

AN INVESTIGATION INTO THE EFFICIENT
NUMERICAL INTEGRATION OF STIFF
HYDRAULIC SYSTEMS CONTAINING
DISCONTINUITIES

By

CRAIG EDWARD WENZEL

Bachelor of Science

Milwaukee School of Engineering

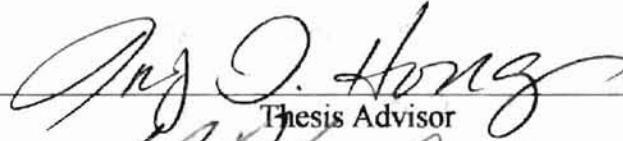
Milwaukee, Wisconsin

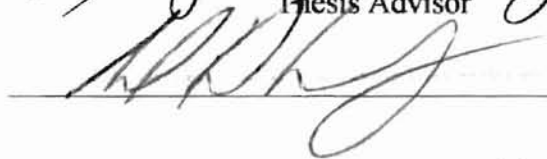
1990

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1999


AN INVESTIGATION INTO THE EFFICIENT
NUMERICAL INTEGRATION OF STIFF
HYDRAULIC SYSTEMS CONTAINING
DISCONTINUITIES

Thesis Approved:


Thesis Advisor



Prabhakar - pagilla


Dean of the Graduate College

ACKNOWLEDGMENT

I wish to express my sincere gratitude to my principle advisor, Dr. I. T. Hong, for his patience, expert guidance, and support during my thesis effort and throughout the course of my graduate study. My sincere appreciation also extends to Dr. R. K. Tessmann whose insightful suggestions and encouragement were invaluable. In addition, I wish to thank my committee members, Dr. Delahoussaye and Dr. Pagilla, for their time and recommendations. I also wish to thank FES\Bardyne Inc., Stillwater, Oklahoma, for providing excellent laboratory facilities and computing equipment. Particular appreciation is given to the faculty and staff of the Department of Mechanical and Aerospace Engineering for providing a quality educational experience. Finally, I wish to express my gratitude to my family and friends for their support throughout all of my academic pursuits.

TABLE OF CONTENTS

Chapter	Page
1. INTRODUCTION.....	1
Problem Statement.....	1
Objective of Study.....	3
2. LITERATURE REVIEW.....	5
Introduction.....	5
Mathematical Stiffness.....	5
Numerical Integration of Stiff ODEs.....	12
Discontinuities and Stiff ODE Solvers.....	14
Summary.....	17
3. MATHEMATICAL MODELS AND CUBIC SPLINES.....	19
Introduction.....	19
Background.....	19
Two-Point Cubic Splines.....	20
Four-Point Cubic Splines.....	24
Comparison of Two-Point and Four Point Splines.....	29
Application Examples.....	30
4. THE EVENT SWITCHING ALGORITHM.....	45
Background.....	45
Hydraulic System State Events.....	46
Development of the Event Switching Algorithm.....	51
Application Example.....	56
5. CASE STUDY: PILOT-OPERATED RELIEF VALVE.....	67
Introduction.....	67
Overview of Stiff ODE Solvers.....	67
The Pilot-Operated Relief Valve.....	69
Numerical Integration Evaluation.....	75
Discussion of Computation Time Results.....	83
Error Analysis.....	85
Discussion of Error Analysis Results.....	87
Summary.....	88

Chapter	Page
6. EXPERIMENTAL VERIFICATION.....	89
Introduction.....	89
Test Component.....	89
Test System.....	91
Mathematical Models and Differential Equations	92
Results of Computer Analysis.....	92
Test Results.....	93
Discussion of Results.....	95
7. CONCLUSION AND RECOMMENDATIONS.....	96
Conclusion.....	96
Suggestions for Further Study.....	98
Works Cited.....	100
Appendix A: Procedure for Generating Cubic Splines to Interpolate Between Four Points.....	102
Appendix B: Experimental Determination of the Pressure and Flow Relationship of an Orifice.....	105
Appendix C: MATLAB Script File for Example 3.1.....	109
Appendix D: MATLAB Script File for Example 3.2.....	113
Appendix E: MATLAB Script File for Example 4.1.....	120
Appendix F: MATLAB Script Files for Pilot-Operated Relief Valve Case Study from Chapter 5.....	127

LIST OF TABLES

Table		Page
5.1	Simulation Results for Example 4.1 Using Different Variations of Gear's Method.....	64
5.2	Compiled Test Results for Variations of the Pilot-Operated Relief Valve Problem.....	83
5.3	Error Analysis Results for Inlet Pressure Response of Test #4.....	86
B-1	Test Results for 0.012 Diameter Orifice.....	108

LIST OF FIGURES

Figure		Page
2.1	Orifice Pressure/Flow Relationship Using Turbulent Flow Equation.....	7
2.2	Fluid Volume Discharge to the Atmosphere through an Orifice.....	7
2.3	Orifice Flow Model with Linear Laminar Flow Region.....	9
2.4	Orifice Flow Model with Cubic Spline Laminar Flow Region.....	11
2.5	Volume Changes for an Extending Hydraulic Cylinder.....	12
2.6	Cubic Spline Used to Smooth a Discontinuity.....	15
3.1	Cubic Spline Interpolation of Data Points.....	20
3.2	Two Functions Requiring a Smooth Connecting Cubic Spline.....	20
3.3	Cubic Spline Providing Slope Continuity Between Two Functions.....	21
3.4	Effect of Moving a Single Data Point (A to A').....	25
3.5	Cubic Spline End Points and their Control Points.....	26
3.6	Cubic Spline Generated with Four-Point Form.....	27
3.7	Flow Chart for Generating Control Points.....	28
3.8	Comparison of Two-Point and Four-Point Cubic Splines.....	30
3.9	Experimental Results of Orifice Test.....	31
3.10	Comparison of Test Results and Turbulent Flow Equation.....	32
3.11	Cubic Spline Based Orifice Model.....	34
3.12	Refined Orifice Model.....	35
3.13	Typical Pressure Relief Valve.....	39
3.14	General Static Relief Valve Model.....	40
3.15	Smoothed Static Relief Valve Model.....	43
4.1	Spring-Mass-Damper System with Mechanical Stops.....	46
4.2	Three States for Spring-Mass-Damper System with Mechanical Stops.....	47
4.3	Flow Chart for Event Switching Algorithm as Applied to General Spring-Mass-Damper System.....	55

Figure	Page
4.4 Hydraulic Circuit Schematic for Example 4.1.....	56
4.5 Cylinder Rod Displacement Response for Example 4.1.....	65
4.6 Cylinder Rod Velocity Response for Example 4.1.....	65
4.7 Pressure Responses for Example 4.1.....	66
5.1 Pilot-Operated Relief Valve.....	69
5.2 Pilot-Operated Relief Valve Test Circuit.....	75
5.3 Pressure Response for Test #1.....	79
5.4 Displacement Response for Test #1.....	79
5.5 Pressure Response for Test #2.....	80
5.6 Displacement Response for Test #2.....	80
5.7 Pressure Response for Test #3.....	81
5.8 Displacement Response for Test #3.....	81
5.9 Pressure Response for Test #4.....	82
5.10 Displacement Response for Test #4.....	82
5.11 Comparison of Error Analysis Results.....	87
6.1 Typical Direct-Acting Pressure Relief Valve.....	90
6.2 Direct-Acting Relief Valve Test Circuit.....	91
6.3 Simulation Results for Direct-Acting Relief Valve.....	93
6.4 Test Results for Direct-Acting Relief Valve.....	94
6.5 Simulation and Test Results Superimposed.....	94
B-1 Hydraulic System for Orifice Testing.....	106

NOMENCLATURE

A	Area
b	Independent Axis Intercept of General Straight Line
b	Damping Ratio
BDF	Backward Differentiation Formula
C_d	Discharge Coefficient
D	Diameter
ESA	Event Switching Algorithm
f	Generic Function Notation
F	Force
$F_{preload}$	Spring Preload Force
h	Interval Size
k	Spring Constant
K_c	Flow Pressure Coefficient
k_v	Lumped Orifice Coefficient
m	Slope of General Straight Line
m	Mass
n	Integer Value Notation
NDF	Numerical Differentiation Formula

N_{Rt}	Transition Reynolds Number
ODE	Ordinary Differential Equation
P	Pressure
P_{cr}	Relief Valve Cracking Pressure
P_{max}	Relief Valve Maximum Pressure
Q	Volumetric Flow Rate
$Q_{@P_{cr}}$	Relief Valve Flow Rate at P_{cr}
Q_{max}	Relief Valve Flow Rate at P_{max}
RK	Runge-Kutta Method
S	Second Derivative (Curvature)
SIRK	Semi-Implicit Runge-Kutta Method
t	Time
V	Fluid Volume
w	Weight
α	Poppet Half-Angle
β	Bulk Modulus of Elasticity
δ	Laminar Flow Coefficient
ΔP	Differential Pressure
ΔP_{tf}	Differential Pressure for Fully Turbulent Flow
ε	Computational Precision
λ	Increment Factor
ν	Kinematic Fluid Viscosity

ρ

Fluid Density

τ

Time Constant

ϕ

Discontinuity Function

CHAPTER 1

INTRODUCTION

Problem Statement

Computer modeling and simulation techniques are important to designers in every engineering discipline. These techniques are particularly valuable to engineers concerned with the dynamic response of physical systems. A number of numerical integration methods are available to solve the ordinary differential equations (ODEs) which describe a dynamical system. In the computer environment, the effect of design changes on dynamic response can be determined without the financial consequences of complicated "breadboard" testing. However, the time required to simulate the dynamic response of a system does have some inherent costs. In addition, the results of these simulations are only as accurate as the mathematical models used to represent the physical system.

As with engineers in many other fields, fluid power system designers have turned to computer modeling and simulation to predict the dynamic response characteristics of hydraulic circuits. A hydraulic circuit uses pressurized fluid to provide a desired mechanical output. In addition, the various pressure and flow control components of a hydraulic system are mechanical in nature. A typical hydraulic circuit, therefore, is comprised of components with widely varying time constants. As a result, the differential equations which describe this combination of hydraulics and mechanics are often numerically *stiff*. Small time constants force the use of relatively small time steps during numerical integration. Because the differential equations are linked, the presence

of large time constants requires a great number of these small steps to capture the overall response. As a result, computation time becomes a concern. In the past, complex hydraulic circuits often required hours or days to complete a single simulation. Computation time must be reduced to enhance the value of computer analysis.

A number of numerical integration methods have been developed to decrease the computation time required to solve stiff differential equations. Gear's method for stiff ODEs and Rosenbrock's semi-implicit Runge-Kutta (SIRK) method are the most prominent among the stiff solvers[1]. The current state-of-the-art for both solvers features an adaptive step size algorithm to further improve computation time. Both of these methods require the calculation of Jacobian matrix for each time step. This Jacobian matrix is not required for standard explicit solvers like Euler's method and the Runge-Kutta (RK) method. The introduction of the Jacobian matrix creates a compatibility problem with hydraulic systems.

Physical constraints in a hydraulic system often cause stiff ODE solvers to fail. These constraints may be fluid-related or mechanical. For example, the turbulent flow equation has an infinite slope at the origin. This infinite slope will produce an ill-conditioned Jacobian matrix when a stiff ODE solver is used. Similarly, a limitation on the displacement of a mechanical component will produce a mathematical discontinuity in the Jacobian matrix during the integration process. These discontinuities often cause a stiff ODE solver to fail or cause the adaptive step size routine to slow the solver dramatically.

A few attempts have been made to apply stiff ODE solvers to the unique boundary constraints of hydraulic systems. Two different approaches have been proposed to eliminate the infinite slope at the origin of the turbulent flow equation. Ellman considers both laminar and turbulent orifice discharge coefficients with a cubic spline used to create a mathematically smooth transition between the two types of flow[2]. Piche', Ellman, and Vilenius also used a cubic spline approach in dealing directly with the laminar and turbulent flow equations[1]. Both of these methods are limited to orifice flow and cannot be adapted to model other flow related discontinuities. Bowns, Tomlinson and Dugdale have suggested a numerical integration technique to deal with mechanical discontinuities[3]. This technique involves step size halving and restarting procedures for the ODE solver when a discontinuity is encountered. However, the proposed method is limited to fixed step ODE solvers and requires a modification to the core solver algorithm.

Objective of Study

The purpose of this study is to develop a unique procedure that eliminates the numerical integration problems caused by the fluid-related and mechanical boundary conditions found in hydraulic systems. A unique curve fitting technique which uses control points is used to create mathematically smooth fluid flow models. The concept of control points involves positioning a point in such close proximity to the end point of a curve that the difference is physically negligible. However, the relative position of the end point and the control point determines the shape of the resulting curve. As such, the control point is physically insignificant but mathematically important. The versatility of

this technique allows hydraulic engineers to create a greater variety of fluid flow models. More importantly, the continuity and smoothness of the resulting mathematical models provide compatibility with stiff ODE solvers and significantly reduce computation time.

Numerical integration problems caused by mechanical boundary conditions will be eliminated by an *Event Switching Algorithm* (ESA). The ESA tracks the displacement of any given mass. If a physical limit is encountered by the mass, the initial conditions are reset and a revised set of ODEs are solved numerically. The sum of the forces acting on the mass is monitored while the mass is at the physical limit. If this sum of forces becomes unbalanced, the algorithm reverts back to the original set of ODEs and the solver is automatically restarted. By continuing in this manner, numerical integration proceeds unimpeded by the mathematical discontinuities associated with mechanical boundaries. In addition, the ESA requires no modification to the core ODE solver code. As a result, it is compatible with commercially available numerical integration software.

CHAPTER 2 Hydraulic systems

LITERATURE REVIEW

Introduction

As with any dynamic system, hydraulic circuits are mathematically defined by a series of ordinary differential equations (ODEs). Unlike most systems, however, hydraulic systems are generally dominated by non-linearities [3]. These highly non-linear ODEs are, for all practical purposes, impossible to solve with conventional analytical techniques [4]. Computerized numerical integration is the only practical approach to solving a set of ODEs accurately describing a hydraulic system. Unfortunately, the numerical integration of hydraulic systems is a notoriously slow process. The nature of hydraulic systems requires relatively small time steps and a great deal of computational effort.

Mathematical Stiffness

Hydraulic systems have been identified as mathematically *stiff* by a number of researchers [1,3,4,5,6,7,8]. Bowns et. al. [3] and Krus [6] define stiffness as a set of ODEs containing widely varying time constants. In other words, the system contains both rapidly and slowly varying transient solutions [8]. The effect of mathematical stiffness on numerical integration is well documented [1,3,4,5]. Piche' and Ellman [5] summarized this effect for hydraulic systems by stating, "Conventional explicit numerical integration methods such as classical Runge-Kutta schemes become numerically unstable unless a very small time increment is used, which leads to excessively long computation times".

Ellman [2] identified two sources of stiffness in hydraulic systems. The first source is stiffness inherent in turbulent flow through an orifice. An orifice is the most basic element in hydraulic control. Pressure and flow control in a hydraulic system are performed by components which use fixed and variable orifices. Orifice flow is dominated by turbulence and, therefore, laminar flow is typically ignored. Turbulent flow through an orifice is governed by the following relationship:

$$Q = C_d A \sqrt{\frac{2}{\rho} \Delta P} \quad (2.1)$$

Q = Flow Through the Orifice

C_d = Discharge Coefficient of the Orifice

A = Flow Area of the Orifice

ρ = Fluid Density

ΔP = Pressure Drop Across the Orifice

Discharge coefficient, flow area, and fluid density are normally considered to be constant. As such, the flow through an orifice is proportional to the square root of the pressure drop across it. A graph of this relationship for an arbitrary orifice is shown in Figure 2.1. The slope of the curve approaches infinity as the pressure drop approaches zero. This characteristic is responsible for mathematical stiffness. An example presented by Krus and Palmberg [7] effectively illustrates the problem. A volume of

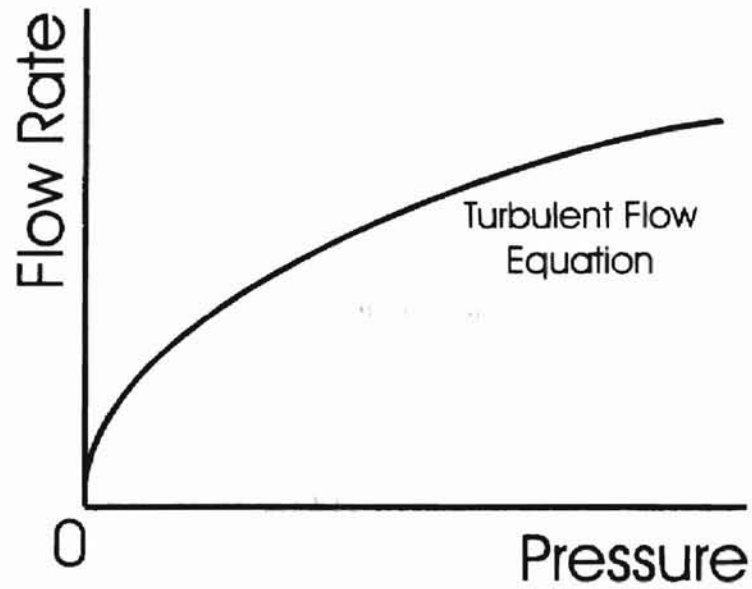


Figure 2.1: Orifice Pressure/Flow Relationship Using Turbulent Flow Equation

fluid is connected to the environment through an orifice as depicted in Figure 2.2. The

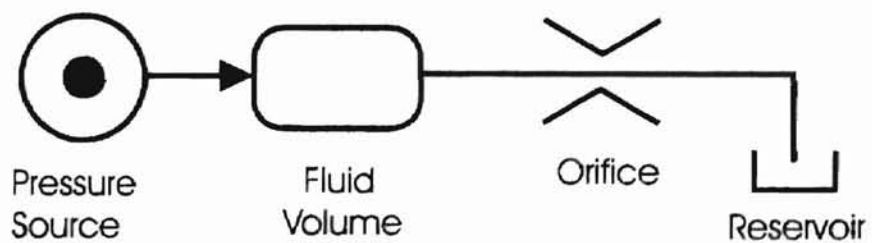


Figure 2.2: Fluid Volume Discharge to the Atmosphere through an Orifice

time constant for emptying the volume is described by the following equation:

$$\tau = \frac{V}{\beta K_c} \quad (2.2)$$

τ = Time Constant

V = Fluid Volume

β = Bulk Modulus of the Fluid

$K_c = \frac{\partial Q}{\partial P}$ = Flow Pressure Coefficient of the Orifice

If the turbulent orifice flow equation (2.1) is used, the term K_c becomes large when the pressure drop is small. The time constant, in turn, becomes small. In a numerical integration situation, relatively small time constants will lead to mathematical stiffness and long computation times. If the pressure drop is zero, K_c becomes infinite and the time constant goes to zero. A zero time constant creates infinite stiffness and causes all classical numerical integration methods to fail [4].

