however, a system of equations may be near singular and the determinant of the A matrix may be a very small value. Such sets of equations are called *ill conditioned* and from a numerical computational standpoint can lead to unreliable results. Ill-conditioned systems are characterised by the fact that a small change in the initial condition can cause a large change in the result.
Methods for the solution of linear simultaneous equations can be classified in two broad categories:

1- direct methods
2- indirect methods

The term direct refers to a numerical procedure that will provide a solution in a finite number of stages.

One of the most widely used direct methods for the solution of simultaneous equations is *Gauss elimination.*

A.2.1 GAUSS ELIMINATION

The Gauss elimination method, the variables are eliminated one at a time to reduce the original system to an equivalent triangular system. The first step of the procedure consists of eliminating x_1 from the last $(n-1)$ equations. In the second step x_2 is eliminated from the last $(n-2)$ equation. The process is continued until the system is reduced to an equivalent triangular form.

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots\dots\dots + a_{1n}x_n &= b_1 \\
a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \dots\dots\dots + a_{2n}^{(1)}x_n &= b_2^{(1)} \\
a_{33}^{(2)}x_3 + \dots\dots\dots + a_{3n}^{(2)}x_n &= b_3^{(2)} \\
\cdot \qquad & \cdot \\
\cdot \qquad & \cdot \\
a_{nn}^{(n-1)}x_n &= b_n^{(n-1)}
\end{aligned}$$

In the preceding equations superscripts are to indicate that the original coefficients a_{ij} and b_i have been replaced by new values. The value of the superscript corresponds to the step number in the forward elimination process.

After the system of equations has been reduced to an equivalent triangular form, the solution can be found from back-substitution. In back-substitution first x_n is determined from the last equation. This value of x_n is then substituted into the $(n-1)$ equation and x_{n-1} is computes this process is continued until each x_i is determined.

For the pass the following relation can be obtained

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

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

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

For a system of n equations it is necessary to eliminate $x_1, x_2, \ldots\ldots, x_{n-1}$ from the last $(n-1)$ equations. We will need to perform $(n-1)$ passes during the forward elimination process. The number of steps in each successive pass of the forward elimination process decreases by one. In the first pass there are $(n-1)$ steps, in the second pass there are $(n-2)$ steps, and so on until the $(n-1)$ pass which will have only one step.

If $a_{kk}^{(k-1)}$ is zero then the foregoing technique cannot be used. Also if $a_{kk}^{(k-1)}$ is small, then in the forward elimination process we are multiplying by very large numbers and round off errors can occur. However, we can remedy both problems by interchanging rows. A practical way to alleviate the difficulty arising from the presence of zero diagonal elements is to find the largest element in the rows. This process is called *partial pivoting*

A.2.2.C program for Gauss Elimination

The C program solves linear simultaneous equations using the Gauss elimination technique with partial pivoting. Input to the program consists of the square matrix A containing the coefficients a_{ij} and the column vector B containing the right hand side constants b_i.

Problem analysis

The program uses the Gauss elimination to solve the system [A]{ x} ={ B}, where A is the matrix of known coefficients, B is the vector of known constant, and x is the column matrix of the unknowns.

The Call to this Function should contain a statement of the form

*Gauss_Elimination (a,b,n,*det)*

The important variables needed to implement Gauss elimination method are as follows:

Program variables

Number of equations to be solved; A must contain N*N values, and B must contain N values (*int n*)

Elements of matrix A (*double a* [ ] [ ])

Elements of B vector (*double b* [ ])

Program

There are two functions in the program: main ( ) and *Gauss_elimination( )*.The main( ) program prints a heading and a brief description of the program. It reads in the number of equations and saves this in the integer variable n . It then reads in the elements of the matrix of known coefficients and stores these in the two-dimensional array a [ ][ ]. This array is declared as type double. In the statement

double a [MAXSIZE][MAXSIZE];

The symbolic constant MAXSIZE represents the maximum number of rows and columns. The constant is defined to have a value of 20 in the pre-processor directive

#define MAXSIZE 20

Function *main ()* also reads in the values of the right-hand side constants. These are stored in the one-dimensional array b[ ] which is declared to have max elements.

The computations are performed in function *Gauss_elimination ( )* . This function is called from *main( )* as follows:

return_val = Gauss_elimination (a,b,n,det);

Based on the value returned by Gauss_elimination ( ), main( ) prints the solution or an error message indication that the matrix is singular.

The function Gauss_elimination ( ) uses the Gauss elimination procedure with partial pivoting to solve the system of equations . During each pass of the forward elimination step, it searches for the maximum coefficient in the pivot column, and if necessary, interchanges the row containing the element, which has the largest absolute value with the row containing the pivot element. The variable npivot is incremented by one each time a row interchange is performed. Thus npivot contains the number of row interchanges that took place during the procedure.

At the end of the row interchange, the function checks the value of the pivot element. If the absolute value of the pivot element is less than a prescribed tolerance value (close to zero), the function sets the error_flag equal to 1 and returns. The tolerance value specified in *Gauss_elimination ( )* is 1e-30.

The next step involves eliminating the coefficients of x_i in the rows $i+1$ through n where i is the current pass number. During this step a multiplier is computed, and new values of the b_i constants and the a_{ij} coefficients are computed. These steps are repeated until the A matrix has been reduced to upper triangular form.

The next operation performed in *Gauss_elimination( )* is backsubstitution. The results of the are stored in the array b [] thus the function returns the solution in the b [] array, which is destroyed upon return from the function.

The function returns an integer value representing the status of the computation. a return value of 1 indicates that the matrix is singular and a return value of 0 indicates that a solution was obtained the function also returns the determinant through the pointer variable ptr_det.

A.3 Roots of non-linear equations

There are many applications in science and engineering, which involve finding the roots of an equation of the form

$$y = f(x)$$

The function $f(x)$ on the left side of the equation is usually a non-linear function or a transcendental function.

One of the more common tasks in science and engineering consists of finding the roots of non-linear equation. The roots of an equation are defined as the values of x to satisfy an equation of the form $y = f(x) = 0$. In general, the roots may be real, complex, or both. Also, the number of roots may be finite or infinite. The real roots of an equation are represented by the points where the graph of the function $y = f(x)$ crosses the x-axis.

There are many sophisticated techniques for determining the roots of non-linear equations on computers. Most of these methods are based on an iterative approach, which means that we have to specify an initial guess of the root, and the method will compute and improved estimate of the root. This procedure is repeated until the desired accuracy is achieved.

A.3.1NEWTON-RAPHSON METHOD

One of the most widely used root finding techniques is the Newton-Raphson method. The Newton-Raphson method is illustrated in figure A3.1. Let x_1 represent an arbitrary first trial. A value for the second trial is obtained by drawing a tangent to the curve at point A. The intersection of this tangent line with the x axis is the second trial. By definition, the slope of the tangent at A is

$$slope = \tan\theta = \frac{f(x_1)}{x_1 - x_2}$$

From which

$$x_2 = x_1 - \frac{f(x_1)}{\tan\theta}$$

The slope tan is also the derivative of the function at point x_1 , which is symbolised by $f'(x_1)$. Thus the Newton-Raphson method for the second trial becomes

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

An iterative scheme can now be set up as follows:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

This formula can be used repeatedly to find improved approximations to the real root x_r.

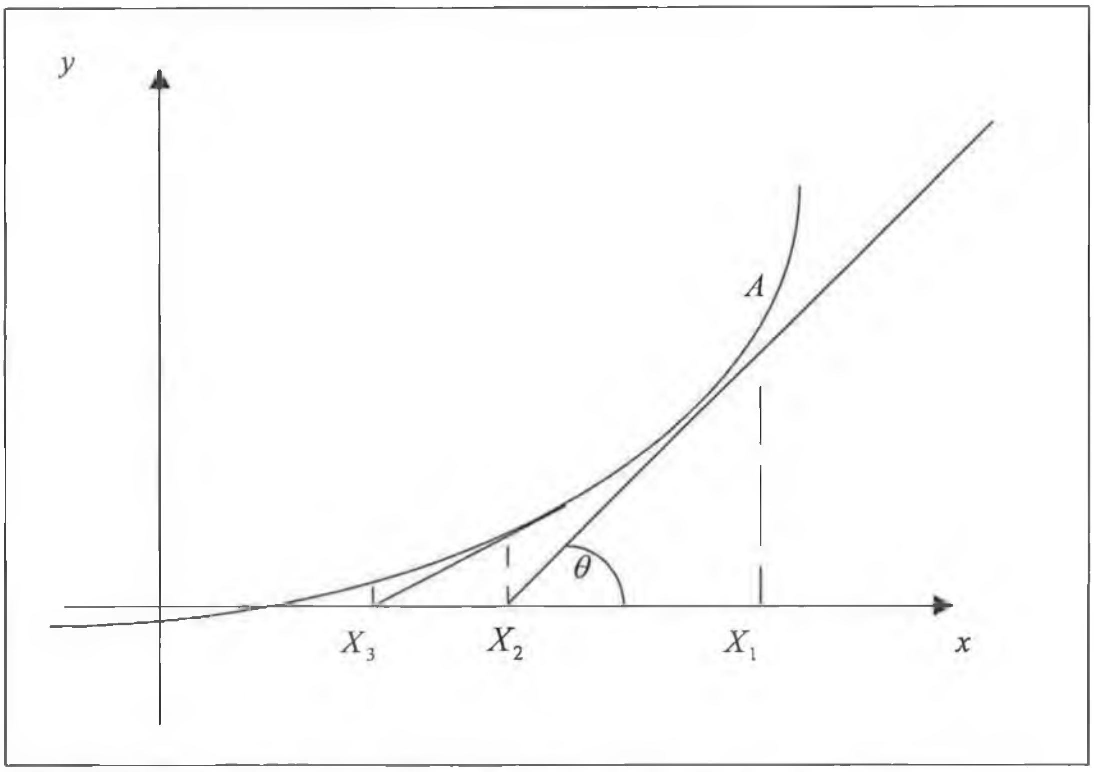


Figure A.3.1 **Newton's method for finding the root of an equation**

A problem with the Newton-Raphson method is that it may fail to converge under some circumstances. The choice of location for the starting point will greatly influence the speed of convergence. The method has difficulty in converging if the slope of the curve $f'(x)$ is small. It can be shown that if the second derivative $f''(x)$ goes to infinity, the method will fail to perform properly. Newton's method does not converge for the case of multiple roots, since the conditions for this case are $f(x) = 0$ and $f'(x) = 0$.

Despite the foregoing limitations, the Newton-Raphson method is the most popular method for finding a root of an equation. The attraction of the Newton-Raphson method is that it converges very rapidly. When the errors are small, each error is inversely proportional to the square of the previous error, which gives much faster convergence than the linear relationship that exists for some of the other root finding techniques.

For the purpose of practical computation several tests must be performed when using Newton's method the test for convergence can be based on the condition that

$$|f(x_{n+1})| < \varepsilon$$

Where ε is a small number or on

$$|x_{n+1} - x_n| < |x_{n+1} \cdot \varepsilon_1|$$

Where ε_1=1 to 5 percent. In addition to the above test for convergence, we will also need to evaluate the performance of the method at each iteration to determine whether it is possible to calculate the iteration using Newton's method. This can be accomplished by checking the absolute value of the derivative $f'(x_{n+1})$ to determine if it is close to zero and checking if the value of the function evaluated at x_{n+1} is greater than the value at x_n.

$$|f(x_{n+1})| > |f(x_n)|$$

If either of these conditions is true, it may not be fruitful to continue the iteration .In addition to the foregoing two conditions, it may also be necessary to limit the number of iterations by setting an upper bound on the number of iterations.

A.3.2Computer use with Darcy-Weisbach equation Roots of non-linear equations

Because the equations for determining f for smooth and transitional flow are implicit, requiring that problem solved by Darcy-Weisbach equation be solved by trial , methods easily adapted to computer computations are described in this section. One very effective method for obtaining f in computer applications is to obtain an estimate of f from equation (2.8) initially assuming rough flow, and then iteratively correcting this value of f by equation (2.11). A computer algorithm implementing this approach will be discussed in details in this section.

The Newton- Raphson method is an iterative scheme which starts with an estimate to the solution and repeatedly computes better estimate.

In using the Newton method the equation containing the unknown (which we will call x when describing the method in general), is expressed as a function which equal to zero when the correct solution is substituted into the equation or $f(x)=0$. For instance the friction factor equation (2.11) in the transition region would be written as,

$$F(f)=\frac{1}{\sqrt{f}}-1.14+2\log_{10}(\frac{e}{D}+\frac{9.35}{R_e\sqrt{f}})=0$$

And the derivative needed to solve equation (2.11) is

$$F''(f) = -\frac{1}{2f\sqrt{f}} - \frac{9.35\log_{10} \in}{f(\frac{e}{D}+\frac{9.35}{R_e\sqrt{f}})R_e\sqrt{f}}$$

in solving for f, equation (2.11) may be used whether the turbulent flow is smooth, transitional, or rough. However, since the Newton-Raphson method does require an initial guess, and this can explicitly be supplied by equation for turbulent rough flow, it is desirable to be able to distinguish rough flow from transitional flow without looking at a moody diagram. A close approximation of the curve on the moody diagram, which separates these flows, is

$$\frac{V\sqrt{f/8}}{v}e = 100$$

Problem statement

A computer program for solving for f for the Darcy-Weisbach equation using the Newton-Raphson method should include the following features:

1. Read in the specification such as *D*, *e* (or *e/d*) ,V (or *Q* or *Re*),*v*, and *L*

2.Compute *Re* and test whether *Re*<2100. If so f= 64/*Re* otherwise

3.Compute an initial value for f from the rough equation, equation (2.8)

4.Compute $\left(eV\sqrt{\frac{f}{8}}\right)/v$ and if this quantity is greater than 100, then the f from step 3 is correct, otherwise

5.Solve equation (2.11) by Newton – Raphson method. Appendix B shows the computer program, which accomplishes this.

