

## **INFORMATION TO USERS**

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

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

ProQuest Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600

**UMI<sup>®</sup>**



## **NOTE TO USERS**

**This reproduction is the best copy available.**

UMI



**The Development of a Simulation Model of  
Pipeline Network Systems with Check Valve**

**Tianhe Wen**

**A Thesis**

**in the Department of**

**Mechanical Engineering**

**Faculty of Engineering and Computer Science**

**Presented in Partial Fulfillment of the Requirements**

**for the Degree of Master of Applied Science at**

**Concordia University**

**Montreal, Quebec, Canada**

**August 2001**

**© Tianhe Wen, 2001**



**National Library  
of Canada**

**Acquisitions and  
Bibliographic Services**

**395 Wellington Street  
Ottawa ON K1A 0N4  
Canada**

**Bibliothèque nationale  
du Canada**

**Acquisitions et  
services bibliographiques**

**395, rue Wellington  
Ottawa ON K1A 0N4  
Canada**

*Your file Votre référence*

*Our file Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-64071-X

**Canada**

## ABSTRACT

### **The Development of a Simulation Model of Pipeline Network Systems with Check Valve**

Tianhe Wen

The objective of this thesis is to design and implement a software package to model and simulate pipeline network systems with check valves. The application package is written under the window's environment to provide the hydraulic engineer a user friendly interface for ease of simulation and analysis without requiring to write code.

Discussion of the phenomenon of hydraulic transients, derivation of the differential equations, comparison of different kinds of analysis methods, and investigation of the method of characteristics solution and basic boundary conditions are all established. The check valve dynamic equation is investigated for the analysis of pipeline transients, and formulae for flow torque acting on the valve discs are derived by introducing the orifice sequence model. Dynamic behavior of pumps is considered during the pipeline transients, and the analysis of this behavior combines the method of characteristics with check valve dynamics, pump characteristic and pump boundary conditions both for pump failure and pump start up. Numerical solutions to pipeline transients, check valve dynamics, and pump characteristic are established. Finally, the implementation into a working simulation program of a dynamic model for pipeline network systems with check valves is described, and a user guide for the software *HydroAnalysis* and *HydroGraphic* is presented.

## ACKNOLEDGMENTS

The author wishes to express his gratitude to his supervisors Dr. J. V. Svoboda and Dr. H. Hong, for their guidance and inspiration throughout the author's graduate studies.

The author appreciates the cooperation and support given by Mr. D. Barclay, former President of Ritepro Inc. Also, the author would like to express his gratitude to Mr. P. Major, the Chief Engineer of Ritepro Inc., for his technical assistance throughout the research.

# Table of Contents

Chapter 1	Introduction .....	1
1.1	Summary .....	1
1.2	Classification of Hydraulic Transients .....	2
1.3	Classification of Check valves .....	3
1.4	Review of Previous Work .....	5
1.4.1	Hydraulic Transients .....	5
1.4.2	Check Valves Dynamics .....	6
1.4.3	Computer Program Simulation of Transients in Pipelines and Check Valves.....	7
1.5	Thesis Outline .....	8
Chapter 2	Transients in Pipelines.....	11
2.1	Introduction .....	11
2.2	Basic Equations of Transients in Pipelines .....	12
2.2.1	Continuity Equation .....	12
2.2.2	Equation of motion.....	15
2.3	Comparison of Method of Transient Analysis in Pipelines .....	16
2.4	The Method of Characteristic .....	17
2.4.1	Introduction .....	17
2.4.2	Characteristic Equations.....	18
2.4.3	Finite Difference Formulations .....	20
2.4.4	Basic Boundary Conditions.....	23
2.4.4.1	<i>Reservoir</i> .....	24
2.4.4.2	<i>Series Junction</i> .....	25
2.4.4.3	<i>Branch Junction</i> .....	26
2.4.4.4	<i>Valve or Orifice</i> .....	29
2.4.5	Boundary Conditions for Pump .....	31
2.4.5.1	<i>Introduction</i> .....	31

2.4.5.2	<i>Boundary Conditions for Pump Failure</i> .....	32
2.4.5.3	<i>Boundary Condition for Pump Start Up</i> .....	37
Chapter 3	Check Valve Dynamics .....	39
3.1	Introduction .....	39
3.2	Check Valve Differential Equation .....	40
3.3	Component Torque.....	41
3.3.1	Counterweight Torque.....	42
3.3.2	Spring Torque.....	43
3.4	Check Valve Flow Torque .....	45
3.4.1	Introduction .....	45
3.4.2	Check Valve Pressure Drop .....	46
3.4.2.1	<i>Coefficient of Resistance of a Wafer Swing-Disk Check Valve</i> .....	47
3.4.2.2	<i>Coefficient of Resistance of Forward Flow</i> .....	50
3.4.2.3	<i>Coefficient of Resistance of Reverse Flow</i> .....	53
3.4.3	Swing-Disk Check Valve Flow Torque .....	54
3.5	Boundary Condition for Check Valve.....	57
Chapter 4	Simulation and Model of Pipeline Network	
	Transients with Check Valve System .....	59
4.1	Introduction .....	59
4.2	Algebraic Solution to a Piping System .....	60
4.2.1	Algebraic Representation and Continuous Storage Data .....	60
4.2.2	Indexing .....	63
4.2.3	Simulation of Piping System.....	66
4.3	Numerical Model and Simulation for Check Valve.....	68
4.3.1	Check Valve Dynamic System Model .....	68
4.3.2	The Runge-Kutta Numerical Solution to the Check Valve Dynamic Equation .....	70
4.4	Newton-Raphson Solution to Pump Boundary Condition .....	73

Chapter 5	Software Implementation of Transients in Pipeline Network and Check Valve System .....	77
5.1	Introduction .....	77
5.2	Analysis for Hydro-Analysis Software Package .....	78
5.2.1	Problem Description.....	78
5.2.2	Input Parameters.....	79
5.2.3	Output Requirement .....	81
5.3	Software Design .....	82
5.3.1	Algorithm for the Hydro-Analysis .....	82
5.3.2	Hierarchical Diagram .....	84
5.3.3	Object Oriented Programming Design for Hydro-Analysis.....	87
5.3.3.1	<i>Major Class Description for Graphic User Interface</i> .....	87
5.3.3.2	<i>Class Description of Transients Simulation</i> .....	92
5.3.3.3	<i>Class Description of Graphic Output</i> .....	97
5.4	Introduction to HydroAnalysis Software .....	98
5.4.1	Overview .....	98
5.4.2	User Guide for Software Program Package <i>HydroAnalysis</i> .....	100
5.4.2.1	<i>Getting started</i> .....	100
5.4.2.2	<i>Setting System Parameters</i> .....	100
5.4.2.3	<i>Getting Simulation Output</i> .....	112
Chapter 6	Conclusion.....	116
6.1	Summary .....	116
6.2	Suggestions for Future Work .....	120
REFERENCES	.....	122
APPENDIX A:	Reviews of Method of Transient Analysis in Pipelines .....	127
APPENDIX B:	Pump Characteristics .....	138
APPENDIX C:	Simulation Results.....	143
APPENDIX D:	Head Files for Software Hydro-Analysis .....	155

## List of Figures

Fig. 2.2.1 Notation for Continuity and Motion Equation.....	13
Fig. 2.4.1 Characteristic Lines in x-t Plane.....	20
Fig. 2.4.2 The x-t Finite Difference Grid.....	21
Fig. 2.4.3 Reservoir Boundary: (a) Upstream end; (b) Downstream end.....	24
Fig. 2.4.4 Series Junction.....	26
Fig. 2.4.5 Branching Junction.....	27
Fig. 2.4.6 Valve in line.....	30
Fig. 2.4.7 Pump and Valve in Pipeline Systems.....	32
Fig. 2.4.8 Approximation of Pump Curve by a Straight Line.....	34
Fig. 2.4.9 Pump Start Up.....	38
Fig. 3.2.1 Schematic of Check Valve.....	40
Fig. 3.3.1 Counterweight.....	42
Fig. 3.3.2 Cylindrical Helical Bending spring.....	43
Fig. 3.3.3 Cylindrical Helical Torsion Spring.....	44
Fig. 3.4.1 Orifice Model for Wafer-Type Swing-Disc Check Valve.....	48
Fig. 3.4.2 Orifice Sequence Model.....	51
Fig. 3.4.3 Unsteady Flow Through an Check Valve.....	54
Fig. 4.1.1 Block Diagram of Pump, Check Valve and Pipelines.....	59
Fig. 4.2.1 x-t Diagram for Algebraic Representations in Series System.....	61
Fig. 4.2.2 Branching System with Pump and Check Valve.....	64
Fig. 4.2.3 Indexing of Branching System.....	65
Fig. 4.2.4 Flowchart of Piping System Simulation.....	67
Fig. 4.3.1 Flowchart of Check Valve Numerical Simulation.....	69
Fig. 4.4.1 Newton-Raphson Solution for Pump.....	75
Fig. 5.3.1.1 Flow Chart of Hydro-Analysis.....	84
Fig. 5.3.2.1 (a) Hierarchical Diagram for Hydro-Analysis.....	85
Fig. 5.3.2.1 (b) Hierarchical Diagram for Graphic User Interface.....	85
Fig. 5.3.2.1 (c) Hierarchical Diagram for Transients Simulation.....	86
Fig. 5.3.2.1 (d) Hierarchical Diagram for Output Graphic.....	86

Fig. 5.3.3.1 The operation and attributes of CMainFrame .....	87
Fig. 5.3.3.2 Operation of CChildFrame .....	88
Fig. 5.3.3.3 Operation of ChydroAnalysisView .....	88
Fig. 5.3.3.4 The operation and attributes of CFrontPageFormView .....	89
Fig. 5.3.3.5 Operation of CHydroAnalysisApp .....	90
Fig. 5.3.3.6 Operation of ChydroAnalysisDoc .....	90
Fig. 5.3.3.7 The operation and attributes of PlotPrintoutDlg .....	91
Fig. 5.3.3.8 The operation and attributes of Pipe .....	92
Fig. 5.3.3.9 Operation of PipeNet2Cv & PipeNet2NoCv .....	93
Fig. 5.3.3.10 Operation & Attributes of PumpFail .....	94
Fig. 5.3.3.11 Operation & Attributes of PumpStart .....	95
Fig. 5.3.3.12 Operation & Attributes of CheckValve .....	96
Fig. 5.3.3.13 Description of HydroGraphic Classes .....	99
Fig. 5.4.2.1 Main Window of HydroAnalysis .....	101
Fig. 5.4.2.2 Dialog Box for Connection Parameters .....	102
Fig. 5.4.2.3 Dialog Box of Indexing .....	103
Fig. 5.4.2.4 Dialog Box of Simulation .....	104
Fig. 5.4.2.5 Dialog Box of Plot & Print .....	105
Fig. 5.4.2.6 Dialog Box for Pump Check Valve Parameters .....	106
Fig. 5.4.2.7 Dialog Box of Pipeline Parameters .....	107
Fig. 5.4.2.8 Dialog Box of Reservoir Parameters .....	108
Fig. 5.4.2.9 Dialog Box of Control Valve Parameters .....	108
Fig. 5.4.2.10 Dialog Box of Junction Parameters .....	109
Fig. 5.4.2.11 Menu File Window .....	110
Fig. 5.4.2.12 Save Model As Window .....	111
Fig. 5.4.2.13 Open Model Window .....	112
Fig. 5.4.2.14 HydroGraphic Main Page (With Check Valve) .....	113
Fig. 5.4.2.15 Display of the Plot Data .....	114
 Fig. A 1 Circuit Represent of Lumped Parameter Method .....	 130
Fig. A 2 Block Diagram for Single Pipeline .....	134

Fig. B 1 Polar Diagram of $\theta$ .....	140
Fig. B 2 Complete Pump Characteristics .....	141
Fig. C 1 Simple Pump, Check Valve , and Pipe System.....	144
Fig. C 2 Indexing of Branching Systems .....	151
Fig. C 3 Graphic Display of Branching System Without Check Valve .....	152
Fig. C 4 Graphic Display of Branching System With Check Valve .....	152

## Nomenclature

$A$	Area of pipe, Project area of the disc and disc arm.
$A_0'$	Pump characteristics parameters, Project area of the disc and disc arm normal to the flow stream
$A_I$	Pump characteristics parameters
$A_G$	Area of valve opening
$a$	Wave speed
$a_1, a_2$	Constants to describe pump head-discharge curve
$B$	Pipeline characteristics impedance
$B_M, B_P$	Known constants in compatibility equations
$B_n$	Calculate parameter in compatibility equations
$C$	Pipeline capacity
$C^*, C$	Name of characteristic equations
$C_D, C_d$	Valve or orifice discharge coefficient
$C_M, C_P$	Known constants in compatibility equations
$C_{ND}$	Normal drag coefficient
$C_n$	Calculate parameter in compatibility equations
$C_v, C_v^*$	Valve flow coefficients
$D, d$	Pipe diameter
$F_D, F_R$	Disc forward and reverse flow forces
$f$	Friction factor
$g$	Gravitational acceleration
$H$	Pressure head
$H_R$	Rated pressure head of pump
$H_S$	Pump start up shutoff head
$H_0$	Stead state or mean pressure head
$h$	Dimensionless pressure head for pump failure
$I$	Moment of inertia of rotating parts for pump
$J$	Total moment of inertia for check valve

$J_{CW}$	Moment of inertia of counter weight for check valve
$J_V$	Moment of inertia of the disc and arm assemble for check valve
$K$	Bulk modulus of elasticity
$K, K_0$	Valve coefficient of resistance
$K_1, K_2$	Valve coefficient of resistance; Disc friction constant
$K_3, K_4$	Valve coefficient of resistance
$k_1, k_2, k_3, k_4$	Fourth order Runge-Kutta parameters
$K_S$	Spring constant
$L$	Pipe length
$m$	Mass
$N$	Rotational speed of pump
$N_R$	Rated speed of pump
$n$	Number of reach in pipeline
$Q$	Discharge
$Q_{in}$	Inflow
$Q_n$	Junction or nodal flow
$Q_p$	Pipeline flow; Pump flow
$Q_v$	Check valve or valve flow
$R$	Pipeline resistance coefficient
$T$	Torque on pump or check valve; Time
$T_0$	Stead state torque of pump
$T_b$	Bearing friction torque of check valve
$T_{CW}$	Counterweight torque
$T_F$	Flow torque of check valve
$T_R$	Rated torque of pump
$T_S$	Spring torque of check valve; Pump start up shutoff time
$T_{VW}$	Weight torque due to rotating disc
$t$	Time
$V$	Velocity
$v$	Dimensionless velocity
$W$	Velocity ; Weight

$W_B, W_H$	Dimensionless pump characteristics
$x$	Distance along the pipeline; Angular position in pump characteristic curve
$Z_C$	Characteristic impedance
$z$	Elevation of pipe above datum
$\alpha$	Dimensionless speed ratio for pump failure; Torque ratio for pump start up
$\beta$	Dimensionless torque ratio for pump failure
$\gamma$	Specific weight of fluid
$\delta$	Angle of torsional spring
$\eta$	Pump efficiency
$\theta$	Opening of Check valve
$\lambda$	Multiplier in characteristics method
$\nu$	Dimensionless flow ratio for pump failure
$\rho$	Mass density
$\tau$	Dimensionless valve opening
$\omega$	Angular velocity

# Chapter 1

## Introduction

### *1.1 Summary*

A pipeline system is a combination of piping components with the objective of carrying incompressible or compressible fluid to one or several users. An industrial pipeline system may be considered, for most of its operating time, to be in a steady state of flow. But a rapid velocity (or pressure) change, an accident or a non-normal operation in a pipeline system will result in a hydraulic transient. This can shorten the life span of certain pipeline components and even damage them.

Common examples of the causes of transients in engineering systems include:

1. Opening, closing of valves in a pipeline
2. Starting or stopping the pumps in a pumping system
3. Starting-up a hydraulic turbine, accepting or rejecting load
4. Vibrations of the vanes of a runner or an impeller, or of the blades of a fan.

Check valves are pieces of equipment which were originally developed and fitted to pipelines in order to prevent the lines draining backwards when pumps stopped – They were sometimes also used to prevent downstream reservoirs from emptying. In addition, they prevent reverse rotation of pumps, thereby avoiding damage to seals and to the gear of the driving motor.

The undesired effects of flow reversal vary depending on the application. Some examples where check valves are used include:

- In tank fill-up applications, a check valve is installed on a supply pipe which is below the water line to prevent draining the tank in the case of flow supply pressure drop.
- A check valve installed on the output of a pump would prevent high downstream pressure from driving the pump in reverse in the event of a power failure.
- In an intermittent pumping application where the fluid supply level is below the pump, a check valve is installed at the foot of the inlet suction line to prevent draining of the pipe and hence unnecessary priming of the system.

## *1.2 Classification of Hydraulic Transients*

Hydraulic transients may be classified into the following three categories:

1. Transients in closed conduits
2. Transients in open channels
3. Combined free-surface-pressurized transient flows.

The analysis of transients in closed conduits may be further subdivided into two types:

1. Distributed systems
2. Lumped systems

In distributed systems, the transient phenomenon occurs in the form of traveling waves. Places in which such transients occur include water supply pipes, power plant conduits, and gas-transmission lines.

In the analysis of lumped systems, any change in the flow conditions is assumed to take place instantaneously throughout the system, i.e., the fluid is considered as a solid body. Mathematically, transients in the distributed systems are represented by partial differential equations, whereas the transients in the lumped systems are described by ordinary differential equations. If  $\omega L/a$  is much less than 1, then the system may be analyzed as a lumped system; otherwise, the system must be analyzed as a distributed system. In the previous expression,  $\omega$  = frequency of oscillation,  $L$  = length of the pipeline, and  $a$  = wave velocity.

In this thesis, discussion will focus mainly on transients in closed conduits.

### 1.3 Classification of Check valves

Some of the most commonly used check valves are classified here according to operating principle.

A ball check valve consists of a ball inserted into the flow to permit essentially unrestricted flow in the preferred direction. Flow in the reverse direction carries the ball to the seat, immediately blocking the line. The valve may be spring or gravity loaded so that the minimal flow per operation is controllable. The ball check valve is used for high viscosity liquids, in small size (max. 2") transport lines, and in hydraulic and pneumatic power systems.

A piston check valve consists of a piston similar to a globe valve disc, which, in normal operation, is kept suspended by the flow forces. The piston falls by gravity when the flow forces become insufficient or negative. This type may also be spring or gravity

loaded. The piston check valve is widely used for liquids and gases and is particularly effective for high-pressure steam services.

The swing-disc check valve consists of a disc inserted into the line and rotating about a pivot shaft. In normal operation, the disc is kept open by the flow forces. The disc closes due to gravity when the fluid forces become insufficient. Counterweights, dashpots, or springs may be added to control the disc closing characteristics.

The swing-disc check valve is the most commonly used type in industrial pipelines, in a wide range of pipe dimensions (from 2" to over 48" diameter) and operating pressures. In the following, the investigation is focused mainly on the swing-disc check valve.

The category of Swing-Disc Check Valves can be further subdivided into the following types:

1. Conventional bore or reduced orifice swing check valves
2. Full bore swing check valves
3. Wafer swing check valves, which are essentially conventional bore types, with reduced length and weight and favorable economic and installation characteristics.

The disc of the swing check valve can reach dimensions of over 100 cm diameter and a weight of over 2 tons, for such large disk, counterweights become necessary to open the valve in normal operating conditions and dampers are required to reduce the dynamic forces.

## 1.4 Review of Previous Work

### 1.4.1 Hydraulic Transients

The study of hydraulic transients began with the investigation of the propagation of sound waves in air, the propagation of waves in shallow water, and the flow of water in pipes.

In 1897, Joukowski conducted extensive experiments in Moscow on pipes that were, respectively, 7620 m long and 50 mm in diameter, 305 m long and 101.5 mm in diameter, and 305 m long and 152.5 mm in diameter. Based on his experimental and theoretical studies, he published his classic report [1] on the basic theory of the water hammer. He developed a formula for wave velocity, taking into consideration the elasticity of both the water and the pipe walls. He also investigated the relationship between the reduction of flow velocity and the resulting pressure rise by using two methods: the conservation of energy and the continuity condition. He discussed the propagation of a pressure wave along the pipe and the reflection of the pressure waves from the open end of a branch, and studied the effects of air chambers, surge tanks, and spring safety valves on water hammer pressures. Finally, he found that the pressure rise was a maximum for closing times,  $T_c \leq 2L/a$ , in which  $L$  = *length of the pipeline* and  $a$  = *wave speed*.

Allievi [1] published the general theory of water hammer in 1902. He obtained an expression of the pressure rise at the valve and presented charts for the pressure rise and drop caused by a uniformly closing or opening valve. He also studied the rhythmic movement of a valve. He proved that the pressure cannot exceed twice the static head.

Based on Joukowski's theory, in 1920, Gibson [1] presented a paper that included, for the first time, nonlinear friction losses in the analysis. He also invented an apparatus to measure the turbine discharge using the pressure-time history following a load rejection.

Schnyderc [1] included complete pump characteristics in his analysis of water hammer in pipelines connected to centrifugal pumps. He was the first to include the friction losses in the graphical analysis .

Since 1951, several books [2-10] have been published on the subject. Several methods of analysis have been developed, such as Method of Characteristics [4,7,33]. Lumped Parameter Method [14,15] in time domain, Four-Pole Equations Modeling Method [6] and Transmission Line Modeling Method [37,38,39] in frequency domain.

#### **1.4.2 Check Valves Dynamics**

The investigation of check valve dynamics and check valve related pressure surges have been conducted by Pool, Porwit and Carlton [11]. Their work focused on modeling the check valve dynamics more accurately in order to obtain better surge pressure predictions. Swing-disc and tilting-disc check valves were considered.

Csemniczky [12] studied the hydraulic performance characteristics of pressure drop and momentum for tilting-disc check valves. He derived a semi-empirical formula describing the momentum acting on the disc shaft.

Hong and Svoboda [13, 14], in 1981, presented the computer model of a swing check valve suitable for use in lumped parameter pipe network simulation. The valve is

viewed as a variable resistor with a resistance being the function of the disc angle. The fluid was treated as incompressible.

Cavazzoni [15], in 1983, used an *Orifice Sequence Model* to determine the coefficients of resistance for check valves. The valve dynamic system was represented by a second order, non-linear differential equation. Additional elements such as counterweights and springs were examined and introduced, as additional terms, into the general differential equation.

In 1983, Provoost [16] presented results of the study of the dynamic characteristics of several types of check valves. he noted the importance of the local reverse velocity and the deceleration of the liquid during closure.

Significant contributions were made by Thorley [17] in 1987, where valve characteristics have been studied with a view to defining a boundary condition for computer coding and also to establish the 'right' type of check valve to use in a particular installation.

### **1.4.3 Computer Program Simulation of Transients in Pipelines and Check Valves**

The arrival of the 1960's and the advent of the high-speed digital computer began a new era in transient analysis of pipelines and check valves. In 1967, Streeter and Wylie [4] showed the application of the computer to complete a comprehensive hydraulic transients analysis. Watters [6] gave a FORTRAN program to solve simple pipe and pipe networks in 1979. Cavazzoni [15] simulated pipes and check valves separately by a FORTRAN program in 1984. Hong [14] combined pipes simulation and check valves simulation in

1983. In 1994, Cavazzoni developed a C++ program [18] to solve transient flow in pipeline networks systems using the finite element method. Sadfa designed the FORTRAN program [18] to solve for the transient piping network flow in 1998.

Today the emphasis on pipeline transient analysis is almost entirely concentrated on computer applications and window program. Jenkins [19] developed a window program HAMMER which simulates the effects of the water hammer in a reservoir and pipeline system in which a valve at the downstream end is opened or closed by the user. Faast [20] designed software Faast-3 which is an interactive, graphical finite-element program for analyzing fluid flow in piping systems.

## *1.5 Thesis Outline*

The objective of this thesis is to model and simulate pipeline network systems with check valves. The study focuses on three major areas:

1. Analysis of hydraulic transients for pipeline network system
2. Analysis of check valve dynamics, especially for swing-disc check valve dynamics
3. Computer program implementation of combined pipelines and check valves

This study will begin by developing the governing equations for the transient flow in closed conduits. In Chapter 2, continuity and momentum equations will be introduced. Several assumptions are made and then condensed into a simplified version. The

characteristics method will also be discussed in Chapter 2. Finally, basic boundary conditions including valve and pump will be investigated in this chapter.

Chapter 3 will present the study of pressure drops across the check valve, and flow forces acting on the valve disc. An orifice sequence model will be developed, in order to predict the pressure drop through a valve as a function of a disc angle, for direct and reverse flows. Coefficients of resistance for the valve and for the disc alone will be developed, and the disc flow torque will be investigated to determine a relationship between drag forces and total torque. The check valve dynamic system will be represented by a second order, non-linear differential equation.

Chapter 4 will give the procedure for implementing computer simulation of pipelines with check valves. This chapter will give consideration to the integration of the pipelines simulation, check valves dynamics, and pump characteristic analysis. The solution to pipeline transient by the characteristic method, to check valve dynamics by the Runge-Kutta method, and to pump characteristic equations by the Newton-Raphson method will also be discussed in this chapter. Some algorithms, flow chart and code will also be presented.

Chapter 5 will present analysis of the pipeline network systems from the view point of software engineering; design and implementation will be done using Object-Oriented Programming techniques. The analysis will include some high-level algorithms, hierarchical diagram and detail descriptions of the class and some objects. The user guide for the software *HydroAnalysis* and *HydroGraphic* will also be given in this chapter.

Finally, Chapter 6 will include results and the conclusion to this study in addition to suggestions for further work.

Appendix A gives the review of several solution methods used in hydraulic transient analysis. Most of the major methods used in the analysis are introduced. The lumped parameter, Four-Pole Equations Modeling, and transmission line methods are then given in detail. Appendix B is the pump characteristics which will be used in this study. Appendix C gives the extra simulation results of pump failure and start up. Appendix D lists the head file for the program package HydrauAnalysis and HydrauGraphic.

## Chapter 2

### Transients in Pipelines

#### 2.1 Introduction

Flows in pipes are usually in an unsteady state. Altering the rate of flow can cause large pressure fluctuations which can endanger the integrity of the pipe. Such pressure fluctuations are called *pressure transients* and the state of flow in which they occur is called *transient flow*.

In solid body mechanics, one may analyze the situation of a mass when being acted on by a force. The mass is considered to be concentrated at its center of gravity and Newton's second law is applied, taking into account any frictional forces present. If the force is not co-linear through the center of gravity, angular momentum and angular accelerations must be considered. This technique can be applied to a mass of water with equal validity but, if the mass of water is very long and thin, as occurs in pipelines, such an approach is unrealistic. This is because of the fact that when a force is applied to a long mass of water, that force is propagated through the water by wave action. Consequently this process occurs over a finite period of time and so the entire mass involved does not experience the force at the same time. An analysis that takes account of this effect is called *an elastic analysis*, whereas the usual method of analyzing solid bodies is said to be a *rigid body analysis*.

From the above, there are two basic methods for the analysis of unsteady flows: rigid column theory and elastic theory. Obviously the elastic theory is more accurate than

the rigid column theory. Therefore discussion will focus on the elastic theory in this chapter.

## 2.2 Basic Equations of Transients in Pipelines

Two equations are used to solve problems of unsteady flow in pipes, one being a form of the continuity equation and the other being a form of the momentum equation (or motion equation). Since the flow velocity and pressure in transient flows may be functions of time as well as distance, these equations are a set of partial differential equations.

### 2.2.1 Continuity Equation

To derive the continuity equation, we apply the Reynolds transport theorem [8] for the conservation of mass. First, we chose the control surface and control volume as show in Fig. 2.2.1. where  $x$  is the distance,  $V$  is the velocity, and  $W$  is the velocity of section moving.

Applying Reynolds transport theorem to the above control volume gives

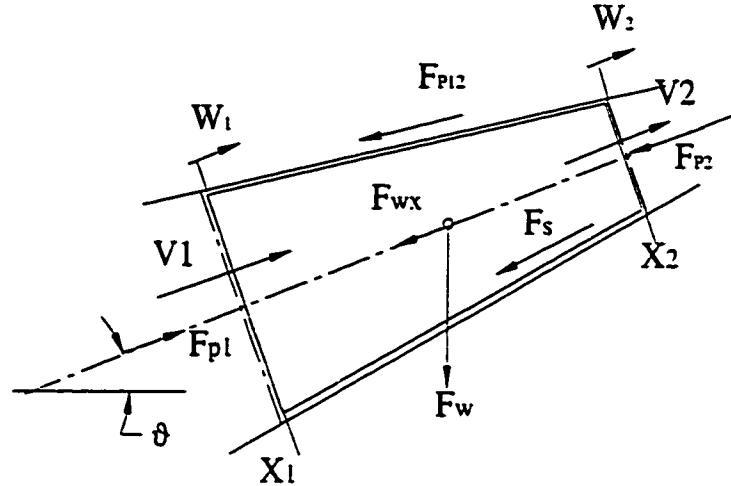
$$\frac{d}{dt} \int_{x_1}^{x_2} \rho A dx + \rho_2 A_2 (V_2 - W_2) - \rho_1 A_1 (V_1 - W_1) = 0 \quad (2.2.1)$$

The above equation can be simplified:

$$\frac{1}{\rho} \frac{d\rho}{dt} + \frac{1}{A} \frac{dA}{dt} + \frac{\partial V}{\partial x} = 0 \quad (2.2.2)$$

This equation holds for converging or diverging tubes as well as for cylindrical

pipes on a slope or horizontal. It is also valid for any fluid, and for rigid or highly deformable tubes, as no simplifying assumptions have been required.



**Fig. 2.2.1 Notation for Continuity and Motion Equation**

To write this equation in terms of the variables of interest, as pressure  $p$  and velocity  $V$ , we express the derivatives of  $\rho$  and  $A$  in terms of  $p$  and  $V$  as following. The first term in (2.2.2) takes into account the compressibility of the fluid. Pressure is introduced with the definition of bulk modulus of elasticity of a fluid.

$$K = \frac{dp}{d\rho / \rho} \quad (2.2.3)$$

This equation can be written as

$$\frac{d\rho}{dt} = \frac{\rho}{K} \frac{dp}{dt} \quad (2.2.4)$$

The second term in (2.2.2) deals with the elasticity of the pipe wall and its rate of deformation with pressure. For prismatic tubes, area is a function of pressure only

$$\frac{dA}{dt} = \frac{dA}{dp} \frac{dp}{dt} \quad (2.2.5)$$

Substituting (2.2.4) and (2.2.5) into (2.2.2) yields

$$\frac{1}{K} \frac{dp}{dt} \left( 1 + \frac{K}{A} \frac{dA}{dp} \right) + \frac{\partial V}{\partial x} = 0 \quad (2.2.6)$$

Now we define [7]: 
$$a^2 = \frac{\frac{K}{\rho}}{1 + \frac{K}{A} \frac{dA}{dp}} \quad (2.2.7)$$

where  $a$  is the speed with which a pressure wave travels back and forth in an elastic conduit filled with slightly compressible fluid.

From (2.2.7), (2.2.6) becomes

$$\frac{dp}{dt} + \rho a^2 \frac{\partial V}{\partial x} = 0 \quad (2.2.8)$$

In the above equation,

$$\frac{dp}{dt} = \frac{\partial p}{\partial x} \frac{dx}{dt} + \frac{\partial p}{\partial t} = V \frac{\partial p}{\partial x} + \frac{\partial p}{\partial t} \quad (2.2.9)$$

The above equation can be approximated by neglecting the spatial variation of  $V$  whenever both spatial and time-varying terms appear in the same equation, because in general, spatial variation is much less than time-varying variation. So that (2.2.9)

becomes: 
$$\frac{dp}{dt} \approx \frac{\partial p}{\partial t} \quad (2.2.10)$$

So (2.2.8) is simplified as:

$$\frac{\partial p}{\partial t} + \rho a^2 \frac{\partial V}{\partial x} = 0 \quad (2.2.11)$$

It is a common practice in hydraulic engineering to compute pressures in a pipeline in terms of the piezometric head,  $H$ , above a specified datum and use the

discharge,  $Q$ , as the second variable instead of the flow velocity  $V$

$$\left. \begin{aligned} p &= \rho g(H - z) \\ Q &= VA \end{aligned} \right\} \quad (2.2.12)$$

in which  $z$  = elevation of the pipe centerline.

From (2.2.12), the slope term is usually small and may be neglected,

$$\frac{\partial p}{\partial x} = \rho g \frac{\partial H}{\partial x} \quad \frac{\partial V}{\partial x} = \frac{1}{A} \frac{\partial Q}{\partial x} \quad (2.2.13)$$

With the above assumption, (2.2.11) becomes

$$\frac{\partial H}{\partial t} + \frac{a^2}{gA} \frac{\partial Q}{\partial x} = 0 \quad (2.2.14)$$

This is the simplified continuity equation.

## 2.2.2 Equation of Motion

For the equation of motion, the Reynolds Transport Theorem [8] for the conservation of momentum can be used. We can obtain

$$\frac{d}{dt} \int_{CV} V \rho dv + [\rho A(V - W)V]_2 - [\rho A(V - W)V]_1 = \sum F \quad (2.2.15)$$

The above equation can be simplified:

$$\frac{dV}{dt} + \frac{1}{\rho} \frac{\partial p}{\partial x} + g \sin \theta + \frac{fV|V|}{2D} = 0 \quad (2.2.16)$$

where  $f$  = Darcy-Weisbach friction factor. The above equation is valid for converging or diverging pipe flow.

Similar reason as (2.2.10), (i.e.,  $\frac{dV}{dt} = \frac{\partial V}{\partial t}$ ), and (2.2.16) reduces to:

$$\frac{\partial V}{\partial t} + \frac{1}{\rho} \frac{\partial p}{\partial x} + g \sin \theta + \frac{fV|V|}{2D} = 0 \quad (2.2.17)$$

From (2.2.12), we can get the following

$$\frac{\partial p}{\partial x} = \rho g \left( \frac{\partial H}{\partial x} - \frac{\partial z}{\partial x} \right) = \rho g \left( \frac{\partial H}{\partial x} - \sin \theta \right) \quad (2.2.18)$$

Substitution into (2.2.17)

$$\frac{\partial V}{\partial t} + g \frac{\partial H}{\partial x} + \frac{fV|V|}{2D} = 0 \quad (2.2.19)$$

Since  $Q=VA$ , we finally derive

$$\frac{\partial Q}{\partial t} + gA \frac{\partial H}{\partial x} + \frac{fQ|Q|}{2DA} = 0 \quad (2.2.20)$$

which is the simplified hydraulic-grade-line form of the equation of motion. It is restricted to less compressible fluid (such as liquids), flowing at a low velocity.

### 2.3 Comparison of Methods of Analysis

The continuity and motion equations are the basis of hydraulic transients analysis. They are quasi-linear, hyperbolic, partial differential equations. A closed-form solution of these equations is not available. However, by neglecting or linearizing the nonlinear terms, several methods have been used for numerically integrating nonlinear, hyperbolic partial differential equations, such as method of characteristics, finite-difference method, finite element method, and linear element method.

The method of characteristics converts the two partial differential equations of motion and continuity into four total differential equations. This method has become

quite popular and is extensively used. For the analysis of systems having complex boundary conditions, this method has proven to be superior to other methods in several aspects, such as its stability, accuracy, easy of programming, and efficiency of computations.

In the finite-difference method, the partial derivatives are replaced by finite-difference approximations such that the unknown conditions at a point at the end of a time step are expressed in terms of the known conditions at the beginning of the time step. Since the algebraic equations for the entire system are solved simultaneously, the analysis of complex boundary conditions by iterative procedures may require a large amount of computing time. The linear element method and finite element method even need 20 times more computation time to solve the same problem [15]. But for complex pipelines, the method of characteristics is superior to the finite difference method, mainly for its efficient handling of boundary conditions. Therefore, the method of characteristics is chosen after comparing with the above methods and with other methods such as the lump parameter method and the transmission line method, which reviewed in Appendix A.

## **2.4 *The Method of Characteristics* [7, 30, 31, 33]**

### **2.4.1 Introduction**

In section 2.3, it was demonstrated that the method of characteristics be superior to the other method. Several methods used for the solution of hydraulic transients will be discussed and compared in Appendix A. The details of the method of characteristics are

presented in this section. In this method, the partial differential equations will be transformed into particular total differential equations. These total differential equations will then be integrated to yield finite difference equations, and furthermore algebraic equations, which can be conveniently handled. Simple, basic boundary conditions are also discussed in this section.

## 2.4.2 Characteristic Equations

The motion and continuity equations (2.2.20) and (2.2.14) can be rewritten respectively:

$$L_1 = \frac{\partial Q}{\partial t} + gA \frac{\partial H}{\partial x} + \frac{fQ|Q|}{2DA} = 0 \quad (2.4.1)$$

$$L_2 = a^2 \frac{\partial Q}{\partial x} + gA \frac{\partial H}{\partial t} = 0 \quad (2.4.2)$$

Let us consider a linear combination of Eq.(2.4.1) and (2.4.2) using a unknown multiplier  $\lambda$

$$\begin{aligned} L &= L_1 + \lambda L_2 \\ &= \left( \frac{\partial Q}{\partial t} + \lambda a^2 \frac{\partial Q}{\partial x} \right) + \lambda gA \left( \frac{\partial H}{\partial t} + \frac{1}{\lambda} \frac{\partial H}{\partial x} \right) + \frac{f}{2DA} Q|Q| = 0 \end{aligned} \quad (2.4.3)$$

Any two real, distinct values of  $\lambda$  will yield two equations in terms of the two dependent variables H and Q that will be equivalent to Eq. (2.4.1) and (2.4.2). Appropriate selection of two particular values of  $\lambda$  leads to a simplification of Eq.(2.4.3).

If  $H = H(x, t)$  and  $Q = Q(x, t)$ , then the total derivatives are

$$\frac{dQ}{dt} = \frac{\partial Q}{\partial t} + \frac{\partial Q}{\partial x} \frac{dx}{dt} \quad \frac{dH}{dt} = \frac{\partial H}{\partial t} + \frac{\partial H}{\partial x} \frac{dx}{dt} \quad (2.4.4)$$

The unknown multiplier  $\lambda$  is defined as

$$\lambda a^2 = \frac{dx}{dt} = \frac{1}{\lambda} \quad (2.4.5)$$

The solution of Eq.(2.4.5) yields two particular values of  $\lambda$ ,

$$\lambda = \pm \frac{1}{a} \quad (2.4.6)$$

Substitution of these values of  $\lambda$  into Eq.(2.4.3) leads to two pairs of equations which are grouped and identified as  $C^+$  and  $C^-$  equations.

$$C^+ : \begin{cases} \frac{gA}{a} \frac{dH}{dt} + \frac{dQ}{dt} + \frac{f}{2DA} Q |Q| = 0 \dots\dots\dots (2.4.7) \\ \frac{dx}{dt} = a \dots\dots\dots (2.4.8) \end{cases}$$

and

$$C^- : \begin{cases} -\frac{gA}{a} \frac{dH}{dt} + \frac{dQ}{dt} + \frac{f}{2DA} Q |Q| = 0 \dots\dots\dots (2.4.9) \\ \frac{dx}{dt} = -a \dots\dots\dots (2.4.10) \end{cases}$$

Thus two real values of  $\lambda$  have been used to convert the original two partial differential equations to two total differential equations in the independent variable  $t$ . However, a price has been paid for this simplification: Eq.(2.4.1) and (2.4.2) were valid everywhere in the  $x$ - $t$  plane: in contrast Eq.(2.4.7) is valid only along the straight line given by Eq.(2.4.8), and (2.4.9) is valid only along the straight line given by Eq.(2.4.10).

In the  $x$ - $t$  plane, Eq.(2.4.8) and (2.4.10) represent two straight lines having slopes  $\pm \frac{1}{a}$ .

These lines are called the *characteristic lines* as shown in Fig. 2.4.1.

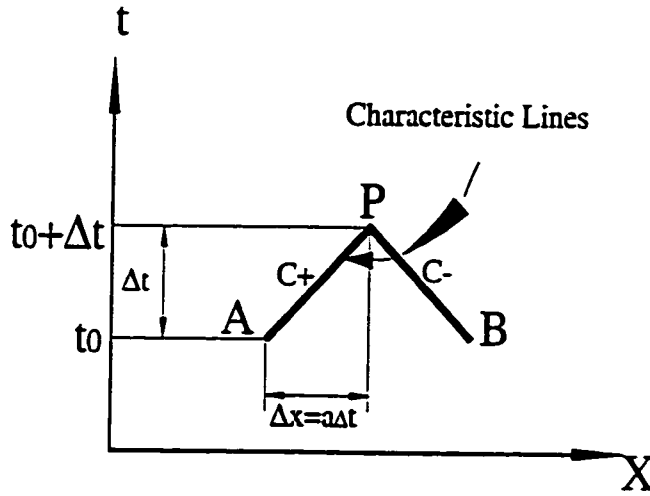


Fig 2.4.1 Characteristic Lines in  $x-t$  Plane

### 2.4.3 Finite Difference Formulations

In practice, the pipeline is divided into an even number of reaches,  $N$ , each  $\Delta x$  in length, as shown in Fig. 2.4.2. Assuming that the values of  $Q$  and  $H$  at point  $A$  and  $B$  are known, and we want to determine their values at point  $P$ , this can be done by solving Eq.(2.4.7) and (2.4.9) as follows.

By multiplying the left-hand side of Eq.(2.4.7) by  $dt$  and integrating, we obtain

$$\frac{gA}{a} \int_{H_A}^{H_P} dH + \int_{Q_A}^{Q_P} dQ + \frac{f}{2DA} \int_{t_A}^{t_P} Q |Q| dt = 0 \quad (2.4.11)$$

in which we use subscripts  $A$  and  $P$  to indicate locations in the  $x-t$  plane. For example,  $H_P$  and  $Q_P$  are the head and discharge at point  $P$ .