A second source of stiffness identified by Ellman [2] involves widely varying oil volumes within a single system. The elasticity of pressurized hydraulic fluid is dependent upon the volume of the trapped fluid. Flow passages within hydraulic control valves are orders of magnitude smaller than the hoses, tubes, housings, actuators, etc. used to contain and utilize the majority of the working fluid. The low elasticity of the fluid contained in these small passages is directly related to small time constants. Small time constants among relatively large time constants within a single system lead to mathematical stiffness.

Various researchers have developed methods to eliminate the stiffness problem caused by the turbulent orifice equation (2.1). Technically, the turbulent orifice equation is not appropriate at low pressure differences because the flow through an orifice is predominantly laminar under these conditions. This fact is usually ignored by designers because very low pressure drops seldom occur in steady state analysis. Dynamically, however, it is advantageous to introduce a laminar flow mechanism at low differential pressures to eliminate the infinite slope problem. Bowns, Tomlinson, and Dorey [8] proposed a small linear region about the origin as depicted in Figure 2.3. This region eliminates the infinite slope at the origin and accurately models the linearity of laminar flow. However, the authors admitted an inherent difficulty in determining the width of the laminar range.

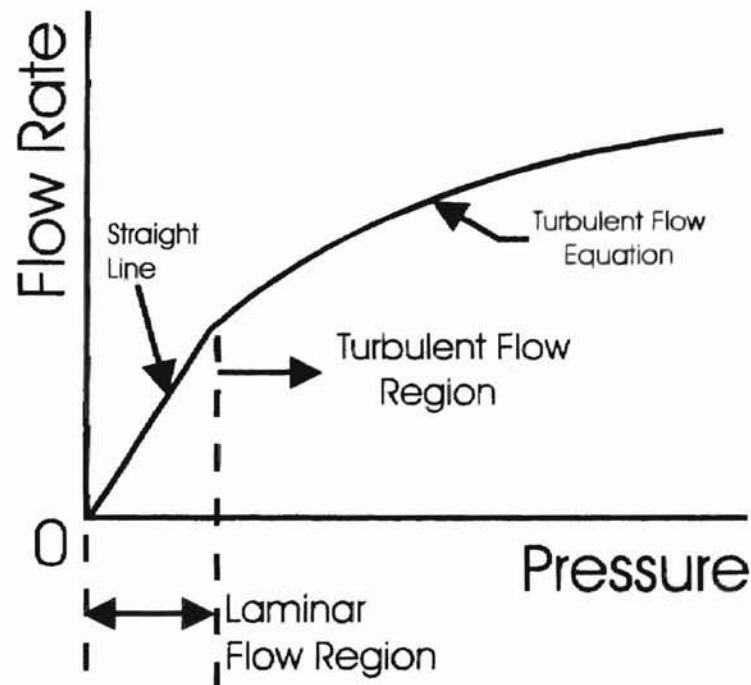


Figure 2.3: Orifice Flow Model with Linear Laminar Flow Region

If the region is not wide enough, mathematical stiffness results. An excessively wide linear region will lead to gross inaccuracy.

Based on studies conducted at the University of Bath [8], an improved laminar flow mechanism was developed at Tampere University of Technology in Finland. Ellman and Vilenius [2,9] proposed the use of a cubic spline to model laminar flow and the region of transition between laminar and turbulent flow. This model was later refined by Piche' and Ellman [5]. The resulting polynomial is presented below:

$$Q = \frac{A v N_{Rt}}{64D} \left\{ 45 \left(\frac{\Delta P}{\Delta P_{tf}} \right)^3 - 150 \left(\frac{\Delta P}{\Delta P_{tf}} \right)^2 + 225 \left(\frac{\Delta P}{\Delta P_{tf}} \right) \right\} \quad (2.3)$$

where, $0 \leq \Delta P \leq \Delta P_{tf}$

ν = Kinematic Viscosity of the Fluid

D = Diameter of the Orifice

N_{Rt} = Transition Reynolds Number

ΔP_{tf} = Pressure Difference for Fully Turbulent Flow

The transition to the turbulent flow equation (2.1) occurs at ΔP_{tf} as defined below:

$$\Delta P_{tf} = \frac{225 N_{Rt}^2 \rho \nu^2}{128 C_d^2 D^2} \quad (2.4)$$

The above laminar/transition flow model for an arbitrary orifice is depicted graphically in Figure 2.4. Piche' suggests a transition Reynolds number (N_{Rt}) of 1000. This model eliminates the infinite slope at the origin and provides slope continuity at the transition

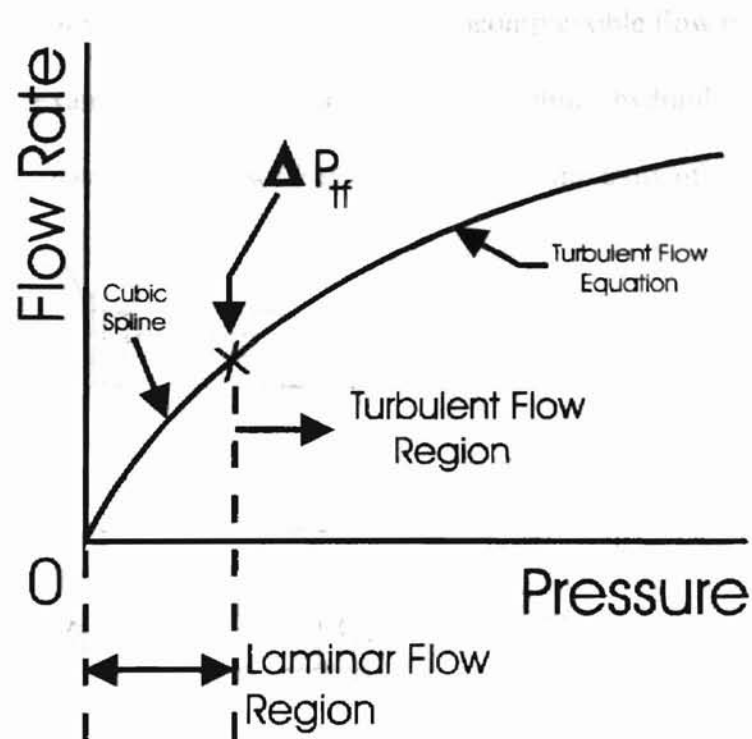


Figure 2.4: Orifice Flow Model with Cubic Spline Laminar Flow Region

point (ΔP_{ff}).

Bowns and Wang [4] proposed an incompressible flow model to eliminate the stiffness problem caused by small oil volumes. Pressure changes occur very rapidly under small volume conditions. The incompressible flow model considers these pressure changes to be instantaneous. In the absence of compressibility, the flow into a pressurized volume is equal to the flow exiting. Bowns and Wang implemented the incompressible flow model with an iterative approach based on this flow continuity property.

Several researchers pointed out an inherent problem with the incompressible flow model [1,5,6]. The dynamic nature of hydraulic systems requires oil volumes to change

during the course of simulation. As a result, the incompressible flow model is not valid at all times. For example, the volume of trapped oil within a hydraulic cylinder will increase as the actuator extends (see Figure 2.5). The complexity of the model must be

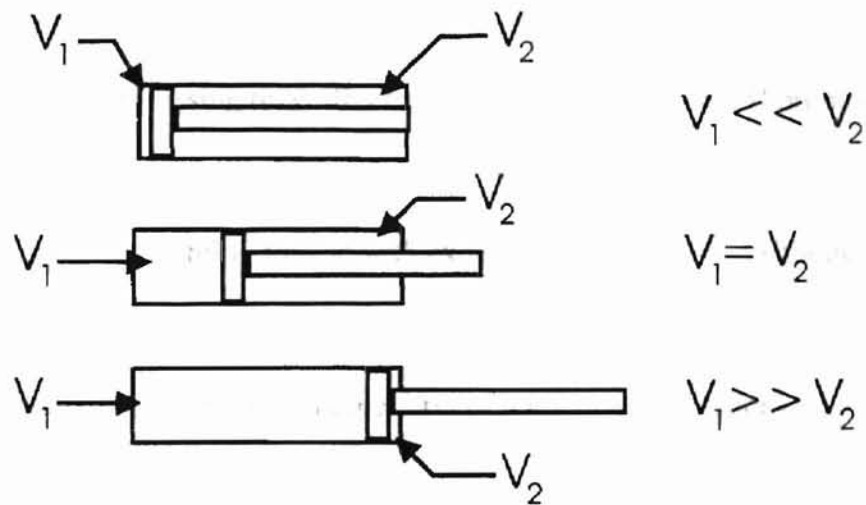


Figure 2.5: Volume Changes for an Extending Hydraulic Cylinder

increased to accommodate varying volumes by switching between compressible and incompressible flow models. As stated by Piche' and Ellman [5], "The problem remains as to how and when to make these transitions smoothly and automatically."

Numerical Integration of Stiff ODEs

In the absence of a sound modeling technique to eliminate stiffness created by widely varying oil volumes, researchers have investigated various numerical integration methods to improve computation time. Simulation efficiency can be improved by using a numerical ODE solver specifically designed for stiff systems of equations. Further efficiency gains may be realized by employing an adaptive step size algorithm.

Gear's method for solving stiff ODEs has been generally accepted as the most efficient algorithm for simulating hydraulic systems [3,8]. The stability features of this

ODE solver are superior to those of classic non-stiff solvers like Runge-Kutta. Mathematically, Gear's method for stiff ODEs is *A-stable*. A-stability means the numerical integration method is stable for any step size as long as the set of ODEs is stable [1]. This added stability allows the solver to use a much larger step size and, therefore, reduces processing time. Recently, Piche' and Ellman [1,5] proposed an *L-stable* Runge-Kutta method for use with fluid power systems. This method provides an even higher degree of stability for use with extremely stiff systems. Although Gear's method is A-stable, the degree of stability becomes small when the ODE system is extremely stiff and problems arise in the form of numerical oscillations. An L-stable method drives modal amplification to zero as the time constant approaches zero. As a result, the L-stable Runge-Kutta method eliminates numerical oscillations associated with certain class of hydraulic systems.

The efficiency gains realized by Gear's method for stiff ODEs and the L-stable Runge-Kutta method do not come without a price. Both of the methods require a significant amount of computational effort. A general system of ODEs may be written as follows.

$$\dot{x}_i = f(x_1, x_2, x_3, \dots, x_m, t) \quad (2.5)$$

In order to achieve improved stability, the numerical integration method must have some knowledge of f at each step [6]. This information is stored in a Jacobian matrix which contains partial derivatives of f with respect to the state variables. The presence of this Jacobian reintroduces the term $\frac{\partial Q}{\partial P}$. If the turbulent flow equation (2.1) is used as an orifice model, the trend toward an infinite slope at the origin will produce an ill-

conditioned Jacobian matrix at low pressure differences. This problem may be solved by using a laminar flow mechanism like the one developed by Piche' and Ellman (Eqs. 2.3 and 2.4).

To further improve processing time, an adaptive step size algorithm may be used in conjunction with a stiff ODE solver. The adaptive step size algorithm automatically adjusts the step length as integration proceeds. The size of the step is controlled by a preset error limit. Piche' and Ellman [5] states that adaptive step size control essentially eliminates numerical stability problems because the algorithm automatically selects a step length small enough to give an accurate solution. Improved efficiency is achieved because a small step size is used only when necessary. Relatively large steps may be used after the dynamics associated with small time constants are damped out.

Discontinuities and Stiff ODE Solvers

The physical nature of hydraulic systems, unfortunately, is not directly compatible with stiff ODE solvers or adaptive step size algorithms. Numerical integration problems arise in the form of discontinuities. Discontinuities affect both the core ODE solver and the adaptive step size algorithm. Abrupt changes to elements in the Jacobian matrix may produce a convergence problem within the integration routine and cause the method to fail [1]. An adaptive step size algorithm will "hunt" around the discontinuity until the step size is reduced sufficiently to cross the discontinuity within the preset error limit. This hunting involves a large number of unsuccessful function evaluations and results in considerable processing time [10].

Discontinuities may be present in the classical mathematical model of a physical entity. A simple solution to this problem is to change the mathematical model. Bowns, Tomlinson, and Dorey [8] have made extensive use of cubic splines to create smooth, continuous models. A generic example is depicted in Figure 2.6. The original model

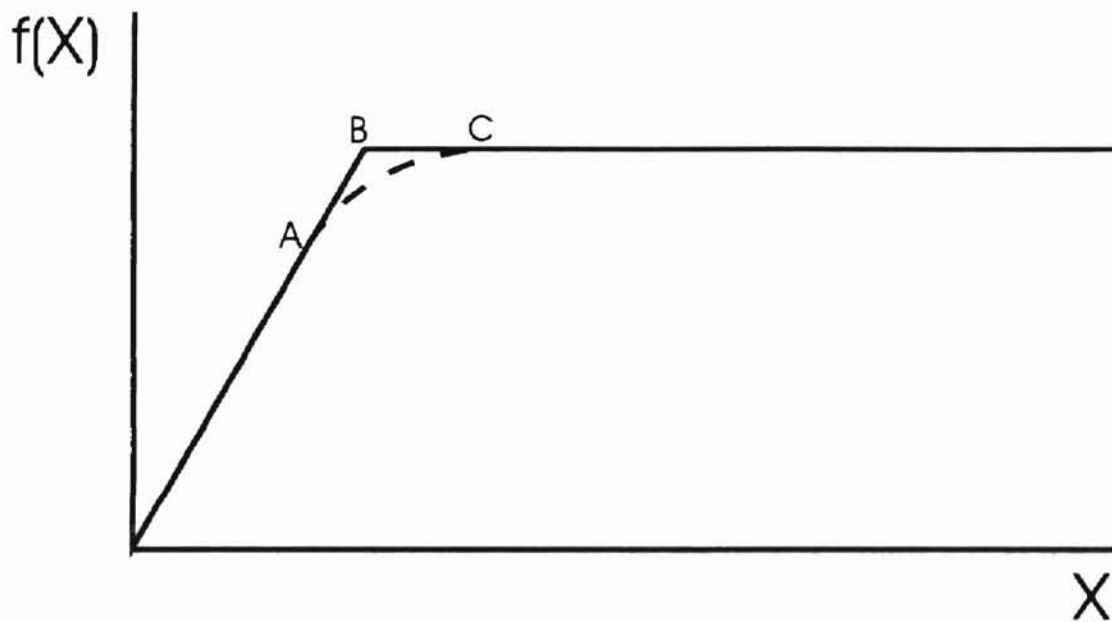


Figure 2.6 Cubic Spline Used to Smooth a Discontinuity

contains a hard non-linearity at point B. A cubic spline connected between points A and C is used to provide a smooth transition across the discontinuity. Some amount of accuracy is sacrificed to improve computational efficiency by allowing a more gradual change to the Jacobian matrix.

Discontinuities may also be encountered during the simulation process. Bowns, Tomlinson, and Dugdale [8] identified two types of discontinuities likely to cause problems for an ODE solver:

- 1) Discontinuities which occur at a known time.
- 2) Discontinuities which occur when a variable reaches a critical value.

Of these, the second discontinuity type is more difficult to handle because the exact time of threshold crossing is unknown. Ellison [11] has labeled this type of discontinuity as a *state event*. A state event involves a switching function which changes terms in the original set of ODEs and defines a new integration problem starting exactly at the switching point [10]. State events in hydraulic systems occur when actuators, loads, or internal valve parts encounter mechanical travel limits. For example, the velocity and acceleration of a hydraulic cylinder rod almost instantaneously drop to zero when the stroke limit is reached. Chaney [12] has recommended restarting the ODE solver when a state event is encountered. This procedure divides the original problem into continuous sections and solves them in a piecewise manner. Preston and Berzins [13] later endorsed this restarting procedure as absolutely necessary "whenever parts of a network suddenly become (in)active."

A difficulty arises in locating the exact time at which the critical value is reached. The critical value invariably falls between two successive time steps. Special measures must be taken to locate the time of the discontinuity, within acceptable error limits, before integration proceeds with a new set of ODEs. It is first necessary to identify a discontinuity by defining a *discontinuity function*. In practice, discontinuities are located

by monitoring sign changes [14]. For a set of ODEs of the form $\dot{x}_i = f(x_1, x_2, x_3, \dots, x_n, t)$, a discontinuity function has the form:

$$\phi_i = f(x_1, x_2, x_3, \dots, x_n, t) \quad (2.6)$$

where the discontinuity occurs at,

$$\phi_i = 0 \quad (2.7)$$

In order to locate the precise time of a zero crossing, a discontinuity handling algorithm is necessary. The goal of this algorithm is to control the step size such that the discontinuity occurs at the end of the step [14]. Chaney [12] proposed an iterative interval halving procedure (bisection) to locate the discontinuity within allowable error limits. More advanced techniques involve interpolation or regula falsi (false position) [11,13,14,15,16,17,18].

Summary

A review of the available literature has shown the value of using cubic splines. Potentially, cubic splines may be used to eliminate infinite slope problems and to smooth hydraulic model discontinuities. As a result, numerical integration of hydraulic systems can be made compatible with ODE solvers designed specifically for mathematically stiff ODEs. However, none of the available literature provides a simple, physically significant, method to generate a variety of cubic splines. Similarly, a physically significant algorithm for handling state events was not contained in the available literature. The complexity of stiff ODE solvers containing adaptive step size and event locating algorithms has forced designers to use commercially available numerical integration packages. These packages are generic and are not specifically designed for

hydraulic systems. An algorithm for handling typical hydraulic system state events would be extremely valuable to hydraulic system designers using commercial integrators.

In order to increase the value of computer analysis, steps must be taken to facilitate compatibility between hydraulic systems and stiff ODE solvers. A versatile cubic spline modeling tool is required to eliminate discontinuities in a variety of function-specific component models. In addition, a procedure for handling state events must be developed to realize the potential gains in computational efficiency offered by stiff ODE solvers.