The program computes the roots of an equation of the form f=*0* using the Newton-Raphson method. Input to the program consists of an initial estimate of the root, x1, the desired tolerance and the maximum number of iterations.

The major tasks performed by the program are the following:

1. Read in input data from the function *linear_Darcy( )*

2. Call *Newton_Raphson( )* function to find the value of f (root of the equation)

3. Prints the root of the equation.

The important variables needed to implement Newton-Raphson method on a computer are the following:

Input Variables

1.Pipe parameters such as

1.1 *e*[] which is equal to e/d	(*double e*)
1.2.Reynolds number	(*double Re*)
2.Desired tolerance	(*double epsilon*)
3.Maximum number of iterations	(*double max_iter*)

Output variables

An estimated value of the friction factor f

There are four functions in the program are as follows:

1. *main ()*- controls operation of the program and calls other functions. Also prints the result
2. *Newton_Raphson( )*-computes the root of an equation of the form $F(f) = 0$ using the Newton-Raphson method.
3. *f( )*- returns the value of a function $F(f)$ evaluated at f .this function is called by *Newton_Raphson( )*
4. *df( )* returns value of derivative of the function $F(f)$

Program

A C program that implements the Newton-Raphson method is given appendix B. The program contains four functions, *main( ),Newton_Raphson( ),f( ),and df( ).* The main program obtains the necessary input, calls function an error message if the function Newton was unsuccessful in obtaining a result because the derivative was close to zero or the number of iterations exceeded the maximum number of iterations. The input to the program consists of an initial guess to the root (*f*), which is calculated from the equation (2.11), the desired tolerance, epsilon, and the maximum number of iterations, *max_iter.*

The computations are performed in the function *Newton_Raphson().* The function header for *Newton_Raphson ( )* is

```
int Newton_Raphson ( double epsilon, double e, double Re, int max_iter,
                              double *ptr_f1, int *ptr_num1_iter)
```

The function expects four arguments: e, Re is the parameters, epsilon is the desired tolerance, and *max_iter* is the maximum number of iterations. The function returns a value of type *int* indicating that the function was successful in computation. A return value of 0 indicates that the function was successful in obtaining a root within the desired tolerance and maximum number of iterations, a value of 1 indicates that the method did not converge because the derivative was close to zero. A return value of 2 indicates that a root(estimated value of *f*) could not be found within the specified number of iterations. The function returns the f in the pointer variable *ptr_f1*. It also returns the number of iterations in the pointer variable *ptr_num1_iter*. The function uses two variable *x_prev* and *x_curr* to store the previous and current estimates of the root.

The computations are performed within the body of while loop:

*While(*ptr_num_iter < max_iter)*

The body of while loop is executed as long as the number of iterations is less than the maximum number of iterations. The function first compute the value of the derivative at *x_prev*. this is stored in the variable *derf*. If the absolute value of *derf* is less than the symbolic constant *NEARLY_ZERO* (which is defined to be a small value close to zero),the function returns a value of 1 : otherwise , it computes

x_curr= x_prev-f(x_prev,e,Re)/derf;

It then checks for convergence using the following if condition:

*If (fabs (x_curr-x_prev)<fabs(x_curr * epsilon)*

If this condition is true, then the function returns a value of 0 indicating that *f* has been found within the requested number of iterations.

If at the end of the while loop convergence is not achieved, the function returns a value of 2.

The function *f( )* and *df( )* compute $F(f)$ and $F''(f)$ for a given value of f .

The function *f* expect three arguments :x which is the value of the x_prev (estimated f) ,and *e* ,*Re* which are the parameters that needed to solve $F(f)$.

The function *df()* computes the value of the derivative $F''(f)$ from the equation (2.8).

To use the program to compute roots of other functions, the expressions for $F(f)$ and $F''(f)$ contained in the function *f( )* and *df( )* will need to be replaced.

A.4.Curve fitting

A common task in engineering is to formulate mathematical models to describe the behaviour of physical systems. These models usually involve relationships between several variables. The functional relationships are often developed by performing experiments, which yield measurements on the variables of interest, and then fitting a curve or series of curves of the data.

In this chapter we present some techniques for fitting a curve to a given set of data. The procedure presented can be divided into two categories depending on the quality of he data. If the set of points have come from observations or measurements, then each data points is subject to experiments errors, which in some situations could be relatively large in magnitude. In this case we are interested in developing a curve that follows the general trend of the data and passes as close as possible but no necessarily through every data point. This approach is called least square regression.

Linear regression analysis

Engineers frequently perform experiments, which yield measurements on two variables x, and y. they then attempt to determine the fundamental relationships between these two variables. The most common model is based on the assumption of a linear relationship between x and y of the form

$$y = a_0 + a_1 x$$

Where a_0 is the intercept and a_1 is the slop of the passing through the data points $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$. The values of a_0 and a_1 are determined so that the straight line passes through the data points with the least errors.

The most widely used technique for fitting a line through a series of observed data points is the least square method.

The basis for the method is represented graphically in Figure A4.1. The calculated (or predicted) values are given by

$$\hat{y} = a_0 + a_1 x_i$$

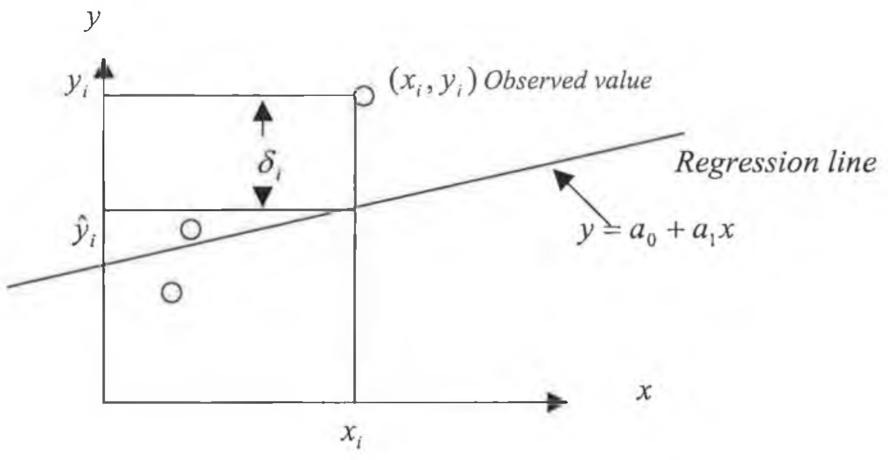


Figure A.4.1 regression line and error associated with point (x_i, y_i).

We can extend the least square method to fit a second-, third-, and higher-order polynomial to the given data set. This is useful for situations where the functional relationship is non-linear and linearization is not possible. The regression coefficients are chosen so as to minimise the sum of the squares of the deviations between the predicted values and the experimental values.

For the problem of fitting a second-order polynomial of the form

$$y(x) = a_0 + a_1 + a_2 x^2$$

The corresponding deviation of the point from the curve is

$$y_i + \hat{y}_i = y_i - (a_0 + a_1 x_i + a_2 x_i^2)$$

And the sum of the squares of the deviations is

$$s = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i)^2$$

Notice that S is a function of the three variables, a_0, a_1, and a_2. We need to take the partial derivatives of S with respect to these three variables and set them equal to zero, that is

$$\frac{\partial S}{\partial a_0} = 0, \qquad \frac{\partial S}{\partial a_1} = 0, \qquad \frac{\partial S}{\partial a_2} = 0$$

This yield the following three linear algebraic equations

$$a_0 n + a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i$$

$$a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum x_i^3 = \sum_{i=1}^{n} x_i y_i$$

$$a_0 \sum_{i=1}^{n} x_i^2 + a_1 \sum_{i=1}^{n} x_i^3 + a_2 \sum x_i^4 = \sum_{i=1}^{n} x_i^2 y_i$$

The solution of these equations can be written in matrix form as

$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x^2{}_i & \sum x_i^3 & \sum x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum y_i x_i \\ \sum y_i x_i^2 \end{bmatrix} \qquad \text{(A4.1a)}$$

The coefficients of the polynomial are calculated by solving the above system of equations by using Gauss elimination technique.

program for polynomial least squares curves fitting

A program to determine the best-fit polynomial of degree N to a set of n data is shown in appendix B .

The important variables needed to implement polynomial least squares curve fitting analysis on a computer are the following:

Variables

Number of data points	(*int num_points*)
Arrays for x_i and y_i values	(*double* *x*[ ] , *Y* [ ])
Degree of polynomial	(*int num_poly*)
Polynomial coefficients	(*double a* [ ])

Algorithm for main

The algorithm for polynomial least square method is as follow:

```
1. read in the number of data points (num_points )
2. read in the degree of the polynomial ( num_poly )
3. read x [ ] and y [ ] arrays
```

Algorithm for poly_leastsqr function

```
1. Compute the sums of products
1.1 Initialise $S$ [0] to num_points
1.2 For $i = 1$ to 2*num_poly
1.2.1       set S[0] to zero
1.2.2       from $j = 0$ to num_points
1.2.3       calculate $S[i] = x[j]$
2.Creat coefficient matrix $c[i]\ [j]$
3.Creat right-hand side vector
4. Call Gauss elimination function for the solution of simultaneous equations
```

5.return the result

   5.1 return to 0 if the computation is successful

   5.2 return to 1 if coefficient matrix is singular

   5.3 return to 2 if the equations are ill conditioned

program

The computations are performed by a separate function called *poly-leastsqr ( ).* This function first assembles the square coefficient matrix and the right hand vector given in equation (A4.1a). The coefficient matrix is saved in the two-dimensional array *C* [ ] [ ], and the right-hand vector is saved in the one-dimensional array *a* [ ]. The function then calls the Gauss elimination routine (presented in section A2.1) to solve the system of equations to obtain the coefficient of the best-fit polynomial.

The function *poly_leastsqr ( )* expects five arguments. The array *x* [ ] and *y*[ ] contain the data values (x_i, y_i).the variables *num_points* represents the number of data points, and the variable *num_poly* represents the degree of the polynomial which is equal to *N-1.* The function returns the coefficients of the best-fit polynomial $a_0, a_1, ...a_N$ in the array *a* [ ].

The function creates two local arrays, a one-dimensional array *S* [ ] and a two-dimensional array *C* [ ] [ ] . The array *S* [ ] used to store the various sums that are needed to create the coefficient matrix. The elements of the array *S* [ ] are obtained from

$$S[K] = \sum_{i=1}^{n} x_i^k$$

When *n* is the number of data points (represented by the variable *num_points* in the program). Thus, $S[1] = \sum x_i$, $S[2] = \sum x_i^2$, and so on. The first element of the array is equal to the number of data points, n.

The elements of the coefficient matrix *C*[ ][ ] are obtained from the array as shown:

$$\begin{bmatrix} S[0] & S[1] & S[2] & \dots\dots & S[N] \\ S[1] & S[2] & S[3] & \dots\dots & S[N+1] \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ S[N] & S[N+1] & S[N+2] & \dots\dots & S[2N] \end{bmatrix}$$

Where N is the degree of the polynomial (represented by the variable *num_poly* in the function). The relationship between the elements of C [ ] [ ] and S [ ] is

$C[i][j] = S[i + j]$

Thus once the various sums are computed it is a relatively easy task to build the coefficient matrix C [ ] [ ] .