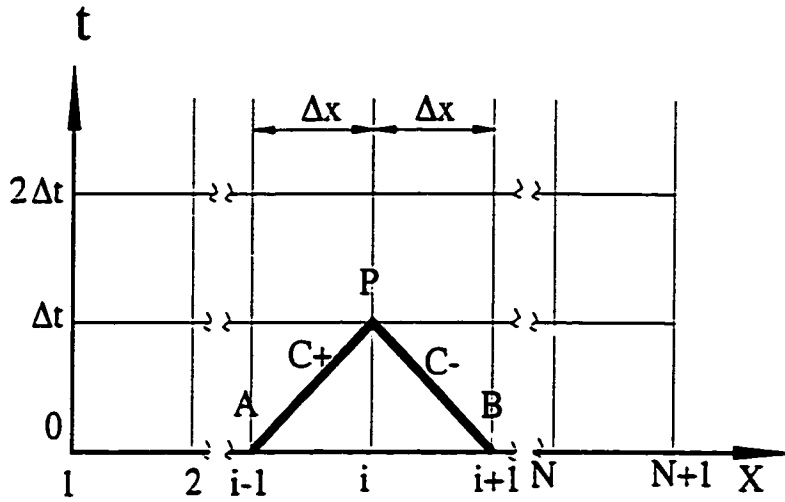


Fig. 2.4.2 The  $x$ — $t$  Finite Difference Grid

The first two integral terms of Eq.(2.4.11) may be evaluated easily. However, this cannot be done for the third term, because the variation of  $Q$  with respect to  $t$  is not explicitly known. Thus an approximation must be introduced in this evaluation. A number of options exist. The objective is to maintain a simple form that will provide reasonable accuracy in general unsteady flow and that will hold conditions steady during steady flow. It is therefore best to use the trapezoidal rule to maintain the linear form of the integrated equations. It is of second-order accuracy, and is a satisfactory approximation for most problems.

$$\int_{t_A}^{t_P} Q | Q | dt = \Delta t Q_P | Q_A | \quad (2.4.12)$$

By substituting Eq.(2.4.12) into (2.4.11), and noting that  $\Delta x = \pm a \Delta t$ , integration along the characteristic line AP, and similarly along the characteristic line BP results in

$$H_p - H_A + \frac{a}{gA}(Q_p - Q_A) + \frac{f\Delta x}{2gDA^2}Q_p |Q_A| = 0 \quad (2.4.13)$$

$$H_p - H_B - \frac{a}{gA}(Q_p - Q_B) - \frac{f\Delta x}{2gDA^2}Q_p |Q_B| = 0 \quad (2.4.14)$$

These two compatibility equations are algebraic relations that describe the transient propagation of the piezometric head and flow in a pipeline. By solving for  $H_p$ , these equations may be written

$$C^+ : \quad H_p = H_A - B(Q_p - Q_A) - RQ_p |Q_A| \quad (2.4.15)$$

$$C^- : \quad H_p = H_B + B(Q_p - Q_B) + RQ_p |Q_B| \quad (2.4.16)$$

In which  $B$  is a function of the physical properties of the fluid and the pipeline, often called the pipeline characteristic impedance:

$$B = \frac{a}{gA} \quad (2.4.17)$$

and  $R$  is the pipeline resistance coefficient:

$$R = \frac{f\Delta x}{2gDA^2} \quad (2.4.18)$$

The friction factor  $f$  may be a constant, or it may be adjusted with the local Reynolds number in accordance with the Moody diagram in each reach at each time step during calculations.

The solution to a problem in liquid transients usually begins with steady-state conditions at time zero, so that  $H$  and  $Q$  are known initial values at each computing section (Fig. 2.4.2), for  $t = 0$ . The solution consists of finding  $H$  and  $Q$  for alternate grid points along  $t = \Delta t$ , then proceeding to  $t = 2\Delta t$ , and so on, until the desired time duration has been covered. At any interior grid intersection point, point P at section  $i$ , the two

compatibility equations are solved simultaneously for the unknown  $Q_i$  and  $H_i$  .

Eq.(2.4.15) and (2.4.16) may be written in a simple form

$$C^+ : \quad H_i = C_p - B_p Q_i \quad (2.4.19)$$

$$C^- : \quad H_i = C_m + B_m Q_i \quad (2.4.20)$$

Where

$$C_p = H_{i-1} + BQ_{i-1} \quad B_p = B + R |Q_{i-1}| \quad (2.4.21)$$

$$C_m = H_{i+1} - BQ_{i+1} \quad B_m = B + R |Q_{i+1}| \quad (2.4.22)$$

solving for  $H_i$  and  $Q_i$  :

$$H_i = \frac{C_p B_m + C_m B_p}{B_p + B_m} \quad (2.4.23)$$

$$Q_i = \frac{C_p - C_m}{B_p + B_m} \quad (2.4.24)$$

Thus the computation procedure uses the current values of H and Q at point  $i - 1$  and  $i + 1$  to compute their values at point  $i$  at one time interval  $\Delta t$  later. Usually, the starting values are known for a steady flow condition, which is called the boundary condition. The discussion concerning the boundary conditions will be presented in the next section and in a later chapter.

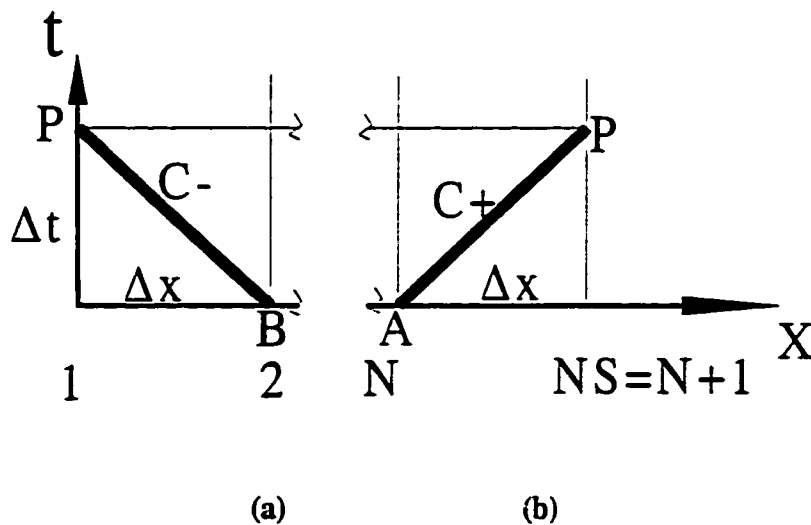
#### 2.4.4 Basic Boundary Conditions

In the last section, it was indicated that boundary conditions are required to determine the transient-state head and discharge at the boundaries. Devices such as pump, valve, reservoir and junctions form these boundaries. They may be located at either end of a

pipeline or at some intermediate point. If they are located at the downstream end ( $x = L$ ), Fig. 2.4.3, the  $C^+$  characteristic equation (2.4.19) can be used, while if it is at the upstream end, the  $C^-$  characteristic equation (2.4.20) can be applied. The following are typical boundary condition equations:

#### 2.4.4.1 Reservoir

1. At the upstream end of the line: Fig 2.4.3a



**Fig. 2.4.3 Reservoir Boundary: (a) Upstream end; (b) Downstream end**

At a large upstream reservoir the elevation of the hydraulic grade line normally can be assumed constant during a short-duration transient. This boundary condition is described as  $H_I = H_R$ , in which  $H_R$  is the elevation of the reservoir surface above the reference

datum. At each time step,  $H_I$  is known, and  $Q_I$  is determined by a direct solution of Eq.(2.4.20)

$$Q_I = \frac{H_I - C_M}{B_M} \quad (2.4.25)$$

The subscript I refers to the upstream section, at point P, Fig. 2.4.3a;  $C_M$  and  $B_M$  are variables in the computational procedure but are dependent only on known values from the previous time step, in this case point B, section 2.

*2. At the downstream of the line Fig. 2.4.3b:*

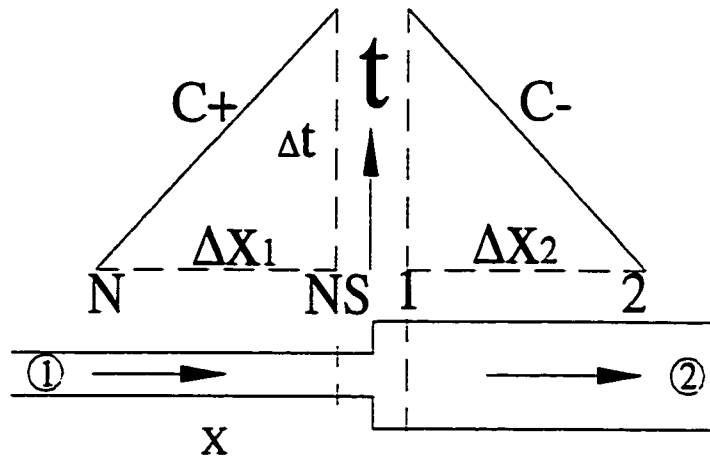
The boundary condition is described as:  $H_{NS} = H_R$ , Similarly, Eq.(2.4.19) is used

$$Q_{NS} = \frac{C_P - H_{NS}}{B_P} \quad (2.4.26)$$

where the subscript NS refers to point P,  $C_P$  and  $B_P$  can be calculated by the previous time step.

#### **2.4.4.2 Series Junction**

This type of junction, although shown in fig. 2.4.4 as a diameter change, applies equally well to a single-diameter pipe with a change in roughness, thickness, or constraint condition, or any combination of these possible variables. At the junction (Fig. 2.4.4), Eq.(2.4.19) is available for pipe 1, and Eq. (2.4.20) is available for pipe 2.



**Fig 2.4.4 Series Junction**

The continuity expression and the condition of a common hydraulic-grade-line elevation provide two equations, as follows:

$$Q_{1,NS} = Q_{2,I} \quad H_{1,NS} = H_{2,I} \quad (2.4.27)$$

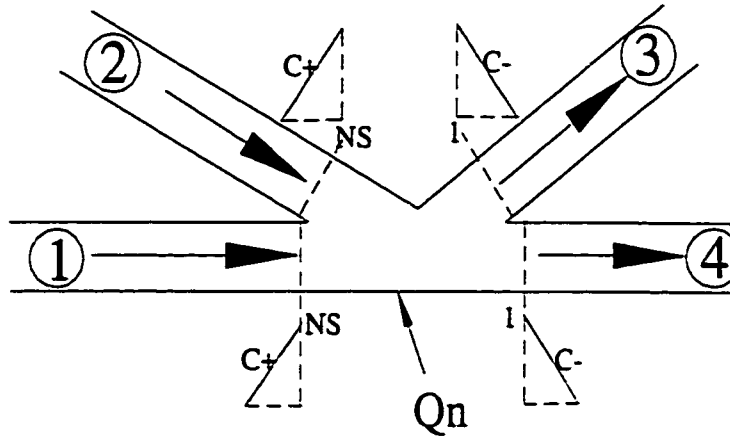
where the first subscript refers to the pipe number and the second one refers to the end condition. By solving these equations simultaneously with Eq.(2.4.19) and (2.4.20), the following is obtained:

$$Q_{2,I} = \frac{C_{P_1} - C_{M_2}}{B_{P_1} + B_{M_2}} \quad (2.4.28)$$

#### **2.4.4.3 Branch Junction**

A junction is visualized as a point connection of elements in a system, at which there are two variables: a nodal flow,  $Q_n$ , which may be zero, with inflow being positive, and a

nodal pressure or hydraulic-grade-line elevation. Fig. 2.4.5 shows a branching junction of pipeline with a potential nodal flow  $Q_n$ .



**Fig. 2.4.5 Branching Junction**

There are two types of junctions or nodes:

1. A constant pressure
2. A constant flow

Referring to Fig. 2.4.5, a continuity equation is written at the junction summing the inflow,

$$\sum Q_{in} = \sum Q_p + Q_n = 0 \quad (2.4.29)$$

in which the  $\sum Q_p$  is the summation of all instantaneous pipeline flows. When minor losses are neglected, we have

$$H = H_{1,NS} = H_{2,NS} = H_{3,I} = H_{4,I} \quad (2.4.30)$$

The compatibility equations are also available in each pipe: Eq. (2.4.19) for pipe 1 and 2, Eq. (2.4.20) for pipe 3 and 4.

For a junction with a known pressure head (unknown nodal flow), such as at a reservoir or pressure tank, the pipeline flows are determined directly from appropriate application of Eq.(2.4.19) and (2.4.20), and the nodal flow  $Q_n$  is then determined directly from Eq.(2.4.29).

For a junction with known  $Q_n$  (may be zero), the compatibility equations are written in the following form:

$$\begin{aligned} Q_{1,NS} &= \frac{C_{P_1}}{B_{P_1}} - \frac{H}{B_{P_1}} \\ Q_{2,NS} &= \frac{C_{P_2}}{B_{P_2}} - \frac{H}{B_{P_2}} \\ -Q_{3,1} &= \frac{C_{M_1}}{B_{M_1}} - \frac{H}{B_{M_1}} \\ -Q_{4,1} &= \frac{C_{M_2}}{B_{M_2}} - \frac{H}{B_{M_2}} \end{aligned}$$

The summation of these equations gives

$$\sum Q_P = S_C - S_B H \quad (2.4.31)$$

in which

$$S_C = \sum \frac{C_P}{B_P} + \sum \frac{C_M}{B_M} \quad S_B = \sum \frac{1}{B_P} + \sum \frac{1}{B_M}$$

and includes all the pipelines connected to the junction. Substitution into Eq.(2.4.29) yields an equation form similar to the compatibility equations to compute the pressure head:

$$H = C_n + B_n Q_n \quad (2.4.32)$$

in which  $C_n = \frac{S_c}{S_b}$  and  $B_n = \frac{1}{S_b}$ . The compatibility equations then yield the flow in each pipe. This method may be applied to any number of pipes and can be extended to non-pipe elements such as valves and pumps.

#### 2.4.4.4 Valve or Orifice [32, 36]

Valves or orifices may be located within a given pipeline, between two different lines, at the reservoir, at pipeline terminations, or in any number of other positions in systems. Fig.2.4.6a shows a valve between two pipelines, and Fig. 2.4.6b shows a schematic representation of the valve with nodes a and b as the interconnecting junctions on both sides of the valve.

Use of the steady state orifice equation neglects any inertia effects from accelerating or decelerating flow through the valve opening and also implies that there is no opportunity for a change in the volume of fluid stored in the valve body. For positive flow, Fig 2.4.6b, with  $H_{1,NS} = H_a$  and  $H_{2,I} = H_b$ , the orifice equation is

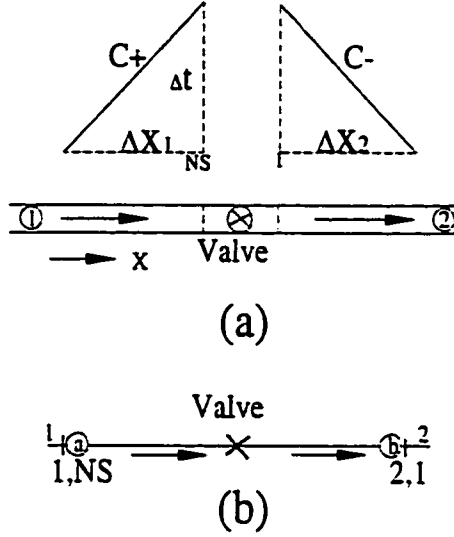
$$Q_{1,NS} = Q_{2,I} = Q_v = C_d A_G \sqrt{2g(H_a - H_b)} \quad (2.4.33)$$

in which  $A_G$  is the area of valve opening and  $C_d$  is the discharge coefficient.

For steady state flow, the similar equation is

$$Q_0 = (C_d A_G)_0 \sqrt{2gH_0} \quad (2.4.34)$$

in which the subscript zero refers to steady conditions for each variable. A dimensionless



**Fig. 2.4.6 Valve in line**

valve opening is useful in specifying valve motions as a function of time and is defined as

$$\tau = \frac{C_d A_G}{(C_d A_G)_0} \quad (2.4.35)$$

For the steady flow of  $Q_0$  and the corresponding energy loss of  $H_0$ ,  $\tau = 1$ , and for no flow with the valve closed,  $\tau = 0$ . By dividing the unsteady Eq. (2.4.33) by the steady state flow Eq.(2.4.34), we obtain

$$Q_{1,NS} = Q_{2,1} = Q_v = \frac{Q_0 \tau}{\sqrt{H_0}} \sqrt{H_a - H_b} \quad (2.4.36)$$

Applying Eq. (2.4.19) in pipe 1 and (2.4.20) in pipe 2, combined with Eq.(2.4.36), we get

$$Q_v = -C_v(B_{P_1} + B_{M_2}) + \sqrt{C_v^2(B_{P_1} + B_{M_2})^2 + 2C_v(C_{P_1} - C_{M_2})} \quad (2.4.37)$$

in which

$$C_v = \frac{Q_0^2 \tau^2}{2H_0} \quad (2.4.38)$$

For flow in the negative direction the orifice equation is

$$Q_{1,NS} = Q_{2,1} = Q_v = -\frac{Q_0 \tau}{\sqrt{H_0}} \sqrt{H_b - H_a} \quad (2.4.39)$$

and when combined with Eq.(2.4.19) and (2.4.20), we can get

$$Q_v = C_v(B_{P_1} + B_{M_2}) - \sqrt{C_v^2(B_{P_1} + B_{M_2})^2 - 2C_v(C_{P_1} - C_{M_2})} \quad (2.4.40)$$

Once the flow is known, Eq.(2.4.19) and (2.4.20) can be used to find the pressure head.

The above describes only the most basic boundary conditions used in analyzing pipeline-check valve transients. The check valve boundary condition will be discussed specifically in detail in Chapter 3.

## 2.4.5 Boundary Conditions for Pump

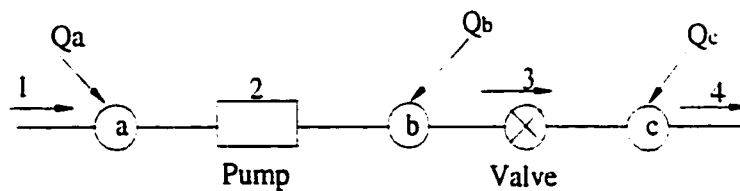
### 2.4.5.1 Introduction

The method of characteristic may be used to analyze the hydraulic transients during pump failure or start. Since the pumping head and discharge depend upon the pump speed, transient-state speed changes must be taken into consideration in the analysis. For this purpose, special boundary conditions for the pump end of a pipeline must be developed. In this section, the dynamic behavior of pumps is considered. A method for

the determination of the boundary conditions for the pump end is developed. This is followed by discussions of pump failure and pump startup for single, series and parallel pump connections.

#### 2.4.5.2 *Boundary Conditions for Pump Failure* [7, 8, 25]

The dynamic behavior of a pump failure depends on the instantaneous pressure-flow response in the piping system, the momentum equation for the rotating masses, and the pump characteristic curves as described in Appendix B. The primary element equations are the torque-angular deceleration equation describing the speed change in response to the resisting torque, and an equation that describes head balance across the pump.



**Fig.2.4.7 Pump and Valve in Pipeline Systems**

Referring to Fig 2.4.7, a pump with discharge valve in a pipeline system will be used to illustrate the handling of a pump failure. If the nodal flows are zero and there is no change in liquid storage between nodes *a* and *c*, the instantaneous flow, *Q*, is the same at the pipe ends and through the pump and valve.

With one pipe only at node *a* the nodal equation is the  $C^+$  compatibility equation for pipe 1:

$$F_a = H_a - C_p + B_p Q = 0 \quad (2.4.41)$$

The comparable equation at node *c* is :

$$F_c = H_c - C_M - B_M Q = 0 \quad (2.4.42)$$

At node *b* the equation is:  $Q = Q_2 = Q_3$ .

The element equation for the valve is:

$$F_3 = H_b - H_c - \Delta H_{pv} = 0 \quad (2.4.43)$$

where

$$\Delta H_{pv} = \frac{Q|Q|}{2C_{vp}} \quad (2.4.44)$$

For a control discharge valve,  $C_{vp} = \frac{Q_R^2 \tau^2}{2\Delta H_0}$ , as discussed in Sec. 2.4.4.4 (Eq. 2.4.38),

with the head loss across the valve equal to  $\Delta H_0$  when the flow is  $Q_R$  and  $\tau = 1$ . The valve position,  $\tau$ , is given as a function of time during the transient calculation.

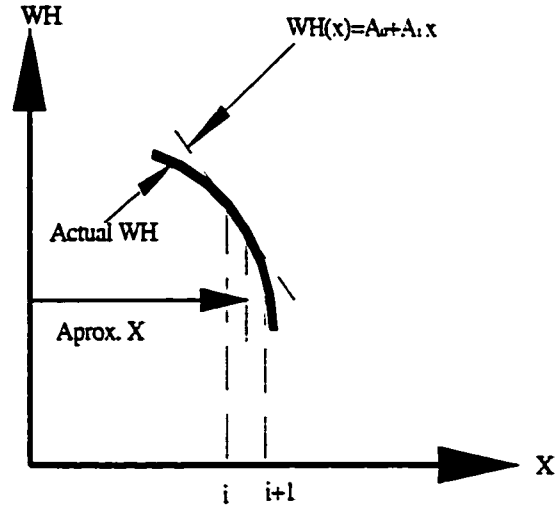
### 1. Head Balance Equation

The head balance relationship across the pump at any instant is

$$H_a + H = H_b \quad (2.4.45)$$

in which  $H$  is the total dynamic head across the pump. This total dynamic head is given by Eq(B.8) and (B.5)

$$H = H_R h = H_R (\alpha^2 + v^2) W_H \quad (2.4.46)$$



**Fig 2.4.8 Approximation of Pump Curve by a Straight Line**

To use the curve of  $W_H$  in Eq.(2.4.46), a straight line is used to replace the curve of  $W_H$  for the proper vicinity of  $x$ . In Fig. 2.4.8 where  $x$  is the approximate location of  $\pi + \tan^{-1} \frac{v}{\alpha}$ , obtained by extrapolating  $v$  and  $\alpha$  from previous calculations, a straight line is defined through the two adjoining data points. as:  $i = \frac{x}{\Delta x} + 1$ .

This is an integer arithmetic expression that locates the subscript for the data point to the left. Then  $(i-1)\Delta x, W_H(i)$  and  $i\Delta x, W_H(i+1)$  are the Cartesian coordinates of the two data points. By substitution into  $W_H = A_0 + A_1 x$ ,

$$\left. \begin{aligned} A_1 &= \frac{W_H(i+1) - W_H(i)}{\Delta x} \\ A_0 &= W_H(i+1) - iA_1\Delta x \end{aligned} \right\} \quad (2.4.47)$$

Now, we know the values of  $A_1$  and  $A_0$  by the pump characteristics curve (or tabular data). The Eq.(2.4.46) becomes

$$H = H_R(\alpha^2 + \nu^2)[A_0 + A_1(\pi + \tan^{-1} \frac{\nu}{\alpha})] \quad (2.4.48)$$

and the Eq.(2.4.45) becomes

$$H_a - H_b + H_R(\alpha^2 + \nu^2)[A_0 + A_1(\pi + \tan^{-1} \frac{\nu}{\alpha})] = 0 \quad (2.4.49)$$

The above equation relates the head on each side of the pump to the variables  $\alpha$  and  $\nu$ .

When combined with Eq. (2.4.41—2.4.43), noticing that  $Q = \nu Q_R$ , the following is obtained:

$$F_H = C_P - C_M - \nu Q_R(B_P + B_M) + H_R(\alpha^2 + \nu^2)[A_0 + A_1(\pi + \tan^{-1} \frac{\nu}{\alpha})] - \frac{\nu |\nu| \Delta H_0}{\tau^2} = 0 \quad (2.4.50)$$

This equation contains unknown  $\alpha$  and  $\nu$  only, and it is valid for control valves.

## 2. Speed Change Equation

The change in rotational speed of the pump depends on the unbalanced torque applied:

$$T = -I \frac{d\omega}{dt}$$

or

$$T = -I \frac{2\pi}{60} \frac{dN}{dt} \quad (2.4.51)$$

in which  $I = \frac{WR_g^2}{g}$ :  $W$  is the weight of rotating parts plus entrained liquid,  $R_g$  is the

radius of gyration of the rotating mass,  $\omega$  is the angular velocity in radians, and  $\frac{d\omega}{dt}$  is

the angular acceleration. On the basis of Eq.(B.5), Eq.(2.4.51) may be written as

$$\beta = -I \frac{2\pi N_R}{60 T_R} \frac{d\alpha}{dt} \quad (2.4.52)$$

In this equation,

$$T_R = \frac{60\gamma H_R Q_R}{2\pi N_R \eta_R}$$

$\gamma$  = specific weight of liquid

$\eta_R$  = pump efficiency at rated conditions.

Since

$$\alpha_0 = \frac{N_0}{N_R} \quad \alpha = \frac{N}{N_R} \quad \beta_0 = \frac{T_0}{T_R} \quad \beta = \frac{T}{T_R} \quad (2.4.53)$$

Eq.(2.4.53) is solved over two time step, by using an average value of  $\beta$  during the time step, and so it become:

$$\frac{\beta + \beta_0}{1} = -I \frac{2\pi N_R}{60 T_R} \frac{\alpha - \alpha_0}{\Delta t}$$

This can be simplified to:

$$\beta = I \frac{\pi N_R}{30 T_R} \frac{\alpha_0 - \alpha}{\Delta t} - \beta_0 \quad (2.4.54)$$

By defining

$$C_T = I \frac{\pi N_R}{30 \Delta t T_R} \quad (2.4.55)$$

Eq.(2.4.54) can be written

$$F_T = \beta + \beta_0 - C_T(\alpha_0 - \alpha) = 0 \quad (2.4.56)$$

Referring to Fig. B.2, the characteristics torque equation is

$$W_B(x) = \frac{\beta}{\alpha^2 + \nu^2} = B_0 + B_1(\pi + \tan^{-1} \frac{\nu}{\alpha}) \quad (2.4.57)$$

in which  $B_0$  and  $B_1$  are found in the same manner as  $A_0$  and  $A_1$ , that is

$$\left. \begin{aligned} B_1 &= \frac{W_B(i+1) - W_B(i)}{\Delta x} \\ B_0 &= W_B(i+1) - iB_1\Delta x \end{aligned} \right\} \quad (2.4.58)$$

So Eq.(2.4.56) become:

$$F_T = (\alpha^2 + \nu^2)[B_0 + B_1(\pi + \tan^{-1} \frac{\nu}{\alpha})] + \beta_0 - C_T(\alpha_0 - \alpha) = 0 \quad (2.4.59)$$

This is the speed change equation in  $\nu$  and  $\alpha$ .

To develop the boundary conditions for a single pump, Eq.(2.4.41) to (2.4.44) and (2.4.49) must be solved simultaneously, or Eq.(2.4.50) and (2.4.59) must be solved directly and simultaneously for  $\nu$  and  $\alpha$  using the Newton-Raphson method [7].

### 2.4.5.3 Boundary Condition for Pump Start Up

Centrifugal pump startup is usually accomplished with the discharge valve closed; as the pump approaches operating speed, the valve is opened slowly.

Analysis for pump startup is generally based on the assumption that the speed rises from zero to  $N_R$  linearly. After a reasonable  $T_S$  is selected,  $\alpha$  is known:

$$\alpha = \frac{T}{T_S} \quad T \leq T_S$$

$$\alpha = 1 \quad T > T_S$$

Thus the pump speed is known during the transient conditions, and we do not have to use the torque characteristics and the polar moment of inertia to determine  $\alpha$

at different time. This simplifies the necessary computations considerably.

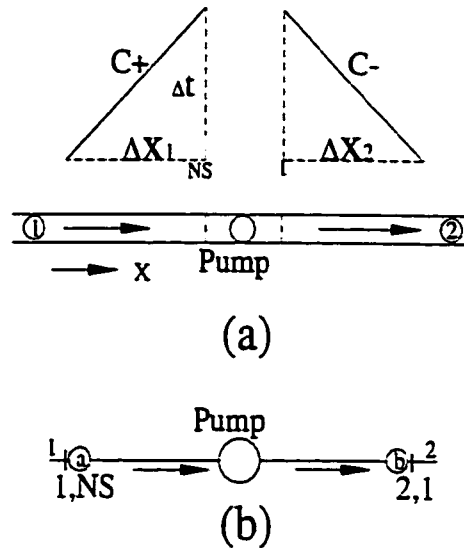


Fig. 2.4.9 Pump Start Up

In homologous form, the curve for the pump in Fig. 2.4.8 takes the form

$$H_{2,1} - H_{1,NS} = \alpha^2 H_S + a_1 \alpha Q_{1,NS} + a_2 Q_{1,NS}^2 \quad (2.4.60)$$

in which  $H_S$  is the shutoff head, and  $a_1$  and  $a_2$  are constant, normally negative, to describe the pump characteristic curve.

When Eq.(2.4.60) is combined with Eq.(2.4.19) and (2.4.20), the discharge may be determined as

$$Q_{1,NS} = \frac{B_R + B_{M_2} - a_1 \alpha}{2a_2} \left( 1 - \sqrt{1 - \frac{4a_2(\alpha^2 H_S + C_R - C_{M_2})}{(B_R + B_{M_2} - a_1 \alpha)^2}} \right) \quad (2.4.61)$$

If a check valve is put near a pump, one must consider the valve head loss and simultaneously solve Eq.(2.4.60) and (3.4.3).

## Chapter 3

### Check Valve Dynamics

#### *3.1 Introduction*

Check valves are placed in pipelines to prevent back flow. They are usually located at pumping stations, but may also be used at other locations in systems. Ideally, when the flow reverses at the location of the check valve, it closes, thus preventing back flow. More realistically, since the valve position is controlled by the flow and valve dynamics, closure occurs after some degree of back flow is established. This causes an instantaneous stoppage of the reverse flow with the corresponding pressure rise. The large pressure rises associated with the reduction of this reverse velocity accelerate the closure, resulting in higher pressure which ultimately slams the moving element onto its seat.

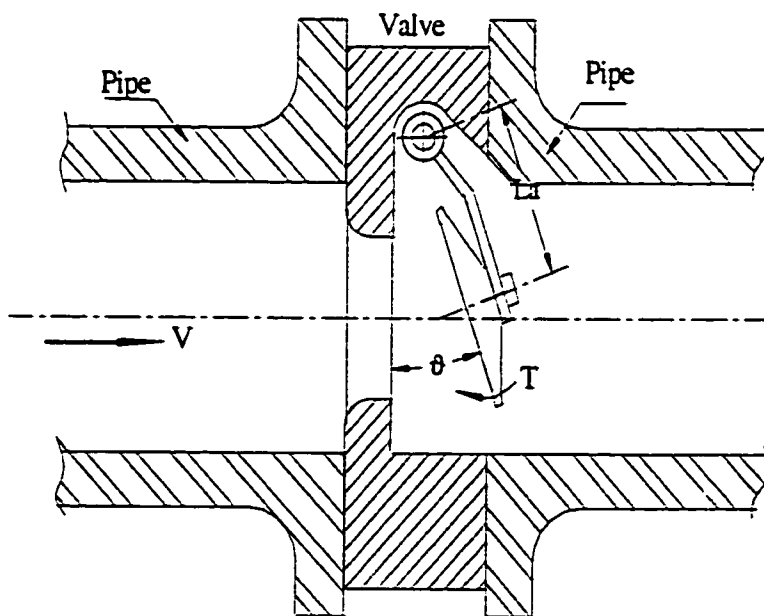
It is important to know the dynamic behavior of the check valve in the analysis of pipeline transients. Dynamic torque or force equation is therefore required in the analysis together with an energy equation to describe the losses across the valve which are accrued by back flow prior to its sealing.

A swing-disk check valve typically contains a relatively freely moving part such as a disk, whose motion is dictated by several elements including the movement of the fluid, its weight, external forces such as a spring or counter weight, friction, and possibly a damping device.

In this chapter, the check valve differential equation of motion will be introduced, followed by a detailed discussion of various torque and forces such as flow force and pressure drop. An orifice sequence model will be introduced for predicting the pressure drop through a valve as a function of disk angle. The disk flow torque will be investigated and an equivalent torque arm and a torque arm coefficient will be introduced to determine the disk torque that depends on coefficient of resistance and flow conditions.

### 3.2 Check Valve Differential Equation [12, 16, 17]

For a swing check valve, Fig. 3.2.1 provides a schematic in which  $\theta$  is the open angle of the valve disk.



**Fig. 3.2.1 Schematic of Check Valve**

The clockwise moments about the pivot point are considered positive. The torque equation becomes:

$$T_{vw} + T_{cw} + T_b + T_f - T_s = (J_v + J_{cw}) \frac{d^2\theta}{dt^2} \quad (3.2.1)$$

where  $J_v$  is the moment of inertia of the disk and arm assembly and  $J_{cw}$  is the moment of inertia about the hinge for an attached counter weight.

The torque due to the weight  $W_r$  of the rotating disk is given by  $T_{vw}$ ,

$$T_{vw} = W_r L_u \cos \theta \quad (3.2.2)$$

$T_b$  is the bearing friction torque. It may be a constant related to a frictional coefficient, the pivot pin radius, and the submerged weight of the gate, but is also likely to depend on at least the angular velocity [21]. The friction torque would be of the form

$$T_b = K_1 + K_2 \left( \frac{d\theta}{dt} \right)^n \quad (3.2.3)$$

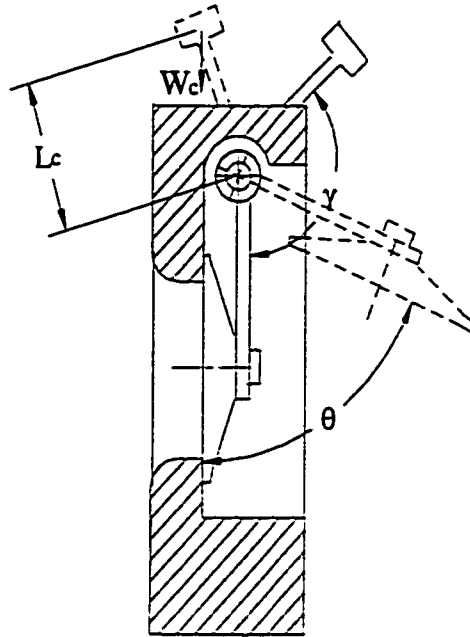
in which  $K_1$  and  $K_2$  are constants and  $n$  an arbitrary but constant power.

$T_{cw}$  is the counterweight torque,  $T_f$  is the flow torque, and  $T_s$  is the spring torque. Some of the above torque can be obtained by theoretical derivation, some of it can only be obtained by experiment. In the next section, the torque components are separately discussed.

### 3.3 Component Torque

The torque component [15] includes the counterweight, external spring and others such as dash-pot. They cause the torques which can be easily determined from valve structure,

dimension and material properties. In this section, only the normal components such as counterweight and external spring are discussed.



**Fig 3.3.1 Counterweight**

### **3.3.1 Counterweight Torque**

A counterweight is an auxiliary, external device, used to modify the static and dynamic characteristics of a clapper-hinge system. It may be used:

1. to reduce the weight torque in the open position
2. to modify the weight torque in the closed position
3. to increase system inertia

The counterweight dynamic equation (referring Fig 3.3.1) is:

$$m_c L_c^2 * \left( \frac{d^2 \theta}{dt^2} \right) = W_c L_c * \sin(\gamma + \theta) \quad (3.3.1)$$

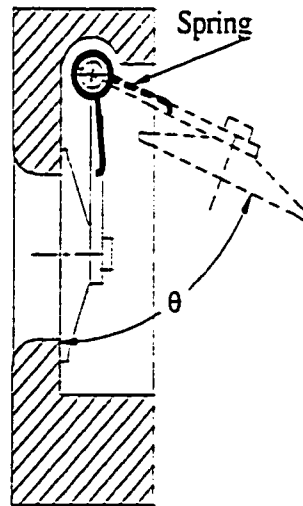
Referring to Eq.(3.2.1), the counterweight inertial and counterweight torque are:

$$J_{cw} = m_c L_c^2 \quad (3.3.2)$$

$$T_{cw} = W_c L_c * \sin(\gamma + \theta) \quad (3.3.3)$$

### 3.3.2 Spring Torque

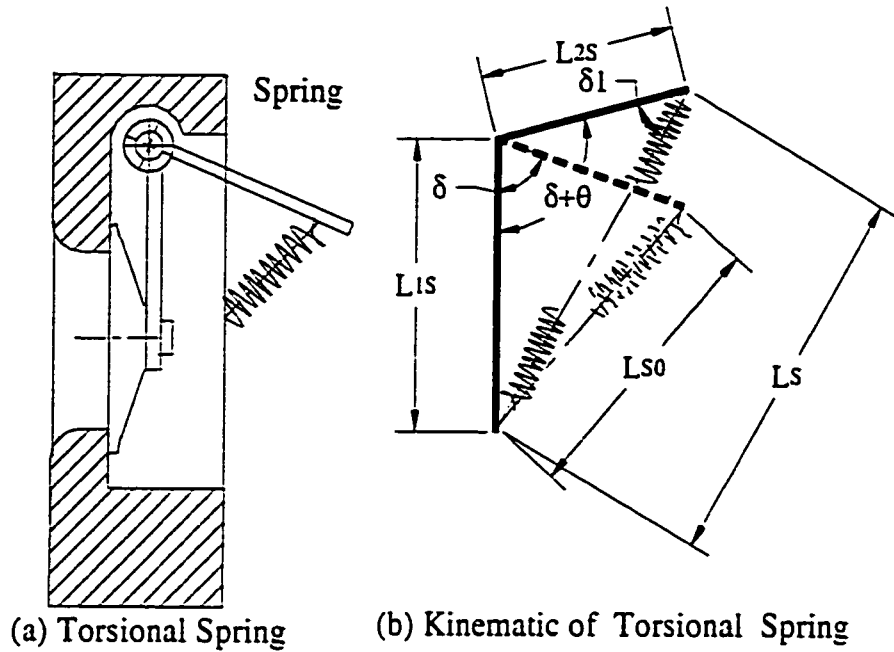
Adding springs is the most common way of reducing the check valve closing time. Springs are also used to increase, or to generate, a closing torque, when the disk is in a near-closed position.



**Fig 3.3.2 Cylindrical Helical Bending spring**

There are two types of springs (Fig. 3.3.2 and Fig. 3.3.3):

1. Cylindrical helical springs subject to bending
2. Cylindrical helical springs subject to torsion



**Fig. 3.3.3 Cylindrical Helical Torsion Spring**

Helical springs subject to bending (Fig. 3.3.2) do not require rigid pivot connections, or external devices, yet they cannot be changed without removing the disk. The bending spring contribution to the valve differential equation is

$$T_s = K_s * (\theta - \theta_o) \quad (3.3.4)$$

in which

$K_s$  is the spring constant

$\theta_o$  is the spring angular pre-load

$T_s$  is the spring torque

Helical springs subject to torsion (Fig. 3.3.3) require a rigid connection, but they can be easily inspected and changed.

The torsion spring torque is

$$T_s = K_s * (L_s - L_{s0}) * L_{2s} * \sin \delta_1 \quad (3.3.5)$$

in which,  $K_s$  is the spring constant and  $L_{s0}$  is the spring free length.

$$L_s = \sqrt{L_{1s}^2 + L_{2s}^2 - 2L_{1s}L_{2s} \cos(\theta + \delta)}$$

$$\delta = \cos^{-1} \frac{L_{2s}^2 + L_{1s}^2 - L_{s0}^2}{2L_{1s}L_{2s}}$$

$$\delta_1 = \cos^{-1} \frac{L_{2s}^2 + L_s^2 - L_{1s}^2}{2L_sL_{2s}}$$

### 3.4 Check Valve Flow Torque

#### 3.4.1 Introduction

With a pipeline in normal operating conditions, flow forces act on the disk surface, keeping it in the open position. These flow forces are a combination of lift and drag effects. The magnitude and point of application of lift forces depend on the fluid-dynamic characteristics of the disk and hinge and on the angle of attack  $\alpha$ . The lift forces increase with the opening angle and reach a maximum for opening angles to  $70^\circ$ . Magnitude, direction and point of application of lift forces depend on the pressure distribution on the upper and lower surfaces of the disk-hinge systems.

The magnitude of the drag forces may be determined when the valve's total pressure losses are known. There are two ways by which to determine drag forces: one is the experimental coefficient representation; the other is by which the orifice sequence model, in which the drag forces can be seen as the sum of five elementary components corresponding to the five contractions and expansions of pipe.

In this section, the orifice sequence model is introduced and used to determine the pressure drop for forward and reverse flow. Afterward, the flow torque for forward and reverse flow can be determined.

### 3.4.2 Check Valve Pressure Drop

A wafer check valve (Fig.3.4.1) may be modeled as a sequence of orifices in which the valve's coefficient of resistance can be assumed as a combination of the coefficient of resistance of the elementary contractions and expansions.