CHAPTER 3

P. H. RAY, J.

MATHEMATICAL MODELS AND CUBIC SPLINES

Introduction

Infinite slopes and discontinuities in mathematical models are a source of problems for state-of-the-art numerical integration packages. Numerical ODE solvers specifically designed for mathematically stiff systems require a Jacobian matrix. Unfortunately, an infinite slope leads to an ill-conditioned Jacobian matrix and causes the solver to fail. Abrupt changes to the Jacobian matrix, in the form of model discontinuities, also cause the ODE solver to fail. In addition, adaptive step size algorithms become extremely inefficient when a discontinuity is encountered and considerable computation time is wasted.

Cubic splines have been used to model problem causing hydraulic system components with some success [5,8]. However, existing cubic spline models are function specific and cannot be used as a general modeling tool. It is necessary to develop a versatile method that can be used to eliminate a variety of infinite slope problems and to create a smooth bridge between any two discontinuous functions. In this way, the dynamic analysis of stiff hydraulic systems can be advanced.

Background

Typically, cubic splines are used as interpolating polynomials for a set of data points [19]. For $n+1$ data points, n third order polynomials are generated to interpolate between the data points as shown in Figure 3.1. At each interior point, the polynomials are continuous in position, slope (1st derivative), and curvature (2nd derivative). At the

end points of the data set, no joining polynomial exists. As a result, the slope and curvature are not constrained.

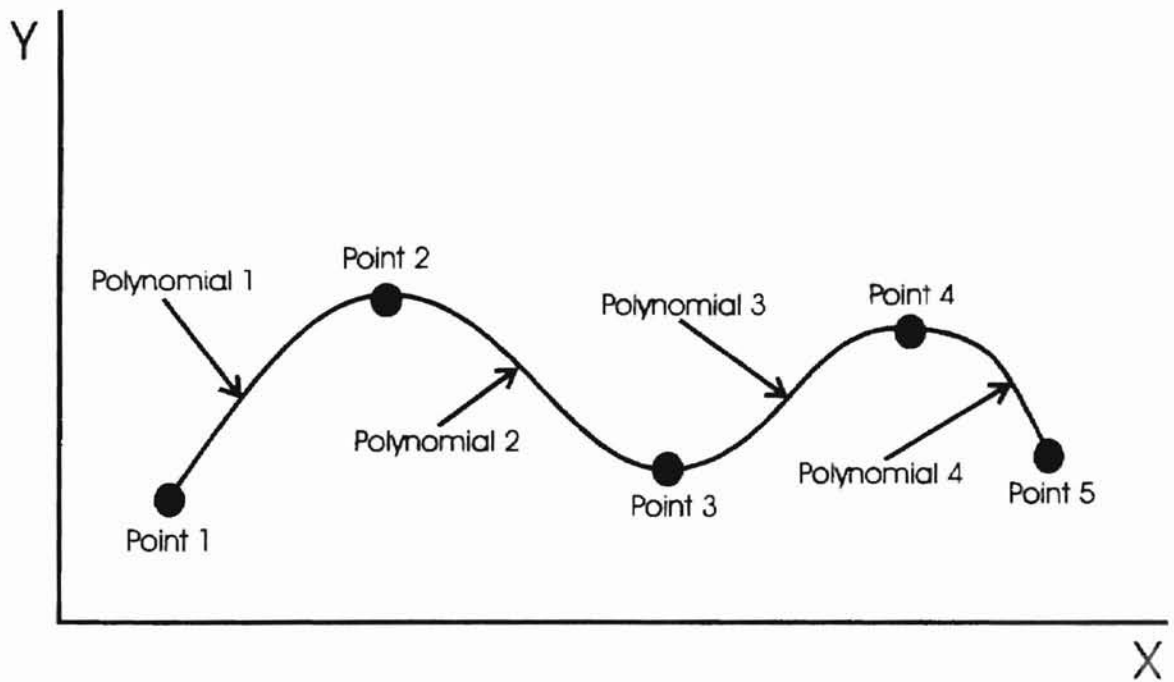


Figure 3.1: Cubic Spline Interpolation of Data Points

Two-Point Cubic Splines

For modeling purposes, it is often necessary to “bridge” two discontinuous functions. A typical example is shown in Figure 3.2. It is desirable to generate a single

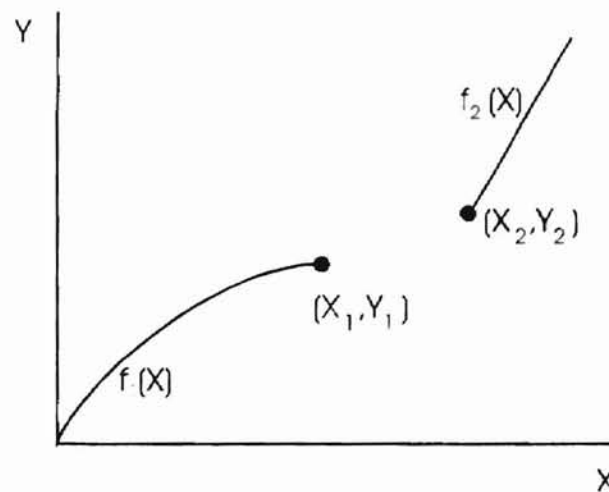


Figure 3.2: Two Functions Requiring a Smooth Connecting Cubic Spline

cubic polynomial with predetermined slope properties at the end points to provide slope continuity as depicted in Figure 3.3. It is possible to force the end point slopes to assume

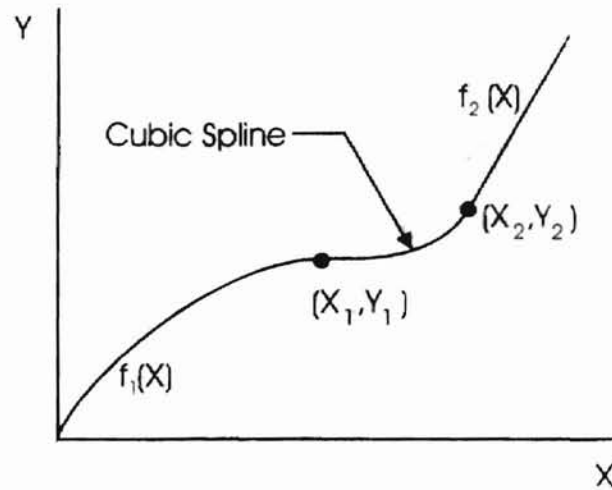


Figure 3.3: Cubic Spline Providing Slope Continuity Between Two Functions

any desired value. The two-point form of a cubic spline with specified end point slopes can be derived from the general case as follows:

A cubic spline connecting two points (x_1, y_1) and (x_2, y_2) takes the form:

$$y = a(x-x_1)^3 + b(x-x_1)^2 + c(x-x_1) + d \quad (3.1)$$

Where, $x_1 \leq x \leq x_2$

The coefficients a , b , c , and d are expressed in terms of the second derivatives (curvatures) at the end points [19]:

$$a = \frac{S_2 - S_1}{6h} \quad (3.2)$$

$$b = \frac{S_1}{2} \quad (3.3)$$

$$c = \frac{y_2 - y_1}{h} - \frac{2hS_1 + hS_2}{6} \quad (3.4)$$

$$d = y_1 \quad (3.5)$$

Where,

S_1 = Second Derivative at (x_1, y_1)

S_2 = Second Derivative at (x_2, y_2)

h = Interval Size: $(x_2 - x_1)$

For cubic spline interpolation on $n+1$ points, the following formulas for forcing end point slopes apply [19]:

$$2h_1S_1 + h_2S_2 = 6 \left(\frac{y_2 - y_1}{h_1} - y'_1 \right) \quad (3.6)$$

$$h_{n-1}S_n + 2h_nS_{n+1} = 6 \left(y'_{n+1} - \frac{y_{n+1} - y_n}{h_n} \right) \quad (3.7)$$

Where,

h_1 = 1st Interval Size: $(x_2 - x_1)$

⋮

⋮

h_n = n^{th} Interval Size: $(x_{n-1} - x_n)$

S_1 = Second Derivative at (x_1, y_1)

⋮

⋮

S_{n-1} = Second Derivative at (x_{n-1}, y_{n-1})

y'_1 = First Derivative at (x_1, y_1)

y'_{n+1} = First Derivative at (x_{n-1}, y_{n-1})

These equations may be simplified for two points as follows:

$$2S_1 + S_2 = \frac{6}{h} \left(\frac{y_2 - y_1}{h} - y'_1 \right) \quad (3.8)$$

$$S_1 + 2S_2 = \frac{6}{h} \left(y'_2 - \frac{y_2 - y_1}{h} \right) \quad (3.9)$$

Let,
$$e_1 = \frac{6}{h} \left(\frac{y_2 - y_1}{h} - y'_1 \right) \quad \text{and} \quad e_2 = \frac{6}{h} \left(y'_2 - \frac{y_2 - y_1}{h} \right) \quad (3.10)$$

Substituting gives:

$$2S_1 + S_2 = e_1 \quad (3.11)$$

$$S_1 + 2S_2 = e_2 \quad (3.12)$$

Solving these equations simultaneously produces:

$$S_1 = \frac{2e_1 - e_2}{3} \quad (3.13)$$

$$S_2 = \frac{2e_2 - e_1}{3} \quad (3.14)$$

Using this result, the cubic spline coefficients from Equations 3.2, 3.3, and 3.4 may be computed to force the slopes at the end points to the specified values of y'_1 and y'_2 .

In order to link two discontinuous functions with a cubic spline and maintain slope continuity, the first derivative of each function at the juncture points must be determined. Unfortunately, hydraulic systems often contain highly non-linear relationships between operating parameters. As a result, obtaining the first derivative analytically is time consuming and impractical. Numerical calculation of the first derivative is the most practical alternative. This task can be effectively accomplished using the standard *forward difference* approximation. This technique uses a finite approximation of the infinitesimally small change in the independent variable that

defines a derivative. Details regarding the forward difference method are available in standard Numerical Analysis texts [19].

Four-Point Cubic Splines

In order to eliminate the first derivative requirement of the two-point method, it is necessary to use a *natural spline*. A natural spline imposes no slope or curvature requirements at the end points. As a result, the end cubics approach linearity at their extremes. Unfortunately, a natural spline connecting only two points is a straight line. Obviously, a straight line is not a useful modeling tool when dealing with the inherent non-linearities of hydraulic systems.

To correct this problem while still using natural splines, a modeling tool based on the four-point form of cubic spline interpolation can be created. The basic concept of this modeling tool is best illustrated with an example. Consider the four points shown in Figure 3.4A. Assume points B and C are fixed and points A and D may be moved. Moving points A and D effects the slopes at points B and C and changes the shape of the curve between points B and C as shown in Figure 3.4B. Therefore, points A and D may be used as "handles" to control the slopes at points B and C.

Returning to the original goal, a cubic spline for modeling purposes connects two points with predetermined slope values at each point. Using the four-point form, the modeler can fix the position of the endpoints with two of the four points. The other two points may be used to control the slope values at these endpoints. This technique is conveniently implemented by locating control points in such close proximity to the true end points that the difference is physically insignificant. Mathematically, however, the

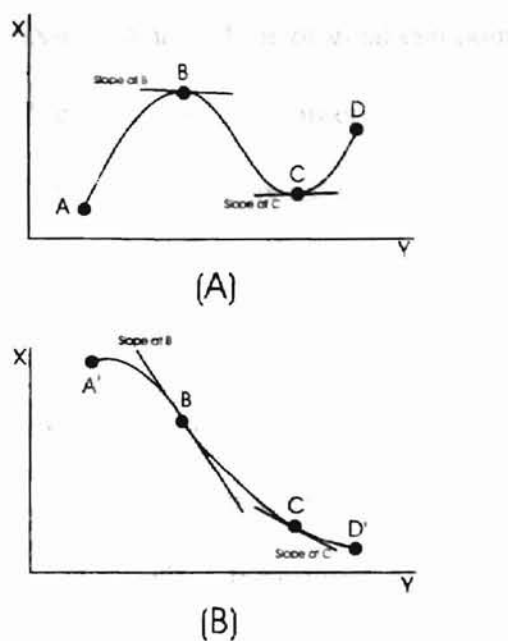


Figure 3.4: Effect of Moving End Points (A to A' and D to D')

difference is instrumental in determining the slope at the endpoints. This difference needs only to be larger than the computational precision of the computer or program used to generate the spline.

The four-point technique is best illustrated by returning to the example depicted in Figure 3.2. Again, suppose it is necessary to provide a smooth connection between the two functions. The first endpoint of the connecting spline is identified as (x_l, y_l) . This point lies on $f_l(x)$. In order to generate a control point for this endpoint, the original function equation ($f_l(x)$) is used. An initial estimate on the order of computational precision is added to x_l as follows:

$$x_{lc} = x_l - \Delta x \quad (3.15)$$

where, x_{lc} = X-coordinate of the control point

$x_l =$ X-coordinate of the original end point

$\Delta x =$ Increase in x on the order of computational precision

The function f_l is then evaluated at x_{lc} and compared to $f_l(x_l)$ as shown below:

$$f_l(x_{lc}) - f_l(x_l) = y_{ldiff} \quad (3.16)$$

where, $y_{ldiff} =$ Difference in y_l as calculated by computer program

If the computer cannot discern y_{ldiff} from zero, Δx is gradually increased until a calculable difference exists. This iterative process is easily accomplished in a computing environment. After this process is complete, the original end point (x_l, y_l) and the control point (x_{lc}, y_{lc}) will be discernibly different in x and y but the difference will be physically negligible. This procedure must be repeated using $f_2(x)$ to obtain a control point for the other end point. A graphical depiction of the endpoints and their control points is shown in Figure 3.5. The relative distances have been exaggerated for visibility. (NOTE: If $f_1(x)$ or $f_2(x)$ is a horizontal line, the corresponding iteration procedure on y is not necessary because y does not vary with x.)

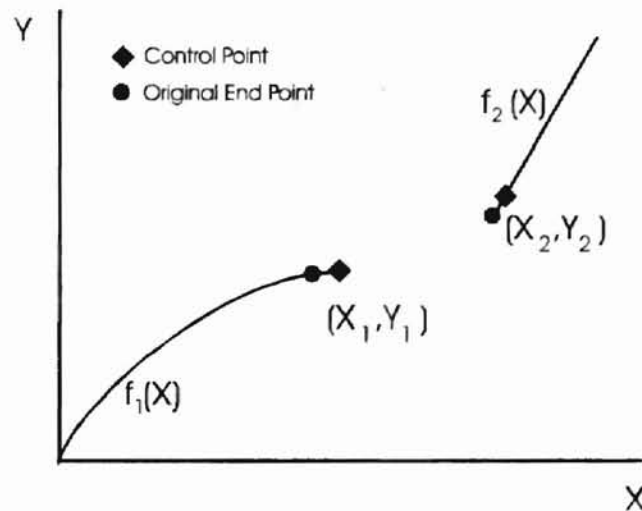


Figure 3.5: Cubic Spline End Points and their Control Points

The original end points, along with their control points, may now be used to generate three cubic spline polynomials using the procedure outlined in Appendix A.

The X and Y matrices take the form:

$$X = \begin{bmatrix} x_1 \\ x_{1c} \\ x_2 \\ x_{2c} \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ y_{1c} \\ y_2 \\ y_{2c} \end{bmatrix} \quad (3.17)$$

Because the true end points are so close to their corresponding control points, the cubics connecting them may be ignored. As a result, the cubic bridging the middle interval is assumed to connect the original endpoints. This assumption is true for all practical purposes. Using the functions $f_1(x)$ and $f_2(x)$ to create the control points forces the slopes at the end points to match the original functions. The resulting cubic spline model is depicted in Figure 3.6

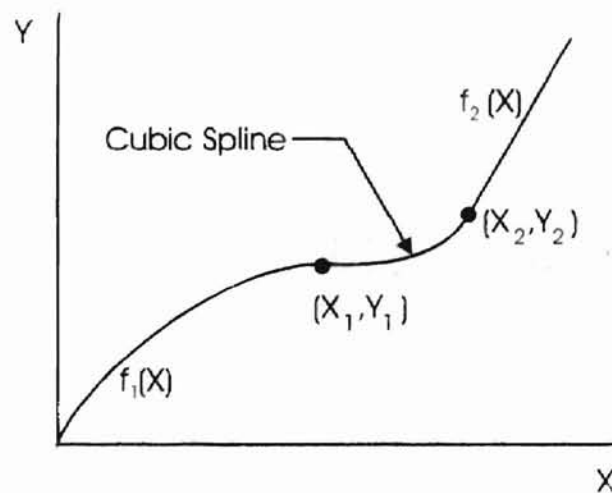


Figure 3.6: Cubic Spline Generated with the Four-Point Form

A complete algorithm for generating control points is displayed as a flow chart in Figure 3.7.