The right-hand vector is saved in the array a [ ] . The elements of a [ ] are obtained from

$$a[K] = \sum_{i=1}^{n} y_i x_i^k$$

The function *poly_leastsqr( )* first creates the arrays S [ ] and a [ ] . It then creates the array C [] [] by placing the elements of S [ ] in their appropriate positions in C [ ] [ ] . It then calls the function *Guass_elimination ( )*

Result=Gauss_elimination (C, a , num_poly+1, & det) ;

To solve the system of equations. The solution is returned in the array a [ ] . Thus, upon return, the array a [ ] contains the coefficient of the best-fit polygon. The function *Guass_elimination( )* returns a value of type *int* indicating the status of the computation. Function *poly_leastsqr ( )* also returns a value of type *int* . The value returned by *poly_leastsqr ( )* is the same value that was returned to it by the function *Guass_elimination().*

Appendix B

(Computer Program Source Code)

```
/*============================================================================*/
/*============================================================================*/
/*     "WELCOME TO THE PIPE NETWORKS SIMULATION COMPUTER PROGRAM "            */
/*         " SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING "               */
/*                       DUBLIN CITY UNIVERSITY                               */
/*                    ( THIS PROGRAM IS DEVELOPED BY )                        */
/*                MR. NASSER EMHMMED  SALEM KHAMKHAM                          */
/*                  CHECKED BY PROF. HASHMI SALIM                             */

/*============================================================================*/

/*============================================================================*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define  MAX_SIZE 150
#define MAXPOINTS 100
#define TRUE 1
#define FALSE 0
#define nearly_zero  1e-40


void main (void);

/*============================================================================*/
void input(char*file_input);

/*============================================================================*/
void output(char*file_input,char*file_output);

/*============================================================================*/

int Linear_Darcy(double QJ[],double QF[],int NP,
                  int NJ,int NL,int JA[],int JB[], int NNJ,int NN[],
                  int JN[][MAX_SIZE],int NPVR,
                  int IFLOW[],int LP[][MAX_SIZE],int LPL[],
                  int NPS,int NLSJ,int LPSL[],int LPS[][MAX_SIZE],
                  double AO[],double ELV[],int NUNIT,int NoR,
                  int NPUMP,int NLPUM,int LPUMSL[],int LPUMS[][MAX_SIZE],
                  double HHO[],double ho[],double D[],double E[],double VIS,
                  double epsilon,
                  double tol,double deq,int max_iter,double KLL[],int NP_PRV[],
                  double L[],double h_loss[],double H_pump[],double QJPV[],
                  double ELT[],double ENGY[],double spg,double H_Jun[],
                  double PRESS[],int pump_line[],double HGL_ELV[] );

/*============================================================================*/
int Linear_Hazen_Williams(double QJ[],int NP,
                   int NJ,int JA[],int JB[],int NL,int NNJ,int NN[],int JN[][MAX_SIZE],
                   int NPVR,int IFLOW[],int LP[][MAX_SIZE],int LPL[],
                   int NPS,int NLSJ,int LPSL[],int LPS[][MAX_SIZE],
                   double AO[],double ELV[],int NUNIT,int NoR,
                   int NPUMP,int NLPUM,int LPUMSL[],int LPUMS[][MAX_SIZE],
                   int NP_PRV[],double HHO[],double ho[],double CHW[],double KLL[],
                   double D[],double L[],double tol,int max_iter,double h_loss[],
                   double H_pump[],double QJPV[],
                   double ELT[],double ENGY[],double spg,
                   double H_Jun[],double PRESS[] );

/*============================================================================*/
```

```
int Gauss_elimination (double a[][MAX_SIZE],double  b[],int n,double *ptr_det);

/*=================================================================================*/
int Newton_Raphson(double epsilon,double e,double Re,
                     int max_er,double *ptr_f1,int *ptr_num1_iter);

int Newton_Raphson2(double f1,double epsilon ,double e,double Re,
          int max_er,double *ptr_f2,int *ptr_num1_iter);

/*=================================================================================*/

 double f(double x,double e,double Re);

 double df(double x,double e,double Re);

/*=================================================================================*/

int poly_leastsqr(double x[],double y[],int num_points,int num_poly ,double ao[]);

/*=================================================================================*/

/*=================================================================================*/
/*   function: main()                                                              */
/*   this function main() determines the type of operations to be                 */
/*   performed and calls the appropriate computational functions, and display      */
/*   list of choice whether to show all the inputs or to print                     */
/*   out the results of the program                                                */
/*=================================================================================*/

/*=================================================================================*/

void main (void )

 {                 /* ------- function main() --------------- */
       char file_input[81];
       char file_output[81];
       int choice;
       while(choice!=3)
       {
                                   /* display list of choice */

      printf("\n \n \t \t MAIN MENU ");
      printf("\n \t 1- Create  a new input file or uptate  an  existing inputfile ");
      printf("\n \t 2- shows list of contents of the output file");
      printf("\n \t 3- Exist the  program");
        choice=0;
             while ( choice<1 || choice>3)
             {
                  printf("\n \t your choice (1,2 or 3)?");
                  scanf("%d",&choice);
             }
              if ( choice!=3)
              {

                 printf("\n \t Enter the filename of the input file  :");
                 scanf("%s",file_input);
                 printf("\n \t Enter the filename of the output file :");
                 scanf("%s",file_output);
                 switch(choice)
                  {
                case 1:
                input(file_input);
                break;
                case 2:
                output(file_input,file_output);
                break;
                  }
              }
        }
 }

/*==============================================================================*/

/*==============================================================================*/
/*                                                                              */
/*      function : input()                                                      */
/*      This function read in the network parameters required for              */
/*      for describing the pipe networks such as number of pipes,               */
/*      number of junctions, the system units which has been used, the type     */
/*      of equations used to define the friction losses,number of loops,etc.    */
/*      This function also stored and write these data in input file , which    */
/*      can be very useful for updating the data for the network if that is     */
/*      necessary.                                                              */
/* =============================================================================*/

/*==============================================================================*/

 void input(char*file_input)

 {


FILE *input_file;
int exit_flag=FALSE;
char buffer[500];
char FLUID[81];
int i,ii,j,k;
int max_iter,PUMUNIT,NP,NJ,NL,NPUMP,NUNIT,NPS,NNJ,NLJ,NLSJ,NLPUM,qq;
int num_poly,num_points,result,NPVR,NoR,LS,LL,NNN,type,Demand_Unit;
int FFF,FF,QAST,J56,J57,k23,k22,ZL,ZL2;
int IFLOW[MAX_SIZE],NN[MAX_SIZE],JN[MAX_SIZE][MAX_SIZE];
int LPL[MAX_SIZE],LPS[MAX_SIZE][MAX_SIZE],LPSL[MAX_SIZE];
int LPUMS[MAX_SIZE][MAX_SIZE],LPUMSL[MAX_SIZE],QQ[MAX_SIZE];
int JA[MAX_SIZE],JB[MAX_SIZE],QAS[MAX_SIZE],NP_PRV[MAX_SIZE];
int pump_line[MAX_SIZE],LP[MAX_SIZE][MAX_SIZE];
double deq,tol,epsilon,spg,VIS;
double D[MAX_SIZE],L[MAX_SIZE],QJ[MAX_SIZE],E[MAX_SIZE],ENG_RV[MAX_SIZE];
double ELV[MAX_SIZE];
double AO[MAX_SIZE],BO[MAX_SIZE],HO[MAX_SIZE],HHO[MAX_SIZE],ho[MAX_SIZE];
double x[MAXPOINTS], y[MAXPOINTS],ao[MAXPOINTS],bo[MAX_SIZE],HTV[MAX_SIZE];
double CHW[MAX_SIZE],KLL[MAX_SIZE],ELT[MAX_SIZE],ENGY[MAX_SIZE],HFF[MAX_SIZE];

/*==============================================================================*/

 input_file=fopen(file_input,"w");

      if(input_file==NULL)
      {
        printf("\n cannot open file %s",file_input);
        return;
      }

 while(!exit_flag)

 {
```

```
printf("          the formula used to compute head losses is      ");
printf("\n        Darcy-Weisbach-------------------->[ 1 ]");
printf("\n        Hazen-Williams-------------------->[ 2 ]");
scanf("%d",&type);

printf("        System Geometery             ");

printf("\n   number of pipes in the network            ");
scanf("%d",&NP);
printf("\n   number of junctions in the network        ");
scanf("%d",&NJ);
printf("\n  number of real loops in the network        ");
scanf("%d",&NL);
printf("\n  number of source pumps in the network      ");
scanf("%d",&NPUMP);
printf("\n number of Resevoirs in the Network          ");
scanf("%d",&NoR);
printf("\n number of Pressure Reducing valves in the network         ");
scanf("%d",&NPVR);

fprintf(input_file,"  %d  %d  %d  %d  %d  %d  %d ",type,NP,NJ,NL,NPUMP,NoR,NPVR);

/*=====================================================================================*/

printf("\n\t  ============[ THE SYSTEM USES  UNITS   ]=================\n ");
printf("\n  pipe diameter in[ Feet    ] &  pipe lenght in [ Feet   ]------> (0) ");
printf("\n  pipe diameter in[ Inches  ] &  pipe lenght in [ Feet   ]------> (1) ");
printf("\n  pipe diameter in[ Meters  ] &  pipe lenght in [ Meters ]------> (2) ");
printf("\n  pipe diameter in[ C-meter ] &  pipe lenght in [ Meters ]------> (3)\n ");

scanf("%d",&NUNIT);
fprintf(input_file,"  %d ",NUNIT);

/*=======================================================================================*/

ii=1;

printf("\n ==========[ INPUT DATA FOR JUNCTIONS ]=================== ");
printf("\n the units of the demand at each junctions  ");

printf("\n   There is a Demand at the junction in [gallon/minute     ]->  [ 1 ]");
printf("\n   There is a Demand at the junction in [cubic feet/second ]->  [ 2 ]");
printf("\n   There is a Demand at the junction in [cubic meter/second]->  [ 3 ]");

      scanf("%d",&Demand_Unit);
      fprintf(input_file,"%d\n",Demand_Unit);




for(i=1;i<=NJ;++i)
{
  printf("\n ==========[ Input data for the junction[%d]================== ",i);

      IFLOW[i]=Demand_Unit;

      fprintf(input_file,"%d\n",IFLOW[i]);

      printf("\n    How many pipes round the  junction                      ");
      scanf("%d",&NNJ);
      fprintf(input_file,"%d \n",NNJ);

      for (j=1;j<=NNJ;++j)
         {
     printf("\n        Number of the pipe at junction                        ");

     printf("\n   if  flow Leaves  the junction ----> The pipe Number is  [ + ]");
     printf("\n   if  flow Enters  the junction-----> The pipe Number is  [ - ]");
     scanf("%d",&JN[i][j]);
     fprintf(input_file,"%d \n",JN[i][j]);
         }
      NN[i]=NNJ;
      fprintf(input_file,"%d \n",NN[i]);
     if(IFLOW[i]==1)
     {
         printf("\n     The flow rates  in    [   GPM ]               ");
         scanf("%lf",&QJ[ii]);

         QJ[ii]=QJ[ii]/449.0;
         fprintf(input_file,"%lf\n",QJ[ii]);
         ii=ii+1;
     }
     if(IFLOW[i]==2)
     {
         printf("\n     The Flow rate in    [   CFS ]                 ");
         scanf("%lf",&QJ[ii]);
         fprintf(input_file,"%lf\n",QJ[ii]);
         ii=ii+1;
     }
 if(IFLOW[i]==3)
     {
         printf("\n    The Flow rate in    [   CMS ]                ");
         scanf("%lf",&QJ[ii]);
         fprintf(input_file,"%lf\n",QJ[ii]);
         ii=ii+1;
     }


 printf("\n         THE ELEVATION OF THE JUNCTION                        ");
       scanf("%lf",&ELT[i]);
       fprintf(input_file,"%lf\n",ELT[i]);

}


     for(i=1;i<=NJ;++i)
     {
         NNJ=NN[i];
     }

/*======================================================================================*/


       tol=0.000001;

       fprintf(input_file,"%lf ",tol);

/*======================================================================================*/



if(type==1)
```

```
{

       printf("\n =========THE FLUID PROPERTIES ===============        ");