Pressure drop and flow through an elementary contraction or expansion are related by the following Darcy Formula:

$$\Delta P = \frac{fL}{D} \frac{\rho V^2}{2} \quad (3.4.1)$$

We can substitute

$$K = \frac{fL}{D} \quad \text{and} \quad V = \frac{Q_v}{A} \quad (3.4.2)$$

into Eq.(3.4.1) to obtain:

$$\Delta P = K \frac{\rho Q_v^2}{2A^2} \quad (3.4.3)$$

The flow rate can be rewritten as:

$$Q_v = \frac{A \sqrt{\frac{2}{\rho}}}{\sqrt{K}} \sqrt{\Delta P} \quad (3.4.4)$$

Letting:

$$K = \frac{1}{C_v^2} \quad (3.4.5)$$

Eq.(3.4.4) becomes:

$$Q_v = C_v A \sqrt{\frac{2}{\rho}} \sqrt{\Delta P} \quad (3.4.6)$$

Referring to (Eq.2.3.38):  $C_v = \frac{Q_v}{2\Delta H}$

$$Q_v = \sqrt{\frac{2C_v}{\rho g}} \sqrt{\Delta P} \quad (3.4.7)$$

Compare Eq. 3.4.6 and Eq. 3.4.7:

$$C_v = \frac{A^2 g}{K} = A^2 g C_v^2 \quad (3.4.8)$$

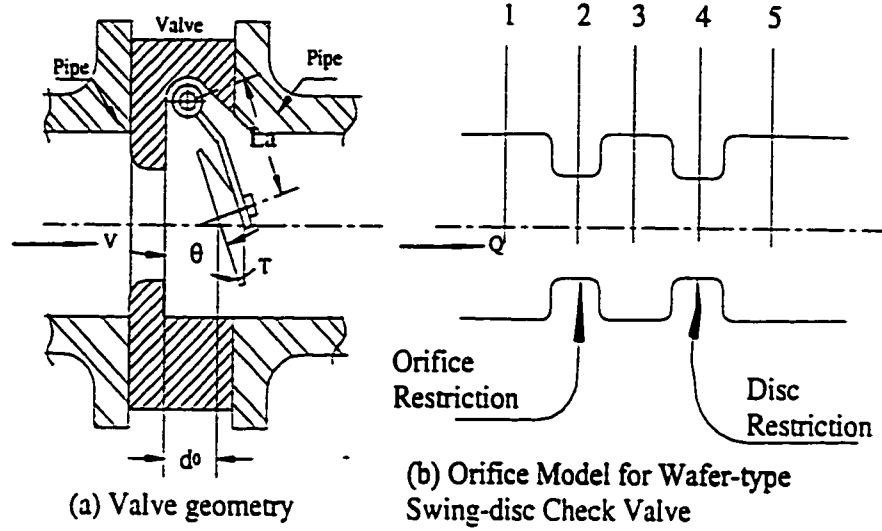
In Eq. 3.4.5, the K is the coefficient of resistance, and represents an equivalent length, in pipe diameter of straight pipe:

$$K = \frac{1}{C_v^2} = f \frac{L}{D} = fK_0 \quad (3.4.9)$$

#### 3.4.2.1 Coefficient of Resistance of a Wafer Swing-Disk Check Valve

Hong [14] studied the pressure losses through a wafer valve, using water as the fluid. The check valve was modeled as two restrictions in series. The upstream restriction was due

to the orifice and the downstream restriction was due to the wide-open disc. Each restriction was composed of an entrance contraction followed by an exit expansion (Fig. 3.4.1).



**Fig. 3.4.1 Orifice Model for Wafer-Type Swing-Disc Check Valve**

The coefficients of resistance for the orifice and disc can be assumed as independent. The overall normalized resistance coefficient is given by:

$$C_v^* = \frac{C_{v1}^* C_{v2}^*}{\sqrt{C_{v1}^{*2} + C_{v2}^{*2}}} \quad (3.4.10)$$

in which

$$C_{v1}^* = \frac{C_{v1}}{\sqrt{\frac{2}{\rho}}} = \frac{\left(\frac{d_2}{d_1}\right)^2}{\sqrt{K_1 + \frac{d_2}{d_1} - 1}} \quad (3.4.11)$$

$$C_{v2}^* = \frac{C_{v2}}{\sqrt{\frac{2}{\rho}}} = \frac{F_0}{\sqrt{K_3 + (F_0 - 1)^2}} \quad (3.4.12)$$

$$F_0 = 1 - \frac{d_3}{d_1} \cos \theta - \frac{4A_0}{\pi d_1^2} \quad (3.4.13)$$

where  $A_0$  is the projected area of the disc and disc arm normal to the flow stream. Its value can be calculated as [14]:

$$\begin{aligned} A_0 = & X_0 \left( \left( \frac{d_1}{2} \right)^2 - X_0^2 \right) + \left( \frac{d_1}{2} \right)^2 \arcsin \left( \frac{2X_0}{d_1} \right) \\ & - \left( X_0 \sqrt{\left( \frac{d_3}{2} \right)^2 - X_0^2} + \left( \frac{d_3}{2} \right)^2 \arcsin \left( \frac{2X_0}{d_3} \right) \right) \cos \theta \\ & - 2L_a X_0 (1 - \cos \theta) + 2d_0 X_0 \sin \theta \end{aligned} \quad (3.4.14)$$

where  $X_0$  and  $L_a$  are the half width and total length of the disc arm and  $d_0$  is the disc shaft displacement with respect to the seat plane, and  $K_1$  and  $K_3$  are given as:

$$K_1 = \frac{0.04}{\left( \frac{d_2}{d_1} \right)^4} \quad (3.4.15)$$

$$K_3 = \frac{\left( 1 - \frac{d_3}{d_1} \right)^2 \sqrt{\sin(90 - \theta)}}{\left( \frac{d_3}{d_1} \right)^4} \quad (3.4.16)$$

Also, we can write  $K_2$  and  $K_4$  as:

$$K_2 = \frac{\left( 1 - \frac{d_2}{d_3} \right)^2}{\left( \frac{d_2}{d_1} \right)^4} \quad (3.4.17)$$

$$K_4 = \frac{(1 - F_0)^2}{F_0^2} \quad (3.4.18)$$

Referring to Eq. (3.4.9), another expression may be used to represent the resistance coefficient when disc angle greater than 15°.

$$C_v^* = \frac{1}{\sqrt{K}} = \frac{1}{\sqrt{fK_0}} = \frac{1}{\sqrt{K_1 + K_2 + K_3 + K_4}} = \frac{1}{\sqrt{f(K_{01} + K_{02} + K_{03} + K_{04})}} \quad (3.4.19)$$

where  $K_0$  is the equivalent length of the valve.

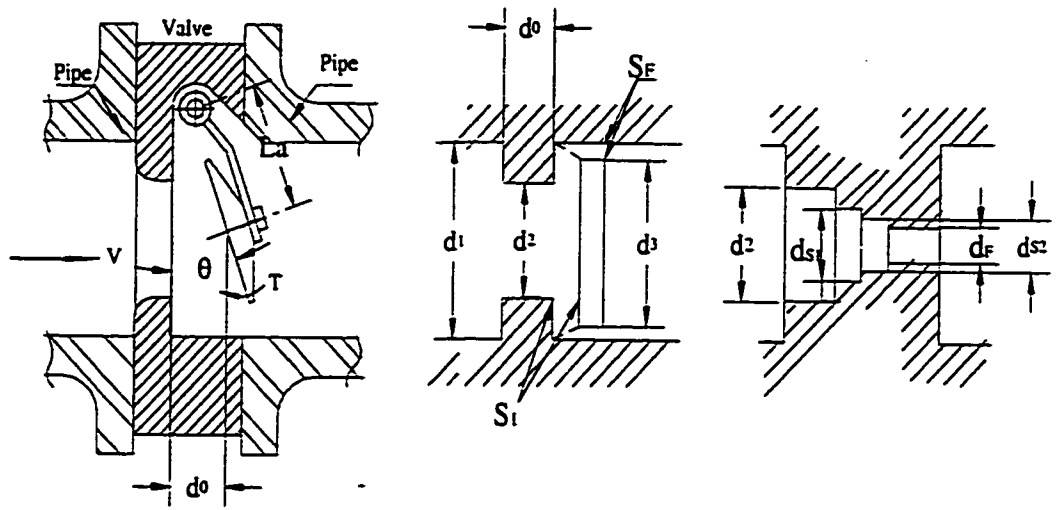
The value of the equivalent length for wafer disk check valves depends on valve sizes and disc opening angles. For a widely open valve, the equivalent length of a wafer check valve can be approximated as:

$$K_0 = 76 \quad (3.4.20)$$

### 3.4.2.2 Coefficient of Resistance of Forward Flow

Cavazzoni [15] had improved the above orifice model with the following assumptions (Fig.3.4.2):

The valve's internal geometry may be idealized as an orifice, the seat area, and a circular disc, and the vertical projection of disc and hinge (the disc arm) area. Four characteristic surfaces can be identified and their area assumed to be the surface area of four orifices in sequence. Using Fig. 3.4.2 notations:



(a) Valve Geometry

(b) Disc Idealization

(c) Sequence of Orifices

Fig 3.4.2 Orifice Sequence Model

$$\psi = 0.7 \quad (\text{form coefficient})$$

$$S_F = \pi \psi d_1^2$$

$$d_0 = L_a \psi n g \theta$$

$$S_1 = \pi d_2 d_0$$

$$d_F = \sqrt{\psi d_1^2}$$

$$d_{s1} = \sqrt{\frac{4S_1}{\pi}}$$

$$d_{s3} = \sqrt{d_1^2 - d_F^2}$$

The coefficient of resistance for each elementary contraction and expansion can be calculated as follows:

$$K_{1c} = \frac{0.04}{\left(\frac{d_2}{d_1}\right)^4} \quad (3.4.21)$$

$$K_{2e} = \frac{\left(1 - \left(\frac{d_2}{d_{s1}}\right)^2\right)^2 \left(\frac{d_1}{d_{s1}}\right)^4}{\left(\frac{d_2}{d_{s1}}\right)^4} \quad (3.4.22)$$

$$K_{2c} = \frac{0.5 \left(1 - \left(\frac{d_{s1}}{d_2}\right)^2\right) \left(\frac{d_1}{d_2}\right)^4}{\left(\frac{d_{s1}}{d_2}\right)^4} \quad (3.4.23)$$

$$K_{3c} = \frac{\left(1 + \left(\frac{d_F}{d_{s1}}\right)^2\right) \sqrt{\sin(90 - \theta)} \left(\frac{d_1}{d_{s1}}\right)}{\left(\frac{d_F}{d_{s1}}\right)^4} \quad (3.4.24)$$

$$K_{4e} = \frac{\left(1 - \left(\frac{d_F}{d_1}\right)^2\right)^2}{\left(\frac{d_F}{d_1}\right)^4} \quad (3.4.25)$$

and

$$K = K_{1c} + K_{2e} + K_{2c} + K_{3c} + K_{4e} \quad (3.4.26)$$

the above formula is valid for :

$$15^\circ < \theta < 70^\circ$$

For disc angle less than  $15^\circ$ , the coefficient of resistance may be considered as

using:

$$K = \frac{15K_{(\theta=15^\circ)}}{\theta} \quad (3.4.27)$$

### 3.4.2.3 Coefficient of Resistance of Reverse Flow

Using the same notations as Fig. 3.4.2, the elementary coefficients of resistance for a reversed flow are:

$$K_{1re} = \frac{\left(1 - \left(\frac{d_2}{d_1}\right)^2\right)^2}{\left(\frac{d_2}{d_1}\right)^4} \quad (3.4.28)$$

$$K_{2re} = \frac{\left(1 - \left(\frac{d_{s1}}{d_2}\right)^2\right)^2 \left(\frac{d_1}{d_2}\right)^4}{\left(\frac{d_{s1}}{d_2}\right)^4} \quad (3.4.29)$$

$$K_{2rc} = \frac{0.5 \left(1 - \left(\frac{d_2}{d_{s1}}\right)^2\right) \left(\frac{d_1}{d_{s1}}\right)^4}{\left(\frac{d_2}{d_{s1}}\right)^4} \quad (3.4.30)$$

$$K_{3re} = \frac{\left(1 - \left(\frac{d_F}{d_{s1}}\right)^2\right)^2 \left(\frac{d_{s1}}{d_1}\right)^4}{\left(\frac{d_F}{d_{s1}}\right)^4} \quad (3.4.31)$$

$$K_{4rc} = \frac{\left(1 - \left(\frac{d_F}{d_1}\right)^2\right)^2}{\left(\frac{d_F}{d_1}\right)^4} \quad (3.4.32)$$

and  $K = K_{1re} + K_{2re} + K_{2rc} + K_{3re} + K_{4rc}$  (3.4.33)

The above formula is valid for:

$$10^\circ < \theta < 70^\circ$$

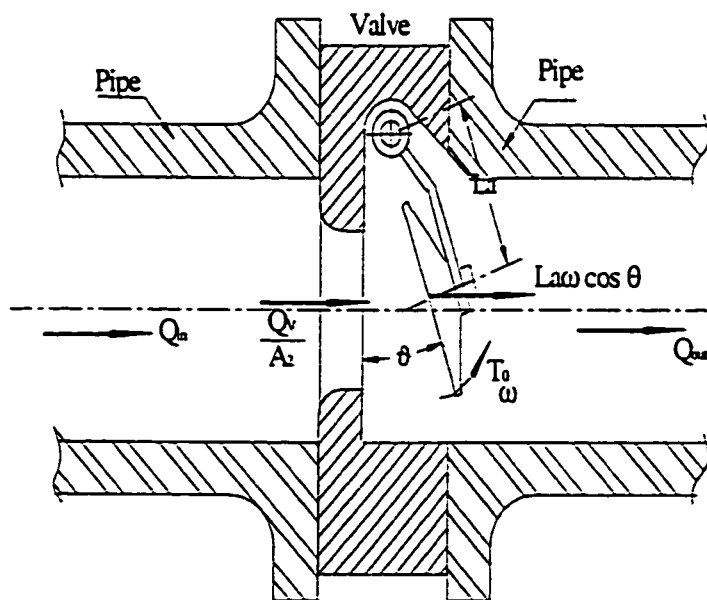
For disc angle less than  $10^\circ$  the coefficient of resistance may be considered as:

$$K = \frac{10K_{(\theta=10^\circ)}}{\theta} \quad (3.4.34)$$

Now that the coefficient of resistance for forward and reverse flow has been obtained, Eq. (3.4.3) can be used to determine the pressure drop for the forward and reverse flow condition.

### 3.4.3 Swing-Disk Check Valve Flow Torque

Fig.3.4.3 shows the important parameters for the unsteady flow condition through a check valve. Due to the fluid forces acting against the valve disc, a torque  $T_0$  is created which acts on the disc shaft forcing the disc to swing with the angular velocity  $\omega$ .



**Fig 3.4.3 Unsteady Flow Through a Check Valve**

The relative uniform fluid velocity  $V_R$  with respect to the angular velocity of the disc forward flow is given by:

$$V_R = V_2^* - L\omega \cos \theta \quad (3.4.35)$$

where  $L_a$  is the hinge arm length and  $\theta$  is the disc opening angle.

And  $V_1^* = \frac{Q_v}{A_2}$  where  $Q_v$  is the flow through the valve and  $A_2$  is the orifice area.

For reverse flow,

$$V_R = V_1^* - L_a \omega \cos \theta \quad (3.4.36)$$

and  $V_1^* = \frac{Q_v}{A_2}$ . Where  $A_1$  is the pipe area.

The flow force is acting normal to the disc center and is expressed in terms of the normal drag coefficient  $C_{ND}$ :

$$F_D = \frac{1}{2} \rho C_{ND} A_3 V_R |V_R| \quad (3.4.37)$$

where  $A_3$  is the disc area and  $\rho$  is the density of water.

This may also be written in the form of  $Q_v$ :

$$F_D = \frac{1}{2} \rho K_{total} \frac{Q_v^2}{A_3} \quad (3.4.38)$$

where  $K_{total}$  is the total coefficient of resistance.

The flow torque acting on the disc shaft for a wafer swing-disc check valve is given as:

$$T_{FD} = F_D L_a \quad (3.4.39)$$

For complete description of the fluid torque acting on the valve disc, the normal drag coefficient must be determined as a function of disc angle and pipe flow Reynolds number. This may be obtained through experimentation.

If the orifice sequence model is introduced, the drag force can be seen as the sum of five elementary components corresponding to the five contractions and expansions.

Referring to Fig. 3.4.2,

$$F_{D1} = \frac{1}{2} \rho K_i \frac{Q_v}{A_i} \quad (3.4.40)$$

and

$$F_{DC} = F_{D1} + F_{D2} + F_{D3} + F_{D4} + F_{D5} \quad (3.4.41)$$

Now, the Torque Arm Coefficient (TAC) is introduced. It can be obtained through experimentation [19], or represented by the following approximation:

$$0^\circ < \theta < 20^\circ \quad TAC = 0.74 \quad (3.4.42)$$

$$20^\circ < \theta < 70^\circ \quad TAC = 0.0209 * \theta + 0.322 \quad (3.4.43)$$

The disc shaft torque can be modified as:

$$T_{FD} = TAC * L_a * F_{DC} \quad (3.4.44)$$

For reverse flow, the flow force can be written as:

$$F_{R1} = \frac{1}{2} \rho K_r \frac{Q_v}{A_i} \quad (3.4.45)$$

and

$$F_{RC} = F_{R1} + F_{R2} + F_{R3} + F_{R4} + F_{R5} \quad (3.4.46)$$

The algebraic expression of TAC is:

$$0^\circ < \theta < 20^\circ \quad TAC = 0.65 \quad (3.4.47)$$

$$20^\circ < \theta < 45^\circ \quad TAC = 0.0117 * \theta + 0.416 \quad (3.4.48)$$

$$45^\circ < \theta < 70^\circ \quad TAC = 0.0583 * \theta - 1.6820 \quad (3.4.49)$$

The reverse flow torque is:

$$T_{FR} = TAC * L_a * F_{RC} \quad (3.4.50)$$

### 3.5 Boundary Condition for Check Valve [21,24]

If the check valve is located inline, referring to section 2.4.4 and section 3.4.2, using

Eq.(3.4.8), Eq.(2.4.40) becomes:

$$Q_v = \frac{A^2 g}{K} (B_{P_1} + B_{M_2}) - \sqrt{\left(\frac{A^2 g}{K}\right)^2 (B_{P_1} + B_{M_2})^2 - 2 \frac{A^2 g}{K} (C_{P_1} - C_{M_2})} \quad (3.5.1)$$

or:

$$Q_v = A^2 g C_v^{*2} (B_{P_1} + B_{M_2}) - \sqrt{\left(A^2 g C_v^{*2}\right)^2 (B_{P_1} + B_{M_2})^2 - 2 A^2 g C_v^{*2} (C_{P_1} - C_{M_2})} \quad (3.5.2)$$

Once the flow is known, Eq.(2.4.19) and (2.4.20) can be used to find the pressure head.

If the check valve is located near a pump, referring to section 2.4.5.2, using Eq.(3.4.8), Eq.(2.4.50) becomes:

$$F_H = C_P - C_M - v Q_R (B_P + B_M) + H_R (\alpha^2 + v^2) [A_0 + A_1 (\pi + \tan^{-1} \frac{v}{\alpha})] - \frac{v |v| Q_R^2 K}{2 A^2 g} = 0 \quad (3.5.3)$$

or:

$$F_H = C_P - C_M - v Q_R (B_P + B_M) + H_R (\alpha^2 + v^2) [A_0 + A_1 (\pi + \tan^{-1} \frac{v}{\alpha})] - \frac{v |v| Q_R^2}{2 A^2 g C_v^{*2}} = 0 \quad (3.5.4)$$

$K$  or  $C_v^*$  is the coefficient of resistance, and may be obtained by using the equation discussed in Sec. 3.4.2.1—3.4.2.3.

It has been demonstrated that all the torque represented in section 3.3 and Eq.(3.2.1) are available for calculation and numerical representation. Therefore, it is possible to represent check valve dynamics by using a set of ordinary differential equations, which can be solved using numerical techniques.

## Chapter 4

### Simulation and Model of Pipeline Network Transients with Check Valve System

#### 4.1 Introduction

Experimental investigation of pipeline transients with a check valve system would be very expensive. In a large pipe and valve system, it also would be very difficult and time consuming. As an alternative, an adequate computer simulation would be a valuable substitution as a design aid and for system performance evaluation.

The complete numerical model consists of the pipeline system model, the check valve dynamics model, and the pump characteristic model. The relation of these three model parts is shown in Fig. 4.1.1.

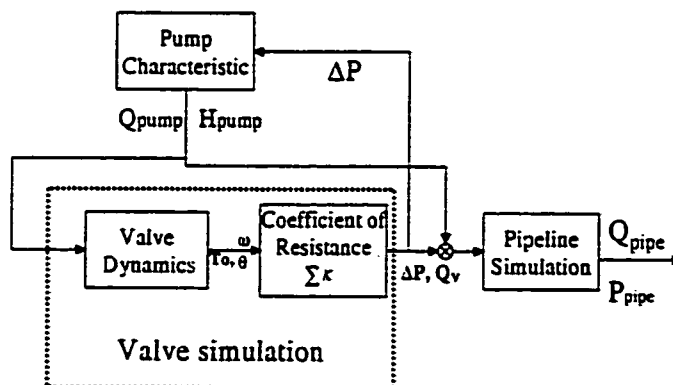


Fig. 4.1.1 Block Diagram of Pump, Check Valve and Pipeline

The simulation of the pipeline, check valve and pump system makes use of the material presented from chapters 2 and 3. In this Chapter, discussion focuses on the solution of the proceeding chapters. The detailed solution of the characteristic method for a pipeline transient will be studied in section 4.2. The check valve dynamic equation will be solved by the Runge-Kutta method in section 4.3. The pump characteristic equation will be solved by the Newton-Ralph method in section 4.4. In this chapter, an example of a network pipeline system will be presented, and the detailed flow chart and an algorithm for the system simulation will be discussed.

## 4.2 *Algebraic Solution to a Piping System* [33, 34]

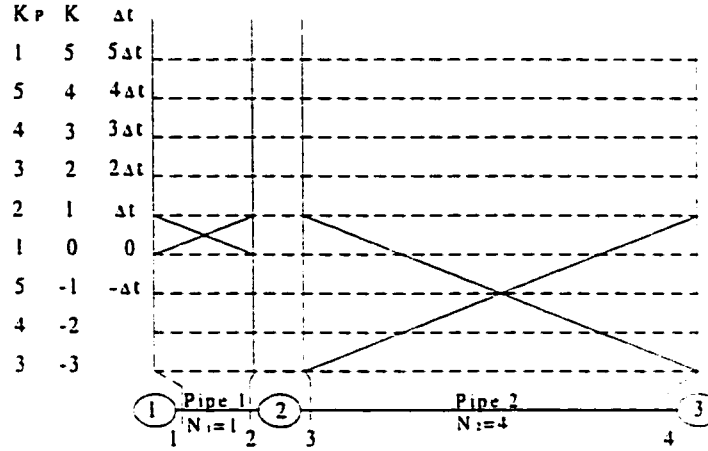
### 4.2.1 **Algebraic Representation and Continuous Storage Data**

The  $C^+$  and  $C^-$  characteristic equations may be transformed into other forms in which time is a subscript, and in which the characteristic lines extend to more than one reach, generally the full pipe length. The equations are called *Algebraic Equations*. These equations may be used to solve most problems handled by the method of characteristics, but they have a form which is particularly convenient for transient calculations in piping systems and networks.

The algebraic forms of the equations have the following properties:

1. They are compatibility equations applied over one or more reaches of a pipe.
2. Time is indicated by an integer subscript.

3. The time increment is  $\Delta t = L/(Na)$ , with  $N$  being an integer.
4. The section numbers along the system may also be subscripted or, for simple systems, the location of head and discharge may be identified by a change in variable name (e.g.  $H_A$ ,  $Q_A$ ).



**Fig. 4.2.1 x—t Diagram for Algebraic Representations in Series System**

There are several forms of numerical studies, in the following focus will be on discussion of *double-subscripted notation*. The schematic diagram of a series system is shown in Fig.4.2.1 with nodes at pipe ends, continuous sectioning, and an integer number of normal reaches for each pipe,  $N = L/(a \Delta t)$ . It is noted that each pipe needs only two computational sections, one at each end. With the data shown, at node 2, the equations extending the full pipe length become:

$$C^+ : \quad H_{2,K} = H_{1,K-N_1} - B_1(Q_{2,K} - Q_{1,K-N_1}) - R_1 | Q_{1,K-N_1} | Q_{2,K} \quad (4.2.1)$$

$$C^- : \quad H_{3,K} = H_{4,K-N_2} + B_2(Q_{3,K} - Q_{4,K-N_2}) + R_2 | Q_{4,K-N_2} | Q_{3,K} \quad (4.2.2)$$

in which the first subscript indicates section number and the second indicates the time counter  $K = t / \Delta t$ . The pipeline characteristic impedance and resistance coefficient may be applied to the total pipe length, so that  $B = \frac{a}{gA}$  and  $R = \frac{fL}{2gDA^2}$ .

Referring to Fig. 4.2.1, each of the variables  $H$  and  $Q$  is dimensioned to the size of the maximum value  $K_M$  in the entire system,  $K_M = L/(a \Delta t) + 1$ . The integer  $K$  is used as a counter for each time step, and  $K_P$  is used as a pointer to identify the position in vector  $H$  or  $Q$  at which current values of the variables are stored. At each iteration when  $K$  is incremented, so is  $K_P$ , the value of the pointer. When  $K_P$  exceeds  $K_M$ , the value is set back to zero, so that it points to the first position in the vector. In this example, information must be retained in pipe 2 for  $N_2 + 1 = 5$  computational steps. Since, for this system, the characteristic lines of pipe 2 extend back the farthest,  $K_M = 5$ . By starting the  $x-t$  diagram  $K_M - 1$  computational increments before the transient starts (Fig. 4.2.1), and storing the steady state values, the equations may be solved at each pipe end, with the incorporation of boundaries and nodal conditions.

The advantages of these algebraic representations over the normal characteristic equations are:

1. They are normally applied over several reaches, with no need to calculate the transient at the intervening sections.
2. They are computationally efficient for solution of multipipe systems such as networks, because interior point calculations are avoided.
3. They use a small  $\Delta t = L/(Na)$ , so that the boundary condition detail is preserved.

But there are some disadvantages inherent in this method, because the characteristic lines are generally extended over more than one reach, the friction term may be less accurate for high-friction systems. Also, since the information must be retained from time steps extending backward in time more than one time step, more computer storage is needed to complete the calculation. However, because of the sophistication of modern computers which are used in these simulations, memory storage is not a crucial issue. It therefore is sensible to choose the algebraic equations to calculate for the pipeline-check valve transient simulation.

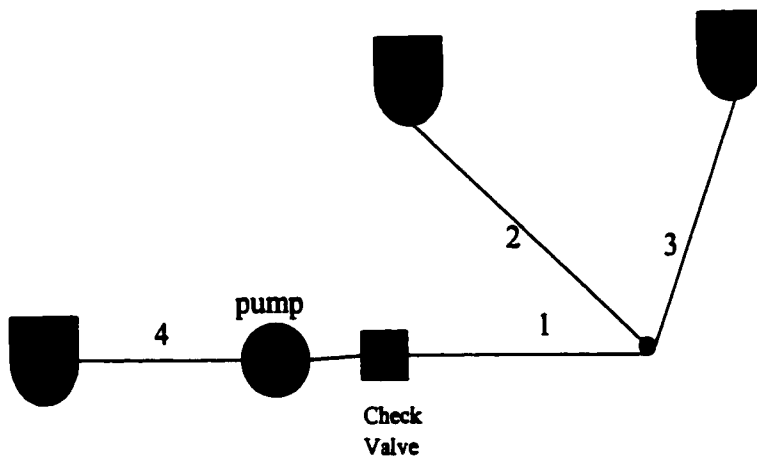
#### **4.2.2 Indexing [18]**

If the algebraic form of the equations in multi-pipe systems is to be used, first the method for storing the system topology must be established. This is called the *indexing* method. An index vector is provided which offers one form of bookkeeping to describe the system connection. It is oriented around the nodes of the system and, for each node, the following information is given: the node number, the number of elements attached to the node, and an element number with a sign to indicate whether it is the upstream or downstream end of the element, and the number of the element section adjacent to the node for each element attached.

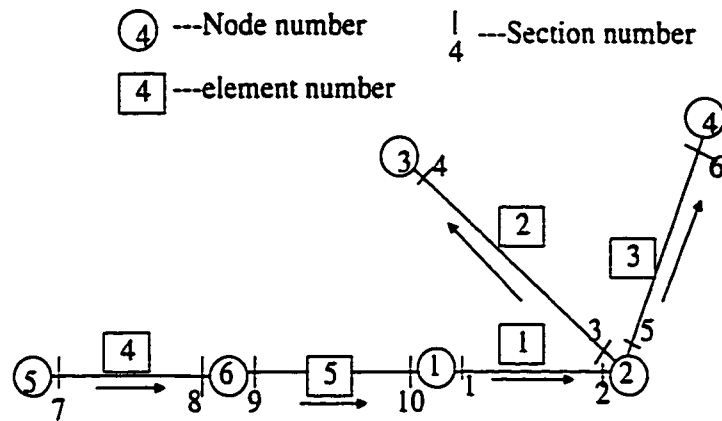
The form of the index vector is:

1. node number (a node represents a connection point, such as a pump, valve, accumulator, reservoir, and pipe junction)
2. number of elements (number of pipes connected to the node)

3. element number (pipe number: - sign for flow out from the node  
+ sign for flow into the node.)
4. element section number adjacent to the node
5. second element number
6. element section number adjacent to the node
7. if there 3 or more elements connected to the node, then steps 5 and 6 are repeated as needed.



**Fig. 4.2.2 Branching System with Pump and Check Valve**



**Fig. 4.2.3 Indexing of Branching System**

Referring to Fig. 4.2.2 and 4.2.3, for a check valve represented by node number 1; there are two pipe elements connected to this node. The first is pipe number 1, in which flow is out from the node by this element, so its sign is minus (-1) and the section number for this pipe adjacent to the node is 1. Element number 5 is the small length of pipe that connects the pump to the check valve. Flow is into the node by element 5 and the section number adjacent to the node is 10.

The index for node 1 is:

1, 2, -1, 1, 5, 10

The above index describes that at node number 1 we have 2 elements. The first element is -1, section number 1; the second element is 5, section number 10.

For the pipe branch represented by node 2, the index is:

2, 3, 1, 2, -2, 3, -3, 5

The above index means that at node 2, there are 3 elements (pipes), pipe 1 is flow into the node, the sign is positive, and the section number is 2. The other two pipes are

flow out of the node, so they are -2 and -3, and the section numbers are 3 and 5 respectively.

For the reservoir represented by end node 4, the index is:

4, 1, 3, 6

The index indicates that node number 4 has 1 pipe element number 3 connected to it, and the adjacent section is number 6.

The index vector does not provide all of the information needed for transient analysis, but it does give the necessary data to enable calculations at junctions and boundaries, especially in connection relationship. The physical parameters that describe each element are needed, in addition to nodal information such as the nodal flow, pressure, and elevation.

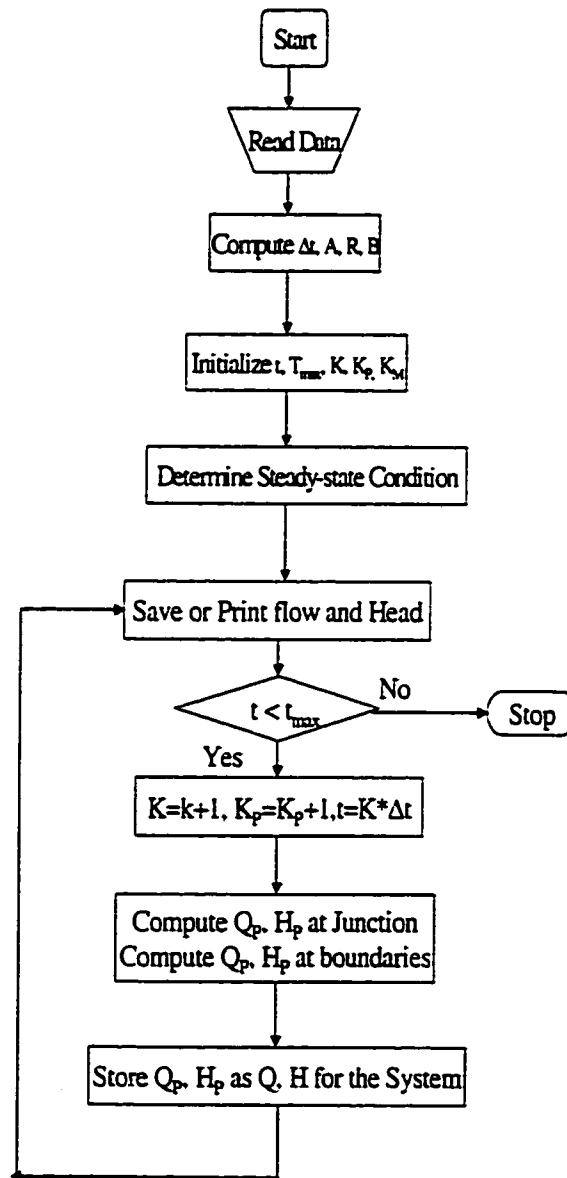
#### 4.2.3 Simulation of Piping System

To compute transient-state conditions in a piping system, the shortest pipe in the system is divided into a number of reaches so that a desired computational time interval,  $\Delta t$ , is obtained [25,26] by:

$$\Delta t = \frac{L_i}{a_i n_i} \quad (i = 1 \text{ to } N) \quad (4.2.3)$$

The remaining pipes in the system are divided into reaches having equal lengths by using Eq. (4.2.3). If necessary, the wave velocities are adjusted to satisfy Eq. (4.2.3) so that characteristics pass through the grid points. The steady-state discharge and pressure head at all the sections are then computed, and their values are saved or printed.

The time is then incremented, and the transient conditions are computed at all the interior points and at the boundaries. This process is continued until transient conditions for the required time are computed. Fig. 4.2.4 show the flowchart of this procedure.



**Fig. 4.2.4 Flowchart of Piping System Simulation**

## 4.3 Numerical Model and Simulation for Check Valve

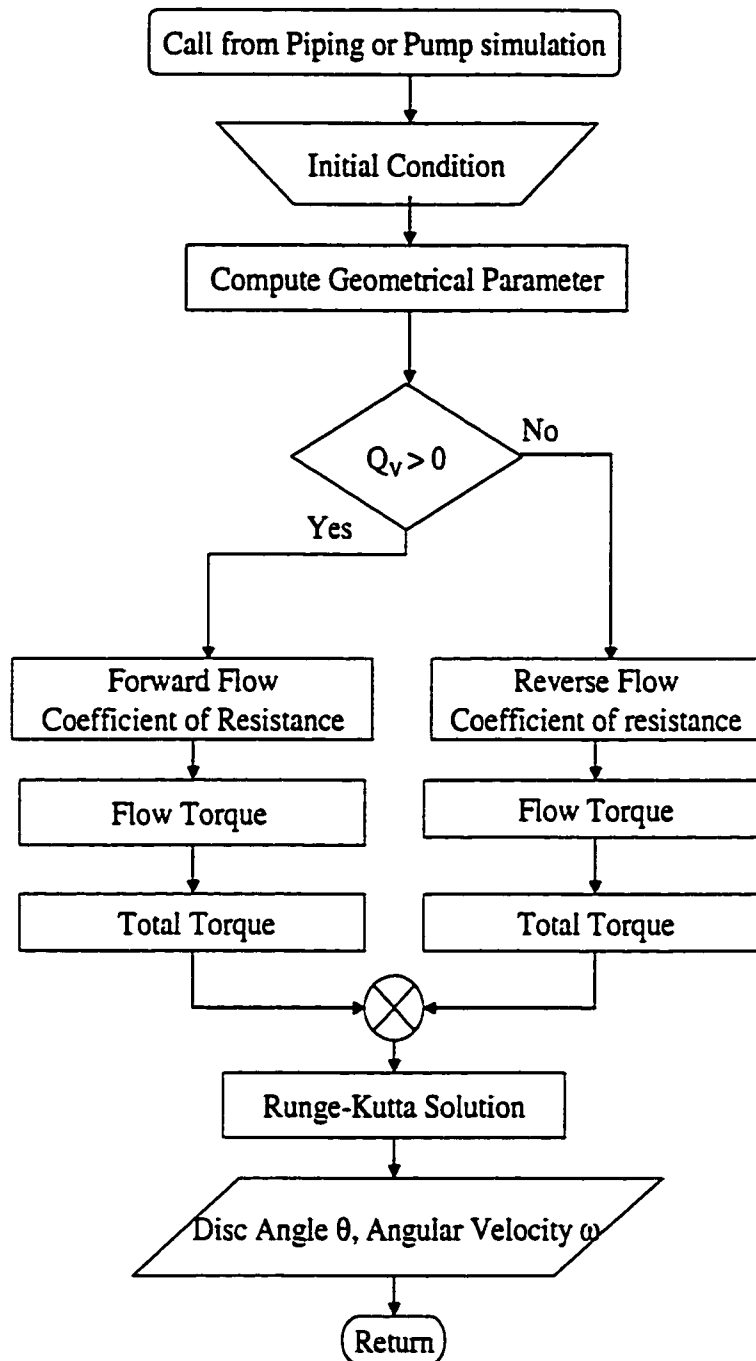
### 4.3.1 Check Valve Dynamic System Model

The complete numerical model consists of the combination of a valve dynamics model with a model of the piping system in unsteady flow conditions. While the piping system model has been investigated in the last section, in this section the check valve numerical model is developed. The Runge-Kutta method is used to solve the second order differential equation, representing a general valve system operating under unsteady conditions.

The check valve dynamic system is represented by a second order, nonlinear, differential equation, described in Section 3.2, Eq. (3.2.1). Before this dynamic equation is solved, the relationships between the pump and check valve, and the piping system and check valve, must be considered.

Fig. 4.1.1 shows these relations. The check valve dynamics requires information for torque  $T_0$  and flow rate  $Q_v$  initial conditions. After simulation, the check valve disc opening angle  $\theta$  and disc angular velocity  $\omega$  can be obtained. Then, from the disc angle information, the check valve coefficients of resistance  $K$  are calculated. Given the check valve pressure drop  $\Delta p$  and the flow coefficient, the flow rate  $Q_v$  through the check valve is calculated. The flow torque  $T_0$  created at the disc arm is then calculated using the check valve flow and coefficients of resistance. The total torque must include the appropriate components of the disc weight torque, counterweight torque, spring torque, damper weight torque and the inertia. As described above, the flowchart for numerical simulation

of the check valve is shown in Fig. 4.3.1.



**Fig 4.3.1 Flowchart of Check Valve Numerical Simulation**

### 4.3.2 The Runge-Kutta Numerical Solution to the Check Valve Dynamic Equation

In section 3.2, the check valve differential equation was described as a second order ordinary differential equation. Referring to Eq. (3.2.1), it may be written in the general form as a nonlinear function of torque:

$$J \frac{d^2\theta}{dt^2} = T(t, \theta, \omega) \quad (4.3.1)$$

$$\frac{d\theta}{dt} = \omega \quad (4.3.2)$$

Therefore Eq. (4.3.1) can be rewritten as:

$$J \frac{d\omega}{dt} = T(t, \theta, \omega) \quad (4.3.3)$$

where  $J$  is system total moment of inertia, and  $T$  is system total torque as a function of disc angular velocity  $\omega$ , disc opening angle  $\theta$ , and time  $t$ .

$\theta$  is disc opening angle

$\omega$  is disc angular velocity

The system total moment of inertia are the sum of the moments of inertia of all moving parts, including disc, counterweight, damper, and so forth. The system total torque term  $T$  is represented by a set of relations describing:

- Coefficient of resistance
- Flow torque
- Counterweight effect
- Spring effect
- Damper effect

- Valve geometric and dynamic parameters

The Runge-Kutta fourth order algorithm [27,28] can be applied to Eq. (4.3.2 and 4.3.3) simultaneously.

Consider a first ordinary differential equation with initial condition  $y(0)$ ,

$$\frac{dy}{dt} = f(y, t) \quad y(0) = y_0 \quad (4.3.4)$$

We can calculate  $y_{n+1}$  at the next time increment  $t_{n+1} = t_n + h$ , with a known value of  $y_n$  at the previous time step  $t_n$ . The fourth order Runge-Kutta formula is written as:

$$\begin{aligned} k_1 &= hf(y_n, t_n) \\ k_2 &= hf\left(y_n + \frac{k_1}{2}, t_n + \frac{h}{2}\right) \\ k_3 &= hf\left(y_n + \frac{k_2}{2}, t_n + \frac{h}{2}\right) \\ k_4 &= hf(y_n + k_3, t_n + h) \end{aligned} \quad (4.3.5)$$

$$y_{n+1} = y_n + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$$

For Eq. (4.3.2), the first derivative variables are:

$$h = \Delta t \quad y = \theta \quad f(y, t) = \omega(\theta, t)$$

and at the same time for Eq.(4.3.3), the second derivative variables are:

$$h = \Delta t \quad y = \omega \quad f(y, t) = T(\omega, \theta, t)$$

and the known initial condition are:

$$\omega(t = 0) = \omega_0 \quad \theta(t = 0) = \theta_0$$

According to Eq. (4.3.5), the following is the C++ code for the Runge-Kutta method:

```
double K1, K2, K3, K4, L1, L2, L3, L4, Tr, theta, thetaR, OmegaR;
```

```
ToJ = GetTotalTorque(Q[Jpump-1], itheta, Omega);
```

```
K1 = ToJ * Dr; // for coefficient of Eq. (4.3.3)
L1 = Omega * Dr; // for coefficient of Eq. (4.3.2)
Tr = T + 0.5 * Dr;
theta = itheta * Pi / 180; //  $\theta$  in radian at  $t$ ,  $\theta_R$  is at  $t+Dt$ 
thetaR = theta + 0.5 * L1;
thetaD = thetaR * 180 / Pi; //  $\theta$  in degree
OmegaR = Omega + 0.5 * K1; //  $\omega$  is the angular velocity at  $t$ ,  $\omega_R$  is at  $t+Dt$ 
```

```
Qcv = GetValveFlow ( Tr, thetaD);
ToJ = GetTotalTorque(Qcv, thetaD, OmegaR);
```

```
K2 = ToJ * Dr;
L2 = OmegaR * Dr;
```

```
Tr = T + 0.5 * Dr;
thetaR = theta + 0.5 * L2;
thetaD = thetaR * 180 / Pi;
OmegaR = Omega + 0.5 * K2;
```

```
Qcv = GetValveFlow ( Tr, thetaD);
ToJ = GetTotalTorque(Qcv, thetaD, OmegaR);
```

```
K3 = ToJ * Dr;
L3 = OmegaR * Dr;
```

```
Tr = T + Dr;
thetaR = theta + L3;
thetaD = thetaR * 180 / Pi;
OmegaR = Omega + K3;
```

```
Qcv = GetValveFlow ( Tr, thetaD);
ToJ = GetTotalTorque(Qcv, thetaD, OmegaR);
```

```
K4 = ToJ * Dr;
L4 = OmegaR * Dr;
```

```
fDiscAngularVelocity = Omega + (K1 + 2 * K2 + 2 * K3 + K4) / 6;
fdiscAngleRad = theta + (L1 + 2 * L2 + 2 * L3 + L4) / 6; // these two variables will be stored
// in Systems
```

The above program can be called at a pump junction or a piping junction according to the valve placement.