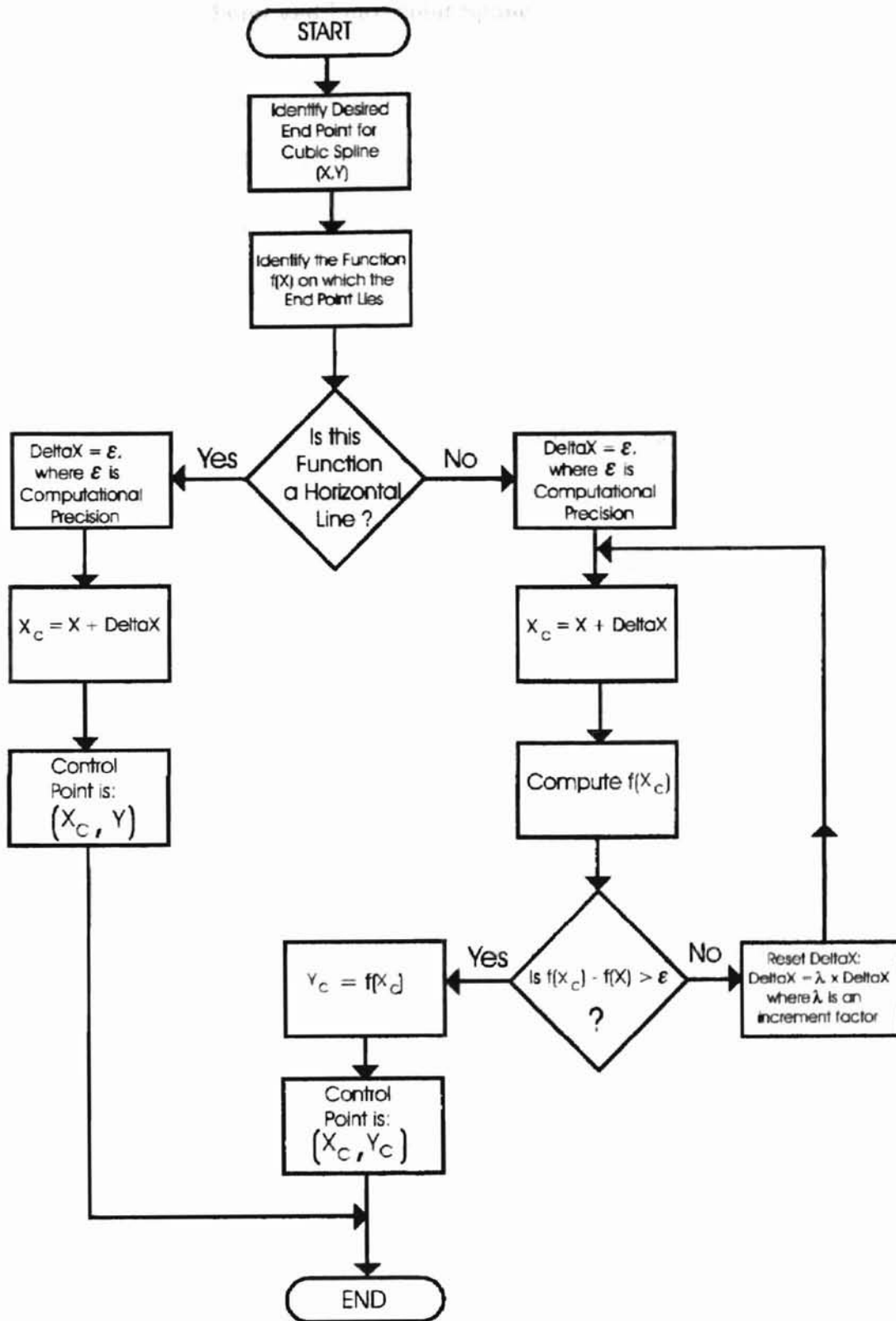


Figure 3.7: Flow Chart for Generating Control Points

Comparison of Two-Point and Four-Point Splines

The two-point and four-point methods for generating cubic splines may both be used as modeling tools. Each of these methods has advantages over the other in terms of computation. The two-point method requires knowledge of end point first derivatives while the four-point method does not require derivatives. Unlike the two-point method, however, the four-point method requires the solution of a set of equations.

A preferred method may be determined by evaluating the resulting cubic splines. The two-point and four-point methods may not produce the same cubic function. The difference between the methods is best illustrated with an example. The following straight line equations are given:

$$f(x_1) = 0 \quad (3.18)$$

$$f(x_2) = 0.9x - 720 \quad (3.19)$$

It is desired to generate a cubic spline between the points (720,0) on $f(x_1)$ and (825, 22.5) on $f(x_2)$. Applying both methods to this problem produces the results shown in Figure 3.8. Slope continuity at the juncture points can be maintained adequately by using either method. However, the cubic spline created using the two-point method “dips” below zero before heading in a positive direction. The potential for this behavior exists when “tight turns” are involved. This characteristic is undesirable when dealing with flow rate on the Y-axis. The negative sign reverses the flow direction. Flow reversal does not exist in practical hydraulic systems. The four-point spline creates a more direct bridge between the two discontinuous functions. As a result, the four-point method is preferred

as a modeling tool when the relatively sharp corners of a typical hydraulic model are involved.

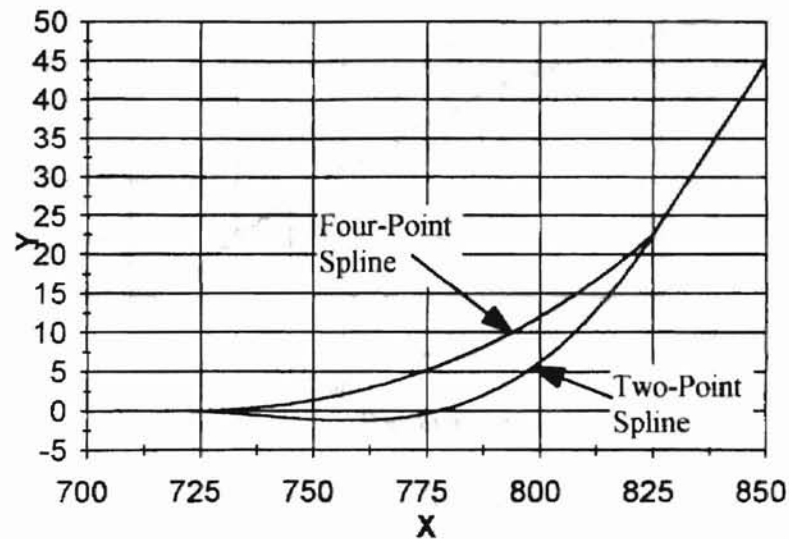


Figure 3.8: Comparison of Two-Point and Four-Point Cubic Splines

Application Examples

The usefulness of cubic splines as a modeling tool will be illustrated by two examples from the world of hydraulics. In the first example, an empirically based orifice model is developed. The second example involves smoothing discontinuities in a relief valve model.

Example 3.1 - An Empirically Based Orifice Flow Model

PART A

An orifice with a diameter (D) of 0.012 inch is known to have a discharge coefficient (C_d) of 0.63. Laboratory testing was performed to determine the pressure/flow relationship of this orifice using SAE 10W oil at 73°F (See Appendix B). The measured data have been plotted in Figure 3.9. It is desired to develop a mathematical model based on this experimental data.

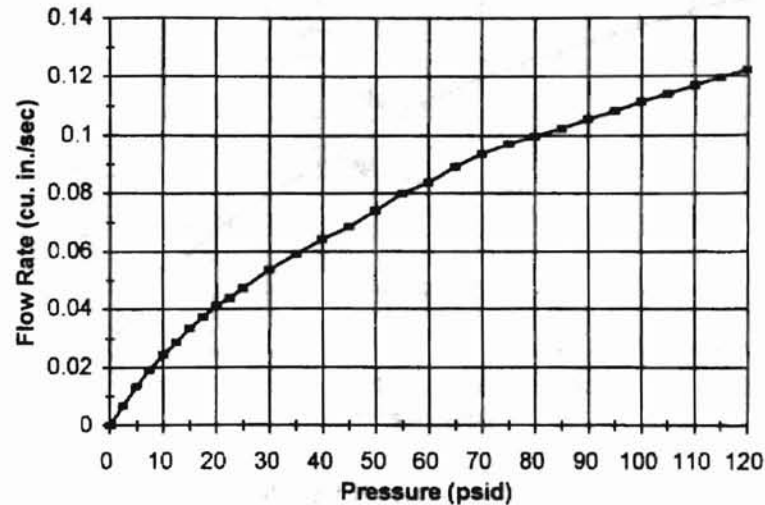


Figure 3.9: Experimental Results of Orifice Test

In order to develop a representative model, it is first necessary to determine where the turbulent flow equation is valid. This task may be accomplished by plotting the experimental data along with the turbulent flow equation. The turbulent flow equation is [20]:

$$Q = C_d A \sqrt{\frac{2}{\rho} \Delta P} \quad (3.20)$$

Q = Flow Through the Orifice (Dependent Variable) (in³/sec)

C_d = Discharge Coefficient = 0.63

A = Flow Area of the Orifice = $\frac{D^2 \pi}{4}$ (in²)

ρ = Fluid Density = 8.171×10^{-5} (lbf-sec²/in⁴) for 10W oil at 73°F

ΔP = Pressure Drop Across the Orifice (Independent Variable) (psid)

The results are depicted in Figure 3.10. From Figure 3.10, the turbulent flow equation

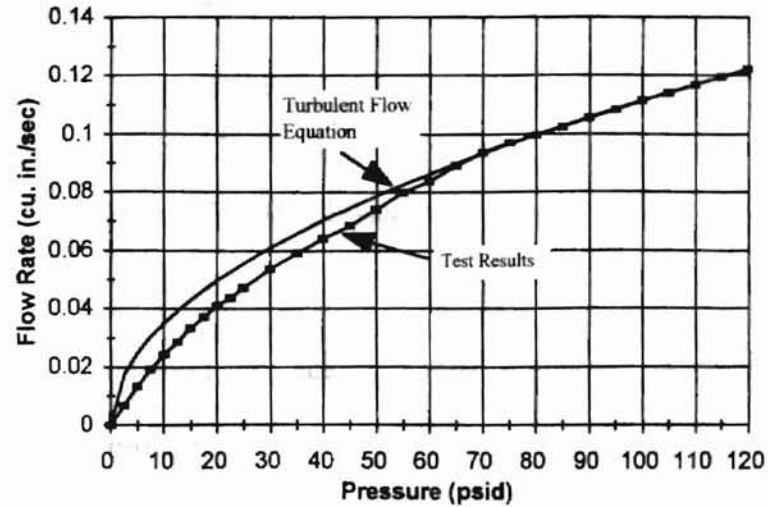


Figure 3.10: Comparison of Test Results and Turbulent Flow Equation

and the test data match almost exactly at 85 psid and above. Using the turbulent flow equation at low pressure differences would obviously be inaccurate. More importantly, the infinite slope at the origin would cause a stiff ODE solver to fail. Therefore, the turbulent flow equation will be used to model the orifice flow at pressures above 85psid. The turbulent flow model is not accurate at pressures lower than 85 psid because laminar and transitional flow are dominant. In order to model the portion of the flow/pressure relationship, a cubic spline must be used. It is first necessary to determine the end points of the cubic spline. From Figure 3.10, the first end point is obviously at the origin. The second end point can be determined by calculating the flow rate at 85 psid with the turbulent flow equation (3.20).

$$Q = C_d A \sqrt{\frac{2}{\rho} \Delta P} \quad (3.21)$$

Q = Flow Through the Orifice (in³/sec)

$$C_d = 0.63$$

$$A = \frac{0.012^2 \pi}{4} \text{ (in}^2\text{)}$$

$$\rho = 8.171 \times 10^{-5} \text{ (lbf-sec}^2\text{/in}^4\text{)}$$

$$\Delta P = 85 \text{ psid}$$

With the two end points in place, the control points may be generated. To obtain a smooth transition to turbulent flow at 85 psid, the control point is determined using the turbulent flow equation (3.20) and the algorithm outlined in Figure 3.7. There are no predetermined slope requirements at the origin because no joining function exists. Physically, laminar flow is dominant near the origin. The equation for laminar flow is linear [20]. Therefore, it is reasonable to choose a linear relationship to create the control point at the origin. The slope of this line is determined using the original test data. From Appendix B, the data point $(\Delta P, Q)$ nearest the origin is:

$$(2.5 \text{ psid}, 0.00693 \text{ in.}^3\text{/sec})$$

The equation of a line passing through this point and the origin is simply:

$$Q = \frac{0.00693}{2.5} \Delta P \quad (3.22)$$

Simplifying gives:

$$Q = 0.00277(\Delta P) \quad (3.23)$$

The algorithm described in Figure 3.7 along with equation 3.23 may now be used to generate a control point at the origin.

Using a computer to calculate the end point and control point values gives:

End Point 1: (0,0)

Control Point 1: $(2.220446049250313 \times 10^{-13}, 6.150635556423367 \times 10^{-16})$

End Point 2: $(8.500000000000000 \times 10^1, 1.027731761455279 \times 10^{-1})$

Control Point 2: $(8.500000000000222 \times 10^1, 1.027731761455292 \times 10^{-1})$

The number of significant figures has been carried to an extreme to show the difference between the original end points and their control points. Obviously, the differences are physically insignificant. However, these differences are critical when generating the cubic spline because they force the slopes at the end points to the desired values.

Using these four points and the procedure outlined in Appendix A, the following cubic spline was created:

$$Q = 1.327 \times 10^{-7} \Delta P^3 - 2.965 \times 10^{-5} \Delta P^2 + 2.770 \times 10^{-3} \Delta P \quad (3.24)$$

The procedure actually produces three third order polynomials. However, the cubics for the intervals between the end points and their control points may be ignored because the points are so close together. The resulting mathematical model is displayed in Figure 3.11.

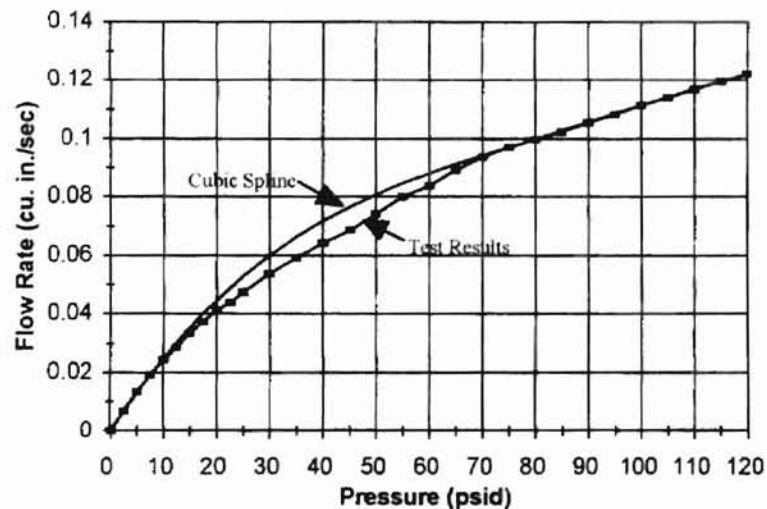


Figure 3.11: Cubic Spline Based Orifice Model

The model may be refined by experimenting with the departing slope at the origin. If the slope in equation 3.23 is changed from 0.00277 to 0.00220, the model is improved as shown in Figure 3.12.

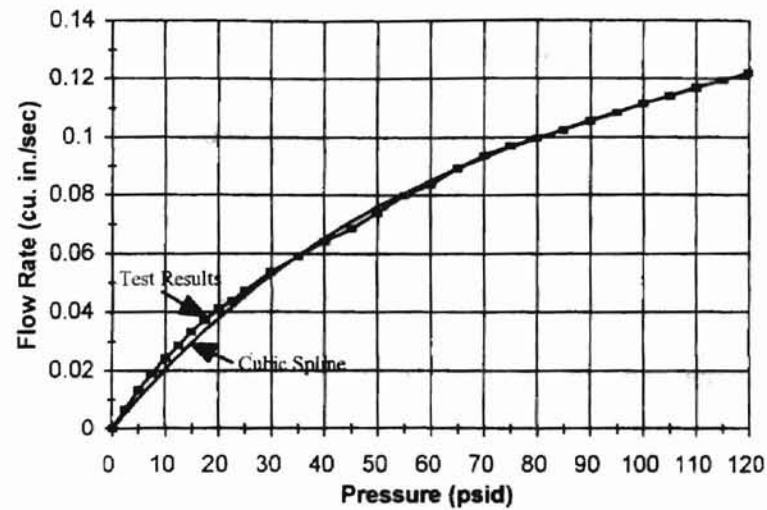


Figure 3.12: Refined Orifice Model

PART B

Using the results from Part A, it is desired to create a generalized mathematical model for flow through any orifice. This task may be accomplished by relating the results of Part A to Reynolds Number (N_r). Flow characteristics at a particular Reynolds Number are consistent for the flow of any fluid through any size orifice. Therefore, Reynolds Number provides a tool to generalize the orifice model developed in Part A.

The function used to force the departing slope at the origin was based on the linearity of laminar flow. The laminar flow equation is [20]:

$$Q = \frac{2\delta^2 DA}{\rho\nu} \Delta P \quad (3.25)$$

Q = Flow Through the Orifice

δ = Laminar Flow Coefficient

D = Hydraulic Orifice Diameter

A = Flow Area of the Orifice = $\frac{D^2 \pi}{4}$

ρ = Fluid Density

ν = Kinematic Fluid Viscosity

ΔP = Pressure Drop Across the Orifice

For Reynolds Numbers less than 100, the following relationship is generally accepted [20]:

$$C_d = \delta \sqrt{N_R} \quad (3.26)$$

Rearranging gives:

$$\delta = \frac{C_d}{\sqrt{N_R}} \quad (3.27)$$

Substituting into Equation 3.25 produces:

$$Q = \frac{2C_d^2 DA}{\rho \nu N_R} \Delta P \quad (3.28)$$

The slope of this line is simply:

$$\text{slope} = \frac{2C_d^2 DA}{\rho \nu N_R} \quad (3.29)$$

From Part A, a slope of 0.00220 was used to generate the final mathematical model. If this slope and the rest of the constants from Part A are substituted into Equation 3.29, it is possible to solve for Reynolds Number. The only new parameter is kinematic viscosity

(ν). For the 10W oil used in Part A, kinematic viscosity is 0.1623 (in.²/sec). Performing this calculation results in a Reynolds Number of 36.92. In the general case, therefore, the following equation may be used to create a control point at the origin:

$$Q = \frac{2C_d^2 DA}{\rho \nu (36.92)} \Delta P \quad (3.30)$$

The position of the second end point must also be related to Reynolds Number.

Reynolds Number is defined by the following equation:

$$N_R = \frac{(Q/A)D}{\nu} \quad (3.31)$$

From Part A, the second end point was located at:

$$P = 85 \text{ psi} \quad \text{and} \quad Q = .10277 \text{ in}^3/\text{sec}$$

Using this value for Q and the known values for A , D and ν , the Reynolds Number at the end point is easily calculated as 67.19. With this Reynolds Number known, the flow rate at the second end point for any orifice may be calculated by rearranging Equation 3.15 as follows:

$$Q = \frac{67.19 A \nu}{D} \quad (3.32)$$

The corresponding pressure is calculated by rearranging the turbulent flow equation (3.20):

$$\Delta P = \frac{\rho}{2} \left(\frac{Q}{C_d A} \right)^2 \quad (3.33)$$

Equations 3.32 and 3.33 represent a generalized method to calculate the location of the second end point for any orifice/fluid combination. The turbulent flow equation (3.20) may now be used to create the control point as outlined in Part A.

To further generalize, it is desirable to allow the user to select the Reynolds Number at which the turbulent flow equation applies (N_{Rt}). This feature is made possible by using a ratio of the two Reynolds Numbers calculated above. These values are:

$$\text{Reynolds Number Used in Laminar Flow Equation } (N_{Rl}) = 36.92$$

$$\text{Reynolds Number Used in Turbulent Flow Equation } (N_{Rt}) = 67.19$$

$$\text{Ratio} = \frac{36.92}{67.19} = 0.55 \quad (3.34)$$

Using this information, Equations 3.30 and 3.32, may be further generalized as follows:

$$Q = \frac{2C_d^2 DA}{\rho v(0.55N_{Rt})} \Delta P \quad (3.35)$$

$$Q = \frac{N_{Rt} A v}{D} \quad (3.36)$$

The value of N_{Rt} may now be defined by the modeler without effecting the basic shape of the cubic spline model.