       printf("\n\t TYPE OF FLUID                                       ");
       scanf("%s",FLUID);
       fprintf(input_file,"%s ",FLUID);
       printf("\n\t THE FLUID VISCOSITY                                 ");
       scanf("%g",&VIS);
       fprintf(input_file,"%g ",VIS);
       printf("\n\t THE  FLUID SPECIFIC GRAVITY                         ");
       scanf("%lf",&spg);
       fprintf(input_file,"%lf ",spg);
       epsilon=0.001;
       fprintf(input_file,"%lf \n",epsilon);
       deq=0.1;
       fprintf(input_file,"%lf \n",deq);


for(i=1;i<=NP;++i)
{
printf("\n Pipe [%d]",i);
          printf("\n node no 1 connects the pipe");
          scanf("%d",&JA[i]);
          fprintf(input_file,"%d ",JA[i]);
          printf("\n node no 2 connects the pipe");
          scanf("%d",&JB[i]);
          fprintf(input_file,"%d ",JB[i]);
          printf("\n               Pipe Diameter  ");
          scanf("%lf",&D[i]);
          fprintf(input_file,"%lf ",D[i]);
          printf("\n                  Pipe Lenght   ");
          scanf("%lf",&L[i]);
          fprintf(input_file,"%lf ",L[i]);
          printf("\n   Relative roughness of pipe   ");
          scanf("%lf",&E[i]);
          fprintf(input_file,"%lf ",E[i]);
          printf("\n        Minor Lose Coefficient  ");
          scanf("%lf",&KLL[i]);
          fprintf(input_file,"%lf ",KLL[i]);

}
}

/*=============================================================================*/

if(type==2)
{

       for(i=1;i<=NP;++i)
       {
    printf("\n Pipe [%d]",i);
          printf("\n node no 1 connects the pipe");
          scanf("%d",&JA[i]);
          fprintf(input_file,"%d ",JA[i]);
          printf("\n node no 2 connects the pipe");
          scanf("%d",&JB[i]);
          fprintf(input_file,"%d ",JB[i]);
          printf("\n              Pipe Diameter in  ");
          scanf("%lf",&D[i]);
          fprintf(input_file,"%lf ",D[i]);
          printf("\n               Pipe Lenght  in  ");
          scanf("%lf",&L[i]);
          fprintf(input_file,"%lf ",L[i]);
          printf("\n   Hazen-Williams coefficient   ");
          scanf("%lf",&CHW[i]);
          fprintf(input_file,"%lf ",CHW[i]);
          printf("\n              Minor Lose Coefficient  ");
          scanf("%lf",&KLL[i]);
          fprintf(input_file,"%lf\n ",KLL[i]);

        }


}
/*=====================================================================================*/

    max_iter=100;
    fprintf(input_file,"%d ",max_iter);

/*=====================================================================================*/

  printf("\n ================ Input Data for the Real Loops ===================\n ");


        for(i=1;i<=NL;++i)
        {
printf("\n ================ Input Data for the Real Loop [ %d ] ============\n ",i);

        printf("\n How many  pipes in the Real  loop  [ %d ]",i);
        scanf("%d",&NLJ);
        fprintf(input_file,"%d \n",NLJ);

         printf("\n the direction of the loop always is ***Clockwise****");

         printf("\n if the direction of the flow in the pipe Clockwise -------[+] ");
         printf("\n if the direction of the flow in the pipe anti-Clockwise---[-] ");

            for(j=1;j<=NLJ;++j)
            {
             printf("\n the number of the pipe in the loop");
             scanf("%d",&LP[i][j]);
             fprintf(input_file,"%d \n",LP[i][j]);
            }

           LPL[i]=NLJ;
           fprintf(input_file,"%d \n",LPL[i]);
        }

/*=====================================================================================*/

for(i=1;i<=NP;++i)
{
    k22=JA[i];
    k23=JB[i];
    if((k22+k23)<=abs(k22-k23))
    {


        printf("\n Eneter the Reservoir elevation which connected to pipe   [%d ] ",i)
;
        scanf("%lf",&ENGY[i]);
        fprintf(input_file,"%lf \n",ENGY[i]);
    }
```

```
}



/*=============================================================================*/

/*=============================================================================*/
        printf("\n ==========[   Input data for  pumps        ]==========  ");


   for(i=1;i<=NPUMP;++i)
   {
            printf("\n enter the pipe number contains the pump");
            scanf("%d",&pump_line[i]);
            fprintf(input_file,"%d \n",pump_line[i]);
            printf("\n type of the operator for the pump");
            printf("\n type of pump data   :( [1]--> operating data               ");
            printf("\n                     :( [2]--> performance operating data   ");
            printf("\n                     :( [3]--> usefull horse power          ");

            scanf("%d",&PUMUNIT);
            fprintf(input_file,"%d \n",PUMUNIT);
            printf("\n number of pipes round the pump");
            NLPUM=2;
            fprintf(input_file,"%d \n",NLPUM);



  for(j=1;j<=NLPUM;++j)
  {                printf("\n forming th transformation equations for the pumps");
                   printf("\n in the form  -Q+G=B/2A    ");

                   printf("\n\n the number of the pipe in the loop");
                   printf("\n the sign [ - ] goes with the pipe number         ");
                   printf("\n the sign [ + ] goes with the number of the pump  ");
                   scanf("%d",&LPUMS[i][j]);
                   fprintf(input_file,"%d \n",LPUMS[i][j]);

                   QQ[j]=LPUMS[i][j];

                   qq=abs(QQ[j]);

                   if(PUMUNIT==1)
                   {
                      if(qq>NP)
                      {
                       printf("\n the pump is described by a Quadratic Equation   ");
                       printf(" Hp = AoQ**2 +BoQ + Ho, In which Ao ,Bo and Ho are ");
                       printf(" constants detemined from the pump curve           ");
                       printf("\n constant A ");
                       scanf("%lf",&AO[qq]);
                       fprintf(input_file,"%lf \n",AO[qq]);
                        printf("\n constant B ");
                        scanf("%lf",&BO[qq]);
                        printf("\n constant HO ");
                        scanf("%lf",&HO[qq]);
                        HHO[qq]=BO[qq]/(2*AO[qq]);
                        fprintf(input_file,"%lf \n",HHO[qq]);
                      ho[qq]=HO[qq]-((pow(BO[qq],2.0))/(4*AO[qq]));
                      fprintf(input_file,"%lf \n",ho[qq]);
                      }
```

```
                }

     if(PUMUNIT==2)
     {
               printf("\n the pump is described by performance operating data ");

                  if(qq>NP)
                  {
                    printf("degree of polynomial ");
                    scanf("%d",&num_poly);
                    printf("number of data points ");
                    scanf("%d",&num_points);
                        for(k=1;k<=num_points;++k)
                        {
                    printf("\n enter x[%d];",k);
                    scanf("%lf",&x[k]);
                    printf("\n enter y[%d];",k);
                    scanf("%lf",&y[k]);
                        }

               result=poly_leastsqr(x,y,num_points,num_poly,ao);

               if(result==1)
                {
                  printf("\n the cofficient matrix is singular ");
                  return ;
                }
                   if(result==2)
                   {
                     printf("\n The equations are ill");
                     return ;
                   }
               printf("\n the cofficient of the best fit polynomial are");
                        for(k=1;k<=num_poly+1;++k)
                        {
                              printf("\n a(%d) = %lf ",k,ao[k]);
                        }
                    for(k=1;k<=num_poly+1;++k)
                        {
                        bo[k]=ao[k];
                        }

                          HO[qq]=ao[1];
                          BO[qq]=ao[2];
                          AO[qq]=ao[3];
                          fprintf(input_file,"%lf \n",AO[qq]);

                          HHO[qq]=BO[qq]/(2*AO[qq]);
                          fprintf(input_file,"%lf \n",HHO[qq]);
                          ho[qq]=HO[qq]-((pow(BO[qq],2.0))/(4*AO[qq]));
                          fprintf(input_file,"%lf \n",ho[qq]);
                  }

     }
   }
    LPUMSL[i]=NLPUM;
fprintf(input_file,"%d \n",LPUMSL[i]);

    }

/*===============================================================================*/

    if(NoR==0 && NPUMP==0)
```

```
      {
         LS=0;
         NPS=LS;
      }
      else
      {
      LS=NoR+NPUMP-1;
      LL=LS+NL;
      NNN=NJ+LL;
        if(NNN>NP)
        {
         LS=LS-(NNN-NP);
        }
         NPS=LS;
         fprintf(input_file,"%d \n",NPS);
      }
/*=====================================================================================*/

printf("\n==============================================================");

        printf("\n\n the Network needs [%d] pesudo loops  ",LS);

printf("\n==============================================================");


/*================================================================================*/
for(i=1;i<=NP;++i)
{

    ENG_RV[i]=0.0;
}


if (NPVR>0)
{
    for(i=1;i<=NPVR;++i)
    {
        printf("\n the pipe numper contains the PRV");
        scanf("%d",&NP_PRV[i]);
        fprintf(input_file,"%d\n",NP_PRV[i]);
        ZL=NP_PRV[i];
        printf("\n the elevation of the artificial reservoir");
        scanf("%lf",&ENG_RV[ZL]);


    }
}
/*=====================================================================================*/

printf("\n ============[   data for pseudo loop  ]============================== ");


 for(i=1;i<=NPS;++i)
 {
     printf("\n   Suggest a path to connect the two reservoir for pseudo loop [%d] ",i
);

     printf("\n\n if the path contains a pump [ sign the pump by Np+1] ( + )->if the
");
     printf("  flow in the pipe which contains the pum in the direction of the path ")
;

         printf("\n How many pipes in the path");
```

```
scanf("%d",&NLSJ);
fprintf(input_file,"%d \n",NLSJ);

HFF[i]=0.0;

  for(j=1;j<=NLSJ;++j)
{
   printf("\n the number of the pipe in the loop");
   printf("\n   [ + ] if flow in the same direction of the energy line ");
   printf("\n   [ - ] if flow oppesite the energy line direction       ");
   scanf("%d",&LPS[i][j]);
   fprintf(input_file,"%d \n",LPS[i][j]);



     LPSL[i]=NLSJ;
     fprintf(input_file,"%d \n",LPSL[i]);

     FFF=LPS[i][1];
     FF=abs(FFF);

     QAS[j]=LPS[i][j];

     QAST=abs(QAS[j]);


 if(QAST<=NP)
 {
        J56=JA[QAST];

        J57=JB[QAST];

          if((J56+J57)<=abs(J56-J57))
          {
          if(QAST==FF)
          {
             HTV[1]=ENGY[QAST];

          }
        else
         {
             HTV[2]=ENGY[QAST];

         }

          }
                if(NPVR>0)
                {
                   for(k=1;k<=NPVR;++k)
                   {

               ZL2=NP_PRV[k];
                      if(ZL2==QAST)
                      {
                HTV[2]=ENG_RV[ZL2];

                      }
                   }
                }

 }

       if(QAST>NP)
```

```
                {

                 if(QAS[j]>0)
                 {

                HFF[i]=HFF[i]-ho[QAST];

                 }
                else
                {

                HFF[i]=HFF[i]+ho[QAST];

                }
                }


            }

 ELV[i]=HTV[1]-HTV[2]+HFF[i];

 fprintf(input_file,"%lf \n",ELV[i]);


 }

/*================================================================================*/


        printf("\n do u want to enter more data- Y/N? " );

          scanf("%s",buffer);

          if ( buffer[0]=='N' ||buffer[0]=='n')
             exit_flag=TRUE;


 }

           fclose(input_file);

   }     /* end of the main function void(char filename) */

/*================================================================================*/

/*=============================================================================*/
/*    function : output()                                                      */
/*                                                                             */
/*    This function reads in the data for the network from the function       */
/*    input(). Also prints the result of the program.                         */
/*    This function has two sub-functions, function Linear_Darcy() if the     */
/*    Darcy equations is used to define the friction losses, and function     */
/*    Linear_Hazen_William() if the Hazen-William equations is used.          */
/*                                                                             */
/*                                                                             */
/*=============================================================================*/
```

```
/*==================================================================*/

void output(char*file_input,char*file_output)




{


    FILE *outfile;
    FILE *input_file;