#### 4.4 Newton-Raphson Solution to Pump Boundary Condition [7]

In chapter 2, the head balance equation and speed change equation for the pump conditions were derived as:

$$F_H = C_P - C_M - v Q_R (B_P + B_M) + H_R (\alpha^2 + v^2) [A_0 + A_1 (\pi + \tan^{-1} \frac{v}{\alpha})] - \frac{v |v| \Delta H_0}{\tau^2} = 0 \quad (2.4.50)$$

$$F_T = (\alpha^2 + v^2) [B_0 + B_1 (\pi + \tan^{-1} \frac{v}{\alpha})] + \beta_0 - C_T (\alpha_0 - \alpha) = 0 \quad (2.4.59)$$

These equations can be solved simultaneously for  $\alpha$  and  $v$  as following by using the Newton-Raphson method:

$$F_H + \frac{\partial F_H}{\partial v} \Delta v + \frac{\partial F_H}{\partial \alpha} \Delta \alpha = 0 \quad (4.4.1)$$

$$F_T + \frac{\partial F_T}{\partial v} \Delta v + \frac{\partial F_T}{\partial \alpha} \Delta \alpha = 0$$

As a first step in the solution  $\alpha$  and  $v$  are approximated at the new time step by an extrapolation of the previously computed values, two time steps,  $\alpha_0$ ,  $v_0$ , and four time steps,  $\alpha_{00}$ ,  $v_{00}$ , earlier.

$$v = 2v_0 - v_{00} \quad \alpha = 2\alpha_0 - \alpha_{00} \quad (4.4.2)$$

Next the coefficients  $F_H, \frac{\partial F_H}{\partial v}, \frac{\partial F_H}{\partial \alpha}, F_T, \frac{\partial F_T}{\partial v}, \frac{\partial F_T}{\partial \alpha}$  are evaluated for this  $\alpha$  and  $v$ .

The derivatives are:

$$\frac{\partial F_H}{\partial v} = -Q_R (B_P + B_M) + H_R 2v [A_0 + A_1 (\pi + \tan^{-1} \frac{v}{\alpha})] + H_R A_1 \alpha - \frac{2\Delta H_0 |v|}{\tau^2} \quad (4.4.3)$$

$$\frac{\partial F_H}{\partial \alpha} = H_R 2\alpha [A_0 + A_1 (\pi + \tan^{-1} \frac{v}{\alpha})] - H_R A_1 v \quad (4.4.4)$$

$$\frac{\partial F_T}{\partial \nu} = 2\nu[B_0 + B_1(\pi + \tan^{-1} \frac{\nu}{\alpha})] + B_1\alpha \quad (4.4.5)$$

$$\frac{\partial F_T}{\partial \alpha} = 2\alpha[B_0 + B_1(\pi + \tan^{-1} \frac{\nu}{\alpha})] - B_1\nu + C_T \quad (4.4.6)$$

Eq. (4.4.1) can be solved for  $\Delta\alpha$  and  $\Delta\nu$ :

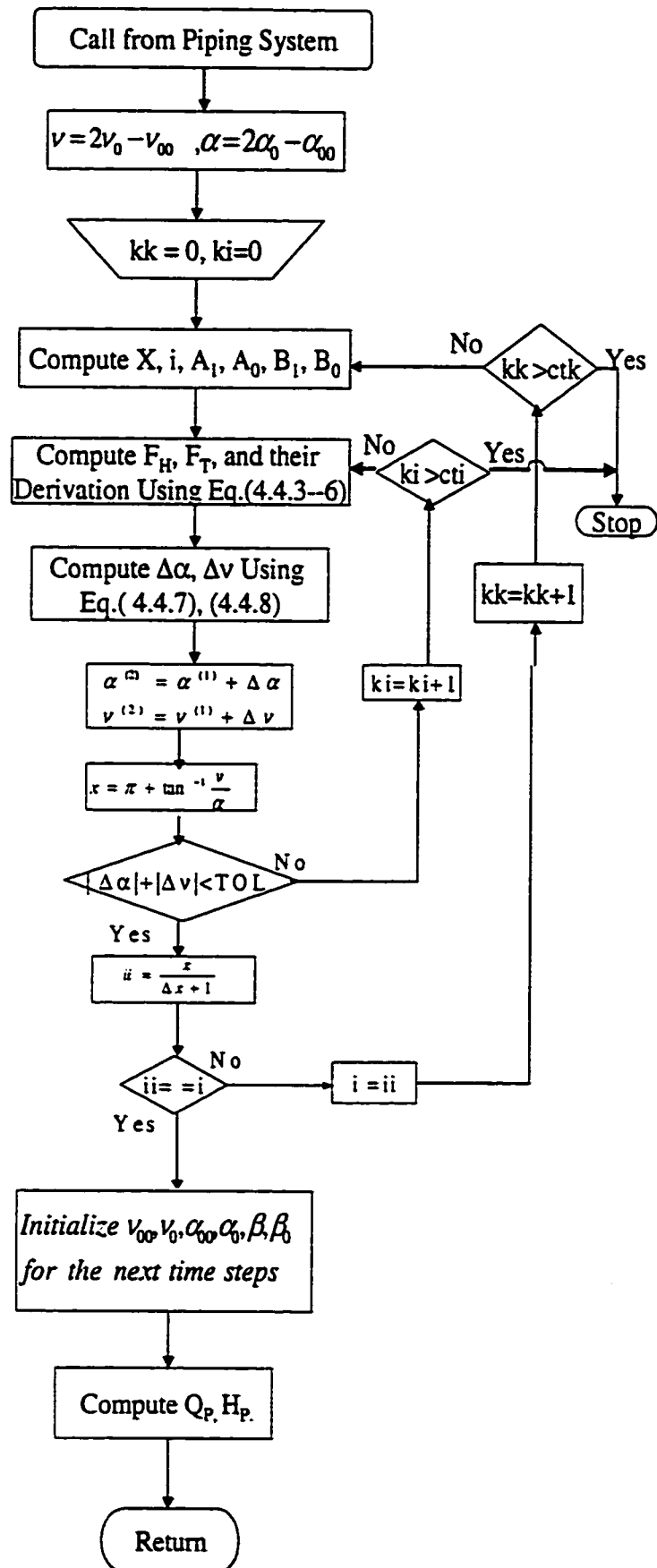
$$\Delta\alpha = \frac{F_T \left/ \frac{\partial F_T}{\partial \nu} - F_H \left/ \frac{\partial F_H}{\partial \nu} \right.}{\frac{\partial F_H}{\partial \alpha} \left/ \frac{\partial F_H}{\partial \nu} - \frac{\partial F_T}{\partial \alpha} \left/ \frac{\partial F_T}{\partial \nu} \right.} \quad (4.4.7)$$

$$\Delta\nu = -\frac{F_H}{\frac{\partial F_H}{\partial \nu}} - \Delta\alpha \frac{\frac{\partial F_H}{\partial \alpha}}{\frac{\partial F_H}{\partial \nu}} \quad (4.4.8)$$

Let  $\alpha^{(1)}$  and  $\nu^{(1)}$  be the initial estimated values, and  $\alpha^{(2)}$  and  $\nu^{(2)}$  be the values after first iteration. Then, the following equations yield improved values of  $\alpha$  and  $\nu$ :

$$\begin{aligned} \alpha^{(2)} &= \alpha^{(1)} + \Delta\alpha \\ \nu^{(2)} &= \nu^{(1)} + \Delta\nu \end{aligned} \quad (4.4.9)$$

This procedure is repeated either a fixed number of times or until the sum of the absolute values of the corrections  $\Delta\alpha$  and  $\Delta\nu$  is less than an acceptable tolerance (e.g., 0.0002). The flowchart of Fig. 4.4.1 illustrates the above procedure.



**Fig. 4.4.1 Newton-Raphson Solution for Pump**

In this chapter, the numerical aspect of algebraic representation for the characteristic method, check valve dynamics equation, and pump boundary conditions has been discussed. An elegant solution for pipeline transient analysis was shown to be achieved in the use of algebraic solution, continuous data storage, and indexing for system topology. A flowchart and C++ code were given for clear and direct illustration to the check valve and pump dynamic simulation. The software design which implements this analysis will be given in the following chapter.

## Chapter 5

# **Software Implement of Transients in Pipeline Network and Check Valve System**

### *5.1 Introduction*

In the previous chapters, the main focus was on the theory and methods for dealing with hydraulic transients. The basic equations, the methods of analysis, the specific solution, and specific boundary conditions were investigated. As well, pipeline configuration, the pump characteristics, and the check valve dynamics were discussed. Hydraulic transients in the pipeline network system are implemented by, and interacted with the pump and check valve dynamic behavior and software program. Therefore, the following chapter will focus on the analysis of the pipeline network system with a view of software engineering and design and coding for the Hydro-Analysis program package using Object-Oriented Programming techniques.

The analysis, which includes the problem specification and requirement analysis, simulation analysis, input and output analysis, and the user interface analysis will be performed in Section 5.2. The design will include high level algorithm and hierarchical diagrams that will be given in Section 5.3. Also in this section, C++ program language will be used to perform transient analysis. As well, visual C++ will be used to design the user interfaces for user-friendly input and graphics output. The introduction to all classes used in this package will be given in this section as well, and Section 5.4 will simulate a

pipeline network system. What follows is an in-depth step-by-step description of how to use this software package.

## **5.2    *Analysis for Hydro-Analysis Software Package***

### **5.2.1    Problem Description**

The application should input the pipeline connection information, the individual pipes, pump, and valve geometrical and characteristic data, and then simulate the system for a period of time. Next it should determine the pressure at certain nodes and flow rate at certain sections during the occurrence of transients. After finishing the simulation, the application should save the output data, such as the Head (pressure) and Flow rate with respect to time, and also produce a graphic output for the Head and Flow rate.

The pipeline network transients analysis and simulation should combine the pipeline transients, pump and check valve dynamic simulation. This simulation should be performed at different situations, such as in pump start-up, pump failure, and cases in which the control valve is closed and when it is opened. The application can also handle different network connections: the series system, the parallel system, the branch system, and network system. It can deal with different pump arrangements, including single pump, series pumps, the parallel pumps, and complex pumps. Finally, it can simulate the system for different sizes and types of check valves.

### 5.2.2 Input Parameters

As described in Section 4.2.2, the connect relationship for a network pipeline system is given by an index vector. The flow direction for a pipe and the type of junctions should be specified. The input data for the connection, pipe, junction, pump, check valve, valve and reservoir is as follows:

The connection parameters include:

1. Index data
2. Number of index
3. Number of pipes
4. Number of junctions
5. Number of pumps
6. Upstream junction numbers of a pipe
7. Downstream junction numbers of a pipe

The junction parameters include:

1. Type of a junction
2. Junction head
3. Junction flow rate
4. Junction elevation

The pipeline parameters include:

1. Pipe length
2. Pipe diameter
3. Pipe friction factor  $f$

The pump parameters include:

1. Pump start up
2. Pump failure
3. Pump normal run
4. Pump stop

For the pump start up, the following must be known:

- Pump coefficient  $A_0$ ,  $A_1$ , and  $A_2$
- Time to gain speed  $T_s$
- Pump shut off Head  $H_s$

For the pump failure:

- Rated pump flow rate
- Rated pump head
- Rated pump torque
- Rated pump speed
- Pump specific speed
- Pump characteristic data

For the pump normal operation:

- Pump coefficient  $A_0$ ,  $A_1$ , and  $A_2$

The check valve parameters include:

1. Check valve size
2. Pipe diameter
3. Disc weight
4. Hinge length

5. Auxiliary device parameters such as counterweight, extension spring, dash pot and so on

The control valve parameters include:

- Valve performance character  $\tau$
- Valve closure time  $T_c$

The reservoir parameter is:

- Elevation of the reservoir

The simulation parameters are:

- Start time of simulation
- Stop time of simulation

The output parameters are:

- Plot location
- Printing or saving data location

For the plot locations, the following must be known:

- Junction number for head vs time
- Pipe section number for flow rate vs time

Finally, for the print out and save location, the following must be known:

- Junction number for head vs. time (Maximum 3)
- Pipe section number for flow rate vs. time (Maximum 3)

### **5.2.3 Output Requirement**

The output of the application includes the file output and the graphic output.

The file output gives the following data:

1. Time
2. Flow rate at the pump element
3. Flow rate at certain pipe locations (the pipe number are indicated by the user, maximum 3)
4. Head at certain junctions (the junction numbers are indicated by the user, maximum 3)
5. Check valve disc opening angle (if there are check valves in this system)

The graphic output should have the following features:

1. Plot the Head vs. Time and Flow rate vs. Time simultaneously on the screen.
2. Display the following information:
  - pump start-up, pump failure, or pump normal run
  - with or without the check valve
  - the pipe number for the flow rate plot, the junction number for the head plot
3. A plot from which the user may easily obtain the data
4. Easy control of graphic output display such as line type, color, and style

## **5.3 Software Design**

### **5.3.1 Algorithm for the Hydro-Analysis**

The high level algorithm can be summarised as following:

*begin the application*

*read data (information for the network connection, pipe, junction, pump, check valve, valve, reservoir)*

*read information on saving results and plotting results*

*construct the network system*

*determine the initial flow rate for each pipe and head for each junction*

*save and print the initial flow rate, head and check valve disc angle*

*While the time is less than time maximum (the time of simulation)*

*if the element is check valve*

*do check valve dynamic analysis*

*get check valve disc angle*

*end if*

*calculate the pipe element flow rate and head*

*calculate the pump element flow rate and head*

*store the flow rate, head and check valve disc angle for next time step use*

*save and print the flow rate, head of indicated location and check valve disc angle*

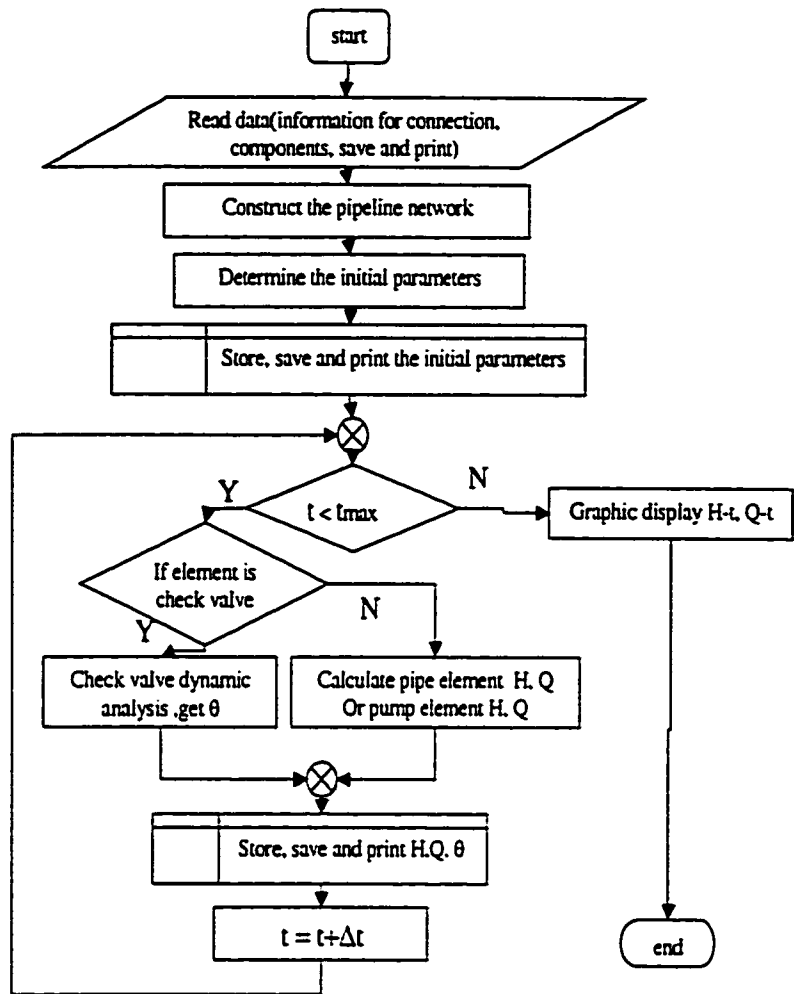
*time increment for the next step*

*end while*

*graphic display the result of flow rate Vs. time and head Vs. time*

*end the application*

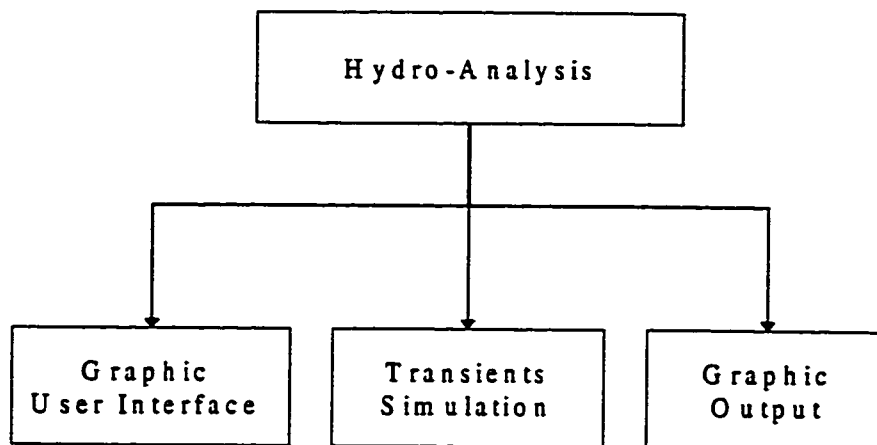
Fig 5.3.1.1 shows the flow chart of the above algorithm:



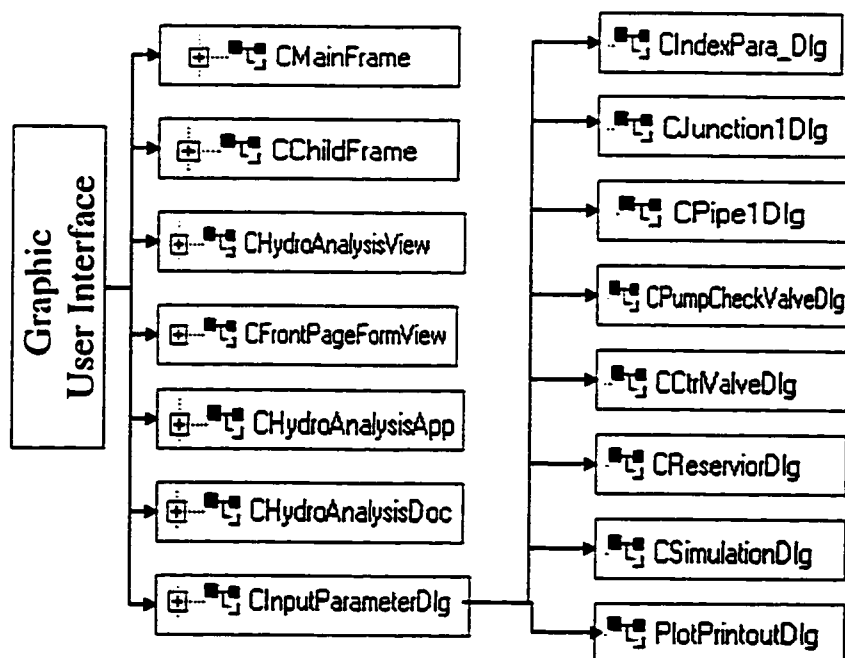
**Fig 5.3.1.1 Flow Chart of Hydro-Analysis**

### 5.3.2 Hierarchical Diagram

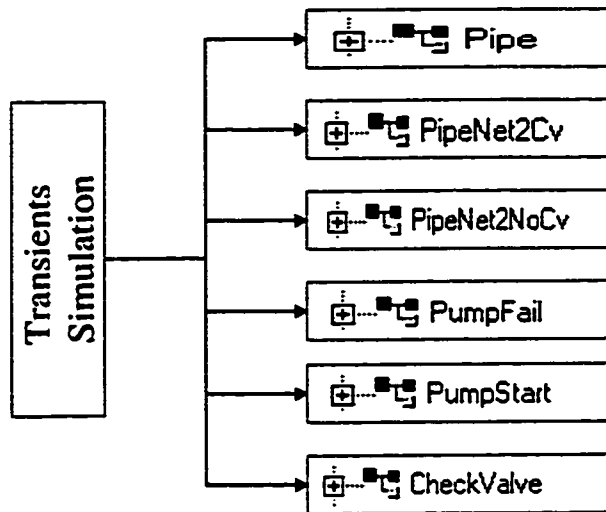
Before outlining the object-oriented-programming design, the hierarchical diagram for the Hydro-Analysis (Fig. 5.3.2.1) is presented below:



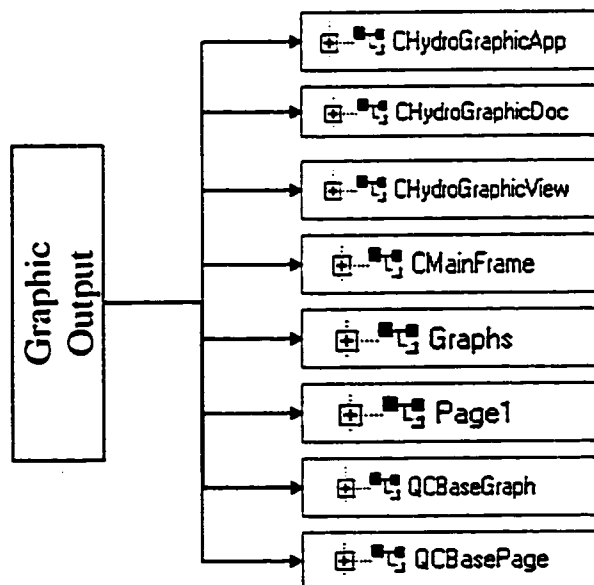
**Fig.5.3.2.1 (a) Hierarchical Diagram for Hydro-Analysis**



**Fig. 5.3.2.1 (b) Hierarchical Diagram for Graphic User Interface**



**Fig. 5.3.2.1 (c) Hierarchical Diagram for Transients Simulation**



**Fig. 5.3.2.1 (d) Hierarchical Diagram for Output Graphic**

### 5.3.3 Object-Oriented Programming Design for Hydro-Analysis

In this project, the Object Oriented Programming (OOP) [29] technique is used to design the Graphic User Interface, Transient Analysis and Graphic output.

#### 5.3.3.1 Major Class Description for Graphic User Interface

##### 1. *CMainFrame*:

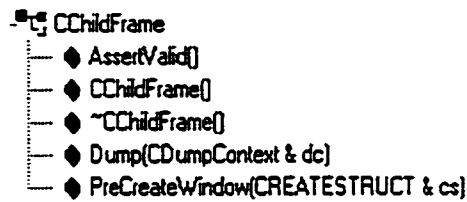
This class provides an enclosed space on the desktop within which all application interactions take place. This frame window is the frame to which the menu and toolbars are attached.

```
class CMainFrame
{
public:
    CMainFrame()
    ~CMainFrame()
    Dump(CDumpContext & dc)
    OnCreate(LPCREATESTRUCT lpCreateStruct)
    PreCreateWindow(CREATESTRUCT & cs)
private:
    m_pFrontWnd
    m_wndStatusBar
    m_wndToolBar
}
```

Fig 5.3.3.1 The operation and attributes of CMainFrame

##### 2. *CChildFrame*:

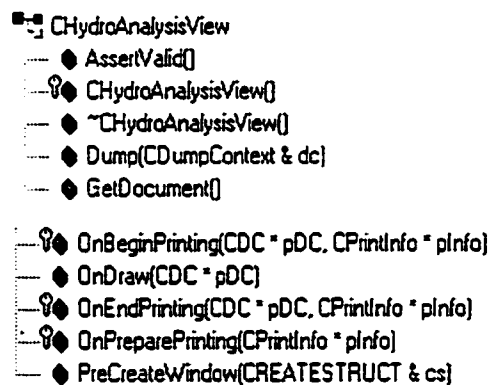
*CChildFrame* is the frame that holds the *CHydroAnalysisView* class and passes messages and events to the *CHydroAnalysisView* class for processing or display.



**Fig. 5.3.3.2 Operation of CChildFrame**

### 3. *CHydroAnalysisView*:

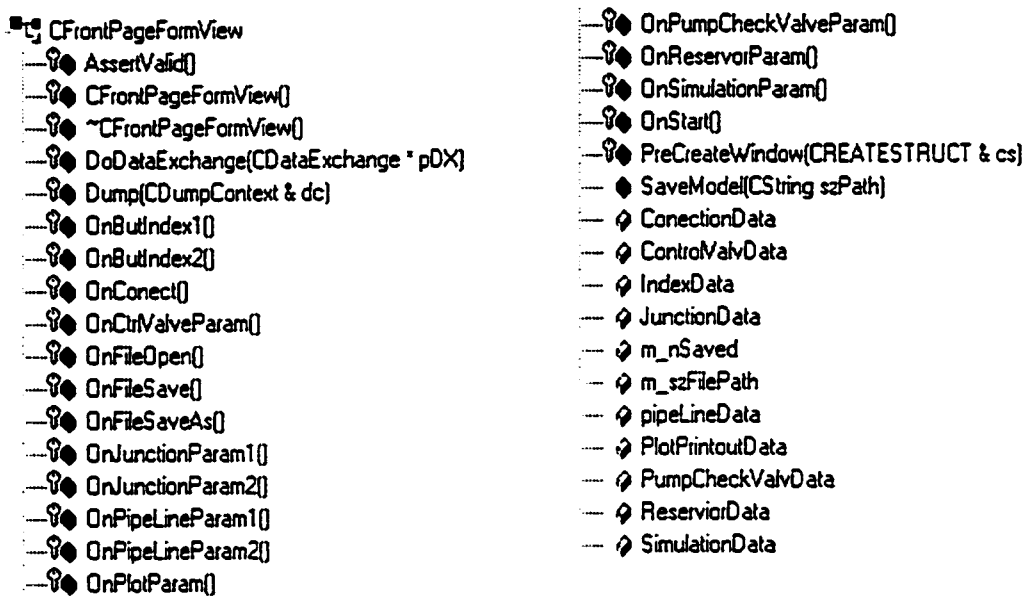
This is the class that displays the visual representation of the document for the user. This class passes input information to the *CHydroAnalysisDoc* class and receives display information from the *CHydroAnalysisDoc*.



**Fig 5.3.3.3 Operation of CHydroAnalysisView**

#### 4. *CFrontPageFormView*

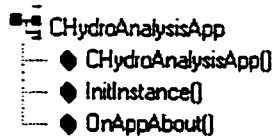
The *CFrontPageFormView* class is the form frame. It holds visible objects attached to the form. It is the base class for views containing controls. It is used to provide form-based documents in applications. It coordinates and interacts with the transient analysis (such as *PipeNet*, *CheckValve*, *PumpFail*, *PumpStart*) class and graphic output application.



**Fig 5.3.3.4 The operation and attributes of CFrontPageFormView**

#### 5. *CHydroAnalysisApp*

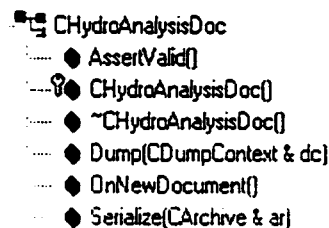
The *CHydroAnalysisApp* class creates all of the other components in the application. It is also the class that receives all event messages and then passes the messages to the *CFrontPageFormView* and *CHydroAnalysisView* classes.



**Fig 5.3.3.5 Operation of CHydroAnalysisApp**

## 6. *CHydroAnalysisDoc*

The *CHydroAnalysisDoc* class houses the document. This is the location of all the data structures necessary to house and manipulate the data that makes up the document. This class receives input from the *CHydroAnalysisView* class and passes display information to the *CHydroAnalysisView* class. Finally, this class is also responsible for saving and retrieving the document data from files.



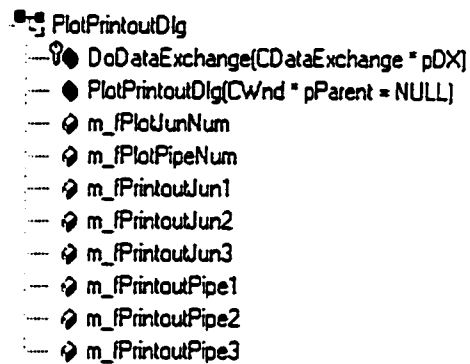
**Fig 5.3.3.6 Operation of CHydroAnalysisDoc**

## 7. *CInputParameterDlg*

*CInputParameterDlg* includes the following classes:

- *CIndexPara\_Dlg*

- *CJunctionDlg*
- *CPipeDlg*
- *CPumpCheckValveDlg*
- *CCtrlValveDlg*
- *CReserviorDlg*
- *CSimulationDlg*
- *PlotPrintoutDlg*



**Fig 5.3.3.7 The operation and attributes of PlotPrintoutDlg**

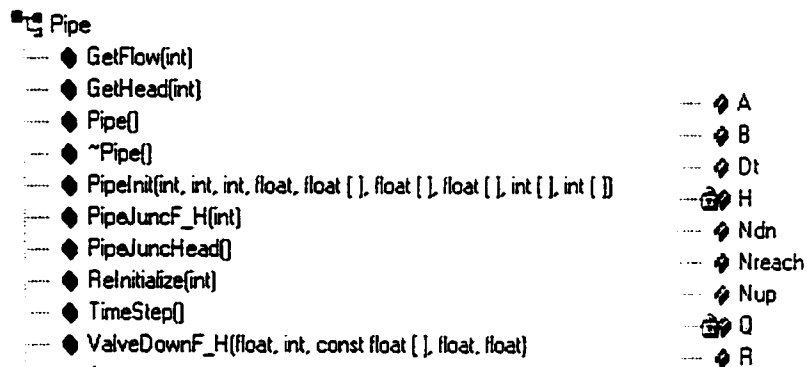
All of these classes receive data which is inputted via the dialog box, and then transfer the data to the *CFrontPageFormView* class. Fig. 5.3.3.7 is the operation and attributes of one of these classes. Other classes perform similar operations but with different attributes.

### 5.3.3.2 Class Description of Transients Simulation

#### 1. Pipe

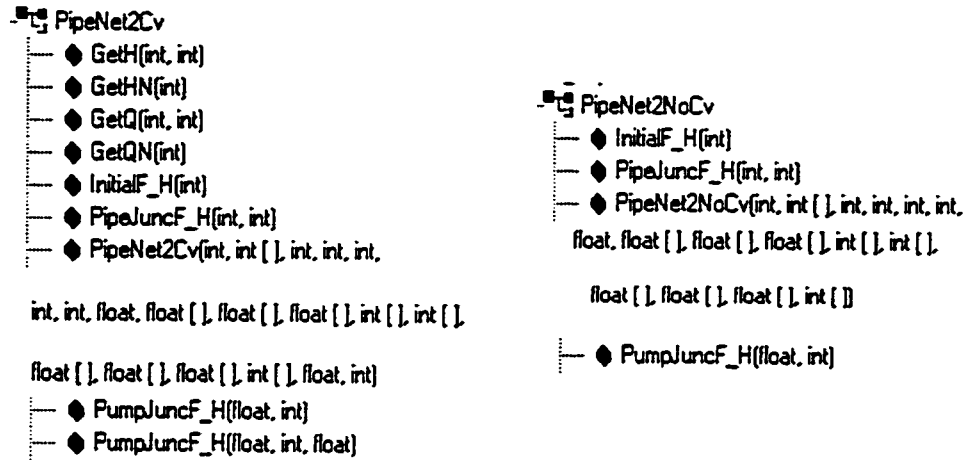
The Pipe class is the base class for pipe simulation. It analyzes one element only. It can calculate flow rate  $Q$  and head  $H$  for each reach of pipe or pump. It also calculates  $A$ ,  $B$ , and  $R$ , where  $A$  is the adjusted sound wave,  $B$  is the pipeline characteristic impedance, and  $R$  is the pipeline resistance coefficient (referring to Eq. 2.4.17 and 2.4.18). Other attribute such as  $Ndn$ ,  $Nreach$ , and  $Nup$  record number of down junctions, reach and up junctions for this pipe.

Fig. 5.3.3.8 shows the operation and attributes of the class Pipe. *PipeInit(...)* obtains the data from the *CFrontpageFormView* class and transfers it to PipeNet2Cv. *ValveDownF\_H()* then proceeds to analyze the system if there is a control valve in this pipe.



**Fig 5.3.3.8 The operation and attributes of Pipe**

## 2. *PipeNet2Cv* and *PipeNet2NoCv*

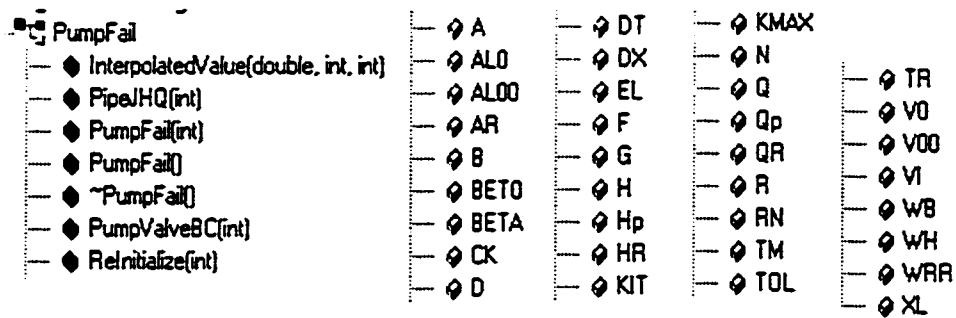


**Fig.5.3.3.9 Operation of *PipeNet2Cv* & *PipeNet2NoCv***

*PipeNet2Cv* class inheritance the class *Pipe*, *PumpStart*, *PumpFail* and *CheckValve*. This is the class that connects all pipes, pumps and check valves together, forming a whole pipeline network system. *PipeNet2Cv* class simulates the system that includes check valves, while the class *PipeNet2NoCv* does not include the check valve. The operation of these two classes is shown in Fig. 5.3.3.9.

## 3. *PumpFail*

*PumpFail* class is another base class for pipeline network simulation. It simply concentrates the hydraulic transient analysis during pump failure, interacting with the class *Pipe* if there is no check valve located after pump, and class *CheckValve* when there is a check valve located after the pump. Referring to Fig.5.3.3.10, the operation is:



**Fig. 5.3.3.10 Operation & Attributes of PumpFail**

*PumpFail()* and *~PumpFail()* are respectively, the constructor and destructor of the PumpFail class. *PumpFail (int)* overloads the constructor. Therefore, only when there is a check valve after the pump, *PumpValveBC(int)* will Analyze the system. Otherwise, *PipeJHQ(int)* will suffice.

The attributes:

A--the wave speed

AL0, AL00 -- dimensionless pump speeds  $\alpha$ , at successive time step

AR, XL--pipeline area, length

B--pipeline impedance, referring to Eq. (2.3.17)

BET0,BETA--previous and current dimensionless torque

CK--valve loss coefficient in velocity head

DT--time step

DX--dimensionless pump data spacing, rad

EL--elevation of the pump node

F--Darcy-Weisbach friction factor

G--gravitational acceleration

H, Hp, Q, Qp, N--head and flow rate and speed during the transients

HR, QR, TR, RN--pump rated head, flow rate, torque and speed

KIT, KMAX--number and maximum number of Newton's method

TM--duration of transients

TOL--tolerance in Newton's method

V, V0, V00--dimensionless discharges at successive time steps

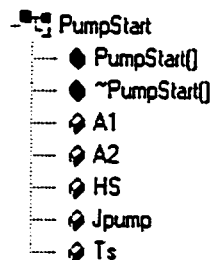
WB, WH--homologous torque and head data

WRR--rotational moment of pump

#### 4. *PumpStart*

*PumpStart* class is a base class, it gets the data from *CFrontpageFormView* class, and then initializes the pump characteristic curve, so that the pump flow rate or head can be determined by using this characteristic curve.

Referring to Eq. (2.3.60), the attributes  $A1$  and  $A2$  are constants for the characteristic curve,  $H_s$  is the pump shutoff Head, and  $T_s$  is the time for the pump to gain speed.  $J_{\text{pump}}$  is the initial element number for pump.



**Fig. 5.3.3.11 Operation & Attributes of *PumpStart***

## 5. CheckValve

The base class *CheckValve* performs the check valve dynamic analysis. The check valve dynamic analysis considers operation of added components, such as: *counterweight(CTWTorque(float))*, *spring(TorqueFromExtensionSpring(float))*, and *DiscTorque(float)*. Using the specifics discussed in Section 3.3, other appropriate components can be added.

Fig. 5.3.3.12 shows the operation of the CheckValve class. *CheckValve()*, and *~CheckValve()* are, respectively, the constructor and destructor of this class. *CheckValveInit(float,int)* creates the check valve with different valve size and maximum disc opening angle.



Fig. 5.3.3.12 Operation & Attributes of CheckValve

Referring to Eq. 3.2.1, before performing the check valve dynamic analysis, the flow torque  $T_F$ , and the bearing friction torque  $T_b$  must both be known. In this analysis, the bearing friction torque will not be considered, because it is a very small term compared with other component terms such as the counterweight and spring.

The operation *DirectFlow(float)*, *DirectTorqueArm(int)*, *ReverseFlow()*, *ReverseTorqueArm(int)*, and *FlowTorque(float,float,float)* are all for the analysis of the flow torque  $T_F$  in Section 3.4.3. *GeomericParameter(float)* and *GetCoefficientsOfResistance()* are used to determine the coefficients of resistance for the check valve dynamic analysis, as discussed in Section 3.4.2. *GetTotalTorque(float,float)*, *GetJoverT(float,float)*, *GetDiscAngle()* and *ValveDynamics(float,float,float,float)* are invoked in the final stage of the check valve dynamics analysis. *Rk4(.....)* is used to solve the dynamic equation by Runge-Kutta method. The attributes of this class are shown in Fig. 5.3.3.12. The meanings are clear and will not be discussed.

### 5.3.3.3 Class Description of Graphic Output

The graphic output application is separated from the HydroAnalysis software package. During a run, the HydroAnalysis starts the HydroGraphic program and then sends the analysis results to it. HydroGraphic receives the data and displays the Head and Flow Rate plot.

In the Hydro-Graphic application, the classes *CHydroGraphicApp*, *CHydroGraphicDoc*, *CHydroGraphicView*, and *CMainFrame* have the same

functionality as the *CHydroAnalysisApp*, *CHydroAnalysisDoc*, *CHydroAnalysisView*, and *CMainFrame* in the Hydro-Analysis.

*QCBaseGraph* and *QCBasePage* are two base classes for graphic output. *BuildGraph(PGRAPH\_DEF,HDC)* is used for initializing the base graph, and *BuildPage(PPAGE\_DEF)* is used for initializing the base page.

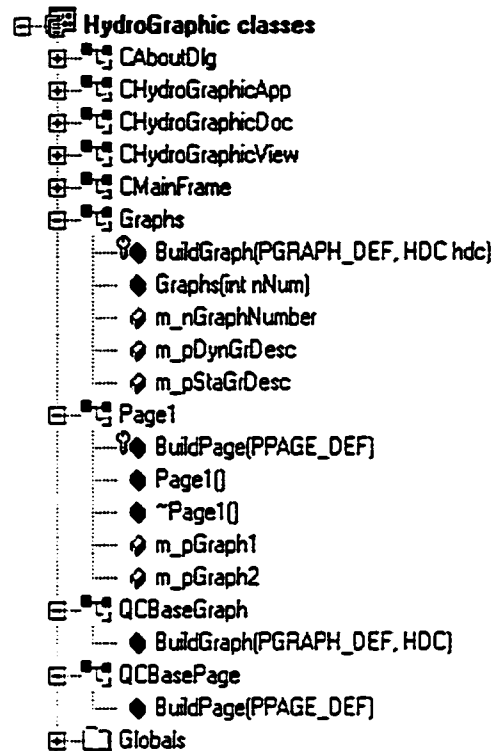
Class *Graphs* is inherited from the class *QCBaseGraph*. *BuildGraph(PGRAPH\_DEF,HDC hdc)* implements the plot. It gets the data from HydroAnalysis application, and then plots two graphs on the same page. The attributes of this class are as follows: *m\_nGraphNumber* is the graph number on the page; *m\_pDynGrDesc* is the pointer to graph descriptions for dynamic graphic plotting, and *m\_pStaGrDesc* is the pointer to graph descriptions for static graphic plotting.

Class *Page1* is inherited from class *QCBasePage*. *BuildPage(PPAGE\_DEF)* implements the page setting, indicating the page size, color, and position. The attributes of this class are as follows: *m\_pGraph1* and *mpGraph2* are the graphic identity in this same page. Fig 5.3.3.13 shows the description of HydroGraphic Class.

## **5.4 Introduction to HydroAnalysis Software**

### **5.4.1 Overview**

*HydroAnalysis* is a program which simulates the hydraulic transient in pipeline networks with a check valve system. The program can solve the transient flow problems in simple, branch and complex pipeline systems.



**Fig. 5.3.3.13 Description of HydroGraphic Classes**

The program *HydroAnalysis* is completely interactive using the Microsoft WINDOW Graphical User Interface. The user can set and change all of the parameters which describe the pipeline network and check valve system. The simulation of the hydraulic transients caused by pump start up, pump failure, or valve operation can only be undertaken once all of the parameters describing the pipeline, pump, check valve, valve and reservoir have been defined.

When running the simulation, the pressure head at indicated junctions (node) and flow rate at indicated pipe are both written to a text file at each time step. The file is saved to the working directory, and the user can read and print it at any time later. The

results of simulation can be visualized as a graph plotting the pressure head versus time at an indicated junction, and flow rate versus time for an indicated pipe.

The solution procedure adopted in the program does not make any allowance for water column separation, which occurs when the pressure in the pipeline reaches the saturated vapor pressure of the fluid. Therefore, care should be taken when dealing with negative pressures produced by the program. It should be noted that this program is to be used strictly for liquid simulation, because the compressibility of the fluid was not considered.

To run the *HydroAnalysis* program, the user will require a computer system running either Microsoft WINDOWS 95 or later, or WINDOWS NT 4.0 or later.