Example 3.1 was developed in the MATLAB computing environment. The MATLAB script files used to complete this example are contained in Appendix C.

Example 3.2 - Static Relief Valve Model

A common direct acting pressure relief valve is depicted in Figure 3.13. When the force due to pressure at port P becomes large enough to overcome the spring preload against the poppet, fluid begins to flow to port T. This pressure is known as the *cracking*

pressure. As the combined forces due to pressure and flow increase, the spring continues to compress until the mechanical stop is reached.

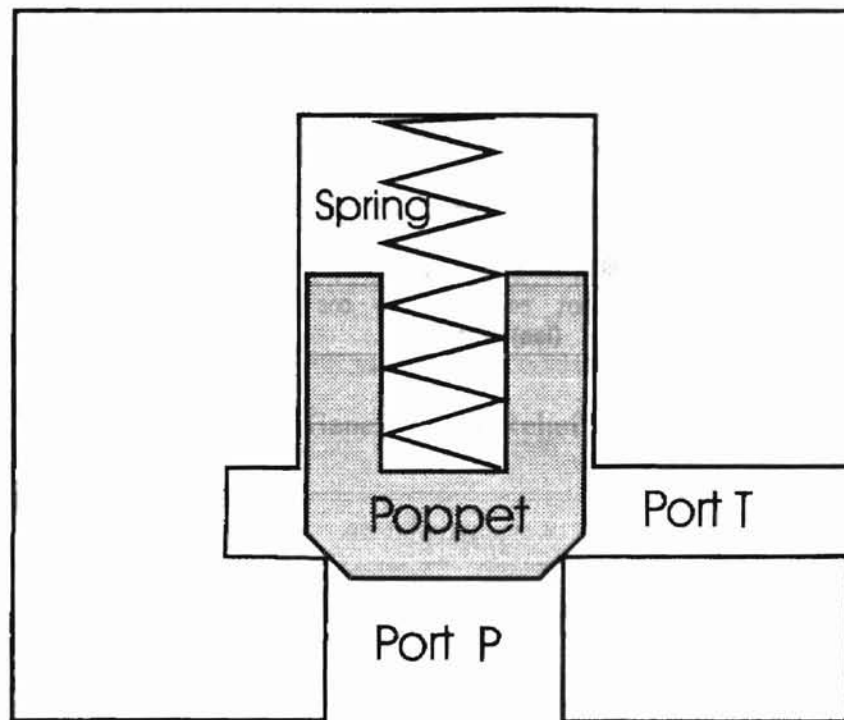


Figure 3.13: Typical Pressure Relief Valve

In many hydraulic circuits, it is acceptable to ignore the dynamics of the poppet. The simplified mathematical model is called a *static relief valve model*. A typical static relief valve model is presented in Figure 3.14. The flow through the valve remains at zero until the cracking pressure (P_{cr}) is attained. At this point, a linear relationship

between pressure and flow is assumed as the pressure at port P works against the spring. When the upper mechanical stop is reached (P_{max}), the poppet is static valve begins and the turbulent flow equation applies.

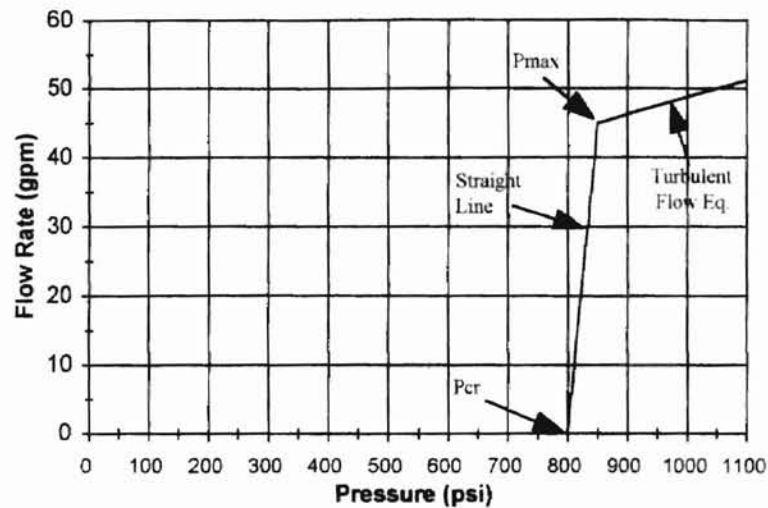


Figure 3.14: General Static Relief Valve Model

The static relief valve model contains two discontinuities which could severely limit the computational speed of a numerical ODE solver. The goal of this example is to smooth these discontinuities using cubic splines. For this example, the following information is known:

$$P_{cr} = 800 \text{ psi}$$

$$P_{max} = 850 \text{ psi}$$

$$Q_{max} = 45.05 \text{ in}^3/\text{sec}$$

$$Q_{@P_{cr}} = 0 \text{ in}^3/\text{sec}$$

$$P \text{ at Port T} = 0 \text{ psi}$$

The first step is to develop working mathematical relationships. From zero pressure to P_{cr} the relationship is obviously:

$$Q = 0 \quad \text{for } 0 \leq P \leq P_{cr} \quad (3.37)$$

The relationship from P_{cr} to P_{max} is simply a straight line equation:

$$Q = mP + b \quad \text{for } P_{cr} \leq P \leq P_{max} \quad (3.38)$$

where,

$$m = \frac{Q_{max}}{P_{max} - P_{cr}} = \frac{45.05}{850 - 800} = 0.90 \text{ in}^3/\text{sec-psi} \quad (3.39)$$

$$b = Q_{max} - mP_{max} = 45.05 - 0.90(850) = -719.95 \text{ in}^3/\text{sec} \quad (3.40)$$

For pressures above P_{max} the turbulent flow equation is utilized. The turbulent flow equation (3.20) may be simplified by lumping the constants together as shown in Equation 3.41. In this case, ΔP equals P because port T is at zero pressure.

$$Q = k_v \sqrt{P} \quad \text{for } P \geq P_{max} \quad (3.41)$$

Given P_{max} and Q_{max} , k_v is easily calculated:

$$k_v = \frac{Q_{max}}{\sqrt{P_{max}}} = \frac{45.05}{\sqrt{850}} = 1.55 \text{ (in}^3/\text{sec)/psi}^{(1/2)} \quad (3.42)$$

With the mathematical relationships in place, it is now possible to generate cubic splines using the four-point method. The discontinuity at P_{cr} will be considered first. Reasonable end points must be selected to bridge the discontinuity. The discontinuity occurs at P_{cr} (800 psi). End point pressure values of 720 psi and 825 psi will be selected because this range spans the discontinuity with enough room to create a relatively smooth curve between the functions. This choice is only one of many possibilities. Using a computer algorithm allows the modeler to test any number of combinations. The flow

rates corresponding to the chosen endpoint pressures may be calculated using Equations 3.34 and 3.35. A computer algorithm based on Figure 3.7 was used to compute the end point values and the control point values:

$$\text{End Point 1: } (7.200000000000000 \times 10^2, 0)$$

$$\text{Control Point 1: } (7.200000000000002 \times 10^2, 0)$$

$$\text{End Point 2: } (8.250000000000000 \times 10^2, 2.252250000000004 \times 10^1)$$

$$\text{Control Point 2: } (8.500000000000002 \times 10^1, 2.252250000000015 \times 10^1)$$

An inspection of End Point 1 and its control point reveals that no migration is necessary for flow rate (dependent variable) because the original function (Equation 3.37) has a slope of zero.

These points may now be used to generate the cubic spline according to the procedure outlined in Appendix A. Again, the resulting cubics between the end points and their control points may be ignored because the difference between these points is physically negligible. The resulting cubic spline equation for the middle interval is:

$$Q = 6.440 \times 10^{-6} P^3 + 1.367 \times 10^{-3} P^2 + 1.943 \times 10^{-16} P \quad (3.43)$$

To complete the model, the entire procedure must be repeated using Equations 3.38 and 3.41. Using endpoints at $P = 840$ psi and $P = 900$ psi, the following cubic spline equation is produced:

$$Q = 5.206 \times 10^{-5} P^3 - 8.592 \times 10^{-3} P^2 + 5.000 \times 10^{-1} P + 3.604 \quad (3.44)$$

The resulting static relief valve model is presented below (eqs. 3.45-3.49) and depicted graphically in Figure 3.13. Although some small amount of accuracy is lost, the resulting model contains no discontinuities. This feature provides compatibility with stiff ODE solvers and adaptive step size algorithms.

$$Q = 0 \quad \text{for } 0 \text{ psi} \leq P \leq 720 \text{ psi} \quad (3.45)$$

$$Q = 6.440 \times 10^{-6} (P-720)^3 + 1.367 \times 10^{-3} (P-720)^2 + 1.943 \times 10^{-16} (P-720) \quad (3.46)$$

$$\text{for } 720 \text{ psi} < P \leq 825 \text{ psi}$$

$$Q = 0.90P - 719.95 \quad (3.47)$$

$$\text{for } 825 \text{ psi} < P \leq 840 \text{ psi}$$

$$Q = 5.206 \times 10^{-5} (P-840)^3 - 8.592 \times 10^{-3} (P-840)^2 + 5.000 \times 10^{-1} (P-840) + 3.604 \quad (3.48)$$

$$\text{for } 840 \text{ psi} < P \leq 900 \text{ psi}$$

$$Q = 1.545 \sqrt{P} \quad (3.49)$$

$$\text{for } 900 \text{ psi} < P$$

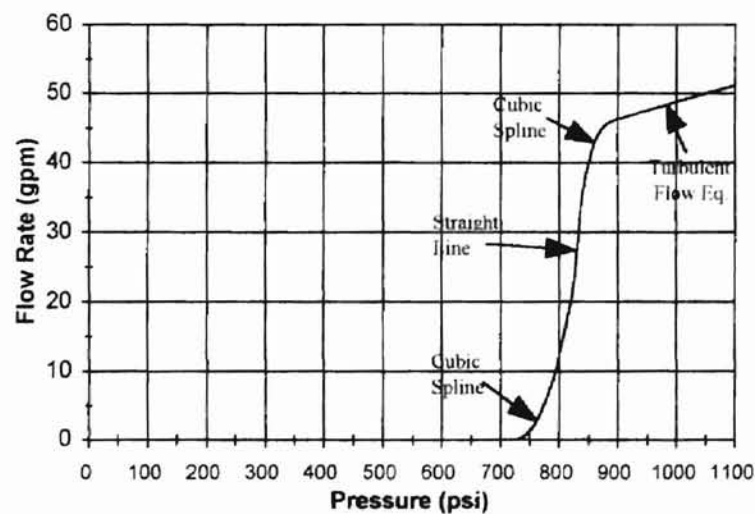


Figure 3.15: Smoothed Static Relief Valve Model

As with Example 3.1, Example 3.2 was developed in the MATLAB computing environment. The MATLAB script files used to complete this example are contained in Appendix D.

THE EVENT SWITCHING ALGORITHM

Background

The computational efficiency of a stiff numerical integration routine may be severely impeded by the presence of *state events*. A state event is a discontinuity which occurs when a variable reaches a physical limit during numerical integration. State events signal a change to the original set of ODEs. A new integration problem is defined at the exact time the critical value is achieved. The sudden switch to a new set of ODEs produces an abrupt change to the Jacobian matrix of a stiff solver. This abrupt change often causes the ODE solver to fail. In addition, adaptive step size algorithms tend to "hunt" around these state events in an effort to traverse them by reducing the step size. This hunting process requires an inordinate amount of processing time.

To combat these problems, researchers have recommended restarting the ODE solver each time a state event is encountered [12,13]. This process divides the original problem into continuous sections and solves them in a piecewise manner. Several event location routines have been devised to pinpoint the precise time at which a state event occurs [11,12,13,14,15,16,17,18]. The goal of these routines is to control step size such that the state event occurs at the end of the step. Commercially available integration packages locate events by monitoring sign changes. Therefore, each state event must be defined by a *discontinuity function*. The discontinuity function is a mathematical entity involving the state variables of the original ODEs. The state event occurs when the discontinuity function equals zero.

Commercial numerical integration packages are intended to be generic. As such, it is left to the user to define the discontinuity functions for a given application.

Hydraulic engineers would benefit greatly from a function-specific "book keeping" approach to state event handling. This type of approach may be developed by investigating the physical nature of hydraulic system state events.

Hydraulic System State Events

In the field of hydraulics, state events typically occur when actuators, loads, or internal valve parts encounter mechanical travel limits. A frictionless spring-mass-damper system may be used to investigate the effects of these mechanical stops (See Figure 4.1). The motion of the mass (m) is constrained by the two mechanical stops. If a

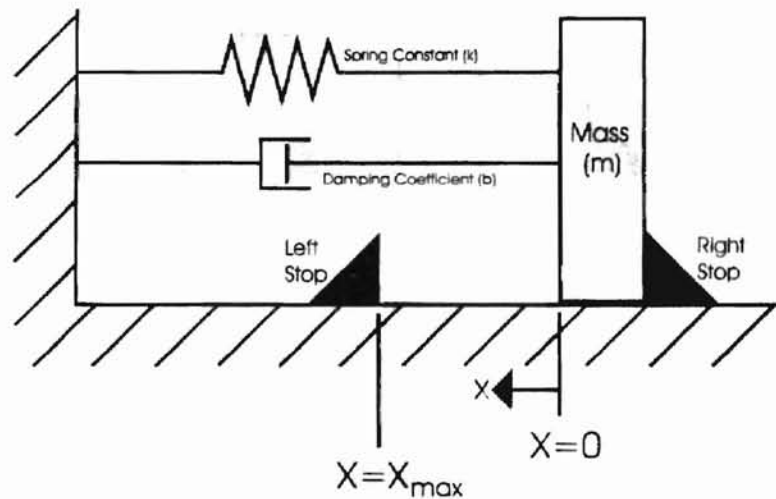


Figure 4.1: Spring-Mass-Damper System with Mechanical Stops

slowly increasing force (F) is applied to the mass, three distinct states are revealed.

These states are depicted in Figure 4.2. From Figure 4.2, State A shows a static condition because the applied force (F) is not large enough to overcome the spring preload ($F_{preload}$)

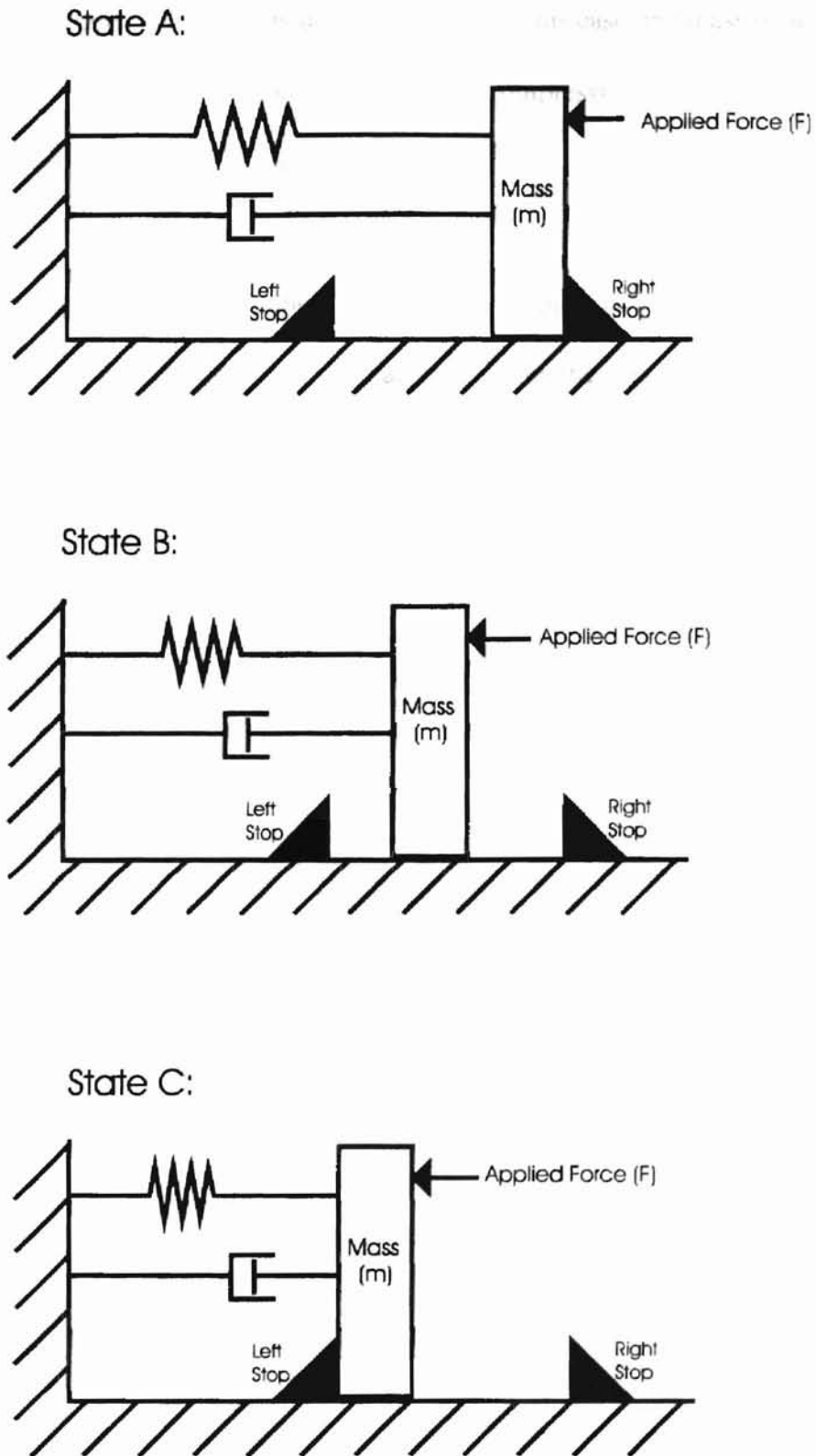


Figure 4.2: Three States for Spring-Mass-Damper System with Mechanical Stops

and no motion occurs. The same is true of State C. In this case, the mass is pinned against the left stop because the force due to spring compression (kx) is limited by this mechanical stop. By definition, therefore, the velocity (\dot{x}) and acceleration (\ddot{x}) of the mass are zero for States A and C.