    int exit_flag=FALSE;
    char FLUID[81];
int i,j,ii;
int NP,NJ,NL,NPUMP,NoR,type,NUNIT,NPVR,result;
int max_iter,PUMUNIT,NPS,NNJ,NLJ,NLSJ,NLPUM,qq;
int k22,k23,npt2,Demand_Unit;
int IFLOW[MAX_SIZE],NN[MAX_SIZE],JN[MAX_SIZE][MAX_SIZE],LP[MAX_SIZE][MAX_SIZE];
int LPL[MAX_SIZE],LPS[MAX_SIZE][MAX_SIZE],LPSL[MAX_SIZE];
int LPUMS[MAX_SIZE][MAX_SIZE],LPUMSL[MAX_SIZE],QQ[MAX_SIZE];
int JA[MAX_SIZE],JB[MAX_SIZE],NP_PRV[MAX_SIZE],pump_line[MAX_SIZE];

double tol,deq,epsilon,spg,VIS;
double QF[MAX_SIZE],D[MAX_SIZE],L[MAX_SIZE],E[MAX_SIZE],QJPV[MAX_SIZE];
double KLL[MAX_SIZE],h_loss[MAX_SIZE],H_pump[MAX_SIZE],H_Jun[MAX_SIZE];
double AO[MAX_SIZE],ELV[MAX_SIZE],CHW[MAX_SIZE],PRESS[MAX_SIZE];
double HHO[MAX_SIZE],ho[MAX_SIZE],ELT[MAX_SIZE],ENGY[MAX_SIZE];
double HGL_ELV[MAX_SIZE],QJ[MAX_SIZE];


/*==================================================================*/


     input_file=fopen(file_input,"r");

      if(input_file==NULL)
      {

        printf("\n cannot open file %s",file_input);
        return;
      }




    outfile=fopen(file_output,"w");

      if(outfile==NULL)
      {
        printf("\n cannot open file  %s",file_output);
        return;
      }
/*==================================================================*/



fscanf(input_file," %d  %d  %d  %d  %d  %d  %d ",&type,&NP,&NJ,&NL,&NPUMP,&NoR,&NPVR);

fscanf(input_file,"  %d    ",&NUNIT);

ii=1;
```

```
fscanf(input_file,"%d \n",&Demand_Unit);
for(i=1;i<=NJ;++i)
{

      fscanf(input_file,"%d\n",&IFLOW[i]);
      fscanf(input_file,"%d \n",&NNJ);



      for (j=1;j<=NNJ;++j)
         {

           fscanf(input_file,"%d \n",&JN[i][j]);
         }
      NN[i]=NNJ;
     fscanf(input_file,"%d \n",&NN[i]);
     if(IFLOW[i]==1)
     {

         fscanf(input_file,"%lf\n",&QJ[ii]);
         ii=ii+1;
     }
     if(IFLOW[i]==2)
     {

         fscanf(input_file,"%lf\n",&QJ[ii]);
         ii=ii+1;
     }
 if(IFLOW[i]==3)
     {

         fscanf(input_file,"%lf\n",&QJ[ii]);
         ii=ii+1;
     }

fscanf(input_file,"%lf\n",&ELT[i]);


}


     for(i=1;i<=NJ;++i)
     {
         NNJ=NN[i];
     }



      fscanf(input_file,"%lf ",&tol);


/*=====================================================================================*/

if(type==1)
{

       fscanf(input_file,"%s ",FLUID);

       fscanf(input_file,"%lf ",&VIS);

       fscanf(input_file,"%lf ",&spg);

       fscanf(input_file,"%lf \n",&epsilon);
```

```
        fscanf(input_file,"%lf \n",&deq);


          for(i=1;i<=NP;++i)
          {
           fscanf(input_file,"%d ",&JA[i]);

           fscanf(input_file,"%d ",&JB[i]);

           fscanf(input_file,"%lf ",&D[i]);

           fscanf(input_file,"%lf ",&L[i]);

           fscanf(input_file,"%lf ",&E[i]);

           fscanf(input_file,"%lf ",&KLL[i]);

          }

}
/*===========================================================================*/

if(type==2)
{

        for(i=1;i<=NP;++i)
        {

           fscanf(input_file,"%d ",&JA[i]);

           fscanf(input_file,"%d ",&JB[i]);

           fscanf(input_file,"%lf ",&D[i]);

           fscanf(input_file,"%lf ",&L[i]);

           fscanf(input_file,"%lf ",&CHW[i]);

           fscanf(input_file,"%lf\n ",&KLL[i]);

        }
}


fscanf(input_file,"%d ",&max_iter);

/*===========================================================================*/

         for(i=1;i<=NL;++i)
         {

         fscanf(input_file,"%d \n",&NLJ);


             for(j=1;j<=NLJ;++j)
             {

             fscanf(input_file,"%d \n",&LP[i][j]);
             }

            LPL[i]=NLJ;
             fscanf(input_file,"%d \n",&LPL[i]);
         }

 /*==================================================================================*/


 for(i=1;i<=NP;++i)
 {
     k22=JA[i];
     k23=JB[i];
     if((k22+k23)<=abs(k22-k23))
     {

        fscanf(input_file,"%lf \n",&ENGY[i]);
     }
 }



 /*==================================================================================*/


 for(i=1;i<=NPUMP;++i)
 {
               fscanf(input_file,"%d \n",&pump_line[i]);

               fscanf(input_file,"%d \n",&PUMUNIT);

               fscanf(input_file,"%d \n",&NLPUM);


      for(j=1;j<=NLPUM;++j)
      {

                   fscanf(input_file,"%d \n",&LPUMS[i][j]);

                   QQ[j]=LPUMS[i][j];

                   qq=abs(QQ[j]);

                   if(PUMUNIT==1)
                   {
                      if(qq>NP)
                      {


                        fscanf(input_file,"%lf \n",&AO[qq]);

                        fscanf(input_file,"%lf \n",&HHO[qq]);

                        fscanf(input_file,"%lf \n",&ho[qq]);
                   }
                      }

                if(PUMUNIT==2)
                {


                      if(qq>NP)
                   {


                          fscanf(input_file,"%lf \n",&AO[qq]);
```

```
                              fscanf(input_file,"%lf \n",&HHO[qq]);

                              fscanf(input_file,"%lf \n",&ho[qq]);
                         }

                    }



          }


   LPUMSL[i]=NLPUM;
fscanf(input_file,"%d \n",&LPUMSL[i]);

}

/*=========================================================================*/


      fscanf(input_file,"%d \n",&NPS);



/*=========================================================================*/

if (NPVR>0)
{
    for(i=1;i<=NPVR;++i)
    {

       fscanf(input_file,"%d",&NP_PRV[i]);


    }
}

/*=========================================================================*/


for(i=1;i<=NPS;++i)
{

         fscanf(input_file,"%d \n",&NLSJ);

         for(j=1;j<=NLSJ;++j)
         {

            fscanf(input_file,"%d \n",&LPS[i][j]);


              LPSL[i]=NLSJ;
              fscanf(input_file,"%d \n",&LPSL[i]);


         }
             fscanf(input_file,"%lf \n",&ELV[i]);

}
```

```
/*===========================================================================================*/

if(type==1)
{
             result=Linear_Darcy(QJ,QF,NP,NJ,NL,JA,JB,
                                NNJ,NN,JN,NPVR,IFLOW,LP,LPL,
                                NPS,NLSJ,LPSL,LPS,AO,ELV,NUNIT,NoR,
                                NPUMP,NLPUM,LPUMSL,LPUMS,
                                HHO,ho,D,E,VIS,
                                epsilon,tol,deq,max_iter,KLL,
                                NP_PRV,L,h_loss,H_pump,QJPV,ELT,ENGY,spg,
                                H_Jun,PRESS,pump_line,HGL_ELV);


             if(result==0)
            {
              //  printf("\n the soluation is ");

            }
       else
       {
           printf("\n \t ********   Warning  ********** ");
            printf("\n did not converge ");
            printf("\n The current estimated flow rates are ");
            for (i=1;i<=NP;++i)
            printf("\n Q(%d)=%lf",i,QF[i]);
            scanf("%d",&NP);

       }


}

/*===========================================================================================*/

/*===========================================================================================*/

if(type==2)
{
          result=Linear_Hazen_Williams(QJ,NP,NJ,JA,JB,NL,
                  NNJ,NN,JN,NPVR,IFLOW,LP,LPL,
                       NPS,NLSJ,LPSL,LPS,AO,ELV,NUNIT,NoR,
                            NPUMP,NLPUM,LPUMSL,LPUMS,NP_PRV,
                                HHO,ho,CHW,KLL,D,L,tol,max_iter,h_loss,H_pump,QJPV,
                                ELT,ENGY,spg,H_Jun,PRESS);

            if(result==0)
            {
                // printf("\n the soluation is ");

            }
           else
           {

            printf("\n \t ********   Warning  ********** ");
            printf("\n did not converge ");
            printf("\n The current estimated flow rates are ");
            for (i=1;i<=NP;++i)
            printf("\n Q(%d)=%lf",i,QJ[i]);
            scanf("%d",&NP);
        }


}

/*=====================================================================================*/


/*=====================================================================================*/

for(i=1;i<=85;++i){
fprintf(outfile,"=");
}

fprintf(outfile,"\n\n\n       WELCOME TO THE PIPE NETWORKS SIMULATION COMPUTER PROGRAM
  \n");
fprintf(outfile,"\n                 SCHOOL OF MECHANICAL & MANUF. ENGINEERING   \n");
fprintf(outfile,"\n                         DUBLIN CITY UNIVERSITY
\n");
fprintf(outfile,"\n          THIS PROGRAM IS DEVELOPED BY :( MR. NASSER EMHMMED  SALEM
 KHAMKHAM )\n");
fprintf(outfile,"\n          CHECKED BY PROF. HASHMI SALIM        \n\n");
for(i=1;i<=85;++i){
fprintf(outfile,"=");
}

/*=====================================================================================*/


if(type==1)
{


fprintf(outfile,"\n\n                PIPE NETWORK DESRIBTION                   ");
fprintf(outfile,"\n\n   TYPE OF FLUID IS         %s",FLUID);
fprintf(outfile,"\n   THE FLUID VISCOSITY  IS  %lf ",VIS);
fprintf(outfile,"\n   THE FLUID SPECIFIC GRAVITY IS  %lf",spg);

fprintf(outfile,"\n\n DARCY-WEISBACH FORMULA USED TO COMPUTE FRICTION LOSS ");

fprintf(outfile,"\n\n THE FOLLOWING RESULTS ARE OBTAINED AFTER TRIALS WITH AN ACCURACY
  %lf ",tol);

fprintf(outfile,"\n\n THE SYSTEM HAS %d PIPES  %d JUNCTIONS %d  REAL LOOPS  %d PSEUDO
LOOPS  ",NP,NJ,NL,NPS);
fprintf(outfile,"\n\n                    PIPE OUTPUT     \n\n");
for(i=1;i<=85;++i){
fprintf(outfile,"=");
}
}

/*=====================================================================================*/

if(type==2)
{


fprintf(outfile,"\n\n PIPE NETWORK DESRIBTION ");
fprintf(outfile,"\n TYPE OF FLUID IS %s",FLUID);
fprintf(outfile,"\n THE FLUID SPECIFIC GRAVITY IS  %lf",spg);
fprintf(outfile,"\n\n HAZEN-WILLIAM FOMULA USED TO COMPUTE FRICTION LOSS");
```

```
fprintf(outfile,"\n\n                          PIPE OUTPUT      \n\n");
for(i=1;i<=85;++i){
fprintf(outfile,"=");
}
}
/*=====================================================================================*/

if(type==1||type==2)
{
fprintf(outfile,"\n PIPE      NODES         LENGTH     DIAMETER       FLOW RATES
  HEAD    ");
fprintf(outfile,"\n NO      FROM      TO                                    (Q)
  LOSS    \n");
/*=====================================================================================*/

if(NUNIT==0)
{
fprintf(outfile,"                               Feet        Feet     ");

if(Demand_Unit==1)
{

    fprintf(outfile,"     Gallon/min           Feet \n");
}
if(Demand_Unit==2)
{

    fprintf(outfile,"     Feet3/s           Feet \n");
}



}
/*=====================================================================================*/
if(NUNIT==1)
{
fprintf(outfile,"                               Feet        Inch     ");

if(Demand_Unit==1)
{

    fprintf(outfile,"     Gallon/min           Feet \n");
}
if(Demand_Unit==2)
{

    fprintf(outfile,"     Feet3/s           Feet \n");
}



}




/*=====================================================================================*/
if(NUNIT==2)
{
fprintf(outfile,"                               Meter        Meter     ");

if(Demand_Unit==3)
{
    fprintf(outfile,"     Meter3/s            Meter \n");
}
}
/*===================================================================================*/
if(NUNIT==3)
{
fprintf(outfile,"                              Meter        C-Meter     ");

if(Demand_Unit==3)
{
    fprintf(outfile,"     Meter3/s            Meter \n");
}
}

/*===================================================================================*/



for(i=1;i<=85;++i){
fprintf(outfile,"=");
}

for (i=1;i<=NP;++i)
{
if(Demand_Unit==1)
{
    for(i=1;i<=NP;++i){
        QF[i]=449.0*QF[i];}

}
fprintf(outfile,"\n %d        %d        %d     %lf   %4.2lf        %lf             %lf\n",i,JA
[i],JB[i],L[i],D[i],QF[i],h_loss[i]);

}

for(i=1;i<=85;++i){
fprintf(outfile,"=");
}

/*===================================================================================*/
fprintf(outfile,"\n\n\n                            JUNCTIONS OUTPUT     \n\n");
for(i=1;i<=85;++i){
fprintf(outfile,"=");
}

fprintf(outfile,"\n JUNCTION      DEMAND        ELEVATION        HEAD             PRESSUR
E        HGL ELV ");
fprintf(outfile,"\n    NO                                        LOSS                  \
n");

if(NUNIT==2||NUNIT==3)
{
fprintf(outfile,"                    Meter3/sec      Meter     ");

if(Demand_Unit==3)
{
    fprintf(outfile,"     Meter          KPa      feet \n");
}
}
/*==================================================================================
=*/
```

```
if(NUNIT==0||NUNIT==1)
{


if(Demand_Unit==1)
{
    fprintf(outfile,"                   Gallon/min          FEET     ");

    fprintf(outfile,"        FEET            Lb/in2          feet  \n");
}
if(Demand_Unit==2)
{
    fprintf(outfile,"                   Feet3/s          FEET     ");

    fprintf(outfile,"        Feet            Lb/in2          feet \n");
}



}


for(i=1;i<=85;++i){
fprintf(outfile,"=");
}

for (i=1;i<=NJ;++i)
{

fprintf(outfile,"\n    %d          %lf         %lf        %lf        %lf     %lf  \n",i,QJ[i],