## **5.4.2 User Guide for Software Program Package *HydroAnalysis***

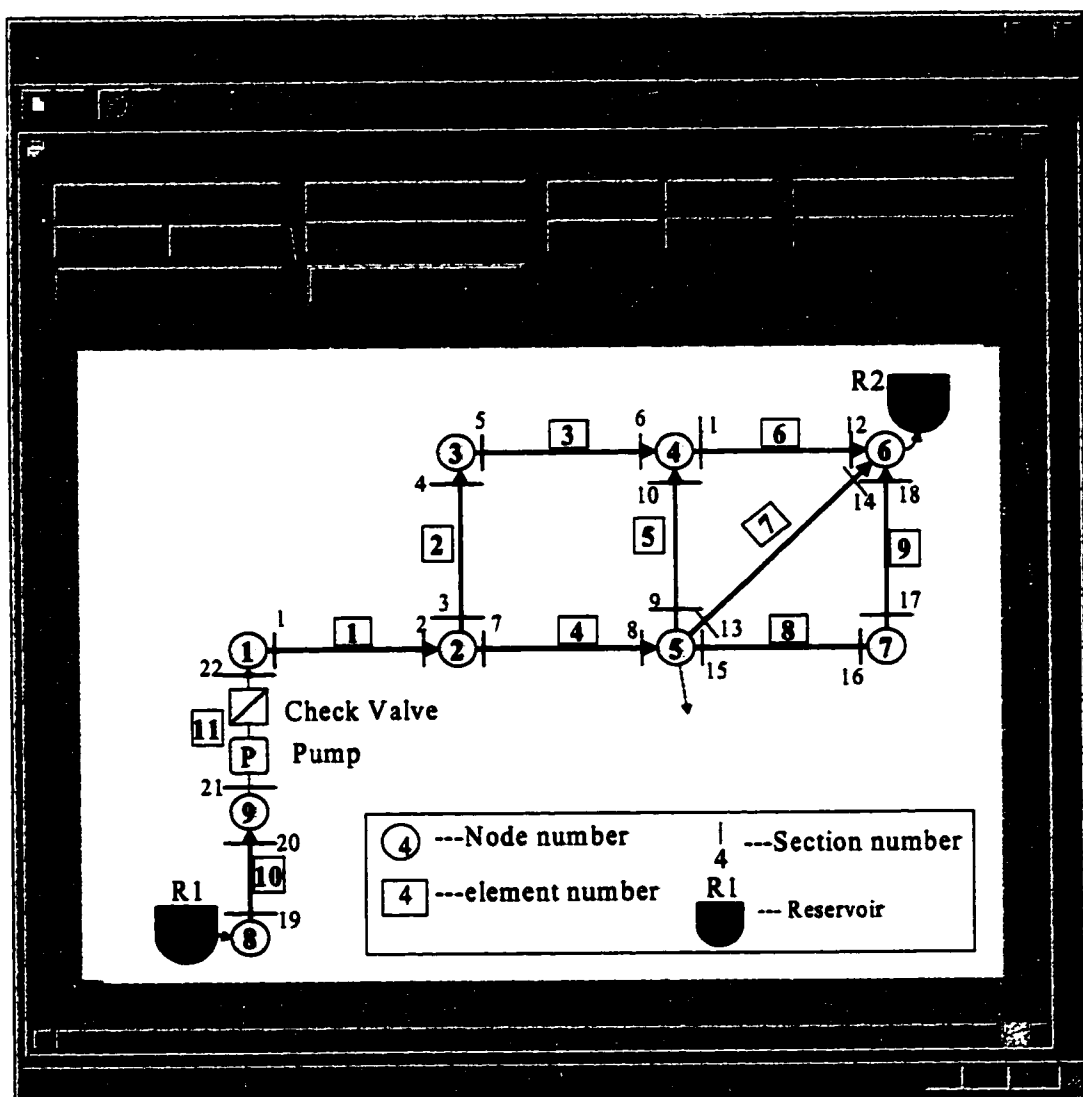
### **5.4.2.1 *Getting started***

To simulate a pipeline network with an check valve system, one needs only need to run the application *HydroAnalysis*. The *HydroGraphic* will be executed automatically during the running of *HydroAnalysis*. After starting the application *HydroAnalysis*, the main window appears (see Fig.5.4.2.1).

### **5.4.2.2 *Setting System Parameters***

#### **1. *Setting the Connection Parameters***

Under the main window, one must set the connection parameter first. Pressing the button *Connection Parameters*, the following dialog box is displayed (Fig. 5.4.2.2):



**Fig. 5.4.2.1 Main Window of HydroAnalysis**

Connection Parameters

10

9

1

62

2

1500

1	2	3	2	5	4	5	5	7	8
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

9	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

2	3	4	5	4	6	6	7	6	9
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

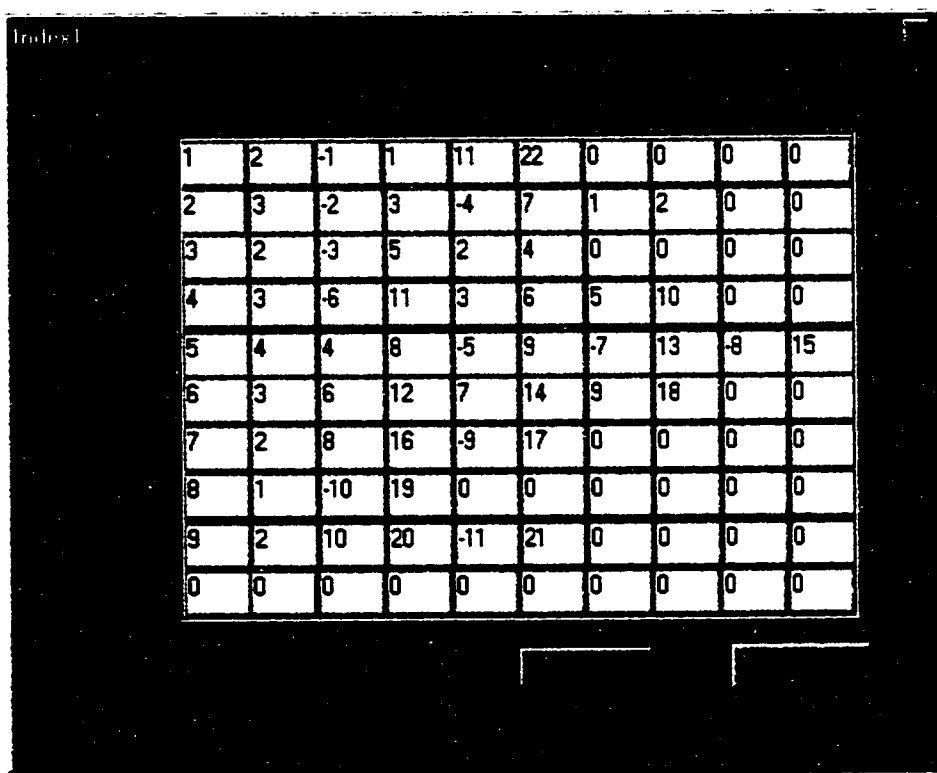
1	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

**Fig. 5.4.2.2 Dialog Box for Connection Parameters**

The pipeline network and check valve system is displayed as shown in the main window (Fig. 5.4.2.1). One may set the number of pipes, junctions, pumps index, and up and down junction numbers of pipe and pumps as shown in Dialog Box (Fig.5.4.2.2). One also must indicate the minimum reach number and sound speed. Normally, the minimum reach number is set to 2 or 3, and the sound speed is set between 1400 to 1550 m/s for water.

## 2. *Setting Index, Plot , Print and Simulation Parameters*

Referring to Section 4.2.2, the index for this system is shown in Fig. 5.4.2.3. It should be noted that the non-pipe elements should be numbered last. Any others pipe elements may be numbered by the user.



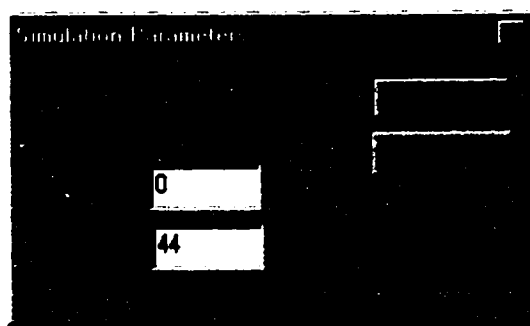
The image shows a screenshot of a software dialog box titled "Index". Inside the dialog box is a 10x10 grid of text boxes, each containing a number. The numbers are arranged in a specific pattern, likely representing an adjacency matrix or a similar system index. The numbers are as follows:

1	2	-1	1	11	22	0	0	0	0
2	3	-2	3	-4	7	1	2	0	0
3	2	-3	5	2	4	0	0	0	0
4	3	-6	11	3	6	5	10	0	0
5	4	4	8	-5	9	-7	13	-8	15
6	3	6	12	7	14	9	18	0	0
7	2	8	16	-9	17	0	0	0	0
8	1	-10	19	0	0	0	0	0	0
9	2	10	20	-11	21	0	0	0	0
0	0	0	0	0	0	0	0	0	0

**Fig. 5.4.2.3 Dialog Box of Indexing**

The user must also choose the duration of simulation. Fig. 5.4.2.4 shows the Dialog Box of Simulation Parameters. The user must define the duration of the simulation before the program will begin that simulation. It should be noted the solution procedure used

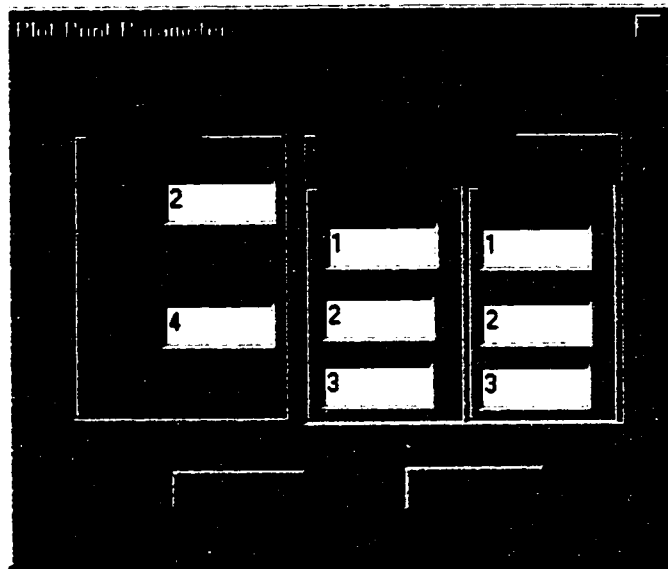
is based on the method of characteristics, and the computational time step is set at the time it takes for an sound wave to travel from one reach to the next along the pipe. The program will save three pressure head records and three flow rate records at each time step into file. The time duration cannot be set too large, because the size of the file depends on the duration of the simulation which is selected by the user.



**Fig. 5.4.2.4 Dialog Box of Simulation**

Fig 5.4.2.5 shows the Dialog Box of Plot Print parameters. The print and plot locations are to be set by the user. The program will plot only one junction location for head and one pipe location for flow rate, but will print (or save) a maximum of three locations for head and three pipe locations for flow rate.

After print out or display, if the user needs to print or display more locations, they can do so by returning to the main window and resetting the Plot and Print dialog box to the desired parameter.



**Fig. 5.4.2.5 Dialog Box of Plot & Print**

### **3. *Setting Pump and Check Valve Parameters***

The pump-check valve parameter dialog box is shown in Fig. 5.4.2.6. The user must set the following data:

- Check valve diameter: the user can set it by choosing from a list box. If the user checks the check box of *WithoutCheckValve*, this list box will be greyed.
- Set pump operation status: the user can choose from the four radio buttons in the box.
  - 1) If the user chooses pump start up, the first group edit box will appear, in which the user can set the pump start up data such as *Ts*, *Hs*, *A0*, *A1*, and *A2*. The other two boxes will be greyed. See Fig. 5.4.2.6 in the left column.

- 2) If the user chooses pump failure, the second group edit box will appear, and the other two will be greyed. The user can set Rated pump flow, head, torque, and speed. See the middle column of Fig. 5.4.2.6.
- 3) If the user chooses pump normal run, the last group edit box will appear, the user will only need to set *A0*, *A1*, and *A2* for this operation status. See the right column of Fig. 5.4.2.6.
- 4) If the user chooses the last radio button Pump Stop, all of these three groups are grayed, and the user is not required to set any data at all.

The screenshot shows a software interface titled "Pump Check Valve Parameters". It consists of three main vertical sections, each containing a set of input fields and checkboxes. The left section has a top field with "16" and a bottom section with fields containing "-1.0175" and "-0.8115". The middle section has a top field with "4" and a bottom section with fields containing "15", "45", "200", "24", and "25". The right section has a top field with "4" and a bottom section with fields containing "-1.2", "1", and "0.5". Each section also includes several empty input fields and checkboxes, some of which are disabled (greyed out).

**Fig. 5.4.2.6 Dialog Box for Pump Check Valve Parameters**

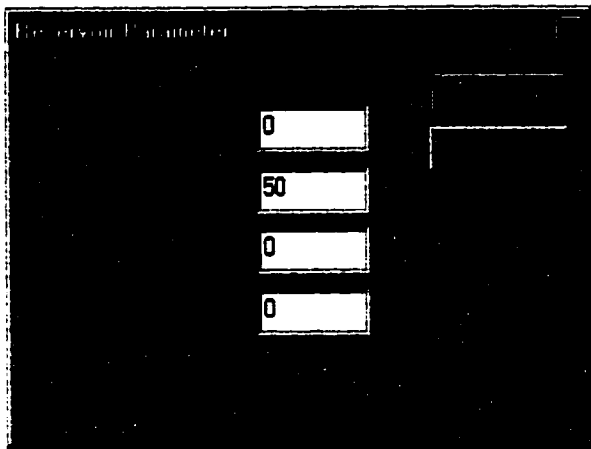
#### 4. *Setting Pipe, Junction (Node), Reservoir, and Control Valve Parameters*

The effect of transients in the pipeline will be different depending on the physical properties of the pipe, junction, reservoir, and valve. It is assumed at the start of the simulation that steady state conditions exist in the pipeline network system. Fig. 5.4.2.7, 5.4.2.8, and 5.4.2.9 show the pipeline, reservoir and junction parameters dialog box.

The image shows a software dialog box titled "Pipe Line Parameter". It contains five groups of input fields, each representing a different pipe segment. Each group has three vertically stacked input boxes. The first four groups each have two columns of three input boxes, while the fifth group at the bottom has only one column of three input boxes. The values entered in the input boxes are as follows:

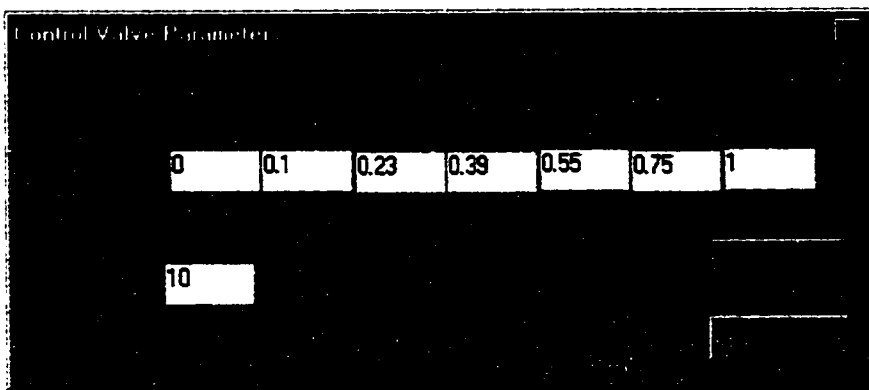
Segment	Column 1	Column 2
1	1000 0.5 0.02	2000 0.5 0.02
2	1500 0.5 0.02	2000 0.5 0.02
3	1000 0.5 0.02	2000 0.5 0.02
4	1500 0.5 0.02	2000 0.5 0.02
5	2000 0.5 0.02	

Fig. 5.4.2.7 Dialog Box of Pipeline Parameters



**Fig. 5.4.2.8 Dialog Box of Reservoir Parameters**

If the transients in the pipeline are caused by the control valve closure or opening, the valve opening and valve closure or open time can be defined by the user. The control valve parameters Dialog box is shown in Fig. 5.4.2.9.



**Fig. 5.4.2.9 Dialog Box of Control Valve Parameters**

The steady state of flow will be controlled by the reservoir head, the diameter, length and friction factor of the pipeline, the junction type, the head, flow, and elevation of junction, and the initial valve settings as showing in Fig. 5.4.2.10 .

Function Parameter

<input type="checkbox"/>	<input type="checkbox"/>
30	30
0	0
0	0

<input type="checkbox"/>	<input checked="" type="checkbox"/>
30	30
0	0
0	30

<input type="checkbox"/>	<input type="checkbox"/>
30	30
0	0
0	0

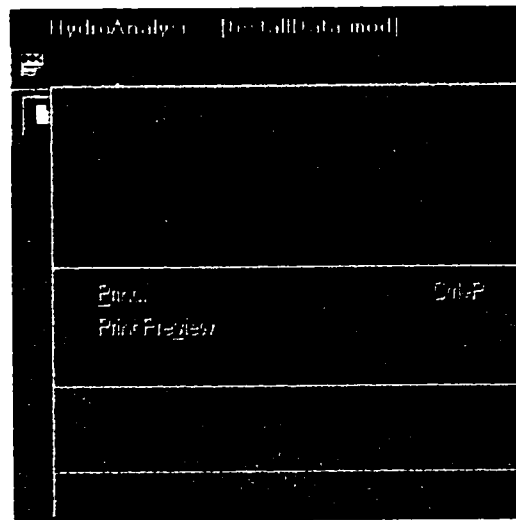
  

<input type="checkbox"/>	<input checked="" type="checkbox"/>
30	0
0	0
0	0

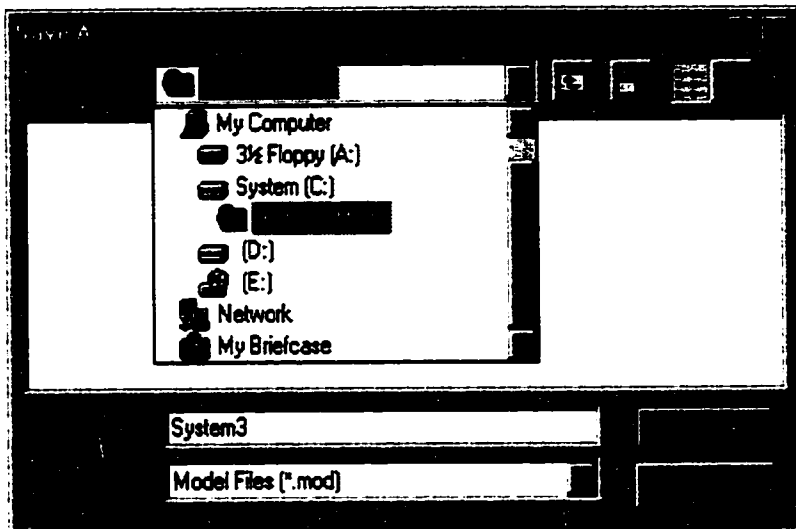
**Fig. 5.4.2.10 Dialog Box of Junction Parameters**

**5. Save Model to a file, open a model from previous file**

After all of the parameters are set by the user, it is recommended that the model be saved to a file for later repeated use. At the main window menu, under *File*, *Save Model As...*, the *Save As* window appears, allowing the user to save the model at any directory indicated by the user, by entering the file name and then saving. The user is not required to give the file extension, because the system will add the *mod* extension automatically. Please refer to Fig. 5.4.2.11 – 5.4.2.12.



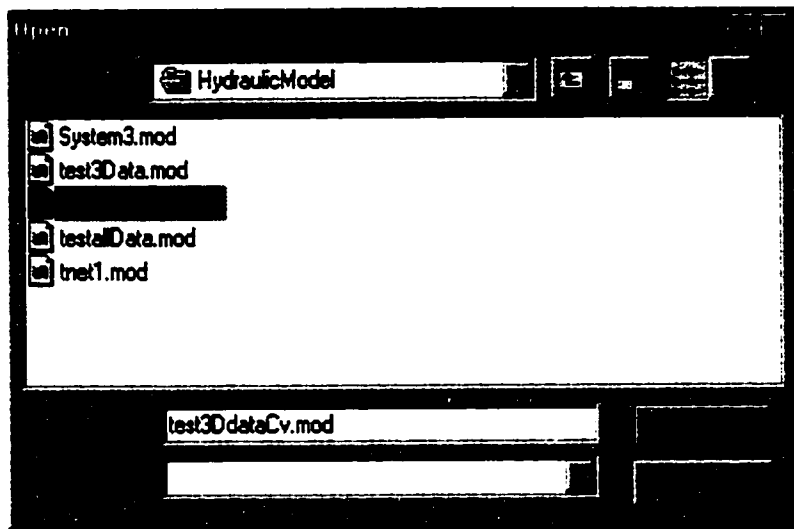
**Fig. 5.4.2.11 Menu File Window**



**Fig. 5.4.2.12 Save Model As Window**

After the user saves the model as a model file, pressing the *Start* button under the main window begins the simulation. Next time, if the user wishes to run a simulation, he or she can start with a *New* or *Open* the model file. Referring to Fig. 5.4.2.11, under the File menu, Open Model, the following Open window will display (Fig. 5.4.2.13).

It should be remembered that initial data must be set by the user before a simulation can be started. Therefore, the *save* function of this program is very useful for users who intend to do multiple simulations of the same system, and the user is only required to open the exiting model.

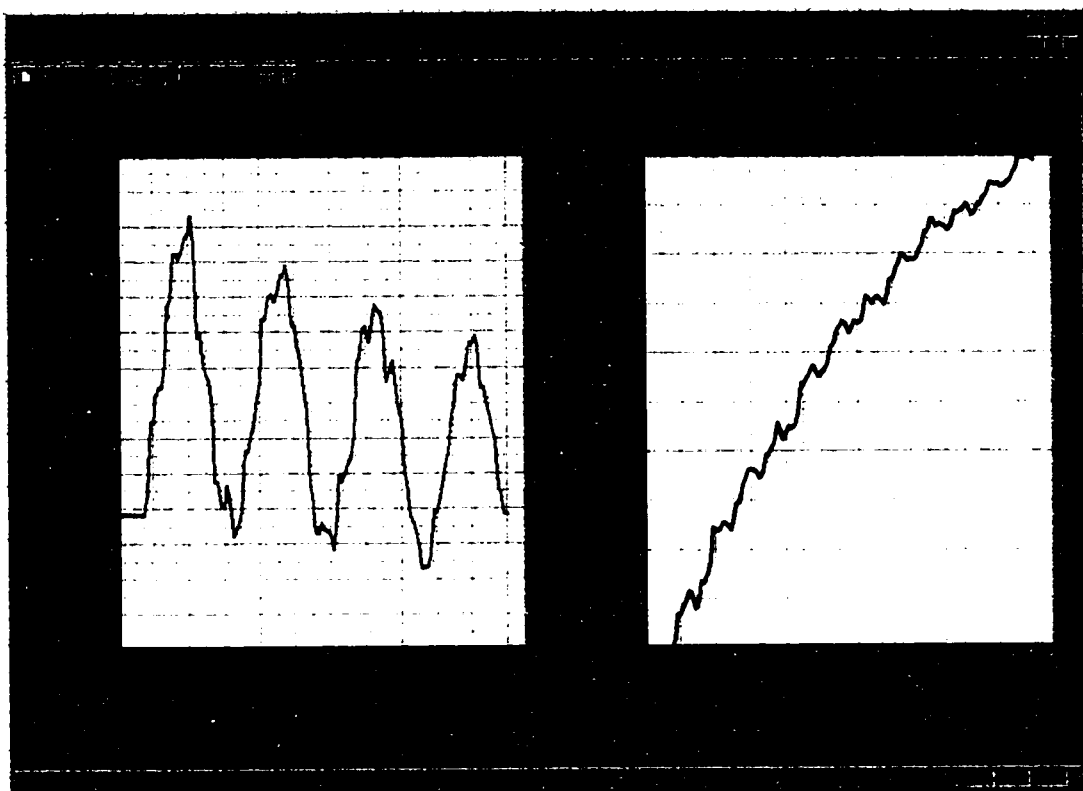


**Fig. 5.4.2.13 Open Model Window**

#### **5.4.2.3 Getting Simulation Output**

After the user finishes inputting all of the data settings and presses the **START** button under the main window of *HydroAnalysis*, the *HydroGraphic* page will display. There are two plots displayed in this page (Fig. 5.4.2.14). One graphs the Head (Pressure) versus Time, and the other graphs Flow Rate (Discharge) versus Time.

In the Head (Pressure) Vs Time Plot for this case, it is shown that this plot is for Pump start, with check valve at the location of junction (Node) number 5. The Flow Rate (Discharge) Vs Time plot shows that the curve is at the location of Pipe Number 4, pump start, and with the check valve. The simulation results of simple pipe with pump failure will be shown in Appendix C. Simulation results of other example pipeline network systems will also be illustrated in this Appendix.



**Fig. 5.4.2.14 HydroGraphic Main Page (With Check Valve)**

The user can change display characteristics of the plot such as the line type or line attributes by double-clicking the plot field under the main page. The plot parameters window will appear. The plot data can also be displayed on the screen by clicking the *Data* Button from the plot parameters window, a list of data will display on the screen. The user can then copy the resulting plot data to an Excel, Word or Text file, for further analysis. (Fig. 5.4.2.15).

Head - Time			FlowRate - Time		
#	Time	Head	#	Time	FlowRate
0	0	0	0	0	1.551e-09
1	0.111111	0	1	0.1111	4.379e-09
2	0.222222	0	2	0.2222	4.379e-09
3	0.333333	1	3	0.3333	0
4	0.444444	0	4	0.4444	0
5	0.555556	0	5	0.5556	0
6	0.666667	0	6	0.6667	0
7	0.777778	0	7	0.7778	2.149e-09
8	0.888889	0	8	0.8889	2.149e-09
9	0.999999	0	9	0.9999	2.149e-09
10	1.111111	30.5011	10	1.1111	1.707e-03
11	1.222222	12.0734	11	1.2222	5.126e-03
12	1.333333	12.0734	12	1.3333	5.126e-03
13	1.444444	12.6648	13	1.4444	6.345e-03
14	1.555556	12.8337	14	1.5556	7.72e-03
15	1.666667	12.8337	15	1.6667	9.186e-03
16	1.777778	13.1474	16	1.7778	8.441e-03
17	1.888889	14.6731	17	1.8889	6.302e-03
18	1.999999	14.8405	18	1.9999	7.738e-03
19	2.111111	35.8291	19	2.1111	1.071e-02
20	2.222222	35.8739	20	2.2222	1.05e-02
21	2.333333	35.7541	21	2.3333	1.193e-02
22	2.444444	35.9745	22	2.4444	1.402e-02

**Fig. 5.4.2.15 Display of the Plot Data**

The system saves Head (Pressure) and Flow Rate (Discharge) of a maximum of three locations automatically. The user can open the file containing this data under the default directory by using NotePad or WordPad. The user can also plot other locations of this system by closing the Main Page, returning back to the Main Window, changing the Plot Parameters, and restarting the simulation.

In this chapter, analyses of the problem, input, and output were first introduced; next, the *Hydro-Analysis* and *Hydro-Graphic* software were implemented by using algorithms, hierarchical diagrams, and description of major classes. A users guide was provided, giving detailed demonstrations for this software. This software creates an effective simulation for hydraulic transients in pipelines with check valve systems. It has a user-friendly graphic interface and allows a user plenty of choices for convenient simulation and analysis of pipeline systems.

## Chapter 6

### Conclusion

#### 6.1 Summary

The purpose of this thesis was to design and implement a software package to model and simulate pipeline networks including check valve systems. This objective was achieved by analyzing hydraulic transients for pipeline network systems, analyzing check valve dynamics, and implementing the software package which combines the pipelines with check valves.

After introducing the continuity and motion equations, Chapter 2 obtained the *Hydraulic-Grade-Line* form of the equations of motion and continuity by using some assumptions and simplifications. The equations are most suitable to the hydraulic transients analysis for pipeline network systems, but are restricted to less compressible fluids flowing at a low velocity.

The method of analysis of hydraulic transient flow in pipeline systems started with the equations of motion, continuity and other physical property relationships. From these basic equations, different methods employing different restrictive assumptions have evolved. The characteristics method was analyzed and compared to the lumped parameter method in the time domain, the four-pole equations and the transmission line modeling method in the frequency domain.

The method of characteristics has several advantages over other methods that are particularly relevant for hydraulic transients problems. These include an explicit solution

and a procedure that is relatively simple with approximations. Therefore, it is particularly suitable for the analysis of systems with complex boundary conditions.

Besides using the basic method of characteristics in this study, some improvement has been added to this method. The Finite Difference Formulations and Algebraic Representation were combined to the method of characteristics. During the implementation of the integrating characteristic equation (2.4.7) and (2.4.9), the trapezoidal rule was introduced for maintaining the linear form of the integrated equations, at the same time keeping the second-order accuracy. At the end of this chapter, the introduction of the pump characteristic and the derivations of the boundary conditions for pump failure and pump start up were included.

In Chapter 3, the check valve dynamic equation was investigated for the analysis of pipeline transients. The torque due to the weight of the rotating disk,  $T_w$ , the bearing friction torque,  $T_b$ , the counterweight torque,  $T_{cw}$ , the flow torque,  $T_F$ , and the spring torque  $T_S$ , were included in the dynamical equation. The flow torque was included in the valve analysis, in which the orifice sequence model was introduced, then determined the pressure drop for forward and reverse flow.

An important issue was to determine the coefficient of resistance of a check valve. The coefficient of resistance of the check valve is a function of the valve geometry and disc opening angle. During the occurrence of transient, the coefficient of resistance is calculated at each time step. Finally, the check valve boundary condition was introduced.

Chapter 4 presented the numerical model used to simulate pipeline network transients including the check valve. This included the pipeline system model, check valve dynamics model, and the pump characteristic model.

The pipeline system model was developed based on the method of characteristics which was investigated in Chapter 2. The solution was algebraic. The use of continuous data storage, connection index vector, and a flowchart of the simulated procedure were introduced.

The advantages of algebraic representations over the normal characteristic equations can be summarized as:

- They are normally applied over several reaches, with no need to calculate the transients at the intervening section; they are therefore a computationally efficient solution for multipipe systems such as networks.
- They use a small  $\Delta t = L/(N\alpha)$ , so that the boundary condition detail is preserved.

The variables  $H$  and  $Q$  of each time step are continually stored by using  $K_M$  ( $K_M = L/(a \Delta t) + 1$ ) which is the maximum size in the system.  $K$  which is used as a counter for each time step, and  $Kp$  which is used as a pointer to identify the position in the vector  $H$  or  $Q$  at which current values of the variables are stored. At each iteration when  $K$  is incremented, so is  $Kp$ , the value of the pointer. When  $Kp$  exceeds  $K_M$ , the value  $Kp$  is set back to 0, so that it points to the first position in the vector. This procedure will save much storage space during computation.

The system topology must be stored before starting the simulation. An indexing method was introduced which offers one form of bookkeeping to describe the system connection. It was oriented around the nodes of the system and, for each node, the following information was given: the node number, the element number (sign) attached to the node, and the element section number adjacent to node.

The check valve dynamic system model was represented by a second order, nonlinear, differential equation. This equation can be solved using the fourth order Runge-Kutta algorithm. It must be solved simultaneously with the pipeline simulation or pump dynamic simulation. Check valve dynamics need torque  $T_0$  and flow rate  $Q_v$  as initial conditions. At the end of each time step of simulation, the check valve disc angle  $\theta$  and disc angular velocity  $\omega$  is obtained. Then, from the disc information, the check valve coefficient of resistance  $K$ , check valve pressure drop  $\Delta P$ , and flow rate  $Q_v$  are calculated. The flow torque  $T_0$  is calculated by  $Q_v$  and  $K$ . The above procedure is illustrated by a flowchart in Fig.4.3.1.

The pump characteristic model includes the head balance and speed change equations. These equations can be solved simultaneously for  $\alpha$  and  $v$  using Newton-Raphson method. The flowchart of this solution was illustrated in Fig.4.4.1.

Chapter 5 described the implementation of the dynamic model of pipeline network systems with check valves into a working simulation program. This program includes three parts: Hydro-Analysis, User Graphic Input, and User Graphic Output, which were involved in two application. One was the *HydroAnalysis* application (Hydro-Analysis and User Graphic Input), and the other was the *HydroGraphic* (User Graphic Output).

An overview of the program was given including the analysis and the design of the *Hydro-Analysis* and *Hydro-Graphic* applications. The analysis application consisted of problem description, input and output requirement. The design was focused on the algorithm, the hierarchical diagram, major class descriptions, and their inter-relationship.

The user interface is essential in achieving the goals of this work. The Microsoft Windows Graphic User Interface was used. The user could easily set and change all of the parameters describing the pipeline network and check valve system through a window dialog box, button control, edit box, and other standard interface methods. This application simulates hydraulic transients caused by pump failure, pump start up, or control valve operation (closing, opening). It can also be used in the system with or without the check valve. The results of simulation can be visualized as a plot in which pressure (head) Vs. time at indicated junction and flow rate Vs. time at indicated pipe. The indicated pressure and flow rate can also be stored in text file which can be printed or read at any time later.

In conclusion, the contributions of this thesis are:

- Provided a hydraulic transient simulation package under the window's environment with graphics user interface for ease of use.
- Included pump dynamics, and check valve dynamics in the pipeline transient analysis.
- Improved algebraic representation and continuous data storage to the method of characteristics.

## ***6.2 Suggestions for Future Work***

The program *HydroAnalysis* is only the first step towards the simulation of pipeline systems with check valves. The solution procedure adopted in the program did not consider the problem of water column separation. Although this only occurs when the

pressure in the pipeline reaches the saturated vapor pressure of the fluid, it should be considered in the method of characteristics. Streeter and Wylie [7] introduced a procedure in which the development, growth, and collapse of a cavity were considered as an internal boundary condition. The pressure of column separation computed by using this approach was, however, found to be higher than those measured on a prototype. Therefore, there remains more work on this topic, especially in program implementation.

An experimental analysis of the Orifice Sequence Model for the check valve dynamic analysis is required. Experimental tuning is also necessary for program *HydroAnalysis*. Since all concepts and most algorithms are only considered for the incompressible flows, compressibility will need to be integrated into the dynamic and transients flow model in the future.

The user interfaces in this application only meet the basic requirement for this project. Throughout this thesis, great effort was made to fulfill basic usable requirements, but there remains room for a great deal more work in this area. At present, an index vector has to be input through the window interface point by point. The complexity of the pipeline networks system is limited by the size of arrays which were used for parameters input of pipes, junctions, pumps, and valves. A well defined database should be created to store the pipe parameters, pump characteristic data, and check valve parameters. A graphic user input should be introduced for the pipeline topology data, so that the user would need only to drag in, and connect each element, and input parameters graphically. Finally, the graphic output should be improved to display and store more plots as freely chosen by the user.

## REFERENCES

1. Wood, F. M., "*History of Water Hammer*", Report No. 65, Department of Civil Engineering, Queen's University at Kingston, Ontario, Canada, April 1970.
2. Rich, G., "*Hydraulic Transients*", McGraw-Hill Book Co., New York, 1951
3. Parmakian, J., "*Water Hammer Analysis*", Prentice-Hall, Inc., Englewood Cliffs, N.J., 1955.
4. Street, V.L., and Wylie, E. B., "*Hydraulic Transients*", McGraw-Hill Book Co., New York, 1967.
5. Fox, J. A., "*Hydraulic Analysis of Unsteady Flow in Pipe Networks*", John Wiley & Sons, Inc., New York, 1977.
6. Watters, G. Z., "*Modern Analysis and Control of Unsteady Flow in Pipelines*", Ann Arbor Science Publications, Ann Arbor, MI, 1979; 2<sup>nd</sup> ed., Butterworth, Boston, London, 1983
7. Wylie, E. B and Streeter, V. L., "*Fluid Transients*", McGraw-Hill Book Co., New York, 1983.
8. Chaudhry, M. H., "*Applied Hydraulic Analysis*", 2<sup>nd</sup> ed., Van Nostrand Reinhold Co., Princeton, N. J., 1987.
9. Fox, J. A., "*Transient Flow in Pipes, Open Channels and Sewers*", John Wiley & Sons, Inc., New York, 1989.

10. Sharp, B. B. and Sharp D.B., "*Water Hammer: practical solutions*", Arnold Publishers, London, 1996.
11. Pool, E. B., Polwit, A. J. and Carlton, J. L., "*Prediction of Surge Pressure from Check Valve for Nuclear Loops*" Paper 62-WA-219, Amer. Soc. of Mech. Engrs, October 1962.
12. Csemniczky, J., "*Hydraulic Investigations of the Check Valves and Bufferfly Valves*", Proceedings of the Fourth Conference on Fluid Machinery, AKADEMIA, 1972, PP. 293-305.
13. Hong, H. and Svoboda, J., "*Model of Swing-disc Check Valve Suitable for Lumped Parameter Piping Network Simulation*", Report, Concordia University, Montreal, Quebec, Canada, 1981.
14. Hong, H., "*The Design of Wafer Swing-Disc Check Valves for Optimal Performance*", Thesis of Master of Engineering in the Faculty of Engineering, Concordia University, Montreal, Canada, 1983.
15. Cavazoni E. M., "*Design Optimization and Performance Evaluation of a Gravity Check Valve for Gas Service*", Thesis of Master of Engineering In the Department of Mechanical Engineering, Concordia University, Montreal, Quebec, Canada, 1984.
16. Provoost, G.A., "*A Critical Analysis to Determine Dynamic Characteristics of Non-Return Valve*", 4<sup>th</sup> Int. Conf. On Pressure Surges, Bath, 275-286, 1983.
17. Thorley, A.R.D. "*Check Valves Behavior under Transient Flow Conditions, a State-of-Art Review*", Trans. A.S.M.E. J Fluids Engineering 111 178-183, 1987.

18. Sidi Ould Sadfa, "*Simulation for the Dynamic Response of a Transient Hydraulic System*", Project report, Dept. of Mechanical Engineering, Aerospace Engineering, Concordia University, 1998
19. "*Simulation of Water Hammer in Pipes with HAMMER*", Version 2.0  
<http://www.civil.bee.qut.edu.au/projects/hammer.html>
20. "*Faast Software for fluid transients*", <http://www.faast.com/>
21. Thorley, A.R.D., "*Dynamic Response of Check Valves*", 4<sup>th</sup> international Conference On Pressure Surges, Bath, 231-242, 1983.
22. V. L. Streeter, and E. B. Wylie, "*Fluid Mechanics*", 8<sup>th</sup> ed., McGraw-Hill, New York, 1985.
23. A. R. D. Thorley, "*Fluid Transients in pipeline Systems*", D. & L. George, Herts, England, 1991.
24. A. R. D. Thotley, and J. H. Oei, "*Dynamic Behavior of a Swing Check Valve*", Proc. 5<sup>th</sup> int. Symp. Water Column Separation, IAHR, Obernach, Germany, Sept. 1981
25. M. Suda, "*Simulation of Valve Closure after Pump Failure in Pipeline*", Journal of Hydraulic Engineering, Vol. 117, No. 3, March 1991
26. Krasimir P. Ivanov and Emil G. Bournaski, "*Combined Distributed and Lumped Parameters Model for Transient Flow Analysis in Complex Pipe Networks*", Comput. Methods in Appl. Mech. Engrg. Vol. 130, 1996
27. B. Camahan, H. A. Luther, and J. O. Wilkes, "*Applied Numerical Methods*", John Wiley & Sons. Inc., New York, 1969.

28. Shoichiro Nakamura, *"Applied Numerical Methods with Software"*, prentice Hall, Inc. New Jersey, 1991.
29. David Budgen, *"Software Design"*, Addison-Wesley, 1994
30. B. W. Karney and D. McInnis, *"Efficient Calculation of transient Flow in Simple Pipe Networks"*, Journal of Hydraulic Engineering, Vol. 118, No. 7, July 1992
31. M. S. Ghidaoui and B. W. Karney, *"Equivalent Differential Equations in Fixed-Grid Characteristics Method"*, Journal of Hydraulic Engineering, Vol. 120, No. 10, October 1994
32. D. A. McInnis, B. W. Karney and D. H. Axworthy, *"Efficient valve Representation in Fixed-Grid Characteristics Method"*, Journal of Hydraulic Engineering, Vol. 123, No. 8, August 1997
33. B. W. Karney and M. S. Ghidoui, *"Flexible Discretization Algorithm for Fixed-Grid MOC in Pipelines"*, Journal of Hydraulic Engineering, Vol. 123, No. 11, November 1997
34. P. Krus, K. Weddfelt and J. O. Palmberg, *"Fast pipeline Models for Simulation of Hydraulic Systems"*, Journal of Dynamic Systems, Measurement, and Control, Vol. 116, March 1994
35. T. Chen and R. F. Boucher, *"Junction Characteristics of Energy Wave Methods"*, Proc Instn. Mech. Engrs, Vol. 211 Part C, 1997
36. A. S. Elansary, W. Silva and M. H. Chaudhry, *"Numerical and Experimental Investigation of Transient Pipe Flow"*, Journal of Hydraulic Research, Vol. 32, No. 5, 1994

37. E. E. Kitsios and R. F. Boucher, "*Transmission Line Modeling of a Hydraulic Position Control System*", Proc Instn. Mech. Engrs, Vol. 200 No. B4, 1986
38. R. F. Boucher and E. E. Kitsios, "*Simulation of Fluid Network Dynamics by Transmission Line Modeling*", Proc Instn. Mech. Engrs, Vol. 200 No. C1, 1986
39. S. M. Beck, H. Haider and R. F. Boucher, "*Transmission Line Modeling of Simulated Drill Strings Undergoing Water hammer*", Proc Instn. Mech. Engrs, Vol. 209, Part C: Journal of Mechanical Engineering Science, 1995

# Appendix A   Review of Methods of Transients Analysis in Pipelines

## A 1. Introduction

There are two kinds of unsteady flow in pipes. One is the *transient state* flow that represents the intermediate-flow condition when flow is changed from one steady state to another. The other is the *oscillatory flow* that may be developed into a *resonance* which depends on the characteristics of the piping system and of the excitation.

## A 2 Methods for Analyzing the Oscillatory Flow

Oscillatory flow may be analyzed either in the time domain or in the frequency domain.

In the time domain, one can use the Method of Characteristics. Using this method, the partial differential equations describing the unsteady flow are converted into ordinary differential equations and then solved. Analysis of the oscillatory flow by this method assumes that the initial steady-state discharge and pressure head in the piping system are equal to their mean values or equal to zero-flow conditions. The specified forcing function is then imposed as a boundary condition, and the system is analyzed by considering one frequency at a time. The disadvantage of this method is that it requires a considerable amount of computer time. The main advantage is that the nonlinear relationships can be included in the analyses.