If the magnitude of the applied force is between the spring preload and the maximum spring force and the mass is not against a stop, the mass is either dynamic or potentially dynamic. This condition is depicted as State B in Figure 4.2. The mass would be potentially dynamic if the applied force (F) was constant. Any change in the applied force would result in motion. The equations of motion for State B may be developed by summing the forces acting on the mass.

$$m\ddot{x} = F - F_{preload} - kx - b\dot{x} \quad (4.1)$$

Solving for acceleration gives:

$$\ddot{x} = \frac{1}{m}(F - F_{preload} - kx - b\dot{x}) \quad (4.2)$$

For numerical integration, a set of first order differential equations is required. This set of equations is obtained by introducing x_1 and x_2 as follows:

$$x_1 = x \quad (4.3)$$

$$\dot{x}_1 = \dot{x} \quad (4.4)$$

$$x_2 = \dot{x}_1 = \dot{x} \quad (4.5)$$

Therefore, the equations of motion for State B are:

$$\dot{x}_1 = x_2 \quad (4.6)$$

$$\dot{x}_2 = \frac{1}{m}(F - F_{preload} - kx_1 - bx_2) \quad (4.7)$$

For States A and C, the velocity and acceleration are zero. The equations of motion for these states are:

$$\dot{x}_1 = 0 \quad (4.8)$$

$$\dot{x}_2 = 0 \quad (4.9)$$

During the course of numerical integration, therefore, it is necessary to switch between these two sets of differential equations. The appropriate set of equations at any given time is dictated by the magnitude of the applied force. Typically, the magnitude of the applied force depends on time-varying hydraulic pressures. As a result, it is necessary to track the magnitude of the applied force during integration and to switch equation sets when a threshold value is achieved. This process assumes instant deceleration when a mechanical stop is encountered. In reality, deceleration is not instantaneous but it is so nearly instantaneous that the assumption is valid.

Switching between two sets of differential equations produces an abrupt change to the Jacobian matrix of a stiff ODE solver. In addition, adaptive step size algorithms require excessive computational effort to traverse a switch discontinuity. As a result, it is necessary to restart the solver when a state event is detected [12,13]. Before the solver is restarted, however, it is necessary to integrate up to the precise time of the state event. Commercial numerical integration packages often locate events by monitoring sign changes. In order to implement a commercial event location scheme, a discontinuity function (ϕ) must be created for each state event such that the discontinuity occurs when

the function equals zero. The discontinuity functions for the spring-mass-damper system must take the form:

$$\phi(x_1, x_2, F) = 0 \quad (4.10)$$

More specifically, the discontinuities for each state may be written as follows:

$$\text{State A: } \phi_A = F_{preload} - F \quad (4.11)$$

$$\text{State B: } \phi_B = x_{max} - x \quad (4.12)$$

or

$$\phi_B = x \quad (4.13)$$

$$\text{State C: } \phi_C = F - F_{preload} - k(x_{max}) \quad (4.14)$$

Zero initial conditions will be assumed to demonstrate the purpose of these discontinuity functions. Under these conditions, the set of equations for State A are applicable (4.8 and 4.9). It is also assumed that the applied force (F) increases with time. At each successive time step, the discontinuity function ϕ_A (4.11) is evaluated. If at any time ϕ_A becomes negative, the event location routine is activated to pinpoint the time at which ϕ_A equals zero. Physically, ϕ_A is zero when the applied force (F) is large enough to balance the spring preload. The numerical integrator is stopped at this point. Using the state variable values at the point of termination as initial conditions, the ODE solver is restarted using the State B differential equations (4.6 and 4.7). Coinciding with the switch to a new set of ODEs, the discontinuity function must be changed. Equations 4.12 and 4.13 are appropriate for State B. One of these discontinuity functions will cross zero if the displacement of the mass falls outside the limits imposed by the mechanical stops. If a sign change occurs, the event locator will again pinpoint the time at which the state

change occurs and the solver may be restarted with the static equations of motion (4.8 and 4.9). In the case of an increasing applied force, the left stop will be encountered. As such, the discontinuity function ϕ_c (4.14) must be invoked. If the applied force later begins to decrease, ϕ_c will define the point at which the applied force is no longer large enough to keep the mass pinned against the stop. The event location and solver restarting procedure may then be repeated as the mass enters State B. Similarly, the entire procedure must be repeated if the mass were to contact the right stop and enter State A.

Development of the Event Switching Algorithm

This ongoing event location and integrator restarting process may be efficiently handled by introducing an *Event Switching Algorithm* (ESA). The ESA takes advantage of physical properties to simplify the "book keeping" required to numerically integrate systems containing state discontinuities. The ESA is developed by comparing the discontinuity functions with the second dynamic equation of motion (eq.4.7). For State A these equations were:

$$\phi_A = F_{preload} - F \quad (4.15)$$

$$\dot{x}_2 = \frac{1}{m} (F - F_{preload} - kx_1 - bx_2) \quad (4.16)$$

The mass is pressed against the right mechanical stop while in State A. Therefore, the displacement (x_1) of the mass is zero. The velocity (x_2) of the mass is also zero.

Inserting this information into Equation 4.16 gives:

$$\dot{x}_2 = \frac{1}{m} (F - F_{preload}) \quad (4.17)$$

If equation 4.17 is set equal to zero, the following is true:

$$0 = \frac{1}{m} (F - F_{preload}) \quad (4.18)$$

$$0 = F - F_{preload} \quad (4.19)$$

The state event occurs when the discontinuity function (ϕ_A) from equation 4.15 equals zero:

$$0 = F_{preload} - F \quad (4.20)$$

Comparing equations 4.19 and 4.20 reveals the following correlation between the discontinuity function (ϕ_A) dynamic equation of motion (\dot{x}_2):

$$\phi_A = -1(\dot{x}_2) = \frac{-1}{m} (F - F_{preload} - kx_1 - bx_2) \quad (4.21)$$

Consequently, the existing equation of motion (4.16) may be used as the discontinuity function in State A. This fortunate circumstance eliminates the need for a separate discontinuity function. In addition, the negative sign or "-1" may be used as a flag to identify State A within the ESA. This negative sign is important because it dictates the direction of zero crossing to the event location routine (i.e. positive to negative).

A similar analysis may be performed for State C using ϕ_c and \dot{x}_2 . These equations are reprinted below for convenience:

$$\phi_c = F - F_{preload} - k(x_{max}) \quad (4.22)$$

$$\dot{x}_2 = \frac{1}{m} (F - F_{preload} - kx_1 - bx_2) \quad (4.23)$$

In this case, x_1 equals x_{max} because the mass is pinned against the left mechanical stop.

The velocity(x_2) is again equal to zero. Substituting this information into equation 4.23 and setting both equations (4.22 and 4.23) equal to zero reveals:

$$\phi_c = +1(\dot{x}_2) = \frac{1}{m} (F - F_{preload} - kx_1 - bx_2) \quad (4.24)$$

Therefore, State C also does not require a separate discontinuity function (ϕ_c). In keeping with State A, a flag of "+1" may be used by the ESA to identify State A.

The discontinuity functions for State B (eqs. 4.12 and 4.13) are related simply to displacement (x_1). This information is readily available during the course of numerical integration. A zero ("0") flag may be used to identify State B within the ESA.

With the discontinuity functions in place, an event location routine may be used to locate the precise time of the state event. The ODE solver will almost always "step over" the state event. At the time step just prior to the state event, the discontinuity function is positive. The discontinuity function then becomes negative after the next time step. Because the state event occurs when the discontinuity function is zero, the actual time of the event is between the two steps. The goal of an event location algorithm is to iteratively shorten the step size until the discontinuity function is zero at the end of the current time step. This task is most simply performed by successively bisecting the time step until the discontinuity function is within some tolerance of zero. More efficient methods often involve interpolation or false position schemes.

The event location routine effectively integrates to the precise moment of the state event. The time at which the state event occurs, as well as all of the corresponding state values, are known. At this point, the ODE solver is stopped. The necessary changes are made to the system of differential equations and integration is restarted using the time and state values obtained by the event location algorithm as initial conditions. The only

change to the initial conditions involves the velocity of the mass. This velocity must be reset to zero when the mass is pinned against either of the mechanical stops.

The resulting Event Switching Algorithm is best represented in flow chart form. This flow chart is displayed in Figure 4.3. The ESA contains a simple method to seamlessly integrate numerically stiff systems containing state event discontinuities. Although useful in many engineering disciplines, the ESA is especially valuable to hydraulic system engineers because stiffness and state discontinuities are commonplace.

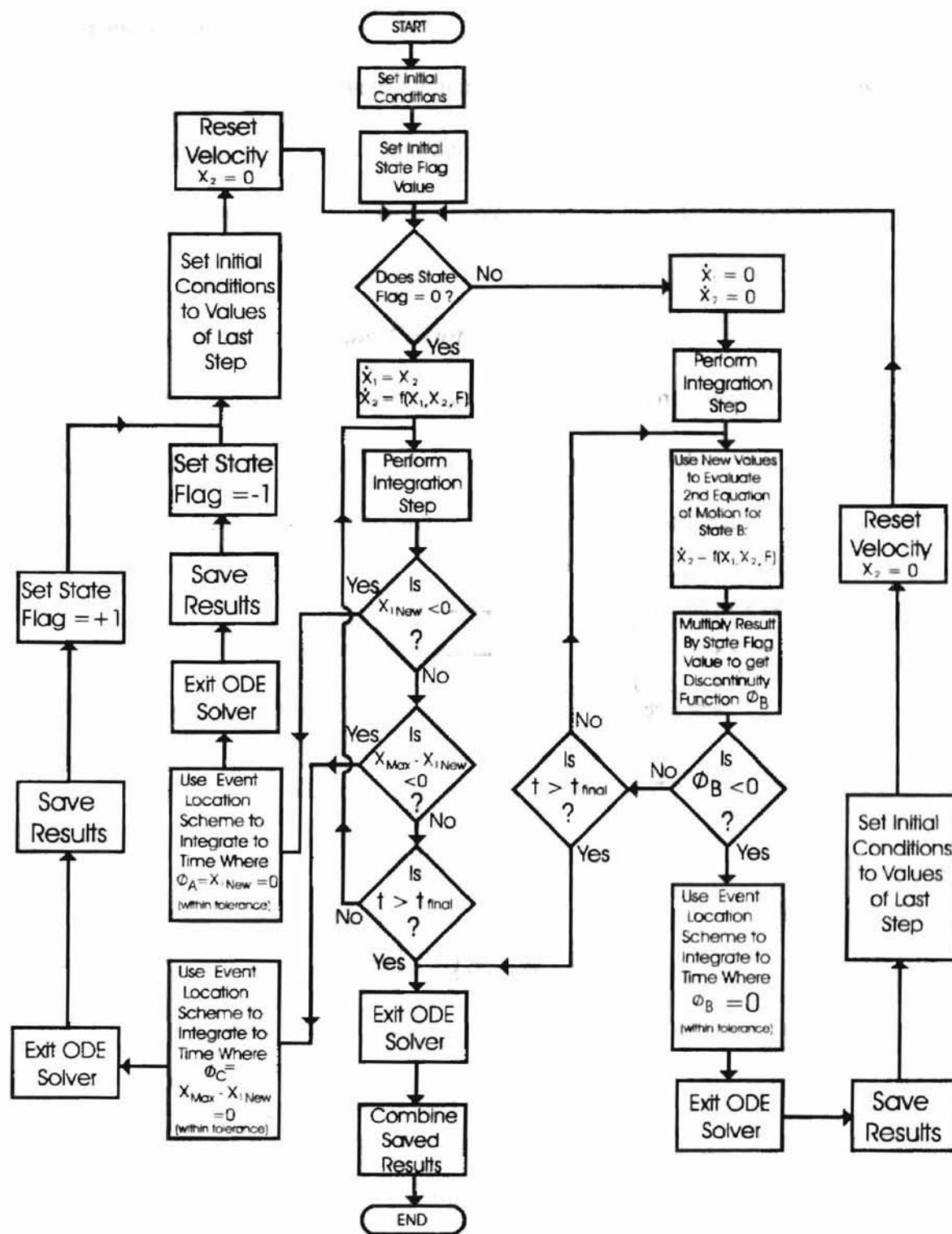


Figure 4.3: Flow Chart for Event Switching Algorithm as Applied to General

Spring-Mass-Damper System

Application Example

The application of the Event Switching Algorithm will be demonstrated with an example. This example involves a hydraulic actuator encountering a travel limit. In this case, an arbitrary stroke limit of 2 inches is placed on the cylinder. Only cylinder extension is considered in this example.

Example 4.1 - Hydraulic Cylinder Circuit

Consider the hydraulic circuit shown in Figure 4.4. This control circuit is designed to impart translational motion to a load using a hydraulic cylinder. Critical pressure nodes have been identified on the schematic. The mathematical

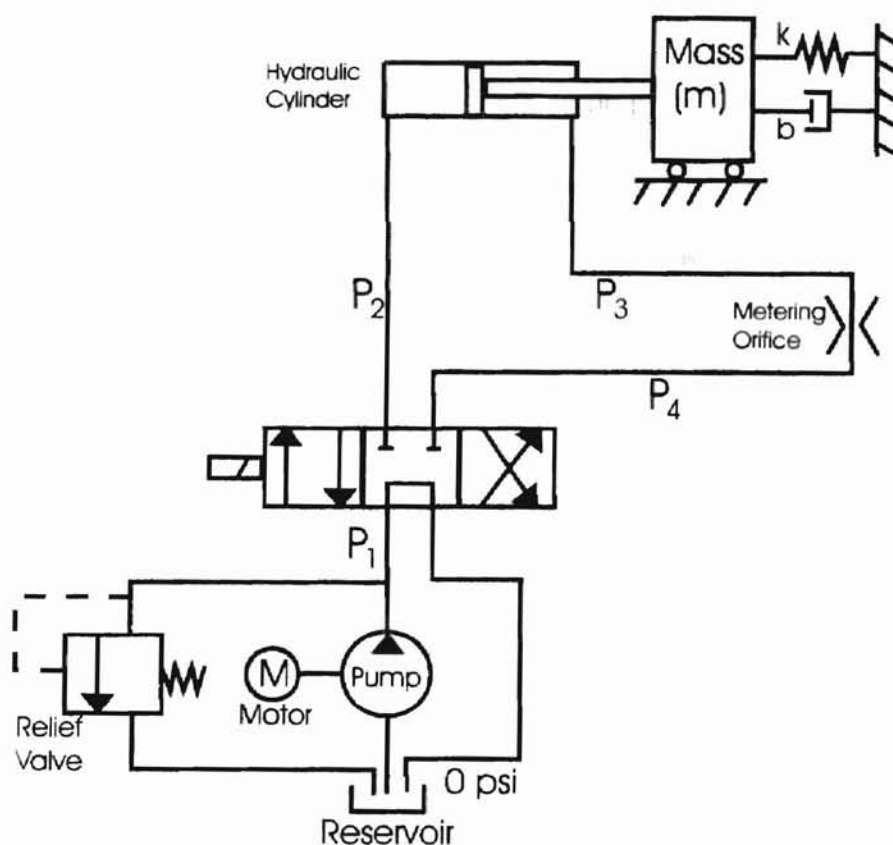


Figure 4.4: Hydraulic Circuit Schematic for Example 4.1

models for the circuit components and the resulting set of differential equations are developed as follows:

Mathematical Models

LOAD & HYDRAULIC ACTUATOR:

A spring-mass-damper system will be used to model the load. The load mass is rigidly fixed to the piston rod of the hydraulic cylinder. As a result, the mass of the rod and of the load must be combined. This combination is called a *lumped* load model. In this case, the following information is known:

Lumped Load Weight (w): 3860 lb

Spring Constant (k): 700 lb/in

Damping Coefficient (b): 200 lb/(in/sec)

The force applied to the load is provided by hydraulic pressure acting against the piston area. Hydraulic pressure is present on both sides of the piston. The resulting applied force is:

$$F = P_2 A_{bore} - P_3 A_{rod} \quad (4.25)$$

The piston dimensions are as follows:

Bore Diameter: 4.0 in.

Rod Diameter: 2.5 in.

Therefore,

$$A_{bore} = (4.0)^2 \cdot \pi / 4 \quad (4.26)$$

$$A_{rod} = A_{bore} - (2.5)^2 \cdot \pi / 4 \quad (4.27)$$

DIRECTIONAL CONTROL VALVE:

When shifted, the directional control valve directs pressurized fluid to either the rod or cap end of the hydraulic cylinder. Depending on the direction the valve is shifted, the cylinder will either extend or retract. In this case, the valve will be shifted to force the cylinder to extend at time zero ($t = 0$). The flow restriction through the open valve may be modeled as an orifice. For this directional control valve, the effective orifice diameter is:

$$D_{dc} = 0.27842 \text{ in.}$$

Making the flow area,

$$A_{dc} = (D_{dc})^2 \cdot \pi / 4 \quad (4.28)$$

Given the following, an orifice model may be created as outlined in Example 3.1 Part B:

$$\text{Orifice Diameter } (D_{or}) = 0.27842 \text{ in.}$$

$$\text{Discharge Coefficient } (C_d) = 0.61$$

$$\text{Fluid Density } (\rho) = 7.95 \times 10^{-5} \text{ (lb}\cdot\text{sec}^2/\text{in}^4)$$

$$\text{Kinematic Viscosity } (\nu) = 0.02 \text{ in}^2/\text{sec}$$

$$\text{Transition Reynolds Number } (N_{Rt}) = 1500$$

The resulting model is:

For $0 \leq (P_1 - P_2) \leq 1.240$ psid,

$$Q_{dcl} = 9.350 \times 10^{-1} (P_1 - P_2)^3 - 4.649 \times 10^0 (P_1 - P_2)^2 + 9.617 \times 10^{-2} (P_1 - P_2) \quad (4.29)$$

For $(P_1 - P_2) > 1.240$ psid.

$$Q_{dcl} = C_d A_{dc} \sqrt{\frac{2}{\rho} (P_1 - P_2)} \quad (4.30)$$

This model will apply to flow through the valve in either direction. The return flow equation is simply:

For $0 \leq (P_4) \leq 1.240$ psid,

$$Q_{dc2} = 9.350 \times 10^{-1} (P_4)^3 - 4.649 \times 10^0 (P_4)^2 + 9.617 \times 10^{-2} (P_4) \quad (4.31)$$

For $(P_4) > 1.240$ psid,

$$Q_{dc2} = C_d A_{dc} \sqrt{\frac{2}{\rho} (P_4)} \quad (4.32)$$

METERING ORIFICE:

The metering orifice creates back pressure against the rod end of the cylinder. This back pressure is responsible for controlled cylinder extension. The following information is known:

Orifice Diameter (D_{or}) = 0.11277 in.