ELT[i],H_Jun[i],PRESS[i],HGL_ELV[i]);

}

/*=====================================================================================*/

if (NPUMP>0)
{

fprintf(outfile,"\n\n      THE HEAD PRODUCED BY PUMPS ARE\n ");
for(i=1;i<=85;++i){
fprintf(outfile,"=");
}


if(NUNIT==0||NUNIT==1)
{


  for (i=1;i<=NPUMP;++i)
       {



npt2=pump_line[i];
fprintf(outfile,"\n    The Pump in Pipe [ %d ]               produced Head      =  %lf
  [ feet ] \n",npt2,H_pump[npt2]);


       }

}
if(NUNIT==2||NUNIT==3)
{


  for (i=1;i<=NPUMP;++i)
        {



npt2=pump_line[i];
fprintf(outfile,"\n    The Pump in Pipe [ %d ]               produced Head      =  %lf
  [ meter ] \n",npt2,H_pump[npt2]);



        }

}


}
 for(i=1;i<=85;++i){
fprintf(outfile,"=");
}

/*=====================================================================================*/
        for(i=1;i<=NP;++i)
        {
            if(NPVR>0)
            {

                              for(j=1;j<=NPVR;++j)
                              {

                                    if(QF[i]==QJPV[j])
                                    {

                                       if(QF[i]>0)
                                       {

                       fprintf(outfile,"\n\n THE PRV[%d] IN PIPE [%d] IS OPERATING NEOMALL
Y \n",j,i);

                                       }

                                        if(QF[i]<0)
                                       {

                                    for(i=1;i<=85;++i){
                                    fprintf(outfile,"=");
                                    }
fprintf(outfile,"\n    ************ (((((( WARNING ))))))***************\n");
                                   for(i=1;i<=85;++i){
                                    fprintf(outfile,"=");
                                    }
fprintf(outfile," \n\nTHE PRV[%d] IN PIPE [%d] IS NOT OPERATING NEOMALLY \n",j,i);

                                        }




                                    }
                              }
            }
        }
```

```
}


for(i=1;i<=85;++i)
{
fprintf(outfile,"=");
}


fclose(input_file);
fclose(outfile);


    }   /* end of main for the function output()   */



/*=================================================================================*/


/*==============================================================================*/
/*     function : Linear_Darcy                                                  */
/*     If the Darcy-Weisbach equation is used, this function will be called     */
/*     This function will set up the mass continuity , loops equations as      */
/*     arrays, and calling the Gauss_eliminations() to solve the simultaneous   */
/*     linear equations.                                                        */
/*     This function computes the flow rates of each pipe in the network based  */
/*     on the use of the Darcy-Weisbach equation for computing the friction     */
/*     loss.                                                                    */
/*                                                                              */
/*                                                                              */
/*==============================================================================*/


/*=================================================================================*/


int Linear_Darcy(double QJ[],double QF[],int NP,int NJ,
                 int NL,int JA[],int JB[], int NNJ,int NN[],int JN[][MAX_SIZE],
                 int NPVR,int IFLOW[],int LP[][MAX_SIZE],int LPL[],
                 int NPS,int NLSJ,int LPSL[],int LPS[][MAX_SIZE],
                 double AO[],double ELV[],int NUNIT,int NoR,
                 int NPUMP,int NLPUM,int LPUMSL[],int LPUMS[][MAX_SIZE],
                 double HHO[],double ho[],double D[],double E[],double VIS,
                 double epsilon,
                 double tol,double deq,int max_iter,double KLL[],int NP_PRV[],
                 double L[],double h_loss[],double H_pump[],double QJPV[],
                 double ELT[],double ENGY[],double spg,double H_Jun[],
                 double PRESS[],int pump_line[],double HGL_ELV[] )


 {



/*-------------------------------------------------------------------------------*/

int max_er=100;
int num1_iter=0;
int num_iter=0;
int tol_exceeded=TRUE;
int ret2_val,return_value,PRVF;
int i,j,IJ,IIJ,ss,RR,WW,result,EEE,asda,npt2;
int NXX,NEXT,N1,N2,N3,MBEG,MJJ,mem,JMAX,NTEP;
int k,J8,J9,J,J1,J2,JJ,ff,NIPE,NUN,NUNO,KL,KJ;

int FF[MAX_SIZE],gg[MAX_SIZE],EE[MAX_SIZE],axa[MAX_SIZE],npt[MAX_SIZE];
int JIJ[MAX_SIZE],JJUN[MAX_SIZE],MPL[MAX_SIZE],JX[MAX_SIZE],flagprv[MAX_SIZE];
int M[MAX_SIZE],NEX[MAX_SIZE],NIX[MAX_SIZE],JPIP[MAX_SIZE],JJI[MAX_SIZE];


double Re,e,f2,f1,BE,AE,EP,det,G2;

double Y[MAX_SIZE],YY[MAX_SIZE];
double m[MAX_SIZE][MAX_SIZE],Q[MAX_SIZE],A[MAX_SIZE][MAX_SIZE];
double ddd[MAX_SIZE],DEQ[MAX_SIZE],Q1[MAX_SIZE],Q2[MAX_SIZE];
double QM1[MAX_SIZE],Q_old[MAX_SIZE],V[MAX_SIZE],V1[MAX_SIZE],V2[MAX_SIZE];
double Re1[MAX_SIZE], Re2[MAX_SIZE];
double EXPP[MAX_SIZE],KP[MAX_SIZE],AR[MAX_SIZE],ARL[MAX_SIZE];
double LL[MAX_SIZE],H_pump2[MAX_SIZE],QJPV2[MAX_SIZE];



/*==========================================================================*/

for(i=1;i<=NP;++i)
{

if(NUNIT==0)
    {
        E[i]=E[i]/( 12.0*D[i]);

    }

    if(NUNIT==1)
    {
          E[i]=E[i]/D[i];

          D[i]=D[i]/12.0;

    }

    if(NUNIT==2)
    {
        E[i]=E[i]*0.01/D[i];

    }

    if(NUNIT==3)
    {
           E[i]=E[i]/D[i];

           D[i]=0.01*D[i];

    }



}


 if (NUNIT==0||NUNIT==1)
     {
       G2=64.4;

     }
    if (NUNIT==2||NUNIT==3)
     {
       G2=19.62;

     }




if (NUNIT==0 || NUNIT==1)
    {
    for(i=1;i<=NP;++i)
    {

           KP[i]=0.00093517*L[i]/pow(D[i],4.87);

           AR[i]=0.78539392*pow(D[i],2);

           ARL[i]=L[i]/(G2*D[i]*pow(AR[i],2));

    }

    }


if (NUNIT==2|| NUNIT==3)
    {
    for(i=1;i<=NP;++i)
        {

           KP[i]=0.00212*L[i]/pow(D[i],4.87);

           AR[i]=0.78539392*pow(D[i],2);

           ARL[i]=L[i]/(G2*D[i]*pow(AR[i],2));


        }

    }


NUN=NP+NPUMP;

NUNO=NUN+1;

KL=NUN;

KJ=NUN;

/*==========================================================================*/

  for(i=1;i<=KL;++i)
  {
        for(j=1;j<=KJ;++j)
        {
           A[i][j]=0.0;
        }
  }
```

```
/*=========================================================================================*/

   if (NPUMP==0 && NoR==0)
   {
       NJ=NJ-1;
   }
   for(i=1;i<=NJ;++i)
 {

             NNJ=NN[i];

        for(j=1;j<=NNJ;++j)
        {

           IJ=JN[i][j];
           IIJ=abs(IJ);
           if(IJ<0)
           {

               A[i][IIJ]=-1.0;

           }

           if(IJ>0)
           {

               A[i][IJ]=1.0;

           }

        }
 }

/*=========================================================================================*/

ss=1;

  for(i=1;i<=NJ;++i)
  {

        if(IFLOW[i]==0)
        {
           A[i][NUNO]=0.0;


        }

     else

      {

       A[i][NUNO]=QJ[ss];

       ss=ss+1;
      }


}

/*=========================================================================================*/

  RR=NJ;
```

```
for(i=1;i<=NL;++i)
      {
            RR=1+RR;
            NNJ=LPL[i];

for(j=1;j<=NNJ;++j)
{
              gg[j]=LP[i][j];
              IIJ=abs(gg[j]);
              if(IIJ<=NP)
              {
                if(gg[j]>0)
                {
                   A[RR][IIJ]=KP[IIJ];

                }
                      if(gg[j]<0)
                      {
                        A[RR][IIJ]=-KP[IIJ];

                      }
            A[RR][NUNO]=0.0;

            }
        if(IIJ>NP)
        {
                if(gg[j]>0)
                {
                        A[RR][IIJ]=AO[IIJ];
                        A[RR][NUNO]=-ho[IIJ];
                }
                if(gg[j]<0)
                {
                       A[RR][IIJ]=-AO[IIJ];
                       A[RR][NUNO]=ho[IIJ];
                }
        }

}

}


/*==============================================================================*/

   for(i=1;i<=NPS;++i)
      {
            RR=RR+1;

            NLSJ=LPSL[i];

        for(j=1;j<=NLSJ;++j)
        {
              FF[j]=LPS[i][j];

              WW=abs(FF[j]);

              if(WW<=NP)
               {
                      if(FF[j]>0)
                      {
                        A[RR][WW]=KP[WW];
```

```
                    }

                    if(FF[j]<0)
                    {

                     A[RR][WW]=-KP[WW];

                    }
               }

              if(WW>NP)
              {

                  if(FF[j]>0)
                  {
                     A[RR][WW]=AO[WW];

                  }

                  if(FF[j]<0)
                  {
                     A[RR][WW]=-AO[WW];

                  }

              }
        }

        A[RR][NUNO]=ELV[i];


    }


/*=====================================================================================*/

    for(i=1;i<=NPUMP;++i)
    {

        RR=RR+1;
        NLPUM=LPUMSL[i];
        for(j=1;j<=NLPUM;++j)
        {
              EE[j]=LPUMS[i][j];
              EEE=abs(EE[j]);

              if(EE[j]<0)
              {
                A[RR][EEE]=-1.0;
              }

              if(EE[j]>0)
              {
                A[RR][EEE]=1.0;
              }

            if(EEE>NP)
            {
               A[RR][NUNO]=HHO[EEE];

            }
        }
    }

/*=======================================================================================*/

    for(i=1;i<=NUN;++i)
    {
         for(j=1;j<=NUN;++j)
         {
           m[i][j]=A[i][j];

         }

    }

    for(i=1;i<=NUN;++i)
     {
         Q[i]=A[i][NUNO];

     }
/*=======================================================================================*/

result=Gauss_elimination(m,Q,NUN,&det);


   if (result== 0)
   {
         //printf("\n\n \t the sluatio of the  smulation of linear equation  is");
         //printf("\n                the sluation is   ");
   }

    else
    {
     printf("\n \t ********   Warning  ********** ");
     printf("\n\n \t    The Matrix is singular   ");
     scanf("%d",&NP);
    }

/*=======================================================================================*/

for(i=1;i<=NUN;++i)
{
    Q_old[i]=0.0;
}

while(tol_exceeded&&num_iter<max_iter)

{


        if(num_iter==0)
        {
        for(i=1;i<=NUN;++i)
        {
```

```
        Q2[i]=Q[i];
        Q1[i]=Q[i];
        Q2[i]=fabs(Q1[i]);

        }
        }
        if(num_iter>0)
        {
        for(i=1;i<=NUN;++i)
        {
          Q_old[i]=Q1[i];
          QM1[i]=0.5*(Q1[i]+Q[i]);
          Q1[i]=QM1[i];
          Q2[i]=fabs(Q1[i]);



        }

        }
  for(i=1;i<=NP;++i)
     {

        V[i]=(Q2[i])/AR[i];

        DEQ[i]=Q2[i]*deq;

        e=E[i];

        ddd[i]=(Q2[i]-DEQ[i]);

        V1[i]=(Q2[i]-DEQ[i])/AR[i];

        Re1[i]=V1[i]*D[i]/VIS;

        Re=Re1[i];

        return_value=Newton_Raphson(epsilon,e,Re,max_er,&f1,&num1_iter);


        V2[i]=(Q2[i]+DEQ[i])/AR[i];

        Re2[i]=V2[i]*D[i]/VIS;
        Re=Re2[i];

 ret2_val=Newton_Raphson2(f1,epsilon,e,Re,max_er,&f2,&num1_iter);

 switch (return_value)
     {
 case 0:
 //printf(" f1 =%16.20le ",f1);
 break;
 case 1:
printf("\n \t ********   Warning  ********** ");
printf("\n the derivative is close to zero");
printf("\n the current estimate ofthe  f1=%16.20le",f1);
 scanf("%d",&NP);
 case 2:
printf("\n \t ********   Warning  ********** ");
 printf("\n \n newton's method did not converge");
 printf("\n maximum number of iterations exceeded");
 printf("\n the current estimate ofthe f1 =%16.20le ",f1);
scanf("%d",&NP);
```

```
      }

switch (ret2_val)
  {
case 0:
//printf(" f2 =%16.20le ",f2);

break;
case 1:
printf("\n \t ********   Warning  ********** ");
printf("\n newton's method did not coverage");
printf("\n the derivative is close to zero");
scanf("%d",&NP);
break;
case 2:
printf("\n \t ********   Warning  ********** ");
printf("\n maximum number of iterations exceeded");
printf("\n the current estimate ofthe  f2=%16.20le",f2);
scanf("%d",&NP);
  }

BE=(log10(f1)-log10(f2))/(log10(Q2[i]+DEQ[i])-log10(Q2[i]-DEQ[i]));
AE=f1*pow((Q2[i]-DEQ[i]),BE);
EP=1.0-BE;
EXPP[i]=EP+1;
LL[i]=(KLL[i]*pow(Q2[i],1))/(G2*pow(AR[i],2));
ARL[i]=L[i]*AE/(G2*D[i]*pow(AR[i],2));
KP[i]=(ARL[i])*pow(Q2[i],EP);
KP[i]=(LL[i]+KP[i]);

}

/*=======================================================================================*/


  RR=NJ;
for(i=1;i<=NL;++i)
      {
            RR=1+RR;
            NNJ=LPL[i];

for(j=1;j<=NNJ;++j)
{
              gg[j]=LP[i][j];
              IIJ=abs(gg[j]);

            if(IIJ<=NP)
            {
                if(gg[j]>0)
                {
                   A[RR][IIJ]=KP[IIJ];
                }
                     if(gg[j]<0)
                     {
                       A[RR][IIJ]=-KP[IIJ];