In the frequency domain, there are two methods of analysis available:

### *1. Impedance Method*

In this method, the *Terminal Impedance*,  $Z_s$ , which is the ratio of the oscillatory pressure head to the discharge, is computed by using the known boundary conditions. An impedance diagram between  $\omega_f$  and  $|Z_f|$  is plotted. The frequencies at which  $|Z_f|$  is maximum are the resonant frequencies of the system.

Because of the lengthy algebraic equations involved, the method is suitable for digital computer analysis only.

### *2. Transfer Matrix Method*

The second method for doing frequency analysis is the transfer matrix method. It has been used for analyzing structural, mechanical vibrations and electrical systems. The method is based on the linearized equations and on sinusoidal flow and pressure fluctuations.

The transfer matrix method is simpler and more systematic than the impedance method. This method is suitable for both hand and digital computations.

## **A 3 Method for Analyzing Transient State Flow**

All methods of analysis of transient flow in pipes start with the equation of motion, continuity, or energy and equations of state and other physical property relationships.

From these basic equations several different methods, each employing different restrictive assumptions, have evolved.

## **Time Domain**

### **1. *Method of characteristics* [7, 31, 32, 33]**

The characteristic method of analysis transforms the partial differential equations of motion and continuity into ordinary differential equations. It has several advantages over other methods that are particularly relevant in water hammer type problems. These include a firmly established stability criterion, an explicit solution so that different elements physically removed from one another in systems are handled independently, and a procedure that is relatively simple for approximations. Therefore, the method is particularly suitable for the analysis of systems having complex boundary conditions.

Its primary disadvantage is the requirement of strict adherence to the time step-distance interval relationship. The method of characteristics has been discussed in detail in Section 2.4.

### **2. *Lumped Parameter Method* [14, 26]**

In the lumped-parameter method, the pipe is divided into  $n$ -sections of equal length, where the fluid inertia, fluid capacitance and pipe elasticity and shear stress between the fluid and pipe walls are lumped. Fig. A 3.1 is a circuit representation of one section of a pipeline. Between each consecutive node a lumped resistance  $R$  and fluid inertance  $I$  represent, respectively, the friction losses and the fluid inertia in the

corresponding sections. At each interior node a capacitance  $C$  connects the node to ground.  $C$  represents the sum of half the capacitance of adjacent lumps.

With reference to Fig. A 1, the coupled non-linear differential equations relating pressure head  $H$  at each interior node to the flow rate  $Q$  through each section, are given by:

$$\frac{dH_i}{dt} = \frac{1}{C}(Q_{i-1} - Q_i) \quad i = 2, 3, \dots, n \quad (\text{A.1})$$

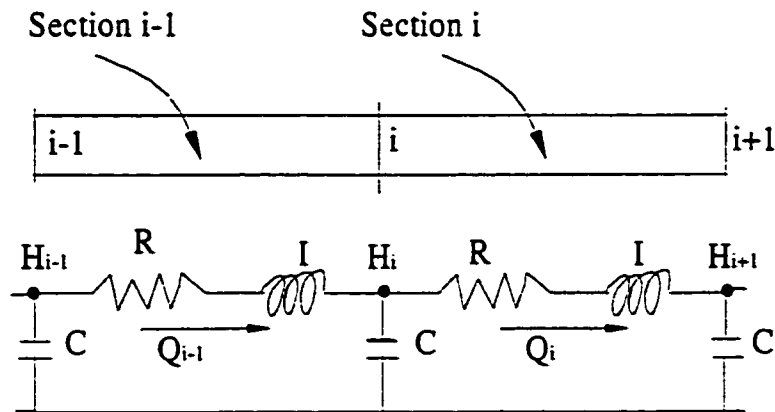
$$\frac{dQ_i}{dt} = \frac{1}{I}(H_i - H_{i+1} - RQ_i^2) \quad i = 1, 2, 3, \dots, n \quad (\text{A.2})$$

In which [14]

$$R = \frac{fL}{2gd_i A_i^2 n} \quad (\text{A.3})$$

$$I = \frac{L}{gA_i n} \quad (\text{A.4})$$

$$C = \frac{gA_i L}{a^2 n} \quad (\text{A.5})$$



**Fig. A 1 Circuit Represent of Lumped parameter method**

where,  $f$  is the Darcy-Weisbach friction factor,  $L$  is the length of the pipe,  $d_i$  is the pipe diameter,  $A_i$  is the pipe area,  $n$  is the section amount,  $a$  is the pressure wave propagation speed for the fluid.

The lumped-parameter method describes the system pressures and flows by coupled non-linear differential equations. This method may be satisfactory at very low frequencies and with a large enough  $n$ , but it requires excessive amounts of computer time.

## Frequency Domain

### 3. Four-pole Equations Modeling Method [6, 34, 35]

In order to derive the four-pole equations, the continuity and motion equations can be solved by using Laplace transformation [22, 23]. One can obtain

$$\frac{As}{E}P(x,s) + \frac{dQ(x,s)}{dx} = 0 \quad (\text{A.6})$$

$$\frac{dP(x,s)}{dx} + \frac{\rho s}{A}Q(x,s)N(s) = 0 \quad (\text{A.7})$$

In which  $E$  is the bulk Modulus for the fluid and

$$N(s) = -\frac{J_0(\gamma R)}{J_2(\gamma R)} = \frac{\alpha}{s} \cdot \frac{\sum_{k=0}^{\infty} \frac{2^k}{(k!)^2} \left(\frac{s}{\alpha}\right)^k}{\sum_{k=0}^{\infty} \frac{2^{k+1}}{k!(k+2)!} \left(\frac{s}{\alpha}\right)^k}$$

$$= \frac{\alpha}{s} + \frac{4}{3} - \frac{s}{18\alpha} + \frac{2s^2}{135\alpha^3} - \dots$$

Where,  $J_n(\gamma R)$  is called Bessel functions of order n.

$$J_n(\gamma R) = \left(\frac{\gamma R}{2}\right)^n \sum_{k=0}^{\infty} \frac{(-\gamma^2 R^2 / 4)^k}{k!(n+k)!}$$

And

$$\gamma^2 = -\frac{s}{\nu} \quad \text{or} \quad \gamma = j\sqrt{\frac{s}{\nu}}$$

where,  $\nu$  is the viscosity.

$$\alpha = \frac{8\nu}{R^2} = \frac{32\nu}{D^2}$$

where,  $\alpha$  is called “viscosity factor”.

An approximation over the whole frequency range is made, especially for the

$$|s| = \omega < 300\alpha$$

$$N(s) = \frac{\alpha}{s} + 1 + \frac{\frac{0.5}{3.3}}{1 + \frac{s}{3.3 \cdot \alpha}} + \frac{\frac{4.05}{25}}{1 + \frac{s}{25 \cdot \alpha}} + \frac{\frac{20}{1000}}{1 + \frac{s}{1000 \cdot \alpha}} \quad (\text{A.8})$$

We introduce the *characteristic impedance*  $Z_c$  and the *sonic velocity*  $a$

$$Z_c = \frac{\rho a}{A} \quad a = \sqrt{\frac{E}{\rho}} \quad N = N(s)$$

The equation (A.2) and (A.3) can be rewritten as the following:

$$P = -\frac{aZ_c}{s} \frac{dQ}{dx} \quad (\text{A.9})$$

$$\frac{dP}{dx} = -\frac{Z_c s}{a} \cdot Q \cdot N \quad (\text{A.10})$$

Elimination of the pressure P from the above

$$\frac{d^2 Q}{dx^2} - \frac{s^2}{a^2} \cdot Q \cdot N = 0 \quad (\text{A.11})$$

The solution of this second-order differential equation

$$Q(x, s) = C_1 e^{\frac{x\sqrt{N}}{a}} + C_2 e^{-\frac{x\sqrt{N}}{a}}$$

$$P(x, s) = -Z_c \sqrt{N} \left[ C_1 e^{\frac{x\sqrt{N}}{a}} - C_2 e^{-\frac{x\sqrt{N}}{a}} \right] \quad (\text{A.12})$$

when  $x=0$ , at the input end of the pipe,  $P = P_i$  and  $Q = Q_i$ , so:

$$C_1 = \frac{1}{2} \left( Q_i - \frac{P_i}{Z_c \sqrt{N}} \right) \quad C_2 = \frac{1}{2} \left( Q_i + \frac{P_i}{Z_c \sqrt{N}} \right) \quad (\text{A.13})$$

Letting  $P = P_o$  and  $Q = Q_o$  at  $x = L$ , the output end of the pipe and introducing the wave propagation time along the pipe line

$$T = \frac{L}{a}$$

The *four-pole equations* are obtained as follows:

$$Q_o = \cosh Ts \sqrt{N} \cdot Q_i - \frac{1}{Z_c \sqrt{N}} \sinh Ts \sqrt{N} \cdot P_i = A_L Q_i + B_L P_i$$

$$P_o = -Z_c \sqrt{N} \cdot \sinh Ts \sqrt{N} \cdot Q_i + \cosh Ts \sqrt{N} \cdot P_i = C_L Q_i + D_L P_i \quad (\text{A.14})$$

We can write above equation in the form of matrix

$$\begin{bmatrix} Q_o \\ P_o \end{bmatrix} = \begin{bmatrix} \cosh Ts \sqrt{N} & -\frac{1}{Z_c \sqrt{N}} \sinh Ts \sqrt{N} \\ -Z_c \sqrt{N} \sinh Ts \sqrt{N} & \cosh Ts \sqrt{N} \end{bmatrix} \cdot \begin{bmatrix} Q_i \\ P_i \end{bmatrix} = \begin{bmatrix} A_L & B_L \\ C_L & D_L \end{bmatrix} \cdot \begin{bmatrix} Q_i \\ P_i \end{bmatrix} \quad (\text{A.15})$$

The block diagram shown in Fig. A.2 can be used to represent the dynamic behavior of one single pipeline.

The four-pole equation provides a practical solution in the frequency domain. It is very useful for the analysis, synthesis and design of hydraulic pipelines.

Solutions in the time domain can be found theoretically. By inverse Laplace transformation, the frequency domain can be transformed into the time domain. However, this procedure turns out to be extremely complicated. The main difficulty comes from the frequency dependent friction factor  $N$ .

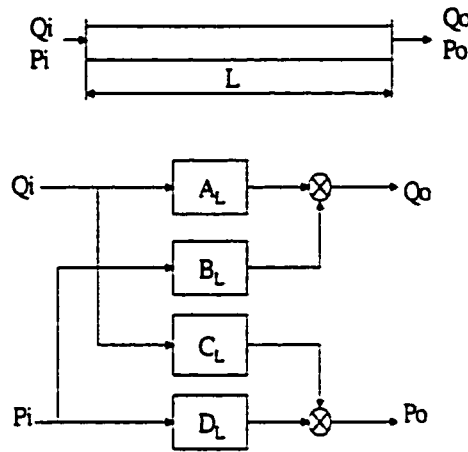


Fig A.2 Block Diagram for Single Pipeline

The use of four-pole equations may be considered for simulation. Using Simulink toolbox in Matlab, one may build block diagram for the single simple pipeline; however, for large and complex pipeline systems, it remains difficult.

#### 4. *Transmission Line Modeling Method* [37, 38, 39]

Transmission Line Modeling (TLM) is essentially a time-delay method, borrowing its main concepts and fundamentals of its computational solution scheme from the field of electrical power lines. In its elementary form, a fluid network is treated as a set of pipes

(or pipe segment) where waves travel with a pure time delay. Connecting the pipes are junctions of various types at which the waves are scattered (transmitted, reflected and /or attenuated).

Similar to the four-pole equation system analysis, The transmission line equation are obtained as follows:

$$\begin{bmatrix} P_2 \\ m_2 \end{bmatrix} = \begin{bmatrix} \cosh TD & -Z_c \sinh TD \\ -\frac{1}{Z_c} \sinh TD & \cosh TD \end{bmatrix} \bullet \begin{bmatrix} P_1 \\ m_1 \end{bmatrix} \quad (\text{A.16})$$

where  $P$  and  $m$  are pressure and mass flow at stations 1 (upstream) and 2 (downstream),  $T$  is the acoustic time delay between stations,  $Z_c$  is the characteristic impedance and  $D$  is the  $D$  operator i.e.,  $e^{TD} F(t) = F(t + T)$ .

Equation (A.16) is rearranged as:

$$P_2 \sqrt{\frac{2}{Z_c}} = \frac{1}{\sqrt{2}} \left[ \frac{P_1}{\sqrt{Z_c}} + m_1 \sqrt{Z_c} \right] e^{-TD} + \frac{1}{\sqrt{2}} \left[ \frac{P_1}{\sqrt{Z_c}} - m_1 \sqrt{Z_c} \right] e^{-TD} \quad (\text{A.17})$$

$$m_2 \sqrt{2Z_c} = \frac{1}{\sqrt{2}} \left[ \frac{P_1}{\sqrt{Z_c}} + m_1 \sqrt{Z_c} \right] e^{-TD} - \frac{1}{\sqrt{2}} \left[ \frac{P_1}{\sqrt{Z_c}} - m_1 \sqrt{Z_c} \right] e^{-TD} \quad (\text{A.18})$$

The first and second bracketed terms of both equations (A.17) and (A.18) represent rightward ( $u$ ) and leftward ( $v$ ) travelling waves respectively,

$$u = \frac{1}{\sqrt{2}} \left( \frac{P}{\sqrt{Z_c}} + m \sqrt{Z_c} \right) \quad (\text{A.19})$$

$$v = \frac{1}{\sqrt{2}} \left( \frac{P}{\sqrt{Z_c}} - m \sqrt{Z_c} \right)$$

Equation (A.19) defines  $u$  and  $v$ , the decoupled wave variables. The pressure and flow are thus simply determined from them as

$$P = \sqrt{\frac{Z_c}{2}}(u + v) \quad (\text{A.20})$$

$$m = \sqrt{\frac{1}{2Z_c}}(u - v) \quad (\text{A.21})$$

The relationship between  $u$  and  $v$  at each end of a transmission line may be confirmed now by substituting for  $P$  and  $m$  in equation (A.16), giving

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} e^{-TD} & 0 \\ 0 & e^{+TD} \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \quad (\text{A.22})$$

Since the  $e^{-TD}$  is the delay operator

$$\begin{aligned} u_2(t) &= u_1(t - T) \\ v_1(t) &= v_2(t - T) \end{aligned} \quad (\text{A.23})$$

Thus,  $u$  and  $v$  waves are linearly related to right and left going  $p$  and  $m$  waves, and therefore undergo only a time delay in the line.

To use TLM method in network modeling, all lines are divided into a number of identical time delay lengths ( $\Delta t$ ) along which  $u$  and  $v$  waves travel unaltered. Each line is modeled computationally by a pair of software resistors, one each for the left- and right-going waves. A feature of the method is that all dynamic elements are modeled as distributed elements—namely as lossless transmission lines. Inductance and capacitance are modeled by equivalent opened- or closed-end transmission line stubs of delay  $\Delta t$ . Resistance (simply, although not essentially, linear) is lumped at the junctions where all wave transformation (via scattering or attenuation) is thus concentrated. Thus the model consists only of resistive junctions and transmission lines (using both actual lines and short stubs representing dynamic elements). The system can be represented by a *general*

*scattering matrix* which acts on the incident wave vector to produce the reflected wave vector.

If  $u_i$  represents incident waves and  $v_i$  emergent waves from a junction, a matrix  $S$  can be constructed which contains all the junction scattering coefficients. This matrix operates on the  $u$  vector of incident waves to produce a  $v$  vector of emergent waves. One time step later this emergent vector of waves becomes the incident vector at the junction. A forcing term  $F$  may be included as following:

$$[V]_{t+\Delta t} = S[u]_t + F[P] \quad (A.24)$$

where  $S$  is scattering matrix,  $F$  is the linear array of forcing coefficients and  $[p]$  is the forcing vector. Thus, the major concern is to find the scattering matrix  $S$  by studying different pipe connections and different elements.

From the above analysis, this kind of method combines the lumped parameter and distributed parameter models, frequency domain analysis, and the time domain implement. It can therefore be used to solve the problem of simple pipeline network, and possible be extend for use in analyzing more complex systems.

There are other methods which may be used in hydraulic transient analysis, such as the *graphical water hammer* method in which friction is neglected in its theoretical development, but taken into account by correction.

## Appendix B

### Pump Characteristics

The relationship between the pump discharge,  $Q$ , and the pumping head,  $H$ , must be specified in a mathematical model. The discharge,  $Q$ , of a centrifugal pump is a function of the rotational speed,  $N$ , and the pumping head,  $H$ , whereas transient-state speed changes depend upon the net torque,  $T$ , and the combined moment of inertia of the pump, motor, and liquid entrained in the pump impeller. Thus, four variables are involved in the characteristics, the pump dynamic head  $H$ , the discharge  $Q$ , torque  $T$ , and the rotational speed  $N$ . Two of these variables may be considered independent (i.e., for a given  $Q$  and  $N$ ,  $H$  and  $T$  are determined from the characteristics).

Since flow conditions may be described for a turbine in the same manner as a pump, we can use turbine characteristics to pump. For a given geometrically similar series of turbine, with  $D$  representing the linear dimension, the homologous equation [8] becomes:

$$\frac{H_1}{(N_1 D_1)^2} = \frac{H_2}{(N_2 D_2)^2} \quad \frac{Q_1}{N_1 D_1^3} = \frac{Q_2}{N_2 D_2^3} \quad (\text{B.1})$$

where the subscripts 1 and 2 refer to two different-sized units of the homologous series.

When restricted to a particular unit, with the subscripts referring to two homologous operating situations, Eq.(B.1) reduces to

$$\frac{H_1}{N_1^2} = \frac{H_2}{N_2^2} \quad \frac{Q_1}{N_1} = \frac{Q_2}{N_2} \quad (\text{B.2})$$

The homologous theory assumes that efficiency does not change with the size of unit, so that

$$\frac{T_1 N_1}{Q_1 H_1} = \frac{T_2 N_2}{Q_2 H_2} \quad (\text{B.3})$$

Combination of Eq.(B.2) and (B.3) obtain:

$$\frac{T_1}{N_1^2} = \frac{T_2}{N_2^2} \quad \frac{H_1}{Q_1^2} = \frac{H_2}{Q_2^2} \quad \frac{T_1}{Q_1^2} = \frac{T_2}{Q_2^2} \quad (\text{B.4})$$

Various variables at the point of best efficiency are referred to as *rated conditions*. By using these values as a reference, the following non-dimensional variables may be defined:

$$h = \frac{H}{H_R} \quad \beta = \frac{T}{T_R} \quad v = \frac{Q}{Q_R} \quad \alpha = \frac{N}{N_R} \quad (\text{B.5})$$

where the subscript  $R$  indicates the rated conditions.

On the basis of Eq.(B.5), the dimensionless-homologous relations may now be expressed as

$$\frac{h}{\alpha^2} \text{ vs. } \frac{v}{\alpha} \quad \frac{\beta}{\alpha^2} \text{ vs. } \frac{v}{\alpha} \quad \frac{h}{v^2} \text{ vs. } \frac{\alpha}{v} \quad \frac{\beta}{v^2} \text{ vs. } \frac{\alpha}{v} \quad (\text{B.6})$$

According to the homologous theory,  $\frac{h}{\alpha^2}$  vs.  $\frac{v}{\alpha}$  is the head-discharge relation for any speed  $\alpha$  of that unit, and similarly  $\frac{\beta}{\alpha^2}$  vs.  $\frac{v}{\alpha}$  is the representation of the torque-discharge relations for any speed  $\alpha$  of that unit.

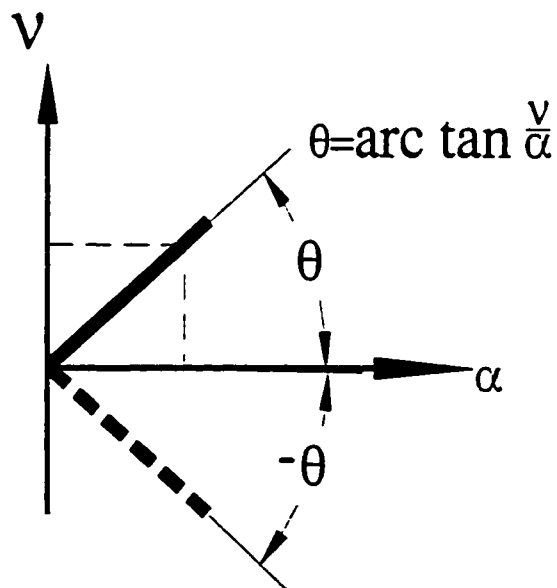
During normal pump operation,  $h$ ,  $\beta$ ,  $v$ , and  $\alpha$  are all positive. However, during the transient-state conditions, they may become negative individually or in groups.

Because the above relations are difficult to handle, the following method may be used to overcome this difficulty.

$$\frac{h}{\alpha^2 + v^2} \text{ vs. } \tan^{-1} \frac{v}{\alpha} \quad \frac{\beta}{\alpha^2 + v^2} \text{ vs. } \tan^{-1} \frac{v}{\alpha} \quad (\text{B.7})$$

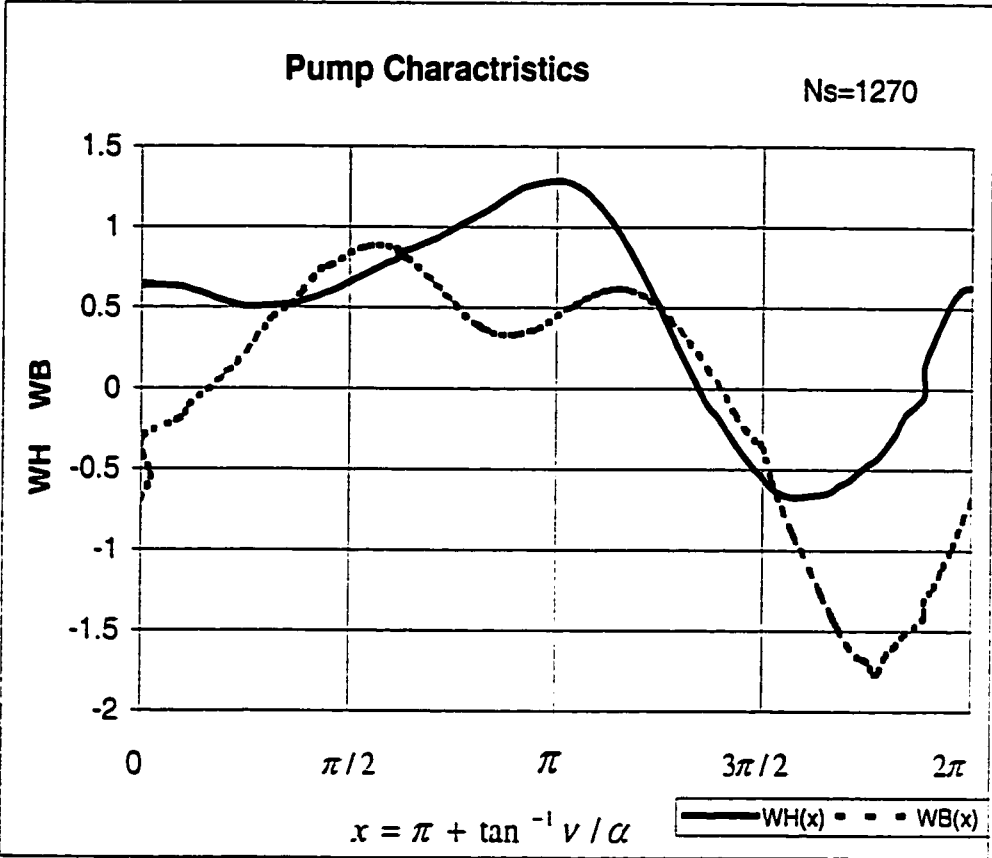
It is noted that for a given  $\frac{v}{\alpha}$  one can develop  $\frac{h}{\alpha^2 + v^2}$  from a plot of  $\frac{h}{\alpha^2}$  vs.

$\frac{v}{\alpha}$ , so that homologous relationships are preserved in Eq.(B.7).



**Fig B.1 Polar Diagram of  $\theta$**

If one had complete pump data for all zones of operation, referring to Fig.(B.1), a polar diagram of  $\theta = \tan^{-1} \frac{v}{\alpha}$  vs.  $r = \frac{h}{\alpha^2 + v^2}$  and vs.  $r = \frac{\beta}{\alpha^2 + v^2}$  would represent the complete characteristics of the pump by two closed curves [7].



$v=0$	$\alpha=0$	$v=0$	$\alpha=0$	$v=0$
Turbine	Dissipation	Normal	Reversed Speed	
Zone	Zone	Zone	Dissipation	
$v \leq 0$	$v < 0$	$v \geq 0$	Zone	
$\alpha < 0$	$\alpha \geq 0$	$\alpha \geq 0$	$v > 0$	
			$\alpha < 0$	
0	$\pi/2$	$\pi$	$3\pi/2$	$2\pi$

**Fig B.2 Complete Pump Characteristics**

The angle  $x = \pi + \tan^{-1} \frac{v}{\alpha}$  may also be plotted as an abscissa against  $W_H(x)$  or

$W_B(x)$ , where

$$W_H(x) = \frac{h}{\alpha^2 + v^2} \quad W_B(x) = \frac{\beta}{\alpha^2 + v^2} \quad x = \pi + \tan^{-1} \frac{v}{\alpha} \quad (\text{B.8})$$

To obtain the rectangular coordinate plot (Fig B.2), the  $\theta = \tan^{-1} \frac{v}{\alpha}$  is an angle between

$-\pi$  and  $+\pi$ . It holds for all real values of  $v$  and  $\alpha$  except  $v=0$  and  $\alpha=0$  at the same time. The parameter  $x$  varies from 0 to  $2\pi$ . We can determine sets of values of  $H$ ,  $Q$ ,  $T$ , and  $N$ , or  $H$ ,  $Q$ ,  $N$ , and efficiency  $\eta$  for various operating points throughout the zone.

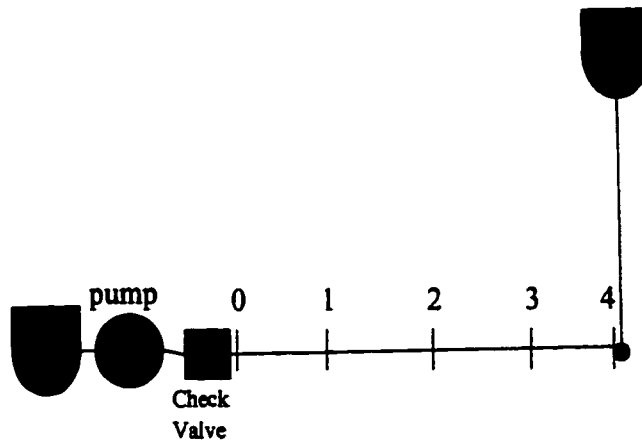
Since  $T\omega = \frac{QH\gamma}{\eta}$  for a pump, and  $T\omega = QH\gamma\eta$  for a turbine,  $T$  may be

determined. Thus the four values  $h$ ,  $\beta$ ,  $v$ , and  $\alpha$  can be obtained, when rated values  $H_R$ ,  $T_R$ ,  $Q_R$  and  $N_R$  are given. From Eq. (B.8),  $x$ ,  $W_H$ , and  $W_B$  may be determined for each operating point.

## Appendix C

### **Simulation Results**

1. *Simple pump, check valve and pipe system simulation during pump failure*



**Fig. C 1 Simple Pump, Check Valve, Pipe Systems**

The results of the simulation are as following:

t	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]	$\theta$
t= 0	4.14276	4.14276	4.14276	4.14276	4.14276	theta=60
t= 0.1	3.46655	4.14276	4.14276	4.14276	4.14276	theta=60
t= 0.2	3.47303	3.46981	3.47305	4.14276	4.14276	theta=62
t= 0.3	3.47981	3.47624	3.47945	3.47629	2.81568	theta=66
t= 0.4	3.48675	3.48298	2.82762	2.82147	2.82721	theta=70
t= 0.5	2.18783	2.834	2.83966	2.83332	2.83939	theta=70
t= 0.6	2.20318	2.19553	2.2036	2.84568	2.85193	theta=70
t= 0.7	2.21916	2.2112	2.21944	2.21188	1.57543	theta=70
t= 0.8	2.23549	2.22735	1.59394	1.58451	1.59354	theta=70
t= 0.9	0.975052	1.60359	1.61255	1.60294	1.61229	theta=70
t= 1	0.994977	0.985031	0.995373	1.62187	1.63141	theta=70
t= 1.1	1.01558	1.00529	1.01584	1.00594	0.382074	theta=70
t= 1.2	1.03657	1.02609	0.403323	0.392514	0.402944	theta=70
t= 1.3	-0.206973	0.414364	0.424771	0.413742	0.424529	theta=70
t= 1.4	-0.185892	-0.196429	-0.185515	0.435546	0.446551	theta=70
t= 1.5	-0.16408	-0.174983	-0.16384	-0.174367	-0.795208	theta=70
t= 1.6	-0.141809	-0.152942	-0.773046	-0.784308	-0.77343	theta=70
t= 1.7	-1.37975	-0.76156	-0.750725	-0.76219	-0.750972	theta=70
t= 1.8	-1.35649	-1.36809	-1.35608	-0.739529	-0.728107	theta=70
t= 1.9	-1.3326	-1.34452	-1.33233	-1.34386	-1.9575	theta=70
t= 2	-1.30837	-1.32046	-1.9316	-1.94472	-1.932	theta=70
t= 2.1	-2.52432	-1.91828	-1.90569	-1.91895	-1.90596	theta=70
t= 2.2	-2.49592	-2.51006	-2.49548	-1.89276	-1.87963	theta=70
t= 2.3	-2.46706	-2.48143	-2.46675	-2.48071	-3.07801	theta=70
t= 2.4	-2.438	-2.45247	-3.04568	-3.06199	-3.04611	theta=70
t= 2.5	-3.61449	-3.0292	-3.01356	-3.02992	-3.01385	theta=70
t= 2.6	-3.57863	-3.59645	-3.57814	-2.99762	-2.9815	theta=70
t= 2.7	-3.54255	-3.56049	-3.54219	-3.55968	-4.13265	theta=70
t= 2.8	-3.5065	-3.52442	-4.09201	-4.11245	-4.09247	theta=70
t= 2.9	-4.6288	-4.07144	-4.05187	-4.07223	-4.05221	theta=70
t= 3	-4.5841	-4.60628	-4.58354	-4.03205	-4.01211	theta=70
t= 3.1	-4.53948	-4.56162	-4.53904	-4.5607	-5.10296	theta=70
t= 3.2	-4.49516	-4.51715	-5.05309	-5.07812	-5.0536	theta=70
t= 3.3	-5.55202	-5.028	-5.00411	-5.02889	-5.0045	theta=70
t= 3.4	-5.49806	-5.5248	-5.49741	-4.98002	-4.95586	theta=70
t= 3.5	-5.44453	-5.47106	-5.444	-5.46999	-5.977	theta=70
t= 3.6	-5.39164	-5.41785	-5.91796	-5.94755	-5.91853	theta=70
t= 3.7	-6.37549	-5.88837	-5.86023	-5.88939	-5.86068	theta=70
t= 3.8	-6.31277	-6.3438	-6.31201	-5.83194	-5.80363	theta=70
t= 3.9	-6.25085	-6.28149	-6.25021	-6.28024	-6.74929	theta=70
t= 4	-6.1899	-6.22006	-6.68195	-6.71566	-6.68259	theta=70
t= 4.1	-7.09667	-6.64829	-6.61634	-6.64945	-6.61686	theta=70
t= 4.2	-7.02639	-7.06112	-7.02553	-6.5843	-6.55228	theta=70
t= 4.3	-6.95726	-6.99142	-6.95651	-6.99	-7.42	theta=70
t= 4.4	-6.88942	-6.92295	-7.34581	-7.38291	-7.34652	theta=70
t= 4.5	-7.7181	-7.30878	-7.27374	-7.31009	-7.27434	theta=70
t= 4.6	-7.64191	-7.67952	-7.64094	-7.23865	-7.20361	theta=70
t= 4.7	-7.56716	-7.60406	-7.56631	-7.60246	-7.99367	theta=70
t= 4.8	-7.49399	-7.53012	-7.91441	-7.95403	-7.91518	theta=70
t= 4.9	-8.24598	-7.8749	-7.83761	-7.87635	-7.83828	theta=70
t= 5	-8.16573	-8.20531	-8.16467	-7.8003	-7.76306	theta=70
t= 5.1	-8.08716	-8.12592	-8.08621	-8.12415	-8.47785	theta=70
t= 5.2	-8.01037	-8.04825	-8.39541	-8.43661	-8.39625	theta=70
t= 5.3	-8.68889	-8.35433	-8.31568	-8.35591	-8.31642	theta=70
t= 5.4	-8.60643	-8.64707	-8.60529	-8.27703	-8.23845	theta=70
t= 5.5	-8.52582	-8.56555	-8.52478	-8.56365	-8.88183	theta=70
t= 5.6	-8.44711	-8.48591	-8.79801	-8.8399	-8.79891	theta=70
t= 5.7	-9.05653	-8.75625	-8.71707	-8.75794	-8.71788	theta=70
t= 5.8	-8.97359	-9.01444	-8.97239	-8.6779	-8.6388	theta=70
t= 5.9	-8.89256	-8.93247	-8.89146	-8.93046	-9.21556	theta=70

t	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]	θ
t= 18	-10.5421	-10.5449	-10.553	-10.5557	-10.5532	theta=70
t= 18.1	-10.558	-10.5502	-10.5479	-10.5506	-10.548	theta=70
t= 18.2	-10.5532	-10.5556	-10.553	-10.5454	-10.5428	theta=70
t= 18.3	-10.5484	-10.5507	-10.5482	-10.5504	-10.5578	theta=70
t= 18.4	-10.5434	-10.5459	-10.5531	-10.5555	-10.5533	theta=70
t= 18.5	-10.5576	-10.5506	-10.5485	-10.5509	-10.5487	theta=70
t= 18.6	-10.5533	-10.5554	-10.5531	-10.5463	-10.544	theta=70
t= 18.7	-10.5489	-10.5511	-10.5488	-10.5508	-10.5573	theta=70
t= 18.8	-10.5445	-10.5467	-10.5532	-10.5553	-10.5533	theta=70
t= 18.9	-10.5571	-10.5509	-10.5491	-10.5512	-10.5492	theta=70
t= 19	-10.5533	-10.5552	-10.5532	-10.5471	-10.545	theta=70
t= 19.1	-10.5494	-10.5513	-10.5493	-10.5511	-10.5569	theta=70
t= 19.2	-10.5455	-10.5474	-10.5532	-10.5551	-10.5533	theta=70
t= 19.3	-10.5567	-10.5512	-10.5495	-10.5515	-10.5497	theta=70
t= 19.4	-10.5533	-10.555	-10.5532	-10.5478	-10.5459	theta=70
t= 19.5	-10.5499	-10.5516	-10.5497	-10.5513	-10.5565	theta=70
t= 19.6	-10.5463	-10.5481	-10.5532	-10.5549	-10.5533	theta=70
t= 19.7	-10.5563	-10.5514	-10.5499	-10.5517	-10.55	theta=70
t= 19.8	-10.5533	-10.5548	-10.5532	-10.5484	-10.5467	theta=70
t= 19.9	-10.5502	-10.5517	-10.5501	-10.5515	-10.5561	theta=70
t= 20	-10.5471	-10.5486	-10.5532	-10.5547	-10.5533	theta=70
t= 20.1	-10.5559	-10.5516	-10.5503	-10.5518	-10.5504	theta=70
t= 20.2	-10.5533	-10.5546	-10.5531	-10.5489	-10.5474	theta=70
t= 20.3	-10.5505	-10.5519	-10.5504	-10.5516	-10.5558	theta=70
t= 20.4	-10.5477	-10.5491	-10.5531	-10.5545	-10.5532	theta=70
t= 20.5	-10.5556	-10.5517	-10.5505	-10.5519	-10.5506	theta=70
t= 20.6	-10.5532	-10.5544	-10.5531	-10.5493	-10.548	theta=70
t= 20.7	-10.5508	-10.552	-10.5507	-10.5518	-10.5554	theta=70
t= 20.8	-10.5483	-10.5495	-10.5531	-10.5543	-10.5532	theta=70
t= 20.9	-10.5553	-10.5518	-10.5508	-10.552	-10.5509	theta=70
t= 21	-10.5532	-10.5542	-10.5531	-10.5497	-10.5485	theta=70
t= 21.1	-10.551	-10.552	-10.5509	-10.5519	-10.5551	theta=70
t= 21.2	-10.5487	-10.5498	-10.553	-10.5541	-10.5531	theta=70
t= 21.3	-10.555	-10.5519	-10.551	-10.5521	-10.5511	theta=70
t= 21.4	-10.5531	-10.554	-10.553	-10.55	-10.5489	theta=70
t= 21.5	-10.5512	-10.5521	-10.5511	-10.5519	-10.5549	theta=70
t= 21.6	-10.5492	-10.5501	-10.553	-10.554	-10.5531	theta=70
t= 21.7	-10.5547	-10.552	-10.5512	-10.5521	-10.5512	theta=70
t= 21.8	-10.553	-10.5539	-10.553	-10.5503	-10.5493	theta=70
t= 21.9	-10.5513	-10.5522	-10.5512	-10.552	-10.5546	theta=70
t= 22	-10.5495	-10.5504	-10.553	-10.5538	-10.553	theta=70
t= 22.1	-10.5545	-10.5521	-10.5513	-10.5522	-10.5514	theta=70
t= 22.2	-10.553	-10.5537	-10.5529	-10.5505	-10.5497	theta=70
t= 22.3	-10.5514	-10.5522	-10.5514	-10.5521	-10.5544	theta=70
t= 22.4	-10.5498	-10.5506	-10.5529	-10.5537	-10.553	theta=70
t= 22.5	-10.5543	-10.5521	-10.5514	-10.5522	-10.5515	theta=70
t= 22.6	-10.5529	-10.5536	-10.5529	-10.5507	-10.55	theta=70
t= 22.7001	-10.5516	-10.5522	-10.5515	-10.5521	-10.5542	theta=70
t= 22.8001	-10.5501	-10.5508	-10.5529	-10.5535	-10.5529	theta=70
t= 22.9001	-10.5541	-10.5521	-10.5515	-10.5522	-10.5516	theta=70
t= 23.0001	-10.5529	-10.5535	-10.5528	-10.5509	-10.5503	theta=70
t= 23.1001	-10.5517	-10.5523	-10.5516	-10.5521	-10.554	theta=70
t= 23.2001	-10.5504	-10.551	-10.5528	-10.5534	-10.5529	theta=69
t= 23.3001	-10.5529	-10.5522	-10.5516	-10.5523	-10.5517	theta=68
t= 23.4001	-10.5509	-10.5524	-10.5519	-10.5511	-10.5505	theta=67
t= 23.5001	-10.5487	-10.5504	-10.5498	-10.5513	-10.552	theta=66
t= 23.6001	-10.5465	-10.5482	-10.549	-10.5505	-10.5491	theta=65
t= 23.7001	-10.5467	-10.5472	-10.5459	-10.5476	-10.5461	theta=64
t= 23.8001	-10.5427	-10.5454	-10.544	-10.5444	-10.5429	theta=63
t= 23.9001	-10.5385	-10.5414	-10.5399	-10.5424	-10.542	theta=62