Discharge Coefficient (C_d) = 0.61

Fluid Density (ρ) = 7.95×10^{-5} (lb·sec²/in⁴)

Kinematic Viscosity (ν) = 0.02 in²/sec

Transition Reynolds Number (N_{Rt}) = 1500

Given the above data, it is again possible to develop an orifice model using the general procedure outlined in Example 3.1 Part B. The resulting model is:

For $0 \leq (P_3 - P_4) \leq 7.560$ psid,

$$Q_{or} = 2.031 \times 10^{-3} (P_3 - P_4)^3 - 5.339 \times 10^{-2} (P_3 - P_4)^2 + 6.390 \times 10^{-1} (P_3 - P_4) \quad (4.33)$$

For $(P_3 - P_4) > 7.560$ psid,

$$Q_{or} = C_d A_{or} \sqrt{\frac{2}{\rho} (P_3 - P_4)} \quad (4.34)$$

RELIEF VALVE:

The relief valve simply limits pump discharge pressure. A model for this relief valve was developed in Example 3.2. This model is as follows:

$$Q_{rv} = 0 \quad \text{for } 0 \text{ psi} \leq P_1 \leq 720 \text{ psi} \quad (4.35)$$

$$Q_{rv} = 6.440 \times 10^{-6} (P_1 - 720)^3 + 1.367 \times 10^{-3} (P_1 - 720)^2 + 1.943 \times 10^{-16} (P_1 - 720) \quad (4.36)$$

for $720 \text{ psi} < P_1 \leq 825 \text{ psi}$

$$Q_{rv} = 0.90 P_1 - 719.95 \quad (4.37)$$

for $825 \text{ psi} < P_1 \leq 840 \text{ psi}$

$$Q_{rv} = 5.206 \times 10^{-5} (P_1 - 840)^3 - 8.592 \times 10^{-3} (P_1 - 840)^2 + 5.000 \times 10^{-1} (P_1 - 840) + 3.604$$

for $840 \text{ psi} < P_1 \leq 900 \text{ psi}$ (4.38)

$$Q_{rv} = 1.545 \sqrt{P_1} \quad (4.39)$$

for $900 \text{ psi} < P_1$

PUMP AND MOTOR:

The pump supplies the system with pressurized fluid. An ideal pump model will be used. Ideal pumps experience no internal leakage. Therefore, the pump flow may be calculated using the following equation:

$$Q_{in} = \text{Displacement} \times \text{Rotational Speed} \quad (4.40)$$

Given a motor speed of 1800 rpm and a pump displacement of $1.28 \text{ in}^3/\text{rev}$, the pump flow is:

$$Q_{in} = 1.28 \frac{\text{in}^3}{\text{rev}} \times 1800 \frac{\text{rev}}{\text{min}} \times \frac{1 \text{ min}}{60 \text{ sec}} = 38.5 \frac{\text{in}^3}{\text{sec}} \quad (4.41)$$

This flow rate will be applied to the system as a step input because the direction control valve is shifted at time zero ($t=0$). Prior to time zero, the flow is simply bypassed to tank with a negligible pressure drop through the directional control valve.

Differential Equations

The governing differential equation for pressure is:

$$\dot{P} = \frac{\beta}{V}(\Sigma Q) \quad (4.42)$$

$P =$ Pressure at a Given Point

$\beta =$ Bulk Modulus of Elasticity

$V =$ Fluid Volume

$Q =$ Flow Rate

Bulk modulus is a fluid property. In this case, a fluid with a bulk modulus of 150,000 psi will be used. By convention, a positive flow rate enters a pressure point while a negative flow exits. The differential equations for each pressure point may be developed as follows:

NODE 1:

The pump discharge flow (Q_m) enters Node 1 while the flow through the relief valve and the directional control valve exit. The volume of fluid trapped between the pump outlet

and valve inlet is known to be 10 in^3 . As a result, the differential equation for this point is:

$$\dot{P}_1 = \frac{150000}{10} (Q_{in} - Q_{dc} - Q_{rv}) \quad (4.43)$$

NODE 2:

Node 2 has one incoming flow rate and one outgoing flow rate. The incoming flow originates from the directional control valve (Q_{dc}). The outgoing flow fills the extending cylinder. The volume of fluid in the extending cylinder at any point in time is simply:

$$V = A_{bore} \cdot x \quad (4.44)$$

Therefore, the volumetric flow rate into the cylinder is

$$Q_{cyl\ in} = A_{bore} \cdot \dot{x} \quad (4.45)$$

The conduit connecting the directional control valve and the cylinder is known to have a volume of 10 in^3 . Given this information, the following differential equation may be developed:

$$\dot{P}_2 = \frac{150000}{10 + A_{bore} \cdot x} (Q_{dc} - A_{bore} \cdot \dot{x}) \quad (4.46)$$

NODE 3:

The differential equation for Node 3 may be developed in the same manner as Node 2.

Again, a conduit volume of 10 in^3 will be used. The resulting equation is:

$$\dot{P}_3 = \frac{150000}{10 + A_{rod} \cdot x} (A_{rod} \cdot \dot{x} - Q_{or}) \quad (4.47)$$

NODE 4:

The flow from the metering orifice enters Node 4 and the flow through the directional control valve exits. Given a transmission line volume of 10 in³, the differential equation for Node 4 is simply:

$$\dot{P}_4 = \frac{150000}{10}(Q_{or} - Q_d) \quad (4.48)$$

The standard differential equation of motion for the spring-mass-damper system (eq.4.2) is appropriate for the load:

$$\ddot{x} = \frac{1}{m}(F - F_{preload} - kx - b\dot{x}) \quad (4.49)$$

The applied force (F) is provided by the cylinder. This force was modeled as (eq. 4.25):

$$F = P_2A_{bore} - P_3A_{rod} \quad (4.50)$$

In this case, the preload force ($F_{preload}$) is zero. As a result, the differential equation of motion is:

$$\ddot{x} = \frac{1}{m}(P_2A_{bore} - P_3A_{rod} - kx - b\dot{x}) \quad (4.51)$$

System of Differential Equations

In order to create a vectorized set of differential equations for the computer algorithm, the following substitutions were made:

$$x_1 = P_1$$

$$x_2 = P_2$$

$$x_3 = P_3$$

$$x_4 = P_4$$

$$x_5 = x$$

$$\dot{x}_5 = \dot{x} \quad \text{from results of}$$

$$x_6 = \dot{x}_5 = \ddot{x}$$

The resulting set of linked ODEs is:

$$\dot{x}_1 = \frac{150000}{10} (Q_m - Q_{d1} - Q_{rv}) \quad (4.52)$$

$$\dot{x}_2 = \frac{150000}{10 + A_{bore} \cdot x_5} (Q_{d1} - A_{bore} \cdot \dot{x}_6) \quad (4.53)$$

$$\dot{x}_3 = \frac{150000}{10 + A_{rod} \cdot x_5} (A_{rod} \cdot x_6 - Q_{or}) \quad (4.54)$$

$$\dot{x}_4 = \frac{150000}{10} (Q_{or} - Q_{d2}) \quad (4.55)$$

$$\dot{x}_5 = x_6 \quad (4.56)$$

$$\dot{x}_6 = \frac{1}{m} (P_2 A_{bore} - P_3 A_{rod} - kx_5 - bx_6) \quad (4.57)$$

Results

This system of equations was solved using Gear's method for stiff ODEs in conjunction with the ESA as outlined in Figure 4.3. Adaptive step control was also utilized. The cylinder stroke limit was assumed to be 2.0 inches. Time-based plots of cylinder rod displacement and velocity are displayed in Figures 4.5 and 4.6 respectively. The displacement becomes constant when the stroke limit is reached. In addition, the velocity becomes zero when the stroke limit is encountered. In the absence of the ESA, these sharp changes would cause the stiff numerical integration and adaptive step size algorithms to become inefficient or to fail completely. The ESA, however, produces the

expected results in a reasonable amount of time. In fact, these results were obtained in under 12 seconds using Pentium 200 computer with 32 megabytes of RAM. A detailed study of computation time is left for Chapter 5 of this document. However, 12 seconds is, by no means, unreasonable for a system of this complexity.

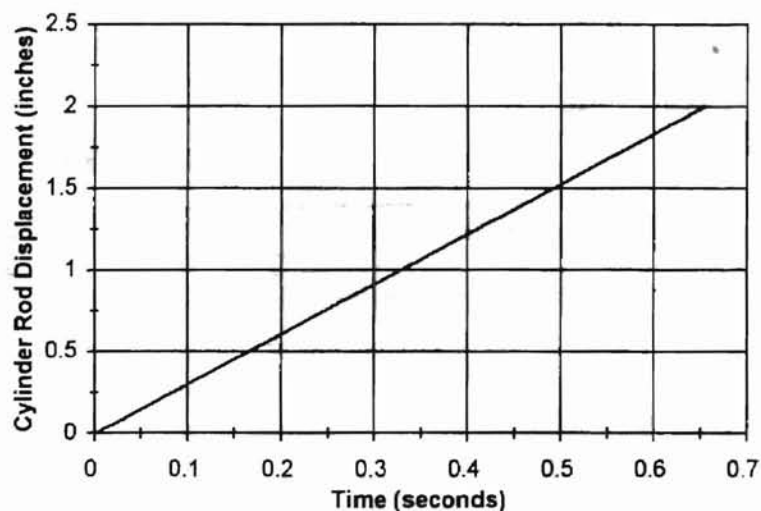


Figure 4.5: Cylinder Rod Displacement Response for Example 4.1

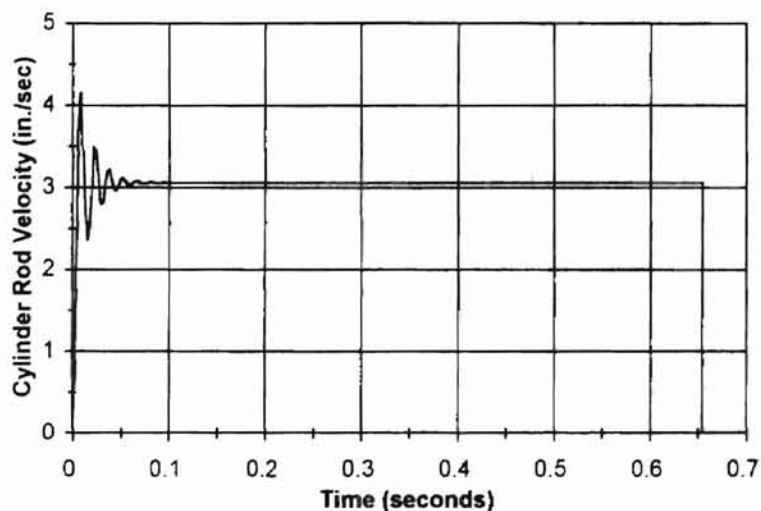


Figure 4.6: Cylinder Rod Velocity Response for Example 4.1

Pressure transients at Nodes 1,2 and 3 are depicted in Figure 4.7. Abrupt changes in pressure are also handled by the ESA. When the cylinder rod reaches its stroke limit, the pressures at Nodes 1 and 2 quickly climb until the relief valve opens. Similarly, pressure at Node 3 drops to zero because the cylinder is no longer forcing fluid through the metering orifice.

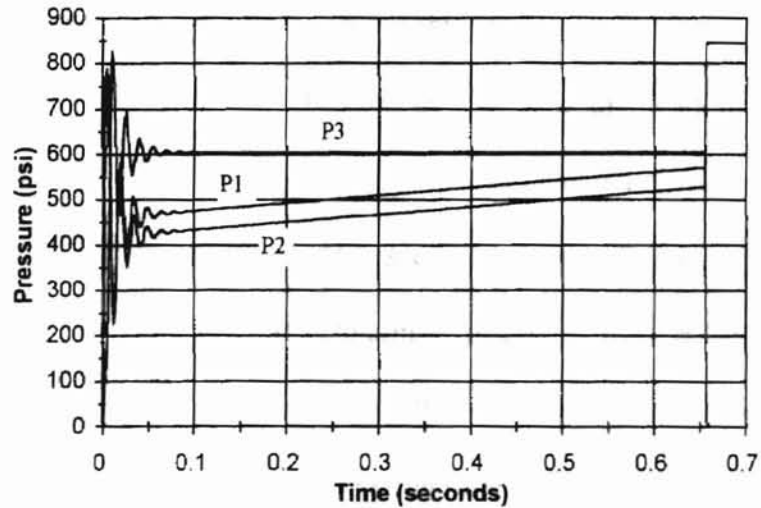


Figure 4.7: Pressure Responses for Example 4.1

Example 4.1 was developed and solved in the MATLAB computing environment. All of the relevant MATLAB script files are contained in Appendix E.

CHAPTER 5

CASE STUDY: PILOT-OPERATED RELIEF VALVE

Introduction

The four-point method for generating cubic splines was developed in this effort as a modeling tool. Its primary function is to eliminate model discontinuities which cause stiff ODE solvers and adaptive step size algorithms to fail or become inefficient.

Similarly, an Event Switching Algorithm was developed in this work to effectively eliminate failures caused by abrupt changes to the Jacobian matrix of a stiff ODE solver.

When combined, these two techniques are used to create compatibility between mathematically stiff hydraulic systems containing discontinuities and stiff ODE solvers with adaptive step size control. These stiff ODE solvers offer a considerable advantage over conventional solvers in terms of processing time.

Overview of Stiff ODE Solvers

Two stiff numerical integration methods are predominant. These methods are Gear's method for stiff ODEs and the Rosenbrock method. Of these, Gear's method is the most common. Gear's method is a predictor-corrector method. The predictor and corrector are based on Backward Differentiation Formulas (BDFs) or Numerical Differentiation Formulas (NDFs) [21]. Though closely related, the NDFs are generally more efficient than the BDFs [22]. The order of the BDFs or NDFs affects stability. During the course of integration, the order is varied by the solver [23]. If a limit is applied to the maximum order, greater stability is achieved. The degree of stability decreases and efficiency increases as the order limit is raised.

To explore the effects of order, Example 4.1 was solved using a variety of order limits. The results are compiled in Table 5.1. A stability problem occurred when the

Table 5.1: Simulation Results for Example 4.1 Using Different Variations of Gear's Method.

Gear's Method: Solver Parameters	Integration Results: Success or Failure	Processing Time[*] (seconds)
NDFs Maximum Order = 1	Success	39.08
NDFs Maximum Order = 2	Success	13.98
NDFs Maximum Order = 3	Success	11.72
NDFs Maximum Order = 4	Success	11.54
NDFs Maximum Order = 5	Failure	Not Applicable

* These results were obtained in the MATLAB computing environment using a Pentium 200 computer with 32 megabytes of RAM. The absolute and relative tolerances were set at 10^{-7} and 10^{-4} respectively.

maximum order was set at 5. This stability problem ultimately led to failure of the method. A review of the processing times reveals a threefold increase in efficiency as the order limit is increased.

Although it is not as numerically efficient as Gear's method, the Rosenbrock method has become popular for two reasons. First, the Rosenbrock method is relatively simple. This method is a single-step solver based on the familiar Runge-Kutta scheme. The need for complicated predictor-corrector formulas with varying order is eliminated. As a result, the method is conceptually easy to understand. Secondly, the Rosenbrock method has stability properties which surpass those of Gear's method. The Rosenbrock method can often solve problems which cause Gear's method to fail.

The Pilot-Operated Relief Valve

model equation

In an effort to investigate the advantages offered by these stiff ODE solvers, the dynamics of a pilot-operated relief valve were studied. In addition, the costs of restarting the numerical integrator in the ESA were explored by varying the travel limits of the poppets. A typical pilot-operated relief valve is depicted in Figure 5.1. The valve

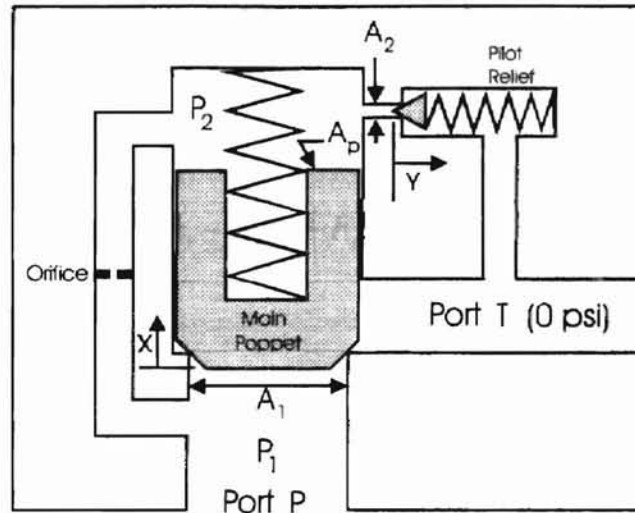


Figure 5.1: Pilot-Operated Relief Valve

functions by controlling the force balance of the main poppet. As the pressure increases, it distributes equally on all surfaces as long as the pilot poppet remains seated. With the aid of a light spring, the closing force is larger than the opening force and the main poppet remains against its seat. The valve opens when the pressure becomes large enough to lift the pilot poppet off of its seat. When the pressure becomes large enough to overcome the pilot spring, flow is established through the orifice. The orifice creates a pressure drop in the spring chamber of the main poppet. As a result, the opening force becomes larger than the closing force and the main poppet lifts off of its seat. The bulk of the flow then passes to tank by flowing around the unseated main poppet.

Dynamically, this process is described by six first-order differential equations.