                     }
            A[RR][NUNO]=0.0;

            }
        if(IIJ>NP)
        {
                if(gg[j]>0)
                {
                        A[RR][IIJ]=AO[IIJ]*fabs(Q[IIJ]);
                        A[RR][NUNO]=-ho[IIJ];
                }
                if(gg[j]<0)
                {
                        A[RR][IIJ]=-AO[IIJ]*fabs(Q[IIJ]);
                        A[RR][NUNO]=ho[IIJ];
                }
        }

}

}

/*=================================================================================*/


for(i=1;i<=NPS;++i)
    {
        RR=RR+1;

        NLSJ=LPSL[i];

        for(j=1;j<=NLSJ;++j)
      {
        FF[j]=LPS[i][j];

        WW=abs(FF[j]);
        if(WW<=NP)
            {
              if(FF[j]>0)
              {
                A[RR][WW]=KP[WW];
                }

              if(FF[j]<0)
              {

              A[RR][WW]=-KP[WW];
              }
          }

         if(WW>NP)
         {
            if(FF[j]>0)
              {
                A[RR][WW]=AO[WW]*fabs(Q[WW]);
                }

              if(FF[j]<0)
              {

              A[RR][WW]=-AO[WW]*fabs(Q[WW]);
              }


         }
       }




      }
```

```
/*=========================================================================================
====*/

    for(i=1;i<=NUN;++i)
     {
         for(j=1;j<=NUN;++j)
        {
          m[i][j]=A[i][j];

         }

    }
 for(i=1;i<=NUN;++i)
    {
      Q[i]=A[i][NUNO];

    }

/*=======================================================================================*/


result=Gauss_elimination(m,Q,NUN,&det);

if (result == 0)
{
//printf("\n\n \t the sluatio of the  smulation of linear equation  is");
//printf("\n               the sluation is    ");
}

else
{
printf("\n \t ********   Warning  ********** ");
printf("\n\n \t the matrix is singular " );
scanf("%d",&NP);
}


/*=======================================================================================*/

  for(i=1;i<=NP;++i)
 {
 QF[i]=Q1[i];
 }

/*=======================================================================================*/
for (i=1;i<=NP;++i)
{

h_loss[i]=KP[i]*QF[i];
}

for (i=1;i<=NUN;++i)
{

H_pump[i]=0.0;
}


 if (NPUMP>0)
 {
        for(i=1;i<=NPUMP;++i)
        {
            npt[i]=pump_line[i];
            npt2=abs(npt[i]);

            for(j=1;j<=2;++j)
            {
                axa[j]=LPUMS[i][j];
                asda=abs(axa[j]);

                if(asda>NP)
                {

                    H_pump2[npt2]=AO[asda]*Q2[asda]*Q2[asda]+ho[asda];
                     H_pump[npt2]= H_pump2[npt2];





                }
            }
        }
    }




/*===================================================================================*/

      for(i=1;i<=NP;++i)
    {

          if(NPVR>0)
          {
              for(j=1;j<=NPVR;++j)
              {
                  flagprv[j]=NP_PRV[j];
                  PRVF=abs(flagprv[j]);

                   if(PRVF==i)
                   {
                          QJPV[j]=QF[i];
                         if(QF[PRVF]<=0)
                         {
                              QJPV2[i]=QF[i];

                        QJPV[i]=QJPV2[i];


                         }

                         else
                         {

                         }

                   }
              }
          }
```

```
}

/*==================================================================================*/

for(i=1;i<=NP;++i)
 {
 YY[i]=KP[i]*QF[i];
 }

for(i=1;i<=NJ;++i)
 {
 JX[i]=i;
 }


NTEP=0;
for (j=1;j<=NP;++j)
{
    NIPE=j;
    J1=JA[j];
    J2=JB[j];
    if ((J1+J2)<=abs(J1-J2))
    {
      NTEP=NTEP+1;
      if(JA[j]>0)
      {
          JJUN[NTEP]=JA[j];




      }
      else
      {
          JJUN[NTEP]=JB[j];


      }


      JPIP[NTEP]=j;

    }
}
JJ=0;
   for (j=1;j<=NJ;++j)
   {
     if(JX[j]!=0)
     {
      JMAX=j;
      JJ=JJ+1;
      JJI[JJ]=j;
      JIJ[j]=JJ;


     }
   }
ff=0;
for(i=1;i<=NJ;++i)
{
  NNJ=NN[i];
   M[1]=1;

   if(i<=NJ-1)
   {
       M[i+1]=M[i]+NNJ;

   }
  for(j=1;j<=NNJ;++j)
  {

    MPL[ff+1]=JN[i][j];


      ff=ff+1;

  }

}

for(i=1;i<=NL;++i)
{

NNJ=LPL[i];
   for(j=1;j<=NNJ;++j)
   {

   MPL[ff+1]=LP[i][j];


      ff=ff+1;

   }


}

for(i=1;i<=NJ;++i)
{
   Y[i]=0.0;
}
NEXT=NTEP;



for(j=1;j<=NEXT;++j)
{
      mem=JJUN[j];

      J8=JIJ[mem];

      NEX[j]=J8;

      J9=JPIP[j];

      Y[J8]=ENGY[J9]+YY[J9]-ELT[J8]+H_pump[J9];
      if(JA[J9]==0)
      {
       Y[J8]=ENGY[J9]-YY[J9]-ELT[J8]+H_pump[J9];

      }
}
NXX=0;
```

```
for(i=1;i<=NJ;++i)
{
MO:
     if(Y[i]==0)
     {

for(i=1;i<=NEXT;++i)

{
     J=NEX[i];

     MBEG=M[J];
if(J<NJ)
{

     MJJ=M[J+1]-1;

     for(k=MBEG;k<=MJJ;++k)
     {

         N1=MPL[k];
         N1=abs(N1);

         N2=JA[N1];

         N3=JB[N1];

         if(N2==J)
         {
              if(N3==0)
              {
                   continue;
              }
              if(Y[N3]!=0)
              {
                   continue;
              }
              Y[N3]=Y[N2]-YY[N1]+H_pump[N1];

              NXX=NXX+1;

              NIX[NXX]=N3;

         }
         if(N2==0)
         {
           continue;
         }
         if(Y[N2]!=0)
         {
              continue;
         }

         Y[N2]=Y[N3]+YY[N1]+H_pump[N1];

         NXX=NXX+1;

         NIX[NXX]=N2;

     }
}
}
NEXT=NXX;
if(NEXT!=0)
{

  for (i=1;i<=NEXT;++i)
  {
      NEX[i]=NIX[i];


  }
  goto MO;
}

    }
}



for(j=1;j<=NJ;++j)

{
    if(NUNIT==0 ||NUNIT==1)
    {
    PRESS[j]=(Y[j])*spg*62.4/144.0;
    }
    if(NUNIT==2 ||NUNIT==3)
    {
    PRESS[j]=(Y[j])*spg*9.807;
    }

}
for(j=1;j<=NJ;++j)

{

    HGL_ELV[j]=Y[j]+ELT[j];


}
for(j=1;j<=NJ;++j)
{
    H_Jun[j]=Y[j];

}


/*=================================================================================*/



tol_exceeded=FALSE;
for(i=1;i<=NP;++i)
{
if(fabs(Q1[i]-Q_old[i])>fabs(Q_old[i]*tol))
tol_exceeded=TRUE;
}
++num_iter;
}                         /*-----------------while -------------*/
return(tol_exceeded);



}
```

```
/*==================================================================================*/

/*=============================================================================*/
/*      function : Linear_Hazen_William                                         */
/*      If the Hazen-William equation is used, this function will be called     */
/*      This function will set up the mass continuity , loops equations as     */
/*      arrays, and calling the Gauss_eliminations() to solve the simultaneous  */
/*      linear equations.                                                       */
/*      This function computes the flow rates of each pipe in the network based */
/*      on the use of the Hazen-William equation for computing the friction     */
/*      loss.                                                                   */
/*                                                                              */
/*                                                                              */
/*=============================================================================*/

/*==================================================================================*/


int Linear_Hazen_Williams(double QJ[],int NP,int NJ,
                     int JA[],int JB[],int NL,int NNJ,int NN[],int JN[][MAX_SIZE],
                     int NPVR,int IFLOW[],int LP[][MAX_SIZE],int LPL[],
                     int NPS,int NLSJ,int LPSL[],int LPS[][MAX_SIZE],
                     double AO[],double ELV[],int NUNIT,int NoR,
                     int NPUMP,int NLPUM,int LPUMSL[],int LPUMS[][MAX_SIZE],
                     int NP_PRV[],double HHO[],double ho[],double CHW[],double KLL[],
                     double D[],double L[],double tol,int max_iter,double h_loss[],
                     double H_pump[],double QJPV[],double ELT[],double ENGY[],
                     double spg,double H_Jun[],double PRESS[] )

 {


int num_iter=0;
int tol_exceeded=TRUE;

int i,j,IJ,IIJ,ss,EEE,RR,result,WW;
int k,J8,J9,J,J1,J2,JJ,ff,NIPE,NUN,NUNO,KL,KJ,PRVF;
int gg[MAX_SIZE],FF[MAX_SIZE],EE[MAX_SIZE];
int JIJ[MAX_SIZE],JJUN[MAX_SIZE],MPL[MAX_SIZE],JX[MAX_SIZE],flagprv[MAX_SIZE];
int M[MAX_SIZE],NEX[MAX_SIZE],NIX[MAX_SIZE],JPIP[MAX_SIZE],JJI[MAX_SIZE];
int NXX,NEXT,N1,N2,N3,MBEG,MJJ,mem,JMAX,NTEP;

double det;
double m[MAX_SIZE][MAX_SIZE],Q[MAX_SIZE],A[MAX_SIZE][MAX_SIZE];
double QM1[MAX_SIZE],Q_old[MAX_SIZE],Q1[MAX_SIZE],Q2[MAX_SIZE];
double KP[MAX_SIZE],DDL[MAX_SIZE];
double Y[MAX_SIZE],YY[MAX_SIZE];


/*==================================================================================*/
for(i=1;i<=NP;++i)
{
    if(NUNIT==1)
    {

          D[i]=D[i]/12.0;

    }



    if(NUNIT==3)
    {

           D[i]=0.01*D[i];

    }
}


/*=======================================================================================*/

if (NUNIT==0|| NUNIT==1)
    {
       for(i=1;i<=NP;++i)
       {

          KP[i]=4.73*L[i]/(pow(D[i],4.87)*pow(CHW[i],1.852));

        }

    }

    if (NUNIT==2|| NUNIT==3)

    {
       for(i=1;i<=NP;++i)
       {

          KP[i]=10.7*L[i]/(pow(D[i],4.87)*pow(CHW[i],1.852));


        }

    }
/*=======================================================================================*/

NUN=NP+NPUMP;

NUNO=NUN+1;

KL=NUN;

KJ=NUN;



for(i=1;i<=KL;++i)
 {
    for(j=1;j<=KJ;++j)
    {
    A[i][j]=0.0;
    }
 }


 if (NPUMP==0 && NoR==0)
   {
       NJ=NJ-1;
   }


/*=======================================================================================*/
```

```
 for(i=1;i<=NJ;++i)
      {

          NNJ=NN[i];

    for(j=1;j<=NNJ;++j)
     {

           IJ=JN[i][j];

        if(IJ<0)
           {
           IIJ=abs(IJ);
           A[i][IIJ]=-1.0;
           }

     if(IJ>0)
             {
           A[i][IJ]=1.0;
           }

   }
}


/*==============================================================================*/

ss=1;
for(i=1;i<=NJ;++i)
{

    if(IFLOW[i]==0)
    {
    A[i][NUNO]=0.0;

    }
    else
  {

    A[i][NUNO]=QJ[ss];

    ss=ss+1;
    }

}

/*==============================================================================*/

RR=NJ;
for(i=1;i<=NL;++i)
      {
            RR=1+RR;

            NNJ=LPL[i];

for(j=1;j<=NNJ;++j)
{
               gg[j]=LP[i][j];

               IIJ=abs(gg[j]);

        if(IIJ<=NP)
        {
```

```
                 if(gg[j]>0)
                 {
                A[RR][IIJ]=KP[IIJ];
                 }


                if(gg[j]<0)
                {
                 A[RR][IIJ]=-KP[IIJ];
                }
               A[RR][NUNO]=0.0;
         }

         if(IIJ>NP)
         {
               if(gg[j]>0)
                 {
                A[RR][IIJ]=AO[IIJ];
                 A[RR][NUNO]=-ho[IIJ];
               }


                if(gg[j]<0)
                {
                 A[RR][IIJ]=-AO[IIJ];
                A[RR][NUNO]=ho[IIJ];
                }

         }

}

}

/*=====================================================================================*/

  for(i=1;i<=NPS;++i)
      {
               RR=RR+1;

               NLSJ=LPSL[i];

               for(j=1;j<=NLSJ;++j)
          {
               FF[j]=LPS[i][j];

               WW=abs(FF[j]);
               if(WW<=NP)
                   {
                     if(FF[j]>0)
                     {
                       A[RR][WW]=KP[WW];
                        }

                     if(FF[j]<0)
                     {

                     A[RR][WW]=-KP[WW];
                     }
                 }

                if(WW>NP)
                {
```

```
                if(FF[j]>0)
                  {
                    A[RR][WW]=AO[WW];
                    }

                  if(FF[j]<0)
                  {

                  A[RR][WW]=-AO[WW];
                  }

              }
          }
              A[RR][NUNO]=ELV[i];

          }

 /*=====================================================================================*/


  for(i=1;i<=NPUMP;++i)
  {

         RR=RR+1;
         NLPUM=LPUMSL[i];
         for(j=1;j<=NLPUM;++j)
         {
           EE[j]=LPUMS[i][j];
             EEE=abs(EE[j]);

         if(EE[j]<0)
           {

           A[RR][EEE]=-1.0;

           }
      if(EE[j]>0)
             {
           A[RR][EEE]=1.0;

           }

      if(EEE>NP)
      {

     A[RR][NUNO]=HHO[EEE];

      }

     }
   }


 /*=====================================================================================*/

     for(i=1;i<=NUN;++i)
     {
     for(j=1;j<=NUN;++j)
     {
     m[i][j]=A[i][j];

     }
     }
```

```
    for(i=1;i<=NUN;++i)
    {
    Q[i]=A[i][NUNO];

    }


 result=Gauss_elimination(m,Q,NUN,&det);


if (result== 0)
{
//printf("\n\n \t the sluatio of the  smulation of linear equation  is");
//printf("\n               the sluation is   ");
}

else
{


printf("\n\n \t the matrix is singular " );
}


 for(i=1;i<=NUN;++i)
{
    Q_old[i]=0.0;
}

while(tol_exceeded&&num_iter<max_iter)