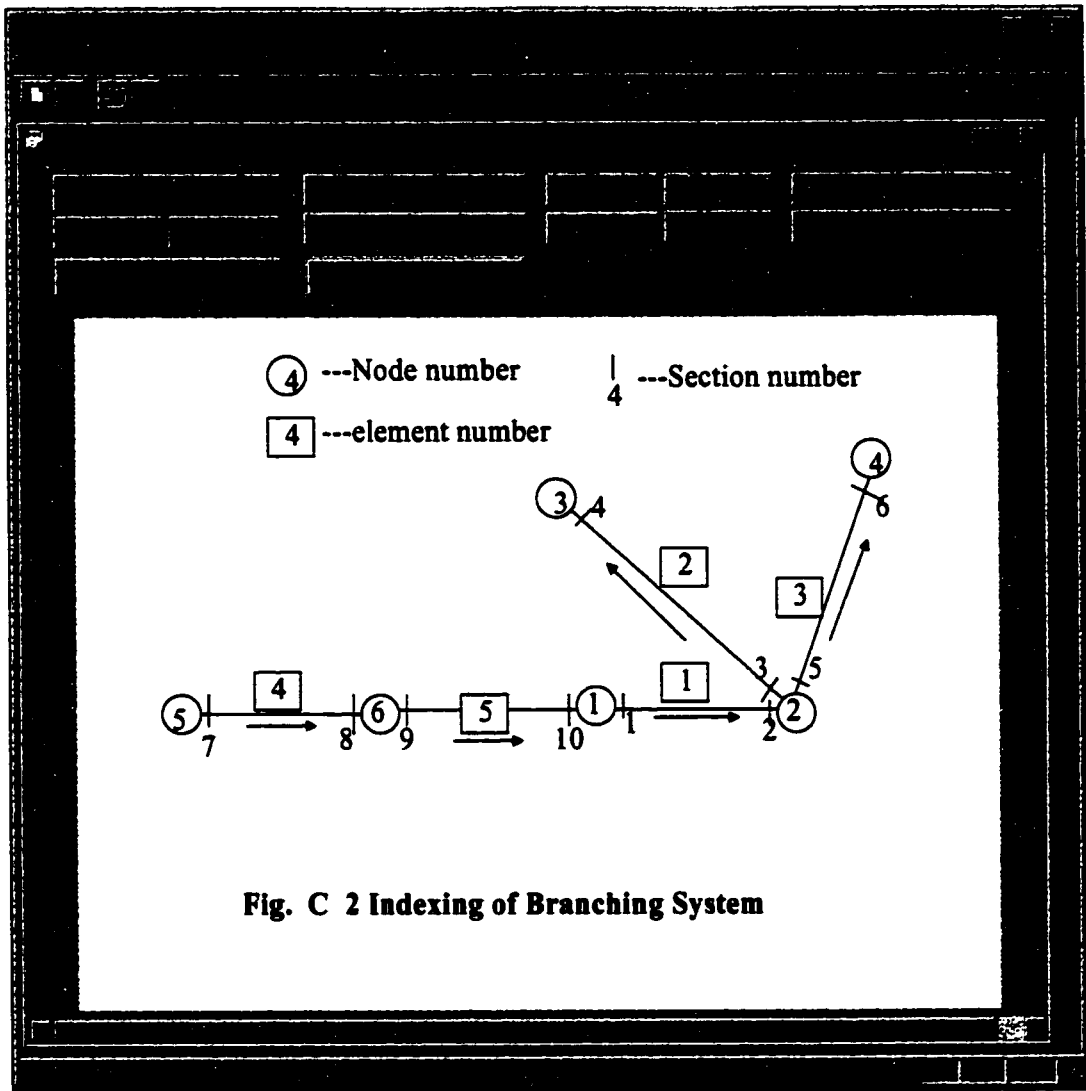
t	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]	θ
t= 24.0001	-10.5341	-10.5371	-10.5367	-10.5394	-10.537	theta=61
t= 24.1001	-10.5318	-10.5338	-10.5315	-10.5344	-10.5318	theta=60
t= 24.2001	-10.5255	-10.5295	-10.5271	-10.529	-10.5263	theta=59
t= 24.3001	-10.5189	-10.5232	-10.5206	-10.5244	-10.5226	theta=58
t= 24.4001	-10.512	-10.5165	-10.5148	-10.5189	-10.5152	theta=57
t= 24.5001	-10.5067	-10.5104	-10.507	-10.5113	-10.5074	theta=56
t= 24.6001	-10.4978	-10.5033	-10.4997	-10.5032	-10.4992	theta=55
t= 24.7001	-10.4883	-10.4942	-10.4904	-10.4957	-10.4924	theta=54
t= 24.8001	-10.4783	-10.4846	-10.4815	-10.4871	-10.482	theta=53
t= 24.9001	-10.4696	-10.4753	-10.4705	-10.4765	-10.4711	theta=52
t= 25.0001	-10.4574	-10.4649	-10.4598	-10.4653	-10.4596	theta=51
t= 25.1001	-10.4445	-10.4525	-10.4471	-10.4543	-10.4491	theta=49
t= 25.2001	-10.4272	-10.4393	-10.4344	-10.4421	-10.4352	theta=47
t= 25.3001	-10.4102	-10.4226	-10.4162	-10.4278	-10.4206	theta=45
t= 25.4001	-10.3892	-10.404	-10.3973	-10.4092	-10.3981	theta=43
t= 25.5001	-10.3666	-10.3827	-10.3723	-10.3865	-10.3752	theta=41
t= 25.6001	-10.3361	-10.3565	-10.3459	-10.3613	-10.3478	theta=39
t= 25.7001	-10.3039	-10.3259	-10.3132	-10.3327	-10.3181	theta=37
t= 25.8001	-10.2658	-10.2917	-10.2779	-10.299	-10.2804	theta=35
t= 25.9001	-10.2237	-10.2526	-10.2351	-10.2598	-10.2398	theta=33
t= 26.0001	-10.1721	-10.2068	-10.188	-10.2156	-10.1921	theta=31
t= 26.1001	-10.1154	-10.154	-10.1318	-10.1651	-10.1388	theta=29
t= 26.2001	-10.0494	-10.0941	-10.0693	-10.1063	-10.0746	theta=27
t= 26.3001	-9.96525	-10.0256	-9.99569	-10.0385	-10.0033	theta=25
t= 26.4001	-9.87474	-9.9365	-9.90348	-9.9615	-9.92073	theta=23
t= 26.5001	-9.77334	-9.84289	-9.80459	-9.86394	-9.80856	theta=21
t= 26.6001	-9.65593	-9.73637	-9.6839	-9.75063	-9.69411	theta=19
t= 26.7001	-9.50437	-9.60507	-9.55168	-9.62889	-9.56525	theta=17
t= 26.8001	-9.33934	-9.45282	-9.39279	-9.48973	-9.41602	theta=15
t= 26.9001	-9.14425	-9.28136	-9.2118	-9.3211	-9.2284	theta=12
t= 27.0001	-8.82381	-9.07709	-8.98932	-9.12145	-9.01696	theta=9
t= 27.1001	-8.31571	-8.73986	-8.64195	-8.8875	-8.76095	theta=6
t= 27.2001	-7.38821	-8.22337	-8.10659	-8.51947	-8.28317	theta=3
t= 27.3001	-5.28199	-7.28074	-7.06123	-7.877	-7.4789	theta=1
t= 27.4001	0	-5.07978	-4.7104	-6.67426	-5.88319	theta=1
t= 27.5001	0	0.353088	1.11192	-3.93778	-2.01179	theta=1
t= 27.6001	0	0.758545	2.67249	3.02963	8.04777	theta=1
t= 27.7001	0	1.91199	6.87294	7.65904	7.27782	theta=1
t= 27.8001	0	4.94767	4.59348	6.50214	5.7392	theta=1
t= 27.9001	0	-0.33873	-1.07109	3.848	1.97525	theta=1
t= 28.0001	0	-0.732097	-2.59339	-2.93594	-7.82526	theta=1
t= 28.1001	0	-1.85943	-6.69448	-7.45275	-7.08724	theta=1
t= 28.2001	0	-4.82242	-4.4824	-6.33874	-5.60223	theta=1
t= 28.3001	0	0.325326	1.0328	-3.76243	-1.94018	theta=1
t= 28.4001	0	0.707225	2.51856	2.8475	7.61459	theta=1
t= 28.5001	0	1.80956	6.52509	7.25721	6.90637	theta=1
t= 28.6001	0	4.7035	4.37673	6.18343	5.47179	theta=1
t= 28.7001	0	-0.312793	-0.996827	3.68078	1.90649	theta=1
t= 28.8001	0	-0.683798	-2.44767	-2.76388	-7.41487	theta=1
t= 28.9001	0	-1.76219	-6.3641	-7.07161	-6.73447	theta=1
t= 29.0001	0	-4.59045	-4.27608	-6.03561	-5.3474	theta=1
t= 29.1001	0	0.301054	0.962974	-3.60278	-1.8741	theta=1
t= 29.2001	0	0.661698	2.38043	2.68472	7.22525	theta=1
t= 29.3001	0	1.71713	6.21091	6.89521	6.57091	theta=1
t= 29.4001	0	4.48284	4.1801	5.89475	5.22866	theta=1
t= 29.5001	0	-0.290039	-0.931069	3.52819	1.84293	theta=1
t= 29.6001	0	-0.640822	-2.31656	-2.60967	-7.04499	theta=1
t= 29.7001	0	-1.67421	-6.06494	-6.72734	-6.41508	theta=1
t= 29.8001	0	-4.38028	-4.08846	-5.76037	-5.11519	theta=1
t= 29.9001	0	0.279687	0.900956	-3.45678	-1.81291	theta=1

	t	H[0]	H[1]	H[2]	H[3]	H[4]	θ
t=	0	315.841	305.63	295.42	285.21	275	theta=60
t=	0.1	-1.78086	305.63	295.42	285.21	275	theta=60
t=	0.2	-1.78754	-10.4598	-19.1455	285.21	275	theta=62
t=	0.3	-1.61016	-10.4731	-19.1654	-27.8378	275	theta=66
t=	0.4	-1.31694	-10.3032	290.05	282.429	275	theta=70
t=	0.5	-0.430787	297.798	290.367	282.621	275	theta=70
t=	0.6	-0.436947	-6.88642	-13.5321	282.746	275	theta=70
t=	0.7	-0.443281	-7.08208	-13.852	-20.3028	275	theta=70
t=	0.8	-0.449869	-7.21272	286.634	280.724	275	theta=70
t=	0.9	-0.0855724	292.665	286.941	280.91	275	theta=70
t=	1	-0.0890889	-5.31939	-10.7352	281.031	275	theta=70
t=	1.1	-0.0928377	-5.5047	-11.038	-16.2689	275	theta=70
t=	1.2	-0.0967416	-5.626	285.112	279.967	275	theta=70
t=	1.3	0.00385429	290.37	285.402	280.144	275	theta=70
t=	1.4	0.00311399	-4.89856	-9.9764	280.258	275	theta=70
t=	1.5	0.00242059	-5.07469	-10.264	-15.1658	275	theta=70
t=	1.6	0.00181129	-5.18686	284.623	279.722	275	theta=70
t=	1.7	0.171385	289.64	284.918	279.901	275	theta=70
t=	1.8	0.165615	-4.15235	-8.65954	280.017	275	theta=70
t=	1.9	0.15982	-4.34125	-8.96812	-13.287	275	theta=70
t=	2	0.154082	-4.46685	282.602	278.707	275	theta=70
t=	2.1	0.573591	286.623	282.914	278.895	275	theta=70
t=	2.2	0.560698	-2.32008	-5.40699	279.019	275	theta=70
t=	2.3	0.547832	-2.52546	-5.74273	-8.62524	275	theta=70
t=	2.4	0.535082	-2.66892	278.949	276.875	275	theta=70
t=	2.5	1.17596	281.162	279.285	277.074	275	theta=70
t=	2.6	1.15274	0.483499	-0.416331	277.211	275	theta=70
t=	2.7	1.12969	0.253958	-0.791887	-1.46395	275	theta=70
t=	2.8	1.10673	0.0846436	273.866	274.325	275	theta=70
t=	2.9	1.92865	273.565	274.237	274.54	275	theta=70
t=	3	1.89143	4.09029	6.02452	274.696	275	theta=70
t=	3.1	1.85481	3.82711	5.59382	7.78861	275	theta=70
t=	3.2	1.81886	3.62281	267.617	271.188	275	theta=70
t=	3.3	2.77478	264.229	268.036	271.427	275	theta=70
t=	3.4	2.72096	8.3022	13.5761	271.608	275	theta=70
t=	3.5	2.66835	7.99683	13.0757	18.6513	275	theta=70
t=	3.6	2.61669	7.7488	260.497	267.614	275	theta=70
t=	3.7	3.65856	253.595	260.975	267.881	275	theta=70
t=	3.8	3.58693	12.9133	21.8841	268.093	275	theta=70
t=	3.9	3.51707	12.5605	21.304	30.6231	275	theta=70
t=	4	3.44882	12.263	252.804	263.752	275	theta=70
t=	4.1	4.53325	242.107	253.348	264.05	275	theta=70
t=	4.2	4.44382	17.7331	30.6161	264.296	275	theta=70
t=	4.3	4.35705	17.3305	29.9512	43.231	275	theta=70
t=	4.4	4.27241	16.9808	244.812	259.739	275	theta=70
t=	4.5	5.36207	230.171	245.424	260.07	275	theta=70
t=	4.6	5.25639	22.5957	39.4792	260.351	275	theta=70
t=	4.7	5.15422	22.1443	38.7311	56.0587	275	theta=70
t=	4.8	5.05534	21.7447	236.756	255.695	275	theta=70
t=	4.9	6.12059	218.14	237.435	256.058	275	theta=70
t=	5	6.00198	27.3688	48.2355	256.374	275	theta=70
t=	5.1	5.88724	26.8739	47.4109	68.7639	275	theta=70
t=	5.2	5.77586	26.4289	228.832	251.718	275	theta=70
t=	5.3	6.79553	206.3	229.57	252.11	275	theta=70
t=	5.4	6.66756	31.9536	56.7027	252.457	275	theta=70
t=	5.5	6.54283	31.4222	55.8125	81.0832	275	theta=70
t=	5.6	6.42289	30.9383	221.183	247.881	275	theta=70
t=	5.7	7.38317	194.868	221.973	248.297	275	theta=70
t=	5.8	7.24834	36.2831	64.7529	248.672	275	theta=70
t=	5.9	7.118	35.7244	63.8105	92.8282	275	theta=70

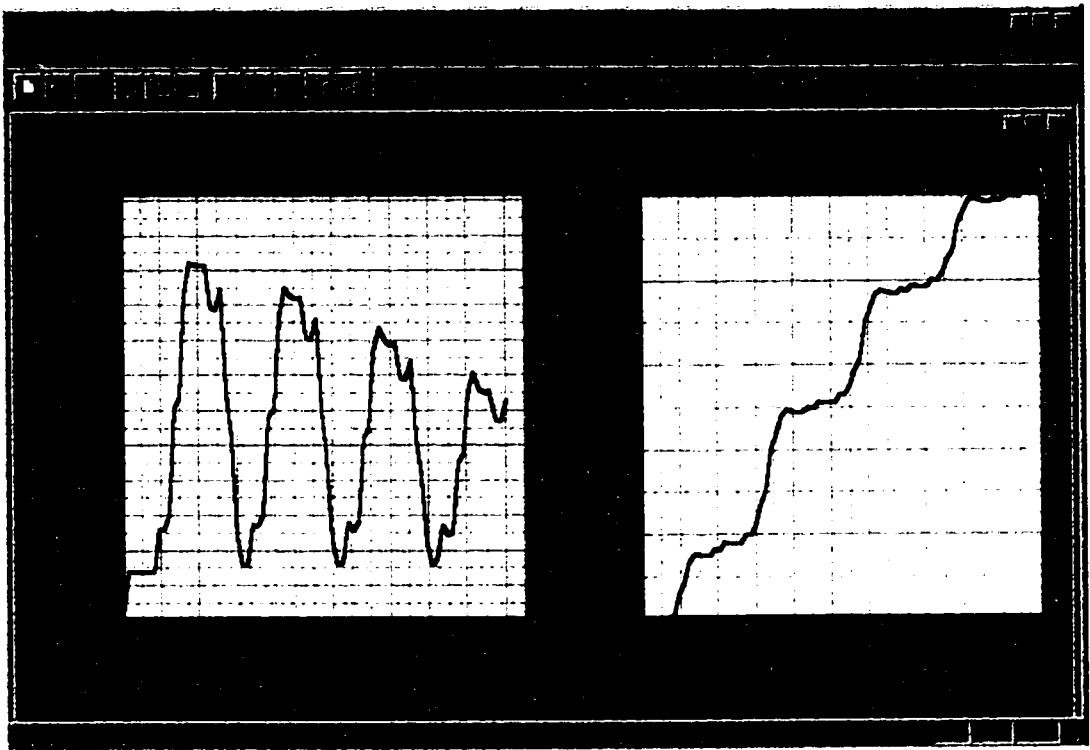
t	H[0]	H[1]	H[2]	H[3]	H[4]	θ
t= 18	10.0038	74.8745	144.881	209.9	275	theta=70
t= 18.1	10.034	79.9421	145.039	209.98	275	theta=70
t= 18.2	10.0251	75.1948	140.278	210.058	275	theta=70
t= 18.3	10.0158	75.1093	140.117	205.284	275	theta=70
t= 18.4	10.0062	75.0248	144.621	209.773	275	theta=70
t= 18.5	10.033	79.5415	144.765	209.847	275	theta=70
t= 18.6	10.0251	75.3133	140.523	209.918	275	theta=70
t= 18.7	10.017	75.236	140.377	205.662	275	theta=70
t= 18.8	10.0084	75.1591	144.39	209.661	275	theta=70
t= 18.9	10.0324	79.1848	144.521	209.728	275	theta=70
t= 19	10.0251	75.4193	140.741	209.792	275	theta=70
t= 19.1	10.0178	75.348	140.609	206	275	theta=70
t= 19.2	10.0102	75.2792	144.183	209.561	275	theta=70
t= 19.3	10.0315	78.8663	144.303	209.622	275	theta=70
t= 19.4	10.0253	75.5134	140.936	209.681	275	theta=70
t= 19.5	10.0184	75.4489	140.815	206.301	275	theta=70
t= 19.6	10.0116	75.3853	143.999	209.472	275	theta=70
t= 19.7	10.0307	78.5823	144.109	209.527	275	theta=70
t= 19.8	10.0248	75.597	141.11	209.581	275	theta=70
t= 19.9	10.0189	75.5383	140.999	206.569	275	theta=70
t= 20	10.0129	75.4801	143.835	209.392	275	theta=70
t= 20.1	10.0302	78.3299	143.936	209.443	275	theta=70
t= 20.2	10.0252	75.6719	141.264	209.493	275	theta=70
t= 20.3	10.0194	75.6183	141.163	206.808	275	theta=70
t= 20.4	10.0143	75.5653	143.69	209.322	275	theta=70
t= 20.5	10.0293	78.1047	143.781	209.367	275	theta=70
t= 20.6	10.0248	75.7375	141.401	209.413	275	theta=70
t= 20.7	10.0201	75.6892	141.31	207.021	275	theta=70
t= 20.8	10.0153	75.641	143.561	209.259	275	theta=70
t= 20.9	10.0287	77.9045	143.644	209.301	275	theta=70
t= 21	10.0246	75.7963	141.523	209.342	275	theta=70
t= 21.1	10.0203	75.752	141.44	207.209	275	theta=70
t= 21.2	10.0164	75.7084	143.445	209.203	275	theta=70
t= 21.3	10.028	77.7259	143.521	209.241	275	theta=70
t= 21.4	10.0244	75.8485	141.633	209.28	275	theta=70
t= 21.5	10.0212	75.8089	141.556	207.378	275	theta=70
t= 21.6	10.017	75.7687	143.343	209.154	275	theta=70
t= 21.7	10.0279	77.5668	143.412	209.189	275	theta=70
t= 21.8	10.0242	75.8954	141.729	209.223	275	theta=70
t= 21.9	10.021	75.8583	141.659	207.529	275	theta=70
t= 22	10.0179	75.8222	143.251	209.109	275	theta=70
t= 22.1	10.0275	77.425	143.314	209.141	275	theta=70
t= 22.2	10.0245	75.9369	141.816	209.173	275	theta=70
t= 22.3	10.0214	75.9039	141.752	207.663	275	theta=70
t= 22.4	10.0183	75.8698	143.169	209.07	275	theta=70
t= 22.5	10.0265	77.298	143.227	209.098	275	theta=70
t= 22.6	10.0243	75.9731	141.892	209.128	275	theta=70
t= 22.7001	10.0216	75.9438	141.834	207.782	275	theta=70
t= 22.8001	10.0191	75.9124	143.096	209.035	275	theta=70
t= 22.9001	10.0264	77.1865	143.15	209.061	275	theta=70
t= 23.0001	10.024	76.0057	141.961	209.089	275	theta=70
t= 23.1001	10.022	75.9793	141.907	207.888	275	theta=70
t= 23.2001	10.0192	75.9498	143.032	209.004	275	theta=69
t= 23.3001	10.4769	77.086	143.081	209.028	275	theta=68
t= 23.4001	10.9615	76.4804	142.461	209.053	275	theta=67
t= 23.5001	11.4843	76.9356	142.885	208.416	275	theta=66
t= 23.6001	12.0481	77.4272	143.983	209.455	275	theta=65
t= 23.7001	12.6617	78.5881	144.13	209.526	275	theta=64
t= 23.8001	13.3105	78.2083	143.687	209.602	275	theta=63
t= 23.9001	14.008	78.7813	144.18	209.092	275	theta=62

t	H[0]	H[1]	H[2]	H[3]	H[4]	θ
t= 24.0001	14.7553	79.3979	145.278	210.075	275	theta=61
t= 24.1001	15.5637	80.6193	145.54	210.202	275	theta=60
t= 24.2001	16.4181	80.4859	145.286	210.336	275	theta=59
t= 24.3001	17.3311	81.2078	145.87	209.955	275	theta=58
t= 24.4001	18.307	81.9825	147	210.902	275	theta=57
t= 24.5001	19.356	83.307	147.397	211.095	275	theta=56
t= 24.6001	20.4622	83.4437	147.347	211.299	275	theta=55
t= 24.7001	21.6402	84.3533	148.048	211.051	275	theta=54
t= 24.8001	22.8937	85.3219	149.242	211.984	275	theta=53
t= 24.9001	24.2345	86.7947	149.799	212.255	275	theta=52
t= 25.0001	25.6449	87.2331	149.973	212.541	275	theta=51
t= 25.1001	27.1409	88.3696	150.819	212.433	275	theta=49
t= 25.2001	30.4655	89.5737	152.111	213.373	275	theta=47
t= 25.3001	34.2115	92.9582	154.548	213.733	275	theta=45
t= 25.4001	38.3889	95.7643	156.971	215.787	275	theta=43
t= 25.5001	43.0457	99.5462	158.73	216.179	275	theta=41
t= 25.6001	48.0265	102.195	160.969	217.534	275	theta=39
t= 25.7001	53.7114	106.741	164.153	218.423	275	theta=37
t= 25.8001	60.272	111.067	167.569	220.7	275	theta=35
t= 25.9001	67.8326	116.701	170.942	221.854	275	theta=33
t= 26.0001	76.4424	122.005	175.068	224.058	275	theta=31
t= 26.1001	86.3089	129.41	180.259	225.984	275	theta=29
t= 26.2001	97.5567	137.06	185.96	229.233	275	theta=27
t= 26.3001	115.087	146.338	191.997	231.696	275	theta=25
t= 26.4001	131.053	160.565	203.676	235.256	275	theta=23
t= 26.5001	149.261	173.994	213.513	243.319	275	theta=21
t= 26.6001	170.256	188.598	220.174	245.172	275	theta=19
t= 26.7001	194.415	201.671	231.304	249.943	275	theta=17
t= 26.8001	224.282	223.78	248.551	256.284	275	theta=15
t= 26.9001	262.325	248.757	267.123	267.138	275	theta=12
t= 27.0001	351.016	280.323	287.854	274.894	275	theta=9
t= 27.1001	512.617	357.67	356.854	287.812	275	theta=6
t= 27.2001	864.23	510.155	495.252	343.426	275	theta=3
t= 27.3001	1757.03	846.014	774.036	425.965	275	theta=1
t= 27.4001	4052.01	1678.51	1520.16	621.264	275	theta=1
t= 27.5001	3722.05	3887.03	3532.24	1170.19	275	theta=1
t= 27.6001	3013.2	3367.63	2472.14	2634.2	275	theta=1
t= 27.7001	1226.48	2119.84	-205.948	130.036	275	theta=1
t= 27.8001	-3397.05	-1085.29	-933.314	-59.2783	275	theta=1
t= 27.9001	-3080.51	-3238.78	-2896.37	-595.509	275	theta=1
t= 28.0001	-2396.38	-2738.44	-1867.64	-2023.17	275	theta=1
t= 28.1001	-658.768	-1527.57	738.973	414.358	275	theta=1
t= 28.2001	3847.71	1594.47	1448.46	598.004	275	theta=1
t= 28.3001	3543.7	3695.71	3364.94	1122.09	275	theta=1
t= 28.4001	2882.81	3213.25	2365.86	2515.29	275	theta=1
t= 28.5001	1191.8	2037.3	-173.017	140.889	275	theta=1
t= 28.6001	-3203.56	-1005.88	-865.447	-37.3816	275	theta=1
t= 28.7001	-2911.26	-3057.41	-2737.61	-549.852	275	theta=1
t= 28.8001	-2272.26	-2591.76	-1766.61	-1910.33	275	theta=1
t= 28.9001	-625.518	-1448.89	707.994	404.192	275	theta=1
t= 29.0001	3664.2	1519.34	1384.13	577.358	275	theta=1
t= 29.1001	3382.86	3523.53	3214.08	1078.7	275	theta=1
t= 29.2001	2764.52	3073.69	2269.69	2408.07	275	theta=1
t= 29.3001	1159.89	1962.2	-143.829	150.428	275	theta=1
t= 29.4001	-3029.27	-934.69	-804.385	-17.8871	275	theta=1
t= 29.5001	-2758.23	-2893.75	-2594.07	-508.559	275	theta=1
t= 29.6001	-2159.39	-2458.81	-1674.95	-1808.31	275	theta=1
t= 29.7001	-594.86	-1377.13	680.452	395.226	275	theta=1
t= 29.8001	3498.45	1451.79	1326.1	558.925	275	theta=1
t= 29.9001	3237.09	3367.77	3077.33	1039.36	275	theta=1

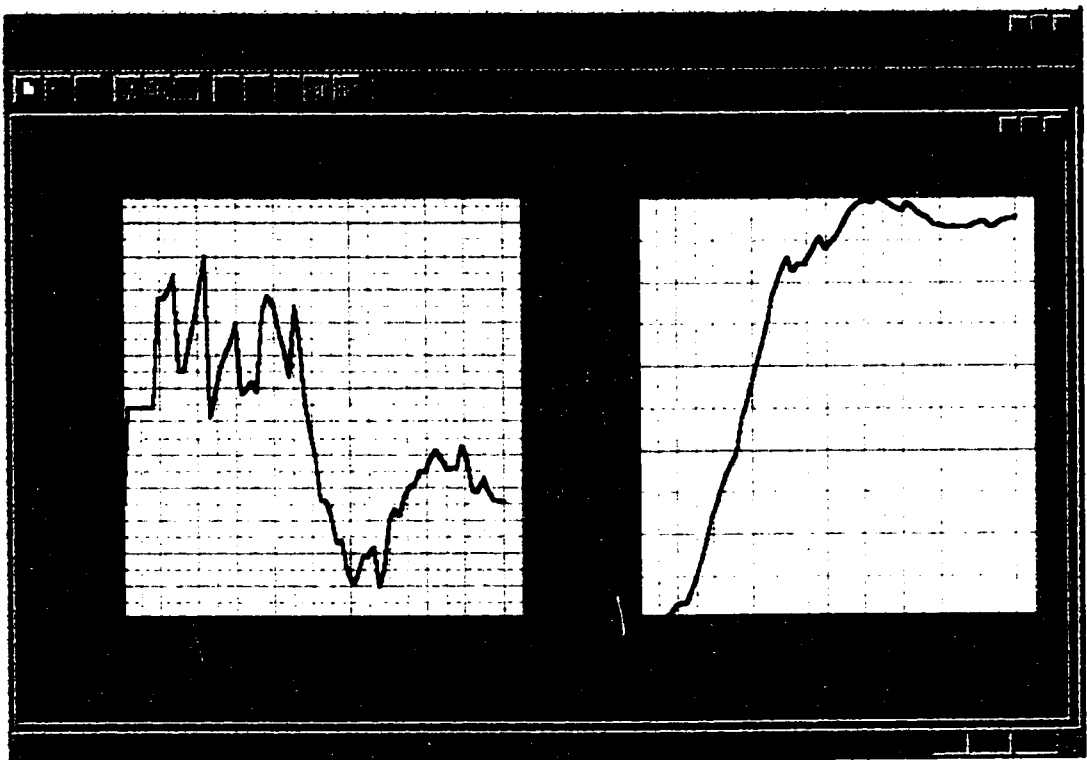
2. *Branching system with check valve and without check valve simulation*



**Fig. C 2 Indexing of Branching System**



**Fig. C 3 Graphic Display of Branching System Without Check Valve**



**Fig C 4 Graphic Display of Branching System With Check Valve**

# Simulation Result of Branching System without Check Valve:

Time	Qpump	Q 1	Q 2	Q 3	Hn 1	Hn 2	Hn 3
0	0	1.14192e-007	1.14192e-007	1.56917	73	73	70
0.275	0	0	-0.0935833	1.60888	73	79.0731	70
0.55	0	0	-0.0935833	1.60888	73	79.0731	70
0.825	0	0	-0.0935833	1.60888	73	79.0731	70
1.1	0	0	-0.0935834	1.60888	85.1462	79.0731	70
1.375	0	0	-0.0935834	1.60888	85.1462	79.0731	70
1.65	0.0377519	0.0377519	-0.0935834	1.60888	87.5963	79.0731	70
1.925	0.272221	0.272221	-0.00312854	1.64991	102.814	85.3492	70
2.2	0.379502	0.379502	-0.00312854	1.64991	109.777	85.3492	70
2.475	0.610683	0.610682	0.0333626	1.66647	124.781	87.8811	70
2.75	0.676052	0.676052	0.259966	1.76927	129.425	103.604	70
3.025	0.672864	0.672864	0.451206	1.85538	129.218	105.11	70
3.3	0.667551	0.667551	0.674531	1.95664	129.038	120.606	70
3.575	0.659796	0.659796	0.740656	1.98662	129.564	125.194	70
3.85	0.741184	0.741184	0.737577	1.98522	123.954	124.98	70
4.125	0.725837	0.725837	0.733677	1.98345	123.973	124.71	70
4.4	0.804642	0.804642	0.733846	1.98353	129.392	124.721	70
4.675	0.797343	0.797343	0.821851	2.02261	128.904	118.672	70
4.95	0.793823	0.793823	0.814584	2.01932	128.657	118.168	70
5.225	0.794058	0.794058	0.929453	2.071	128.673	121.235	70
5.5	0.88316	0.88316	1.14896	2.16787	122.707	106.016	70
5.775	0.883401	0.883401	1.25103	2.21179	122.689	98.9109	70
6.05	1.00356	1.00356	1.47464	2.30953	126.116	84.4173	70
6.325	1.22574	1.22574	1.53933	2.33766	111.107	80.0203	70
6.6	1.35946	1.35946	1.53624	2.33632	106.078	80.2193	70
6.875	1.79221	1.79221	1.61577	2.37225	105.228	86.2624	70
7.15	1.95541	1.95541	1.61134	2.37025	107.253	85.9322	70
7.425	2.15616	2.15616	1.63786	2.38104	120.703	87.8399	70
7.7	2.2082	2.2082	1.83279	2.46909	124.982	102.343	70
7.975	2.19891	2.19891	2.0056	2.54507	124.334	103.696	70
8.25	2.18903	2.18903	2.20012	2.62924	123.883	117.487	70
8.525	2.17567	2.17567	2.25582	2.65165	124.9	121.51	70
8.8	2.24767	2.24767	2.2517	2.64518	119.751	120.529	70
9.075	2.22523	2.22523	2.2436	2.64009	119.492	119.965	70
9.35	2.29249	2.29249	2.24465	2.64063	124.287	120.049	70
9.625	2.2897	2.2897	2.32434	2.67467	123.39	114.375	70
9.9	2.28447	2.28447	2.30702	2.66697	123.01	113.79	70
10.175	2.28976	2.28976	2.4021	2.70805	123.371	116.751	70
10.45	2.37062	2.37062	2.59609	2.79046	117.8	103.051	70
10.725	2.36633	2.36633	2.68616	2.82593	118.058	96.4724	70
11	2.46717	2.46717	2.8882	2.90779	121.428	84.0282	70
11.275	2.6576	2.6576	2.94264	2.92799	107.572	80.1446	70
11.55	2.76895	2.76895	2.93352	2.91975	102.416	80.7188	70
11.825	3.14218	3.14218	2.99902	2.94802	101.206	86.346	70
12.1	3.2828	3.2828	2.98868	2.94341	102.96	85.4775	70
12.375	3.44428	3.44428	3.00209	2.94695	114.64	86.649	70
12.65	3.4815	3.4815	3.15969	3.01823	118.457	99.3088	70
12.925	3.46657	3.46657	3.30895	3.08117	117.285	100.43	70
13.2	3.45231	3.45231	3.46981	3.14687	116.661	112.066	70
13.475	3.4364	3.4364	3.51298	3.16061	118.099	115.476	70
13.75	3.49728	3.49727	3.50688	3.14706	113.539	113.737	70
14.025	3.47096	3.47096	3.49426	3.13681	113.064	113.013	70
14.3	3.52553	3.52554	3.49716	3.13428	117.241	113.234	70
14.575	3.52767	3.52767	3.56576	3.16164	116.034	108.138	70
14.85	3.5197	3.5197	3.53818	3.14966	115.607	107.759	70
15.125	3.52915	3.52915	3.61066	3.17936	116.257	110.741	70
15.4	3.59851	3.59851	3.77213	3.2468	111.248	99.1493	70
15.675	3.58754	3.58754	3.8468	3.27302	111.936	93.3417	70
15.95	3.66578	3.66578	4.01953	3.33897	115.332	83.3456	70
16.225	3.81672	3.81672	4.06164	3.35108	103.147	80.072	70
16.5	3.90241	3.90241	4.04644	3.33452	98.1016	81.0431	70
16.775	4.20745	4.20745	4.09793	3.35372	96.8375	86.1359	70
17.05	4.32297	4.32297	4.08303	3.34334	98.3774	84.6693	70
17.325	4.44398	4.44398	4.08357	3.33968	108.221	85.1227	70
17.6	4.4675	4.4675	4.20407	3.39481	111.523	95.7004	70
17.875	4.44973	4.44973	4.32789	3.44566	109.86	96.6234	70
18.15	4.4325	4.4325	4.45432	3.49605	109.154	106.032	70
18.425	4.41701	4.41701	4.48498	3.50229	110.943	108.851	70
18.7	4.46615	4.46615	4.47694	3.48571	107.078	106.567	70
18.975	4.43795	4.43795	4.46073	3.47169	106.485	105.806	70
19.25	4.47975	4.47975	4.46659	3.46539	110.052	106.213	70
19.525	4.48527	4.48527	4.52339	3.48531	108.647	101.814	70
19.8	4.47457	4.47457	4.48785	3.46645	108.235	101.72	70
20.075	4.48674	4.48674	4.538	3.48524	109.057	104.713	70

# Simulation Results of Branching System With Check Valve:

Time	Qpump	Q 1	Q 2	Q 3	Hn 1	Hn 2	Hn 3	Theta
0	0	0-0.0007	1.56917	73	73	70	0	
0.275	0	0-0.0935833	1.60888	73	79.0731	70	0	
0.55	0	0-0.0935833	1.60888	73	79.0731	70	0	
0.825	0	0-0.0935833	1.60888	73	79.0731	70	0	
1.1	0	0-0.0935834	1.60888	85.1462	79.0731	70	0	
1.375	0	0-0.0935834	1.60888	85.1462	79.0731	70	0	
1.650	0.006825580	0.00682557-0.0935834	1.60888	85.5891	79.0731	70	0	
1.925	0.0195362	0.0195363-0.00312854	1.64991	86.4141	85.3492	70	0	
2.2	0.0222236	0.0222237-0.00312854	1.64991	86.5885	85.3492	70	0	
2.475	0.0226339	0.02263390.00346921	1.65291	86.6151	85.807	70	1.14728	
2.75	0.0432003	0.0432002	0.0157554	1.65848	88.3557	86.6594	70	2.45847
3.025	0.0646612	0.0646612	0.105952	1.69885	89.7484	81.1548	70	3.88487
3.3	0.0896989	0.0896988	0.106349	1.69903	91.4028	81.1823	70	5.42649
3.575	0.116614	0.116614	0.129248	1.70941	93.2048	82.7712	70	7.08334
3.85	0.154208	0.154207	0.149989	1.71881	84.2876	84.2103	70	8.8554
4.125	0.17379	0.17379	0.174407	1.72989	85.5604	85.9045	70	10.7427
4.4	0.201284	0.201284	0.20083	1.74187	87.4479	87.7379	70	12.7452
4.675	0.225749	0.225749	0.243108	1.76038	89.1294	78.5156	70	14.8629
4.95	0.239633	0.239633	0.262045	1.76896	90.1406	79.8296	70	17.0959
5.225	0.254572	0.254572	0.295984	1.78428	91.2294	81.2978	70	19.4441
5.5	0.307753	0.307753	0.332616	1.80078	82.7164	82.1885	70	21.9074
5.775	0.331941	0.331941	0.351385	1.80852	84.3721	82.882	70	24.4861
6.05	0.367872	0.367872	0.367104	1.81564	85.9714	83.9194	70	27.1799
6.325	0.396242	0.396242	0.352317	1.80874	86.3276	79.8161	70	29.989
6.6	0.42283	0.42283	0.397081	1.82884	87.5298	80.1348	70	32.9132
6.875	0.455104	0.455104	0.450772	1.85293	89.643	80.5789	70	35.9527
7.15	0.499634	0.499634	0.493561	1.87205	89.3897	79.9971	70	39.1075
7.425	0.521096	0.521096	0.469079	1.86053	88.1986	84.5133	70	40
7.7	0.542931	0.542931	0.519345	1.8831	86.5783	85.456	70	40
7.975	0.559138	0.559138	0.566455	1.90409	84.2739	85.0316	70	40
8.25	0.536847	0.536847	0.59257	1.91552	88.9303	83.5366	70	40
8.525	0.547622	0.547623	0.605402	1.92043	87.3129	82.499	70	40
8.8	0.55078	0.55078	0.611409	1.92297	84.0383	80.8555	70	40
9.075	0.563886	0.563886	0.5981	1.91709	81.7008	84.9307	70	40
9.35	0.580297	0.580297	0.615738	1.92456	80.8968	82.8666	70	40
9.625	0.590685	0.590685	0.622695	1.92707	79.5383	79.3451	70	40
9.9	0.571265	0.571265	0.628561	1.9292	83.2155	77.4765	70	40
10.175	0.582997	0.582997	0.651297	1.93899	80.7694	76.2602	70	40
10.45	0.590672	0.590672	0.677947	1.95042	77.2951	73.8447	70	40
10.725	0.602241	0.602241	0.716176	1.96702	75.7976	73.7801	70	40
11	0.619727	0.619727	0.705667	1.96177	74.2425	72.7772	70	40
11.275	0.633863	0.633863	0.682315	1.95031	71.0169	71.3171	70	40
11.55	0.640692	0.640692	0.667751	1.94361	68.9167	71.5155	70	40
11.825	0.646714	0.646714	0.68772	1.95264	68.9865	69.7974	70	40
12.1	0.650592	0.650592	0.664229	1.9415	69.2925	69.014	70	40
12.375	0.644466	0.644466	0.629686	1.92543	70.0375	69.6001	70	40
12.65	0.651505	0.651505	0.622409	1.9217	67.4816	70.5329	70	40
12.925	0.649189	0.649189	0.628957	1.92408	68.071	70.6651	70	40
13.2	0.645616	0.645616	0.627146	1.92258	70.665	71.1305	70	40
13.475	0.640202	0.640202	0.629979	1.92288	71.7181	68.847	70	40
13.75	0.634267	0.634267	0.620749	1.91839	71.0402	69.8857	70	40
14.025	0.631113	0.631113	0.616153	1.91574	71.4194	72.5465	70	40
14.3	0.643397	0.643397	0.615618	1.9155	69.7488	73.2833	70	40
14.575	0.638172	0.638172	0.605071	1.91066	71.0468	72.9055	70	40
14.85	0.629381	0.629381	0.589051	1.90318	73.4347	74.1216	70	40
15.125	0.624751	0.624751	0.569718	1.89425	73.9055	74.5022	70	40
15.4	0.619412	0.619412	0.580061	1.89851	73.865	74.7895	70	40
15.675	0.612309	0.612309	0.597808	1.90592	75.6586	75.4548	70	40
15.95	0.609109	0.609109	0.601584	1.90697	77.0851	75.38	70	40
16.225	0.607126	0.607126	0.583483	1.89796	76.573	76.1688	70	40
16.5	0.605215	0.605215	0.596413	1.90352	75.9632	76.6621	70	40
16.775	0.608659	0.608659	0.622614	1.91471	75.8672	76.1796	70	40
17.05	0.604917	0.604917	0.622949	1.91482	77.5868	75.5172	70	40
17.325	0.605115	0.605115	0.609513	1.90879	77.2546	75.656	70	40
17.6	0.607986	0.607986	0.611397	1.90947	75.2594	75.661	70	40
17.875	0.611235	0.611235	0.6156	1.91134	74.7863	76.865	70	40
18.15	0.613409	0.613409	0.620443	1.91303	75.9375	76.2311	70	40
18.425	0.614153	0.614153	0.62062	1.91233	75.8686	74.4107	70	40
18.7	0.606559	0.606559	0.620002	1.9114	76.307	74.1883	70	40
18.975	0.609613	0.609613	0.626424	1.91361	75.5573	75.0634	70	40
19.25	0.616907	0.616907	0.636271	1.91765	74.1991	74.4032	70	40
19.525	0.61883	0.61883	0.645335	1.92093	74.1417	73.7606	70	40
19.8	0.619087	0.619087	0.636566	1.91697	74.617	73.7784	70	40
20.075	0.622764	0.622764	0.624336	1.91158	73.5571	73.6875	70	40

## Appedix D

### **Head File for HydrauAnalysis And HydrauGraphic**

```

// //////////////////////////////////////
//
// @ FrontPageFormView.h : interface of class
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @ Concordia University
//
// //////////////////////////////////////

#ifdef AFX_FRONTPAGEFORMVIEW_H__9F6EBCB8_AEDE_11D2_A4E3_00A0C974979A__INCLUDE
D_)
#define
AFX_FRONTPAGEFORMVIEW_H__9F6EBCB8_AEDE_11D2_A4E3_00A0C974979A__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// FrontPageFormView.h : header file
//
#include "DefinedData.h"
// CFrontPageFormView form view

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif

class CFrontPageFormView : public CFormView
{
protected:
    CFrontPageFormView(); // protected constructor used by dynamic
creation
    DECLARE_DYNCREATE(CFrontPageFormView)

// Form Data
public:
    //{AFX_DATA(CFrontPageFormView)
    enum { IDD = IDD_FRONT_PAGE };
    // NOTE: the ClassWizard will add data members here
    //}AFX_DATA

// Attributes
public:

    RESERVIOR_DATA            ReserviorData;
    CONTROL_VALVE_DATA        ControlValvData;
    SIMULATION_DATA           SimulationData;
    PIPE_LINE_DATA            pipeLineData[17]; // you have 17 pipes
    JUNCTION_DATA             JunctionData[16]; // sixteen now
    PUMP_CHECK_VALVE_DATA     PumpCheckValvData[3]; // 3 now
    PLOT_PRINTOUT_DATA        PlotPrintoutData;
    CONECTION_DATA            ConectionData;
    INDEX_DATA                IndexData[200];

// Operations
public:

    BOOL SaveModel(CString szPath);
    CString m_szFilePath;
    BOOL m_nSaved;