{                                         /*while*/


        if(num_iter==0)
        {
        for(i=1;i<=NUN;++i)
        {
        Q2[i]=Q[i];
        Q1[i]=Q[i];
        Q2[i]=fabs(Q1[i]);
        }
        }
        if(num_iter>0)
        {
        for(i=1;i<=NUN;++i)
        {
          Q_old[i]=Q1[i];
          QM1[i]=0.5*(Q1[i]+Q[i]);
          Q1[i]=QM1[i];
          Q2[i]=fabs(Q1[i]);

        }
        }

    for(i=1;i<=NP;++i)
    {
     DDL[i]=(0.02517*KLL[i]/pow(D[i],4.0))*pow(Q2[i],1.0);
     KP[i]=((4.73*L[i]/(pow(D[i],4.87)*pow(CHW[i],1.852))))*pow(Q2[i],0.852);
     KP[i]=KP[i]+DDL[i];


    }
```

```
/*===========================================================================================*/

for(i=1;i<=NP;++i)
{

        if(NPVR>0)
        {
            for(j=1;j<=NPVR;++j)
            {
               flagprv[j]=NP_PRV[j];
               PRVF=abs(flagprv[j]);

                if(PRVF==i)
                {
                      QJPV[j]=QJ[i];
                     if(QJ[PRVF]<=0)
                     {
                         QJPV[i]=QJ[i];


                     }


                }
            }
        }

}

for (i=1;i<=NP;++i)
{

h_loss[i]=KP[i]*Q2[i];
}

 if (NPUMP>0)
 {
        for (i=NP+1;i<=NUN;++i)
        {
            h_loss[i]=AO[i]*Q2[i]*Q2[i]+ho[i];

        }

 }
if (NPUMP>0)
{
    for (i=1;i<=NPUMP;++i)
    {
        H_pump[i]=h_loss[NP+i];

    }
}

/*===========================================================================================*/


    RR=NJ;
  for(i=1;i<=NL;++i)
      {
            RR=1+RR;
            NNJ=LPL[i];
            for(j=1;j<=NNJ;++j)
            {
              gg[j]=LP[i][j];
              IIJ=abs(gg[j]);

              if(IIJ<=NP)
        {

              if(gg[j]>0)
              {
             A[RR][IIJ]=KP[IIJ];
              }


             if(gg[j]<0)
             {
              A[RR][IIJ]=-KP[IIJ];
             }
            A[RR][NUNO]=0.0;
        }


             if(IIJ>NP)
        {
            if(gg[j]>0)
              {
             A[RR][IIJ]=AO[IIJ]*fabs(Q[IIJ]);
              A[RR][NUNO]=-ho[IIJ];
            }


             if(gg[j]<0)
             {
              A[RR][IIJ]=-AO[IIJ]*fabs(Q[IIJ]);
             A[RR][NUNO]=ho[IIJ];
             }

        }

            }



         }
/*==================================================================================*/
for(i=1;i<=NPS;++i)
      {
            RR=RR+1;

            NLSJ=LPSL[i];

            for(j=1;j<=NLSJ;++j)
         {
            FF[j]=LPS[i][j];

            WW=abs(FF[j]);
            if(WW<=NP)
                {
                  if(FF[j]>0)
                  {
                    A[RR][WW]=KP[WW];
                    }
                    if(FF[j]<0)
                    {

                    A[RR][WW]=-KP[WW];
                    }
                }

               if(WW>NP)
               {
                  if(FF[j]>0)
                    {
                      A[RR][WW]=AO[WW]*fabs(Q[WW]);
                      }

                    if(FF[j]<0)
                    {

                    A[RR][WW]=-AO[WW]*fabs(Q[WW]);
                    }

               }
            }


}



/*======================================================================================*/


    for(i=1;i<=NUN;++i)
     {
         for(j=1;j<=NUN;++j)
        {
         m[i][j]=A[i][j];

         }

    }
  for(i=1;i<=NUN;++i)
    {
      Q[i]=A[i][NUNO];

    }

  result=Gauss_elimination(m,Q,NUN,&det);


if (result == 0)
{
//printf("\n\n \t the sluatio of the  smulation of linear equation  is");
//printf("\n                the sluation is   ");
}

else
{

printf("\n\n\t  ****** Warning ********");
printf("\n\n \t The matrix is singular " );
scanf("%d",&NP);
}
```

```
  for(i=1;i<=NP;++i)
 {
 QJ[i]=Q1[i];
 }

/*=============================================================================*/

for(i=1;i<=NP;++i)
 {
 YY[i]=KP[i]*Q[i];
 }

for(i=1;i<=NJ;++i)
 {
 JX[i]=i;
 }


NTEP=0;
for (j=1;j<=NP;++j)
{
    NIPE=j;
    J1=JA[j];
    J2=JB[j];
    if ((J1+J2)<=abs(J1-J2))
    {
      NTEP=NTEP+1;
      if(JA[j]>0)
      {
          JJUN[NTEP]=JA[j];


      }
      else
      {
          JJUN[NTEP]=JB[j];


      }


      JPIP[NTEP]=j;

    }
}
JJ=0;
   for (j=1;j<=NJ;++j)
   {
     if(JX[j]!=0)
     {
      JMAX=j;
      JJ=JJ+1;
      JJI[JJ]=j;
      JIJ[j]=JJ;


     }
   }
ff=0;
for(i=1;i<=NJ;++i)
```

```
{

  NNJ=NN[i];
   M[1]=1;

   if(i<=NJ-1)
   {
       M[i+1]=M[i]+NNJ;

   }
  for(j=1;j<=NNJ;++j)
  {

    MPL[ff+1]=JN[i][j];


      ff=ff+1;

  }

}

for(i=1;i<=NL;++i)
{

NNJ=LPL[i];
   for(j=1;j<=NNJ;++j)
   {

   MPL[ff+1]=LP[i][j];


     ff=ff+1;

   }


}

/*=============================================================================*/

for(i=1;i<=NJ;++i)
{
   Y[i]=0.0;
}
NEXT=NTEP;



for(j=1;j<=NEXT;++j)
{
     mem=JJUN[j];

     J8=JIJ[mem];

     NEX[j]=J8;

     J9=JPIP[j];

     Y[J8]=ENGY[J9]+YY[J9];

     if(JA[J9]==0)
     {
       Y[J8]=ENGY[J9]-YY[J9];

      }
}
NXX=0;
for(i=1;i<=NJ;++i)
{
MO:
     if(Y[i]==0)
     {


for(i=1;i<=NEXT;++i)

{
     J=NEX[i];

     MBEG=M[J];
     if(J<NJ)
     {

     MJJ=M[J+1]-1;

     for(k=MBEG;k<=MJJ;++k)
     {

        N1=MPL[k];
        N1=abs(N1);

        N2=JA[N1];

        N3=JB[N1];

        if(N2==J)
        {
             if(N3==0)
             {
                  continue;
             }
             if(Y[N3]!=0)
             {
                  continue;
             }
             Y[N3]=Y[N2]-YY[N1];

             NXX=NXX+1;

             NIX[NXX]=N3;

        }
        if(N2==0)
        {
          continue;
        }
        if(Y[N2]!=0)
        {
             continue;
        }

        Y[N2]=Y[N3]+YY[N1];

        NXX=NXX+1;
```

```
              NIX[NXX]=N2;

          }
          }
    }
    NEXT=NXX;

    if(NEXT!=0)
    {

      for (i=1;i<=NEXT;++i)
      {
          NEX[i]=NIX[i];


      }
      goto MO;
    }

          }
    }



    for(j=1;j<=NJ;++j)

    {
        if(NUNIT==0 ||NUNIT==1)
        {
        PRESS[j]=(Y[j]-ELT[j])*spg*62.4/144.0;
        }
        if(NUNIT==2 ||NUNIT==3)
        {
        PRESS[j]=(Y[j]-ELT[j])*spg*9.807;
        }

    }

    for(j=1;j<=NJ;++j)
    {
        H_Jun[j]=Y[j];

    }


    /*==========================================================================================*/

    tol_exceeded=FALSE;
    for(i=1;i<=NP;++i)
    {
    if(fabs(Q1[i]-Q_old[i])>fabs(Q_old[i]*tol))
    tol_exceeded=TRUE;
    }
    ++num_iter;
    }                             /*------------------while --------------*/
    return(tol_exceeded);



    }

    /*==========================================================================================*/
```

```c
/*==============================================================================*/
/*    function : Gauss_elimination                                              */
/*    this function solves the system of equations [A]{X}={B} using the         */
/*    Gauss elimination method with partial pivoting.                           */
/*                                                                              */
/*                                                                              */
/*                                                                              */
/*==============================================================================*/


int  Gauss_elimination (double a[][MAX_SIZE],double  b[],int n,double *ptr_det)

{

double temp,mult,trt,tol;
int npivot,i,j,k,l,error_flag;

*ptr_det=1.0;
tol=1e-30;
npivot=1;




for(k=1;k<=n-1;++k)
{


 for(i=k;i<=n;++i)
    {




     if(fabs(a[i][k])>fabs(a[k][k]))
     {

        ++npivot;
        for(l=1;l<=n;++l)
        {

         temp=a[i][l];
         a[i][l]=a[k][l];
         a[k][l]=temp;

        }

        temp=b[i];
        b[i]=b[k];
        b[k]=temp;
      }

        }


    trt=*ptr_det;

      trt =*ptr_det*a[k][k];
if (fabs(trt)<tol)

  {
  error_flag=1;
```

```
    return(error_flag);
    }




    for(i=2;i<=n;++i)
      {
       if(i!=k)
       {
      mult=a[i][k]/a[k][k];

      b[i]=b[i]-b[k]*mult;



      for(j=1;j<=n;++j)
       {
       a[i][j]=a[i][j]-a[k][j]*mult;

       }
      }
    }
  }


  if (npivot %2==1)
  *ptr_det=*ptr_det*(-1.0);

  /*=======================================================================================*/

  b[n]=b[n]/a[n][n];

  for(i=n-1;i>=1;i--)
  {

   for(j=i+1;j<=n;++j)

       b[i]=b[i]-a[i][j]*b[j];
    b[i]=b[i]/a[i][i];

   }

   error_flag=0;
   return(error_flag);



  }
  /*=======================================================================================*/

  /*===================================================================================*/
  /*  function : Newton-Raphson()                                                     */
  /*  this function computes the root of an equation ofthe form F(x)=0               */
  /*  using the Newton-Raphson mehtod, i.e. computes the value of the friction       */
  /*  factor f.                                                                       */
  /*                                                                                  */
  /*===================================================================================*/

  /*=======================================================================================*/

  int Newton_Raphson(double epsilon ,double e,double Re,
                    int max_er,double *ptr_f1,int *ptr_num1_iter)
{

 double x_prev;
 double x_curr;
 double derf;
 *ptr_num1_iter=0;
 x_prev=1.0/pow(1.14-2.0*log10(e),2);
 x_curr=1.0/pow(1.14-2.0*log10(e),2);

 while ( *ptr_num1_iter < max_er)
     {

 derf=df(x_prev,e,Re);
 if(fabs(df(x_prev,e,Re)) <= nearly_zero )
 return(1);
 x_curr=x_prev-f(x_prev,e,Re)/derf;
 ++ *ptr_num1_iter;
 *ptr_f1=x_curr;
 if (fabs(x_curr-x_prev)<=fabs(x_curr*epsilon))
 return(0);
 x_prev=x_curr;
     }


 return (2);
}
/*=====================================================================================*/


/*=====================================================================================*/

/*function : f()                                                                     */
/* purpose:compute the value of  f[x] at x                                           */
/*                                                                                   */
/*=====================================================================================*/


 double f(double x,double e,double Re)
 {

  double fx;

  fx=1./(sqrt(x))-1.14+2.*log10((e)+9.35/(Re*sqrt(x)));
 return(fx);
 }

/*=====================================================================================*/

/*   function : df()                                                                 */
/*  purpose:         compute the value of  df[x] at x                                */
/*                                                                                   */
/*=====================================================================================*/



 double df(double x,double e,double Re)
 {
 double dfx;

 double ARG;
 double MOR;
 double DAR;
```

```
  ARG=9.35*log10(2.71828);
  MOR=((e)+9.35/(Re*sqrt(x)));
  DAR=x*sqrt(x)*Re;
 dfx= -0.5/(x*sqrt(x))-(ARG/(DAR*MOR));
 return(dfx);
 }



/*=========================================================================
=*/
/*  function: Poly_leastsqr
 */
/*  purpose: if a pump exist in the network,and its characteristic is described by
 */
/*  performance operating data. This function invoked to fit the curve for the
 */
/*  data using least square method
 */
/*
 */
/*=========================================================================
=*/

int poly_leastsqr(double x[],double y[],int num_points,int num_poly,double ao[])

{
double c[ MAX_SIZE][MAX_SIZE];
double s[2*MAX_SIZE];
int i,j;
double det;
int result ;
s[0]=num_points;
for(i=1;i<=2*num_poly;++i)
{
s[i]=0.0;
for(j=1;j<=num_points;++j)
s[i]+=pow(x[j],i);
}


for(i=0;i<=num_poly;++i)
for(j=0;j<=num_poly;++j)
c[i+1][j+1]=s[i+j];
ao[1]=0.0;
for(j=1;j<=num_points;++j)
ao[1]+=y[j];
for(i=1;i<=num_poly;++i)
{
ao[i+1]=0.0;
for(j=1;j<=num_points;++j)
ao[i+1]+=y[j]*pow(x[j],i);
}

result=Gauss_elimination(c,ao,num_poly+1,&det);
return(result);

}

/*========================================================================*/

/*========================================================================*/
```

```
 int Newton_Raphson2(double f1,double epsilon ,double e,double Re,
          int max_er,double *ptr_f2,int *ptr_num1_iter)
{

 double x_prev;
 double x_curr;

 double derf;

  *ptr_num1_iter=0;
 x_prev= f1;
 x_curr= f1;

 while ( *ptr_num1_iter < max_er)
      {

 derf=df(x_prev,e,Re);
 if(fabs(df(x_prev,e,Re)) <= nearly_zero )
 return(1);
 x_curr=x_prev-f(x_prev,e,Re)/derf;
++ *ptr_num1_iter;
 *ptr_f2=x_curr;
 if (fabs(x_curr-x_prev)<=fabs(x_curr*epsilon))
 return(0);
 x_prev=x_curr;
      }


 return (2);
}
```