```

```

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CFrontPageFormView)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
//}}AFX_VIRTUAL

// Implementation
protected:
virtual ~CFrontPageFormView();
#ifdef _DEBUG
virtual void AssertValid() const;
virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
//{{AFX_MSG(CFrontPageFormView)
afx_msg void OnPipeLineParam1();
afx_msg void OnPipeLineParam2();
afx_msg void OnReservorParam();
afx_msg void OnSimulationParam();
afx_msg void OnJunctionParam1();
afx_msg void OnJunctionParam2();
afx_msg void OnStart();
afx_msg void OnCtrlValveParam();
afx_msg void OnPumpCheckValveParam();
afx_msg void OnFileOpen();
afx_msg void OnFileSave();
afx_msg void OnFileSaveAs();
afx_msg void OnPlotParam();
afx_msg void OnConect();
afx_msg void OnButIndex1();
afx_msg void OnButIndex2();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef(AFX_FRONTPAGEFORMVIEW_H__9F6EBCB8_AEDE_11D2_A4E3_00A0C974979A__INCLUDE
D_)

```

```

/////////////////////////////////////////////////////////////////
//
// @ CheckValve.h: interface for the CheckValve class.
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
/////////////////////////////////////////////////////////////////

#if !defined(APX_CHECKVALVE_H__22E8C968_DDEB_11D2_986A_444553540000__INCLUDED_)
#define APX_CHECKVALVE_H__22E8C968_DDEB_11D2_986A_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define KDIRECTTOT 1
#define KDIRECTSEAT 2
#define KDIRECTINTERM 3
#define KDIRECTEXIT 4
#define DIRECTARM 5
#define KREVTOT 6
#define KREVSEAT 7
#define KREVINTERM 8
#define KREVEXIT 9
#define REVERSEARM 10

#define MAXANGLE 70

class CheckValve
{
public:
    CheckValve();
    virtual ~CheckValve();

    float fFlowTorque;
    float fDiscTorque;
    float fCTWTorque;
    float fTotalTorque;
    float fDiscAngularVelocity;
    float fDensity;

    float fPipeDiameter;//D1
    float fPipeArea;
    float fSeatDiameter;//D2
    float fSeatArea;
    float fDiscDiameter;//DCL
    float fDiscArea;
    float fShaftOffset;//DIs
    float fHingeLength;//ALA
    float fHingeWidth;//2 * XD
    float fDiscMomentOfInertia;//AJV
    float fDiscWeight;//WV
    float fDiscCenterOfGravity;//ALG
    int iDiscMaxAngle;//TETAf
    float fDiscOffsetAngle;//ATETAf
    float fPipeFriction;
    float fExitingOrificeDiameter;
    float fIntermediateOrificeDiameter;
    float fPipeToSeatK;
    float fIntermediateOrificeK;
    float fExitingOrificeK;

```

```

float fValveTotalK;
float fCTWWeight;
float fCTWLever;
float fCTWAngle;
float fCTWMomentOfInertia;
float fSpringArmInitialAngle;
float fSpringArmLength;
float fSpringPivotShaftDistance;
float fSpringExtension;
float fSpringExtensionFactor;
float fRestedSpringLength;
float fExtensionSpringTorque;
float fCoilSpringFactor; // Nm/o
float fCoilSpringInitialAngle;//o
float fCoilSpringTorque; //Nm

float fCoefficientsOfResistance[MAXANGLE][10];
float GeometricParameters(float);
float DirectFlow(float);
float ReverseFlow();
float FlowTorque(float, float, float );
float DirectTorqueArm(int);
float ReverseTorqueArm(int);
float DiscTorque(float);
float CTWTorque(float);
float TorqueFromExtensionSpring(float);
float TorqueFromCoilSpring(float);
float ValveDynamics(float, float, float);
float ValveDynamics(float, float, float, float);
float GetDiscAngle();
void CheckValveInit(float, int);

float GetCoefficientsOfResistance();
float GetTotalTorque(float, float);
float GetValveFlow(float, float, float);
void rkl(float *, float, float, float *, float, float (*)(float), float[], float
[], int);
float R_K4(float, float, float, float, float, float);

private:
float fDiscAnglePub;

};

#endif //
!defined(AFX_CHECKVALVE_H__22E8C968_DDEB_11D2_986A_444553540000__INCLUDED_)

```

```

// //////////////////////////////////////
//
// @ ChildFrm.h : interface of the CChildFrame class
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
// //////////////////////////////////////

#ifndef AFX_CHILDFRM_H__B66D4D5F_49D1_11D3_BD31_00E029112567__INCLUDED_
#define AFX_CHILDFRM_H__B66D4D5F_49D1_11D3_BD31_00E029112567__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CChildFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CChildFrame)
public:
    CChildFrame();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChildFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CChildFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
    //{{AFX_MSG(CChildFrame)
    // NOTE - the ClassWizard will add and remove member functions
    here.
    //      DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

// //////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#endif AFX_CHILDFRM_H__B66D4D5F_49D1_11D3_BD31_00E029112567__INCLUDED_

```

```

// ////////////////////////////////////////
//
// @ CconnectionPara.h : interface of class
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
// ////////////////////////////////////////

#ifndef AFX_CONNECTIONPARA_H__AC5C5122_E120_11D2_986A_444553540000__INCLUDED_
#define AFX_CONNECTIONPARA_H__AC5C5122_E120_11D2_986A_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// CconnectionPara.h : header file

// ////////////////////////////////////////
// CConnectionPara dialog

class CConnectionPara : public CDialog
{
// Construction
public:
    CConnectionPara(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    #define IDD_DIALOG_CONNECTION 1001
    enum { IDD = IDD_DIALOG_CONNECTION };
    UINT m_fNumPipe;
    UINT m_fNumJunct;
    UINT m_fNumPump;
    UINT m_fNumIndex;
    UINT m_nDowJNpipe02;
    UINT m_nDowJNpipe01;
    UINT m_nDowJNpipe03;
    UINT m_nDowJNpipe04;
    UINT m_nDowJNpipe05;
    UINT m_nDowJNpipe06;
    UINT m_nDowJNpipe07;
    UINT m_nDowJNpipe08;
    UINT m_nDowJNpipe09;
    UINT m_nDowJNpipe10;
    UINT m_nDowJNpipe11;
    UINT m_nDowJNpipe12;
    UINT m_nDowJNpipe13;
    UINT m_nDowJNpipe14;
    UINT m_nDowJNpipe15;
    UINT m_nDowJNpipe16;
    UINT m_nDowJNpipe17;
    UINT m_nDowJNpipe18;
    UINT m_nDowJNpipe19;
    UINT m_nDowJNpipe20;
    UINT m_nDowJNpipe21;
    UINT m_nDowJNpipe22;
    UINT m_nDowJNpipe23;
    UINT m_nDowJNpipe24;
    UINT m_nDowJNpipe25;
    UINT m_nDowJNpipe26;
    UINT m_nDowJNpipe27;

```

```

UINT    m_nDowJNpipe28;
UINT    m_nDowJNpipe29;
UINT    m_nDowJNpipe30;

UINT    m_nMinReach;
float    m_fSoundSpeed;
UINT    m_nUpJNpipe01;
UINT    m_nUpJNpipe02;
UINT    m_nUpJNpipe03;
UINT    m_nUpJNpipe04;
UINT    m_nUpJNpipe05;
UINT    m_nUpJNpipe06;
UINT    m_nUpJNpipe07;
UINT    m_nUpJNpipe08;
UINT    m_nUpJNpipe09;
UINT    m_nUpJNpipe10;
UINT    m_nUpJNpipe11;
UINT    m_nUpJNpipe12;
UINT    m_nUpJNpipe13;
UINT    m_nUpJNpipe14;
UINT    m_nUpJNpipe15;
UINT    m_nUpJNpipe16;
UINT    m_nUpJNpipe17;
UINT    m_nUpJNpipe18;
UINT    m_nUpJNpipe19;
UINT    m_nUpJNpipe20;
UINT    m_nUpJNpipe21;
UINT    m_nUpJNpipe22;
UINT    m_nUpJNpipe23;
UINT    m_nUpJNpipe24;
UINT    m_nUpJNpipe25;
UINT    m_nUpJNpipe26;
UINT    m_nUpJNpipe27;
UINT    m_nUpJNpipe28;
UINT    m_nUpJNpipe29;
UINT    m_nUpJNpipe30;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CConnectionPara)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CConnectionPara)
    afx_msg void OnButIndex1();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_CONECTIONPARA_H__AC5C5122_E120_11D2_986A_444553540000__INCLUDED_)

```

```

// //////////////////////////////////////
//
// @ CtrlValveDlg.h : interface of class
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
// //////////////////////////////////////

#if
!defined(AFX_CTRLVALVEDLG_H__76576583_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_)
#define AFX_CTRLVALVEDLG_H__76576583_C0BD_11D2_A4E4_00A0C974979A__INCLJDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// CtrlValveDlg.h : header file
// //////////////////////////////////////
// CCtrlValveDlg dialog

class CCtrlValveDlg : public CDialog
{
// Construction
public:
    CCtrlValveDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    //({AFX_DATA(CCtrlValveDlg)
    enum { IDD = IDD_DLG_CTRL_VALVE };
    float m_fTau1;
    float m_fTau2;
    float m_fTau3;
    float m_fTau4;
    float m_fTau5;
    float m_fTau6;
    float m_fTau7;
    long m_nTime;
    //})AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //({AFX_VIRTUAL(CCtrlValveDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //})AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //({AFX_MSG(CCtrlValveDlg)
    // NOTE: the ClassWizard will add member functions here
    //})AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//({AFX_INSERT_LOCATION})
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.
#endif //
!defined(AFX_CTRLVALVEDLG_H__76576583_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_)

```

```

// ////////////////////////////////////////
//
// @ DefinedData.h : interface of class
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
// ////////////////////////////////////////

#ifndef DEFINEDDATA_H
#define DEFINEDDATA_H

typedef struct {
    float fHeadR1;
    float fHeadR2;
    float fHeadR3;
    float fHeadR4;
} RESERVIOR_DATA;

typedef struct {
    float fTau[7];
    int nTime;
} CONTROL_VALVE_DATA;

typedef struct {
    float fStartTime;
    float fStopTime;
} SIMULATION_DATA;

typedef struct {
    UINT fPlotJunNum;
    UINT fPlotPipeNum;

    UINT fPrintoutPipe1;
    UINT fPrintoutPipe2;
    UINT fPrintoutPipe3;
    UINT fPrintoutJun1;
    UINT fPrintoutJun2;
    UINT fPrintoutJun3;
} PLOT_PRINTOUT_DATA;

typedef struct{
    UINT nNumPipe;
    UINT nNumJunct;
    UINT nNumPump;
    UINT nNumIndex;
    UINT nMinReach;
    float fSoundSpeed;
    // int fIndex[200];
    UINT nUpJNPipe[40];
    UINT nDwJNPipe[40];
} CONECTION_DATA;

typedef struct {

    int nIndex;

} INDEX_DATA;

```

```

typedef struct {
    float fDiameter;
    float fFriction;
    float fLength;
} PIPE_LINE_DATA;

typedef struct {
    int nJuncType;
    float fElevation;
    float fFlow;
    float fHead;
} JUNCTION_DATA;

typedef struct {
    int nPumpStatus; // startup, fail, normal, stop
    int nDiameter;    // always effective, it is 0 based index, it is not
the real diameter value
    //you selected, but it is the position, starting from 0, this is
true
    //for the specified speed also
    int nCheckvalve;
    float fA0;          // effective on startup
    float fA1;          // .....
    float fA2;          //.....
    float fHs;          //.....
    float fTs;          //.....
    float fFlow;        // effective on fail
    float fHead;        //.....
    float fTorq;        //.....
    float fSpeed;
    int nSpecifiedSpeed; //.....
    float fNormalA0;     // effective on normal
    float fNormalA1;
    float fNormalA2;
} PUMP_CHECK_VALVE_DATA;

#endif

```

```

// //////////////////////////////////////
//
// @ FrontWnd.h : interface of class
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @ Concordia University
//
// //////////////////////////////////////

#if !defined(AFX_FRONTWND_H__59E54C02_B1FD_11D2_A4E3_00A0C974979A__INCLUDED_)
#define AFX_FRONTWND_H__59E54C02_B1FD_11D2_A4E3_00A0C974979A__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// FrontWnd.h : header file
//

// //////////////////////////////////////
// CFrontWnd window

class CFrontWnd : public CWnd
{
// Construction
public:
    CFrontWnd();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFrontWnd)
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CFrontWnd();

    // Generated message map functions
protected:
    //{{AFX_MSG(CFrontWnd)
    afx_msg void OnPaint();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

// //////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_FRONTWND_H__59E54C02_B1FD_11D2_A4E3_00A0C974979A__INCLUDED_)

```

```

/////////////////////////////////////////////////////////////////
//
// @ HydroAnalysis.h: main header file for the HYDROANALYSIS application
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_HYDROANALYSIS_H__B66D4D59_49D1_11D3_BD31_00E029112567__INCLUDED_
#define AFX_HYDROANALYSIS_H__B66D4D59_49D1_11D3_BD31_00E029112567__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // main symbols

/////////////////////////////////////////////////////////////////
// CHydroAnalysisApp:
// See HydroAnalysis.cpp for the implementation of this class
//

class CHydroAnalysisApp : public CWinApp
{
public:
    CHydroAnalysisApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CHydroAnalysisApp)
    public:
        virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CHydroAnalysisApp)
    afx_msg void OnAppAbout();
    // NOTE - the ClassWizard will add and remove member functions here.
    //      DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#endif(AFX_HYDROANALYSIS_H__B66D4D59_49D1_11D3_BD31_00E029112567__INCLUDED_)

```

```

/////////////////////////////////////////////////////////////////
//
// @ HydroAnalysisDoc.h : interface of the CHydroAnalysisDoc class
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @ Concordia University
//
/////////////////////////////////////////////////////////////////
#ifndef AFX_HYDROANALYSISDOC_H__B66D4D61_49D1_11D3_BD31_00E029112567__INCLUDED_
#define AFX_HYDROANALYSISDOC_H__B66D4D61_49D1_11D3_BD31_00E029112567__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CHydroAnalysisDoc : public CDocument
{
protected: // create from serialization only
    CHydroAnalysisDoc();
    DECLARE_DYNCREATE(CHydroAnalysisDoc)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CHydroAnalysisDoc)
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CHydroAnalysisDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    // Generated message map functions
protected:
    //{{AFX_MSG(CHydroAnalysisDoc)
    // NOTE - the ClassWizard will add and remove member functions here.
    // DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.
#endif //
#ifndef AFX_HYDROANALYSISDOC_H__B66D4D61_49D1_11D3_BD31_00E029112567__INCLUDED_
#endif

```

```

/////////////////////////////////////////////////////////////////
//
// @ HydroAnalysisView.h : interface of the CHydroAnalysisView class
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @ Concordia University
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_HYDROANALYSISVIEW_H__B66D4D63_49D1_11D3_BD31_00E029112567__INCL
    UDED_
#define
AFX_HYDROANALYSISVIEW_H__B66D4D63_49D1_11D3_BD31_00E029112567__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CHydroAnalysisView : public CView
{
protected: // create from serialization only
    CHydroAnalysisView();
    DECLARE_DYNCREATE(CHydroAnalysisView)

// Attributes
public:
    CHydroAnalysisDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CHydroAnalysisView)
public:
    virtual void OnDraw(CDC* pDC); // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CHydroAnalysisView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CHydroAnalysisView)
    // NOTE - the ClassWizard will add and remove member functions here.
    // DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

```

```

#ifndef _DEBUG // debug version in HydroAnalysisView.cpp
inline CHydroAnalysisDoc* CHydroAnalysisView::GetDocument()
    { return (CHydroAnalysisDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_HYDROANALYSISVIEW_H__B66D4D63_49D1_11D3_BD31_00E029112567__INCLUDE
D_)

```

```

// //////////////////////////////////////
//
// @ IndexPara_Dlg.h : header file
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @ Concordia University
//
// //////////////////////////////////////

#ifdef _MSC_VER
#define AFX_INDEXPARA_DLG_H__B0EFC8E2_E3C6_11D2_986A_444553540000__INCLUDED_
#define AFX_INDEXPARA_DLG_H__B0EFC8E2_E3C6_11D2_986A_444553540000__INCLUDED_

#endif

#pragma once
#endif // _MSC_VER >= 1000

// CIndexPara_Dlg dialog

class CIndexPara_Dlg : public CDialog
{
// Construction
public:
    CIndexPara_Dlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    #ifdef AFX_DATA(CIndexPara_Dlg)
    enum { IDD = IDD_DLG_INDEX1 };
    int m_nIndex01;
    int m_nIndex02;
    int m_nIndex03;
    int m_nIndex04;
    int m_nIndex05;
    int m_nIndex06;
    int m_nIndex07;
    int m_nIndex08;
    int m_nIndex09;
    int m_nIndex10;
    int m_nIndex11;
    int m_nIndex12;
    int m_nIndex13;
    int m_nIndex14;
    int m_nIndex15;
    int m_nIndex16;
    int m_nIndex17;
    int m_nIndex18;
    int m_nIndex19;
    int m_nIndex20;
    int m_nIndex21;
    int m_nIndex22;
    int m_nIndex23;
    int m_nIndex24;
    int m_nIndex25;
    int m_nIndex26;
    int m_nIndex27;
    int m_nIndex28;
    int m_nIndex29;
    int m_nIndex30;
    int m_nIndex31;
    int m_nIndex32;
    int m_nIndex33;
    #endif

```

```

        int         m_nIndex34;
        int         m_nIndex35;
        int         m_nIndex36;
        int         m_nIndex37;
        int         m_nIndex38;
        int         m_nIndex39;
        int         m_nIndex40;
        int         m_nIndex41;
        int         m_nIndex42;
        int         m_nIndex43;
        int         m_nIndex44;
        int         m_nIndex45;
        int         m_nIndex46;
        int         m_nIndex47;
        int         m_nIndex48;
        int         m_nIndex49;
        int         m_nIndex50;

    ///}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CIndexPara_Dlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    ///}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(CIndexPara_Dlg)
        // NOTE: the ClassWizard will add member functions here
    ///}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{AFX_INSERT_LOCATION}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_INDEXPARA_DLG_H__B0EFC8E2_E3C6_11D2_986A_444553540000__INCLUDED_

```

```

// //////////////////////////////////////
//
// @ Junction1Dlg.h : header file
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @      Concordia University
//
// //////////////////////////////////////

#ifdef AFX_JUNCTION1DLG_H__76576585_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_
#define AFX_JUNCTION1DLG_H__76576585_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
//
// //////////////////////////////////////
// CJunction1Dlg dialog

class CJunction1Dlg : public CDialog
{
// Construction
public:
    CJunction1Dlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    #if !defined(AFX_DATA(CJunction1Dlg)
    enum { IDD = IDD_DLG_JUNCTION1 };
    BOOL    m_bCheck1;
    BOOL    m_bCheck2;
    BOOL    m_bCheck3;
    BOOL    m_bCheck4;
    BOOL    m_bCheck5;
    BOOL    m_bCheck6;
    BOOL    m_bCheck7;
    BOOL    m_bCheck8;
    float    m_fElev1;
    float    m_fElev2;
    float    m_fElev3;
    float    m_fElev4;
    float    m_fElev5;
    float    m_fElev6;
    float    m_fElev7;
    float    m_fElev8;
    float    m_fFlw1;
    float    m_fFlw2;
    float    m_fFlw3;
    float    m_fFlw4;
    float    m_fFlw5;
    float    m_fFlw6;
    float    m_fFlw7;
    float    m_fFlw8;
    float    m_fHead1;
    float    m_fHead2;
    float    m_fHead3;
    float    m_fHead4;
    float    m_fHead5;
    float    m_fHead6;
    float    m_fHead7;
    float    m_fHead8;
    #endif AFX_DATA

```

```

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CJunction1Dlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    }AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(CJunction1Dlg)
    // NOTE: the ClassWizard will add member functions here
    }AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_JUNCTION1DLG_H__76576585_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_)

```

```

/////////////////////////////////////////////////////////////////
//
// @ MainFrm.h : interface of the CMainFrame class
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__B66D4D5D_49D1_11D3_BD31_00E029112567__INCLUDED_)
#define AFX_MAINFRM_H__B66D4D5D_49D1_11D3_BD31_00E029112567__INCLUDED_
#if _MSC_VER >= 1000
// #include "FrontWnd.h"
#pragma once
#endif // _MSC_VER >= 1000

#include "FrontWnd.h"

class CMainFrame : public CMDIFrameWnd
{
    DECLARE_DYNAMIC(CMainFrame)
public:
    CMainFrame();

// Attributes
public:
// Operations
public:
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL
// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // control bar embedded members
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;
// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    // NOTE - the ClassWizard will add and remove member functions here.
    // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CFrontWnd* m_pFrontWnd;
};
/////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#endif(AFX_MAINFRM_H__B66D4D5D_49D1_11D3_BD31_00E029112567__INCLUDED_)

```

```

// //////////////////////////////////////
//
// @ Pipe.h: interface for the Pipe class.
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @ Concordia University
//
// //////////////////////////////////////

#if !defined(AFX_PIPE_H__22E8C962_DDEB_11D2_986A_444553540000__INCLUDED_)
#define AFX_PIPE_H__22E8C962_DDEB_11D2_986A_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
////////////////////////////////////
#define NPipe 20
#define NPump 3

#define PI 3.1415927
#define DEGTO RAD 0.0174533
#define RADTODEG 57.29578

#include <string.h>
#include <math.h>

#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <fstream.h>
#include <iomanip.h>

#define NumReach 4
#define g 9.81/32.2

class Pipe
{
public:
    Pipe();
    virtual ~Pipe();
    void PipeInit(int,int,int,float,float[],float[], float[],int[], int[]);

    float B[NPipe],R[NPipe],A[NPipe],Dt;
    int Nreach[NPipe+NPump],Nup[NPipe+NPump],Ndn[NPipe+NPump];

    void PipeJuncF_H(int);
    float PipeJuncHead();

    void ValveDownF_H(float,int,const float[],float,float);
    void ValveTauDownFH(float,float,int,const float[],float,float,float,float);

    float PumpJunc();
    void ResvUpF_H(float);
    float ResvUpHead(float);
    float GetFlow(int);
    float GetHead(int);
    void ReInitialize(int);
    float TimeStep();

```

```
private:
    float Q[NumReach+1];
    float H[NumReach+1];
    float Qp[NumReach+1];
    float Hp[NumReach+1];

};

#endif // !defined(AFX_PIPE_H__22E8C962_DDEB_11D2_986A_444553540000__INCLUDED_)
```

```

// //////////////////////////////////////
//
// @ Pipe1Dlg.h : header file
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
// //////////////////////////////////////

#if !defined(AFX_PIPE1DLG_H__76576588_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_)
#define AFX_PIPE1DLG_H__76576588_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
//
//

// //////////////////////////////////////
// CPipe1Dlg dialog

class CPipe1Dlg : public CDialog
{
// Construction
public:
    CPipe1Dlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(CPipe1Dlg)
    enum { IDD = IDD_DLG_PIPE_LINE1 };
    float m_fDiameter1;
    float m_fDiameter2;
    float m_fDiameter3;
    float m_fDiameter4;
    float m_fDiameter5;
    float m_fDiameter6;
    float m_fDiameter7;
    float m_fDiameter8;
    float m_fDiameter9;
    float m_fFriction1;
    float m_fFriction2;
    float m_fFriction3;
    float m_fFriction4;
    float m_fFriction5;
    float m_fFriction6;
    float m_fFriction7;
    float m_fFriction8;
    float m_fFriction9;
    float m_fLength1;
    float m_fLength2;
    float m_fLength3;
    float m_fLength4;
    float m_fLength5;
    float m_fLength6;
    float m_fLength7;
    float m_fLength8;
    float m_fLength9;
    }}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides

```

```

//{{AFX_VIRTUAL(CPipe1Dlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
//{{AFX_MSG(CPipe1Dlg)
    // NOTE: the ClassWizard will add member functions here
//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_PIPE1DLG_H__76576588_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_)

```

```

// ////////////////////////////////////////
//
// @ PipeNet2Cv.h: interface for the PipeNet2Cv class.
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @ Concordia University
//
// ////////////////////////////////////////

#ifndef AFX_PIPENET2CV_H__714869C5_EB39_11D2_986A_444553540000__INCLUDED_
#define AFX_PIPENET2CV_H__714869C5_EB39_11D2_986A_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define NumJ 20
#define NumP 20
#define NIND 200
#define KM 11

#include "pipe.h"

#include "PumpStart.h"
#include "CheckValve.h"

class PipeNet2Cv :
public Pipe,
public PumpStart,
public CheckValve
{
public:
    PipeNet2Cv();
    virtual ~PipeNet2Cv();

    PipeNet2Cv(int, int[], int, int, int, int, float, float[], float[], float[], int[], int[],
               float[], float[], float[], int[], float, int, float, float, float, float);
    void InitialF_H(int);
    void PipeJuncF_H(int, int);
    void PumpJuncF_H(float, int);
    void PumpJuncF_H(float, int, float);

    float GetHN(int);
    float GetQN(int);
    float GetH(int, int);
    float GetQ(int, int);
    float Z[NumJ], CN[NumJ], BN[NumJ], QE[NumP+1], IND[NIND];
    float CC[NumJ][10], BC[NumJ][10];
    int Ntype[NumJ], Ipp[NumJ];

private:
    float QN[NumJ], HN[NumJ], H[2*(NumP+1)+1][KM], Q[2*(NumP+1)+1][KM];
};

#endif //
#ifndef AFX_PIPENET2CV_H__714869C5_EB39_11D2_986A_444553540000__INCLUDED_

```

```

// //////////////////////////////////////
//
// @ PipeNet2NoCv.h: interface for the PipeNet2NoCv class.
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @ Concordia University
//
// //////////////////////////////////////

#if
!defined(AFX_PIPENET2NOCV_H__22E8C967_DDEB_11D2_986A_444553540000__INCLUDED_)
#define AFX_PIPENET2NOCV_H__22E8C967_DDEB_11D2_986A_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
#define NumJ 20
#define NumP 20
#define NIND 200
#define KM 11

#include "Pipe.h"
#include "PumpStart.h"

class PipeNet2NoCv : public Pipe , public PumpStart
{
public:
    PipeNet2NoCv();

    PipeNet2NoCv(int,int[],int,int,int,int,
        float,float[],float[],float[],int[],int[],float[],
        float[],float[],int[],float,float,float,float);
    virtual ~PipeNet2NoCv();
void InitialF_H(int);
void PipeJuncF_H(int,int);
void PumpJuncF_H(float,int);
float QN[NumJ],HN[NumJ],Z[NumJ],CN[NumJ],EN[NumJ],
    QE[NumP+1]/*,Nup[NumP+1],Ndn[NumP+1]*/,IND[NIND];
float CC[NumJ][10],BC[NumJ][10];
int Ntype[NumJ],Ipp[NumJ];

//private:
    float H[2*(NumP+1)+1][KM],Q[2*(NumP+1)+1][KM];

};

#endif //
!defined(AFX_PIPENET2NOCV_H__22E8C967_DDEB_11D2_986A_444553540000__INCLUDED_)

```

```

// ////////////////////////////////////////
//
// @ PlotPrintoutDlg.h : header file
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
// ////////////////////////////////////////
#if
!defined(AFX_PLOTPRINTOUTDLG_H__5BAE5188_E097_11D2_986A_444553540000__INCLUDED_
)
#define AFX_PLOTPRINTOUTDLG_H__5BAE5188_E097_11D2_986A_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ////////////////////////////////////////
// PlotPrintoutDlg dialog

class PlotPrintoutDlg : public CDialog
{
// Construction
public:
    PlotPrintoutDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    {{{AFX_DATA(PlotPrintoutDlg)
    enum { IDD = IDD_DLG_PLOTDLG };
    UINT    m_fPlotJunNum;
    UINT    m_fPlotPipeNum;
    UINT    m_fPrintoutPipe1;
    UINT    m_fPrintoutPipe2;
    UINT    m_fPrintoutPipe3;
    UINT    m_fPrintoutJun2;
    UINT    m_fPrintoutJun1;
    UINT    m_fPrintoutJun3;
    }}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    {{{AFX_VIRTUAL(PlotPrintoutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    }}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    {{{AFX_MSG(PlotPrintoutDlg)
    // NOTE: the ClassWizard will add member functions here
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

{{{AFX_INSERT_LOCATION}}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_PLOTPRINTOUTDLG_H__5BAE5188_E097_11D2_986A_444553540000__INCLUDED_
)

```

```

// ////////////////////////////////////////
//
// @ PumpCheckValveDlg.h : header file
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
// ////////////////////////////////////////

#if
!defined(AFX_PUMPCHECKVALVEDLG_H__76576584_C0BD_11D2_A4E4_00A0C974979A__INCLUDE
D_)
#define
AFX_PUMPCHECKVALVEDLG_H__76576584_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
//////////////////////////////////////
// CPumpCheckValveDlg dialog

class CPumpCheckValveDlg : public CDialog
{
// Construction
public:
    CPumpCheckValveDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    {{{(AFX_DATA(CPumpCheckValveDlg)
    enum { IDD = IDD_DLG_PUMP_CHECK_VALVE };
    CComboBox    m_ListSpeed3;
    CComboBox    m_ListSpeed2;
    CComboBox    m_ListSpeed1;
    CComboBox    m_ListDiameter3;
    CComboBox    m_ListDiameter2;
    CComboBox    m_ListDiameter1;
    float m_fA0;
    float m_fA1;
    float m_fA2;
    float m_fNormalA0;
    float m_fNormalA1;
    float m_fNormalA2;
    float m_fHsPump1;
    float m_fHsPump2;
    float m_fHsPump3;
    float m_fFlowPump1;
    float m_fFlowPump2;
    float m_fFlowPump3;
    float m_fHeadPump1;
    float m_fHeadPump2;
    float m_fHeadPump3;
    float m_fSpeedPump1;
    float m_fSpeedPump2;
    float m_fSpeedPump3;
    float m_fTorqPump1;
    float m_fTorqPump2;
    float m_fTorqPump3;
    float m_fTsPump1;
    float m_fTsPump2;
    float m_fTsPump3;
    float m_fPump2A0;
    float m_fPump3A0;

```

```

float m_fPump2A1;
float m_fPump3A1;
float m_fPump2A2;
float m_fPump3A2;
float m_fPump2NorA0;
float m_fPump3NorA0;
float m_fPump2NorA1;
float m_fPump3NorA1;
float m_fPump2NorA2;
float m_fPump3NorA2;
int m_nPump1Status;
int m_nPump2Status;
int m_nPump3Status;
BOOL m_bNoCheckValve;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CPumpCheckValveDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

public:

    // index for the diameter, and speed
int m_nSpeed3;
int m_nSpeed2;
int m_nSpeed1;
int m_nDiameter3;
int m_nDiameter2;
int m_nDiameter1;

// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CPumpCheckValveDlg)
afx_msg void OnPump1Radio1();
afx_msg void OnPump1Radio2();
afx_msg void OnPump1Radio3();
afx_msg void OnPump1Radio4();
afx_msg void OnPump2Radio1();
afx_msg void OnPump2Radio2();
afx_msg void OnPump2Radio3();
afx_msg void OnPump2Radio4();
afx_msg void OnPump3Radio1();
afx_msg void OnPump3Radio2();
afx_msg void OnPump3Radio3();
afx_msg void OnPump3Radio4();
virtual BOOL OnInitDialog();
afx_msg void OnSelchangeComboDiameter1();
afx_msg void OnSelchangeComboDiameter2();
afx_msg void OnSelchangeComboDiameter3();
afx_msg void OnSelchangeComboSpeed();
afx_msg void OnSelchangeComboSpeed2();
afx_msg void OnSelchangeComboSpeed3();
afx_msg void OnCheckvalve();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    void SetControls();
    void ShowStartupWndPl( BOOL bShow = TRUE);
    void ShowNormalWndPl( BOOL bShow = TRUE);

```

```

void ShowFailWndP1( BOOL bShow = TRUE);
void ShowStartupWndP2( BOOL bShow = TRUE);
void ShowNormalWndP2( BOOL bShow = TRUE);
void ShowFailWndP2( BOOL bShow = TRUE);
void ShowStartupWndP3( BOOL bShow = TRUE);
void ShowNormalWndP3( BOOL bShow = TRUE);
void ShowFailWndP3( BOOL bShow = TRUE);
void ShowCheckvalve1Wnd(BOOL bShow =TRUE);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_PUMPCHECKVALVEDLG_H__76576584_C0BD_11D2_A4E4_00A0C974979A__INCLUDE
D_)

```

```

/////////////////////////////////////////////////////////////////
//
// @ PumpFail.h: interface for the PumpFail class.
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_PUMPFALL_H__549E1951_0B07_11D4_BD5E_00E029112567__INCLUDED_)
#define AFX_PUMPFALL_H__549E1951_0B07_11D4_BD5E_00E029112567__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class PumpFail
{
public:
    PumpFail();
    PumpFail(int);
    virtual ~PumpFail();

public:

    float A, XL, EL, D, F, QR,HR,TR,RN,WRR, G,TM,TOL,VI,R,B,AR,CK;//V
    float DX,V0,V00,AL0,AL00,DT,BETA,BET0;
    float WH[89],WB[89],Q[5],H[5],Hp[5],Qp[5];
    int N,KIT,KMAX;

    void PipeJHQ(int );
    void PumpValveEC(int);
    void DownstreamResoiv(float,int );
    void ReInitialize(int);

    double InterpolatedValue(double, int, int);

};

#endif //
!defined(AFX_PUMPFALL_H__549E1951_0B07_11D4_BD5E_00E029112567__INCLUDED_)

```

```

/// //////////////////////////////////////
//
// @ PumpStart.h: interface for the PumpStart class.
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
////////////////////////////////////

#ifndef AFX_PUMPSTART_H__22E8C964_DDEB_11D2_986A_444553540000__INCLUDED_
#define AFX_PUMPSTART_H__22E8C964_DDEB_11D2_986A_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class PumpStart
{
public:
    PumpStart();

    virtual ~PumpStart();
    void PumpStartInit(float, float, float, float, int, int);

    int Jpump;
    float HS, A1, A2, Ts;

};

#endif //
#ifndef AFX_PUMPSTART_H__22E8C964_DDEB_11D2_986A_444553540000__INCLUDED_

```

```

// ////////////////////////////////////////
//
// @ ReserviorDlg.h : header file
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @          Concordia University
//
// ////////////////////////////////////////
#ifdef AFX_RESERVIORDLG_H__76576582_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_
#define AFX_RESERVIORDLG_H__76576582_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
//
// ////////////////////////////////////////
// CReserviorDlg dialog

class CReserviorDlg : public CDialog
{
// Construction
public:
    CReserviorDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
//{{AFX_DATA(CReserviorDlg)
enum { IDD = IDD_DLG_RESERVOIR };
float m_fHead1;
float m_fHead2;
float m_fHead3;
float m_fHead4;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CReserviorDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
//{{AFX_MSG(CReserviorDlg)
// NOTE: the ClassWizard will add member functions here
//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifdef AFX_RESERVIORDLG_H__76576582_C0BD_11D2_A4E4_00A0C974979A__INCLUDED_

```

```

// ////////////////////////////////////////
//
// @ SimulationDlg.h : header file
// @ author: Tianhe Wen, 1999
// @ Version:2.0
// @ Copyright: Ritepro Inc.
// @      Concordia University
//
// ////////////////////////////////////////
#ifdef AFX_SIMULATIONDLG_H_DC9B7C92_C0D1_11D2_A4E4_00A0C974979A__INCLUDED_
#define AFX_SIMULATIONDLG_H_DC9B7C92_C0D1_11D2_A4E4_00A0C974979A__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
//
//

// ////////////////////////////////////////
// CSimulationDlg dialog

class CSimulationDlg : public CDialog
{
// Construction
public:
    CSimulationDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    #if !defined(AFX_DATA(CSimulationDlg)
    enum { IDD = IDD_DLG_SIMULATION };
    float m_fStartTime;
    float m_fStopTime;
    #endif AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    #if !defined(AFX_VIRTUAL(CSimulationDlg)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    #endif AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    #if !defined(AFX_MSG(CSimulationDlg)
    #endif AFX_MSG
    DECLARE_MESSAGE_MAP()
};

// #if !defined(AFX_INSERT_LOCATION)
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifdef AFX_SIMULATIONDLG_H_DC9B7C92_C0D1_11D2_A4E4_00A0C974979A__INCLUDED_

```

```

/////////////////////////////////////////////////////////////////
//
// @ HydroGraphic.h : main header file for the HYDROGRAPHIC application
// @ Author: Tianhe Wen, 1999
// @ Version: 1.5
// @ Copyright: Ritepro Inc.
// @           Concordian University
//
/////////////////////////////////////////////////////////////////

#if
!defined(AFX_HYDROGRAPHIC_H_F63D25E5_4682_11D3_BD31_00E029112567__INCLUDED_)
#define AFX_HYDROGRAPHIC_H_F63D25E5_4682_11D3_BD31_00E029112567__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // main symbols
#include "qwin.h"
#include "qcwrt.h"

/////////////////////////////////////////////////////////////////
// CHydroGraphicApp:
// See HydroGraphic.cpp for the implementation of this class

class CHydroGraphicApp : public CWinApp
{
public:
    CHydroGraphicApp();
public:
-   HDATA    m_hData1;        // data set handles
    HDATA    m_hData2;
    realtype rNewVals [8];

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CHydroGraphicApp)
public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation
    //{{AFX_MSG(CHydroGraphicApp)
    afx_msg void OnAppAbout();
    // NOTE - the ClassWizard will add and remove member functions here.
    //      DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////
extern CHydroGraphicApp theApp;
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_HYDROGRAPHIC_H_F63D25E5_4682_11D3_BD31_00E029112567__INCLUDED_)

```

```

/////////////////////////////////////////////////////////////////
//
// @ HydroGraphicDoc.h : interface of the CHydroGraphicDoc class
// @ Author: Tianhe Wen, 1999
// @ Version: 1.5
// @ Copyright: Ritepro Inc.
// @           Concordian University
//
/////////////////////////////////////////////////////////////////
#ifdef AFX_HYDROGRAPHICDOC_H__F63D25EB_4682_11D3_BD31_00E029112567__INCLUDED_
)
#define AFX_HYDROGRAPHICDOC_H__F63D25EB_4682_11D3_BD31_00E029112567__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CHydroGraphicDoc : public CDocument
{
protected: // create from serialization only
    CHydroGraphicDoc();
    DECLARE_DYNCREATE(CHydroGraphicDoc)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CHydroGraphicDoc)
    public:
    virtual void Serialize(CArchive& ar);
    //}AFX_VIRTUAL

// Implementation
public:
    virtual ~CHydroGraphicDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    virtual BOOL OnNewDocument();
    // Generated message map functions
protected:
    //{AFX_MSG(CHydroGraphicDoc)
    // NOTE - the ClassWizard will add and remove member functions here.
    // DO NOT EDIT what you see in these blocks of generated code !
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.
#endif //
#ifdef AFX_HYDROGRAPHICDOC_H__F63D25EB_4682_11D3_BD31_00E029112567__INCLUDED_
)

```

```

// ////////////////////////////////////////
//
// @ HydroGraphicView.h : interface of the CHydroGraphicView class
// @ Author: Tianhe Wen, 1999
// @ Version: 1.5
// @ Copyright: Ritepro Inc.
// @           Concordian University
//
// ////////////////////////////////////////

#if
!defined(AFX_HYDROGRAPHICVIEW_H__F63D25ED_4682_11D3_BD31_00E029112567__INCLUDED_)
#define AFX_HYDROGRAPHICVIEW_H__F63D25ED_4682_11D3_BD31_00E029112567__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

// ////////////////////////////////////////
// ...../
// define graph class based on BaseGraph
// ...../
class Graphs : public QCBASEGRAPH
{
public:    // constructor
    Graphs (int nNum);
    PGRAPH_DEF m_pDynGrDesc; // pointer to graph descriptor
    PGRAPH_DEF m_pStaGrDesc;
    int      m_nGraphNumber; // graph number in page
protected:
    virtual void BuildGraph (PGRAPH_DEF, HDC hdc); // user graph building
    function
};

class Page1 : public QCBASEPAGE
{
protected:
    virtual void BuildPage (PPAGE_DEF); // user function initializing page
public:
    Graphs* m_pGraph1;
    Graphs* m_pGraph2;

    Page1() {m_pGraph1 = m_pGraph2 = NULL;};
    ~Page1();
};

// ////////////////////////////////////////

class CHydroGraphicView : public CView
{
protected: // create from serialization only
    CHydroGraphicView();
    DECLARE_DYNCREATE(CHydroGraphicView)

protected:
    Page1 *m_pPg1;           // pointer to the page object
    Page1 *m_pPg2;
    PPAGE_DEF m_pPageDesc;
    PPAGE_DEF m_pPageDesc2;

```

```

// Attributes
public:
    CHydroGraphicDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CHydroGraphicView)
    public:
        virtual void OnDraw(CDC* pDC); // overridden to draw this view
        virtual void OnInitialUpdate(void);
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);
    //}AFX_VIRTUAL

// Implementation
public:
    virtual ~CHydroGraphicView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{AFX_MSG(CHydroGraphicView)
    afx_msg void OnSize(UINT nType, int cx, int cy);
    //afx_msg void OnFileSave();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CRect m_rcPrintRect;
};

#ifdef _DEBUG // debug version in HydroGraphicView.cpp
inline CHydroGraphicDoc* CHydroGraphicView::GetDocument()
{ return (CHydroGraphicDoc*)m_pDocument; }
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//{(AFX_INSERT_LOCATION)}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_HYDROGRAPHICVIEW_H_F63D25ED_4682_11D3_BD31_00E029112567__INCLUDED_

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// @ MainFrm.h : interface of the CMainFrame class
// @ Author: Tianhe Wen, 1999
// @ Version: 1.5
// @ Copyright: Ritepro Inc.
// @           Concordian University
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#if !defined(AFX_MAINFRM_H_F63D25E9_4682_11D3_BD31_00E029112567__INCLUDED_)
#define AFX_MAINFRM_H_F63D25E9_4682_11D3_BD31_00E029112567__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CMDIFrameWnd
{
protected: // create from serialization only

    DECLARE_DYNCREATE(CMainFrame)
// Attributes
public:
    CMainFrame();
// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
//    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL
// Implementation
private:
    int         m_nCount;

public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;
// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnDestroy();
    afx_msg void OnTimer(UINT nIDEvent);

    // NOTE - the ClassWizard will add and remove member functions here.
    //      DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.
#endif //
#endif(AFX_MAINFRM_H_F63D25E9_4682_11D3_BD31_00E029112567__INCLUDED_)

```