These equations are listed below. Flow forces acting on the poppets have been included in this system of equations.

$$\dot{P}_1 = \dot{x}_1 = \frac{\beta}{V_1} (Q_{in} - Q_1 - Q_{or} - A_1 x_4) \quad (5.1)$$

$$\dot{P}_2 = \dot{x}_2 = \frac{\beta}{V_2} (Q_{or} + A_p x_4 - Q_2 - A_2 x_6) \quad (5.2)$$

$$\dot{x} = \dot{x}_3 = x_4 \quad (5.3)$$

$$\ddot{x} = \dot{x}_4 = \frac{1}{m_1} (A_1 x_1 - A_p x_2 - F_{preload1} - b_1 x_4 - k_1 x_3 - F_{flow1}) \quad (5.4)$$

$$\dot{y} = \dot{x}_5 = x_6 \quad (5.5)$$

$$\ddot{y} = \dot{x}_6 = \frac{1}{m_2} (A_2 x_2 - F_{preload2} - b_2 x_6 - k_2 x_5 - F_{flow2}) \quad (5.6)$$

Where,

$\beta =$	Fluid Bulk Modulus of Elasticity
$V_1 =$	Fluid Volume at Inlet Port
$V_2 =$	Fluid Volume of Main Poppet Spring Chamber
$Q_{in} =$	Inlet Flow Rate
$Q_1 =$	Flow Rate Over Main Poppet Seat
$Q_2 =$	Flow Rate Over Pilot Poppet Seat
$Q_{or} =$	Flow Rate Through Orifice
$D_1 =$	Diameter of Main Poppet Seat
$D_2 =$	Diameter of Pilot Poppet Seat

$A_1 =$	Area of Main Poppet Seat
$A_2 =$	Area of Pilot Poppet Seat
$A_p =$	Area of Main Poppet on the Spring Chamber Side
$m_1 =$	Mass of Main Poppet
$m_2 =$	Mass of Pilot Poppet
$F_{preload1} =$	Force of Main Spring Preload
$F_{preload2} =$	Force of Pilot Spring Preload
$F_{flow1} =$	Flow Force Acting on Main Poppet
$F_{flow2} =$	Flow Force Acting on Pilot Poppet
$b_1 =$	Damping Coefficient for Main Poppet
$b_2 =$	Damping Coefficient for Pilot Poppet
$k_1 =$	Spring Rate for Main Spring
$k_2 =$	Spring Rate for Pilot Spring

For the test problem, the following constants are known:

$\beta = 1.03 \times 10^9 \text{ Pa}$	$V_1 = 3 \times 10^{-4} \text{ m}^3$
$V_2 = 1 \times 10^{-7} \text{ m}^3$	$Q_m = 1 \times 10^{-3} \text{ m}^3/\text{sec}$
$D_1 = 0.017 \text{ m}$	$D_2 = 0.005 \text{ m}$
$A_1 = 2.27 \times 10^{-4} \text{ m}^2$	$A_2 = 1.96 \times 10^{-5} \text{ m}^2$
$A_p = 2.35 \times 10^{-4} \text{ m}^2$	$m_1 = 0.045 \text{ kg}$
$m_2 = 0.020 \text{ kg}$	$F_{preload1} = 100 \text{ N}$
$b_1 = 1000 \text{ N}/(\text{m}/\text{sec})$	$b_2 = 50 \text{ N}/(\text{m}/\text{sec})$
$k_1 = 5000 \text{ N}/\text{m}$	$k_2 = 50000 \text{ N}/\text{m}$

The desired cracking pressure (P_{cr}) for the relief valve is 1×10^7 Pascal. Therefore, the spring preload on the pilot poppet must be:

$$F_{preload} = A_2 \cdot P_{cr} \quad (5.7)$$

The flow rate through the fixed orifice (Q_{or}) requires a mathematical model. A suitable model may be developed by using the procedure outlined in Part B of Example 3.1. In this case, the following information is given:

$$\text{Orifice Diameter } (D_{or}) = 0.001 \text{ m}$$

$$\text{Discharge Coefficient } (C_d) = 0.61$$

$$\text{Fluid Density } (\rho) = 845 \text{ (kg/m}^3\text{)}$$

$$\text{Kinematic Viscosity } (\nu) = 14.3 \times 10^{-6} \text{ m}^2\text{/sec}$$

$$\text{Transition Reynolds Number } (N_{Rt}) = 1500$$

The following orifice flow model may be developed from this information:

For $0 \leq (P_1 - P_2) \leq 5.224 \times 10^5$ Pa,

$$Q_{or} = 3.759 \times 10^{-23} (P_1 - P_2)^3 - 7.014 \times 10^{-17} (P_1 - P_2)^2 + 5.863 \times 10^{-11} (P_1 - P_2) \quad (5.8)$$

For $(P_1 - P_2) > 5.224 \times 10^5$ Pa,

$$Q_{or} = C_d A_{or} \sqrt{\frac{2}{\rho} (P_1 - P_2)} \quad (5.9)$$

The flow rate over the main poppet seat (Q_1) is essentially the flow through a variable orifice created by the moving poppet. The flow area of this variable orifice is a function of poppet displacement (Eq. 5.10).

$$A_{flow1} = \pi \cdot D_1 \sin(\alpha_1) \cdot x \quad (5.10)$$

A_{flow1}	=	Flow Area of Main Poppet/Seat
D_1	=	Diameter of Main Poppet Seat
α_1	=	Half Angle of Main Poppet
x	=	Main Poppet Displacement

The mathematical model for this variable orifice may be developed by first assuming a fixed orifice with a diameter equal to the diameter of the main poppet seat (D_1). Using the procedure outlined in Part B of Example 3.1, a cubic spline may be generated. The variability of the orifice is handled by simply multiplying this cubic spline by the ratio of the flow area (A_{flow1}) to the seat area (A_1). In this case, a cubic spline may be developed using the following information:

$$\text{Seat Diameter } (D_1) = 0.017 \text{ m}$$

$$\text{Discharge Coefficient } (C_d) = 0.61$$

$$\text{Fluid Density } (\rho) = 845 \text{ (kg/m}^3\text{)}$$

$$\text{Kinematic Viscosity } (\nu) = 14.3 \times 10^{-6} \text{ m}^2\text{/sec}$$

$$\text{Transition Reynolds Number } (N_{Rt}) = 1500$$

The resulting cubic spline is:

$$1.543 \times 10^{-14} (P_1)^3 - 9.959 \times 10^{-11} (P_1)^2 + 2.881 \times 10^{-7} (P_1) \quad (5.11)$$

Given a main poppet half angle of 30° ($\alpha_1 = \pi/6$ rad.), the variable orifice model is:

For $0 \leq (P_1) \leq 1.808 \times 10^3$ Pa,

$$Q_1 = \frac{\pi D_1 \sin(\alpha_1) x}{A_1} \{1.543 \times 10^{-14} (P_1)^3 - 9.959 \times 10^{-11} (P_1)^2 + 2.881 \times 10^{-7} (P_1)\} \quad (5.12)$$

For $(P_1) > 1.808 \times 10^3 \text{ Pa}$,

$$Q_1 = C_d \pi \cdot D_1 \cdot \sin(\alpha_1) \cdot x \sqrt{\frac{2}{\rho} (P_1)} \quad (5.13)$$

The mathematical model for the variable orifice created by the pilot poppet may be generated in the same fashion as that of the main poppet. The data for the pilot poppet is as follows:

$$\text{Seat Diameter } (D_2) = 0.0005 \text{ m}$$

$$\text{Discharge Coefficient } (C_d) = 0.61$$

$$\text{Fluid Density } (\rho) = 845 \text{ (kg/m}^3\text{)}$$

$$\text{Kinematic Viscosity } (\nu) = 14.3 \times 10^{-6} \text{ m}^2\text{/sec}$$

$$\text{Transition Reynolds Number } (N_{Rt}) = 1500$$

$$\text{Pilot Poppet Half Angle } (\alpha_2) = \pi/6 \text{ radians}$$

The resulting mathematical model is:

For $0 \leq (P_2) \leq 2.090 \times 10^4 \text{ Pa}$,

$$Q_2 = \frac{\pi D_2 \sin(\alpha_2) y}{A_2} \{2.937 \times 10^{-18} (P_2)^3 - 2.192 \times 10^{-13} (P_2)^2 + 7.329 \times 10^{-9} (P_2)\} \quad (5.14)$$

For $(P_2) > 2.090 \times 10^4 \text{ Pa}$,

$$Q_2 = C_d \pi \cdot D_2 \cdot \sin(\alpha_2) \cdot y \sqrt{\frac{2}{\rho} (P_2)} \quad (5.15)$$

With the mathematical models in place, the discontinuity functions can be identified. The pilot-operated relief valve has two spring-mass-damper systems. Both of the masses may encounter state events in the form of mechanical stops. As a result, the ESA must be implemented twice. After each time step, the appropriate discontinuity function for both masses is evaluated. If either discontinuity function becomes negative, the integrator restarting procedure is initiated. For reasons discussed in Chapter 4 of this document, Equations 5.4 and 5.6 will be used by the ESA as the discontinuity functions for their respective poppets when the poppets are pinned against mechanical stops. If either of the poppets is between its mechanical travel limits, the displacement (x_3 or x_5) is monitored until a mechanical stop is reached as dictated by the ESA.

Numerical Integration Evaluation

The pilot-operated relief valve was inserted in a simple test circuit as depicted in Figure 5.2. Shifting the directional control valve at a predetermined time produced a

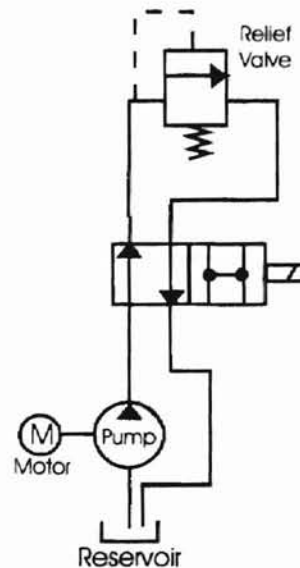


Figure 5.2: Pilot-Operated Relief Valve Test Circuit

duty cycle consisting of an "on" response and an "off" response. A series of tests with varying travel limits on both poppets was performed to evaluate the effectiveness of the stiff ODE solvers and the ESA. In addition to Gear's Method for Stiff ODEs and Rosenbrock's Method, three conventional (nonstiff) solvers were evaluated. An attempt to solve each test problem was made using all of the following ODE solvers:

- 1) Runge-Kutta (4,5): A single-step nonstiff solver credited to Dormand and Prince [22]. This ODE solver uses 4th and 5th order Runge-Kutta methods.
- 2) Runge-Kutta (2,3): A single-step nonstiff solver credited to Bogacki and Shampine [22]. This ODE solver uses 2nd and 3rd order Runge-Kutta methods.
- 3) Adams-Bashforth-Moulton [22]: A nonstiff multistep solver. This ODE solver uses a variable order routine to iteratively predict and correct at each time step.
- 4) Gear's Stiff Method [22]: A variable-order multistep solver based on BDFs or NDFs. For comparison, the following combinations were tested:
 - a) BDFs with order limit of 2.
 - b) BDFs with order limit of 5.
 - c) NDFs with order limit of 2.
 - d) NDFs with order limit of 5.

- 5) Rosenbrock's Method [22]: A second order, single-step stiff ODE solver based on semi-implicit Runge-Kutta Formulas.

TEST CONDITIONS:

Each of these ODE solvers is available in the MATLAB computing package.

MATLAB provided a convenient environment to implement the ESA and test a variety of ODE solvers. The following test conditions prevailed for each test problem:

Computer Specifications:	Pentium 200 with 32 megabytes of RAM
Relative Error Tolerance:	1×10^{-3}
Absolute Error Tolerance:	1×10^{-6}
Adaptive Step Size Control:	Active for all ODE solvers
Event Switching Algorithm:	Utilized for all tests
Initial Conditions:	Zero
Initial Time:	$t = 0$ seconds
Final Time:	$t = 0.08$ seconds
Initial Direction Control Valve Position:	Open to Relief Valve
Directional Control Valve Shift Time:	$t = 0.04$ seconds

The goal of these tests was to explore the potential advantages of using stiff ODE solvers.

For fixed-step integration, the nonstiff solvers may not require mathematically smooth models or an event switching routine because no Jacobian matrix is present. Adaptive step sizing, however, requires both. Because adaptive step size routines are commonplace, adaptive step sizing was applied to all of the tests. This condition provided commonality of mathematical models and event switching. For a given

problem, therefore, any variations in performance among the various ODE solvers were solely due to the solver routines.

TEST PROBLEMS:

Travels limits for each test problem and the corresponding integration results are presented on the following pages. The pressures and poppet displacements were of primary interest. As such, only the transient responses of these parameters were displayed graphically. In each case, the simulation results were identical (within the specified error tolerance) for all of the integration methods. Table 5.2 contains performance information for all of the tests. The number of ODE solver restarts required by the ESA and the computation times for the various ODE solvers are also contained in this table. Computation times presented in Table 5.2 are an average of three runs. As expected, the difference in elapsed time between the three runs was negligible.

Test #1:

Main Poppet Maximum Travel Limit (x_{max}) = 3×10^{-4} meters

Pilot Poppet Maximum Travel Limit (y_{max}) = 0.5×10^{-4} meters

Test #1 Solution:

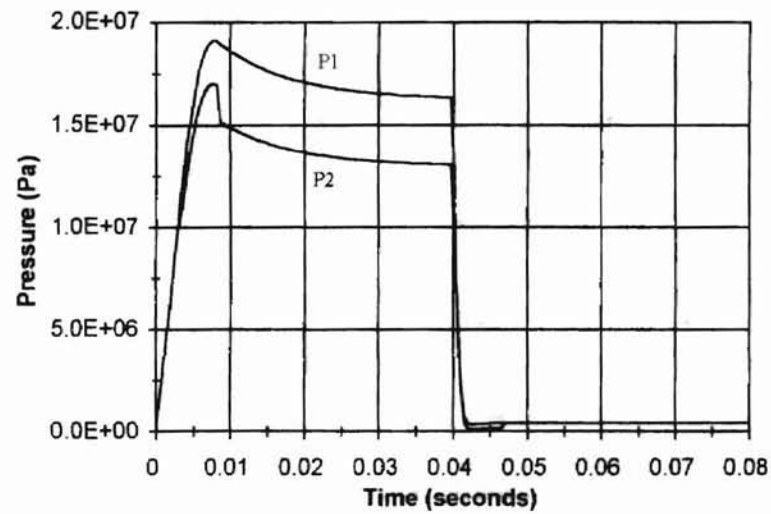


Figure 5.3: Pressure Response for Test #1

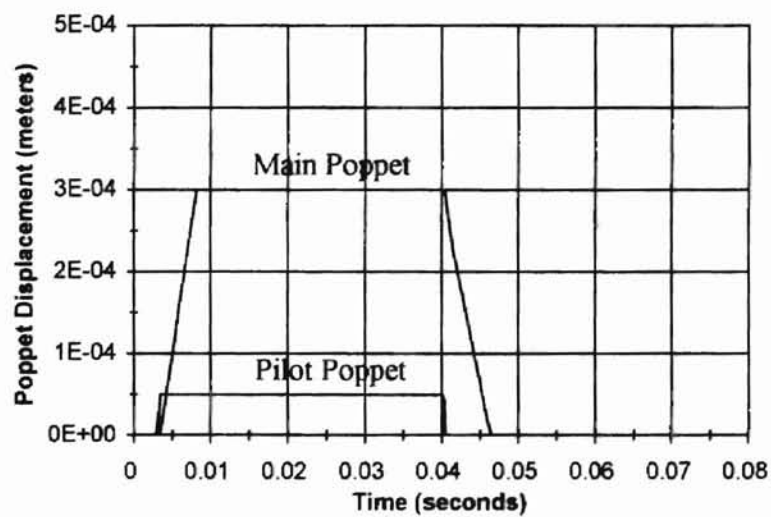


Figure 5.4: Displacement Response for Test #1

Test #2:

Main Poppet Maximum Travel Limit (x_{max}) = 3×10^{-4} meters

Pilot Poppet Maximum Travel Limit (y_{max}) = 0.8×10^{-4} meters

Test #2 Solution:

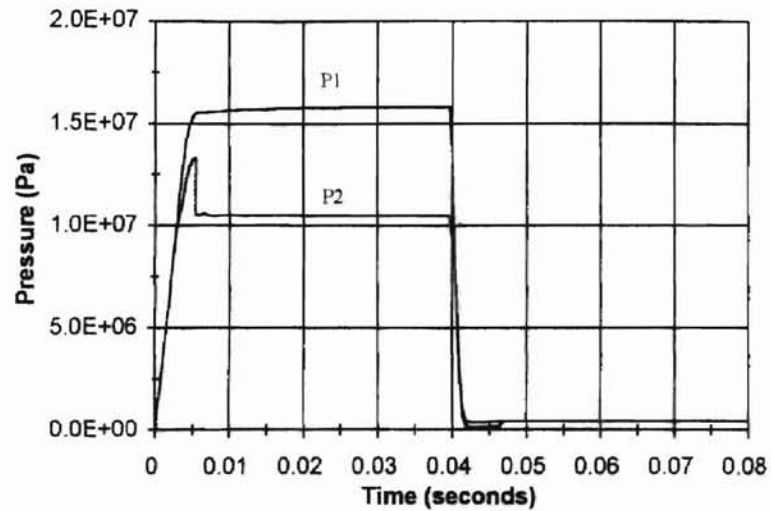


Figure 5.5: Pressure Response for Test #2

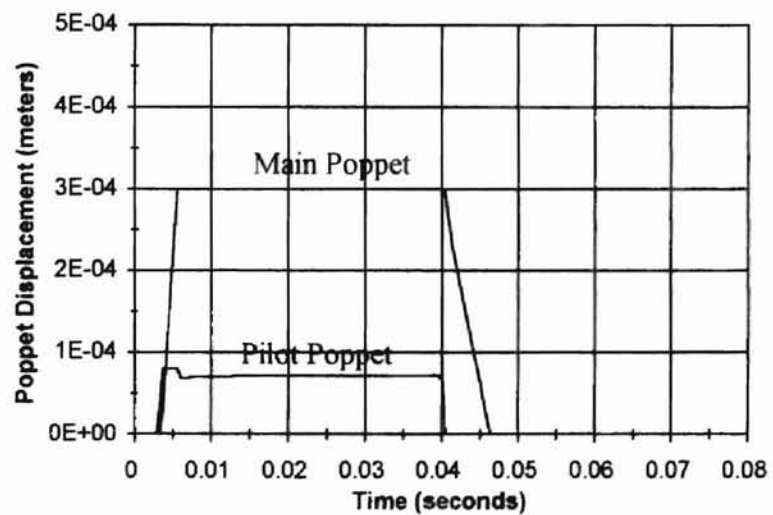


Figure 5.6: Displacement Response for Test #2

Test #3:

Main Poppet Maximum Travel Limit (x_{max}) = 4.5×10^{-4} meters

Pilot Poppet Maximum Travel Limit (y_{max}) = 0.8×10^{-4} meters

Test #3 Solution:

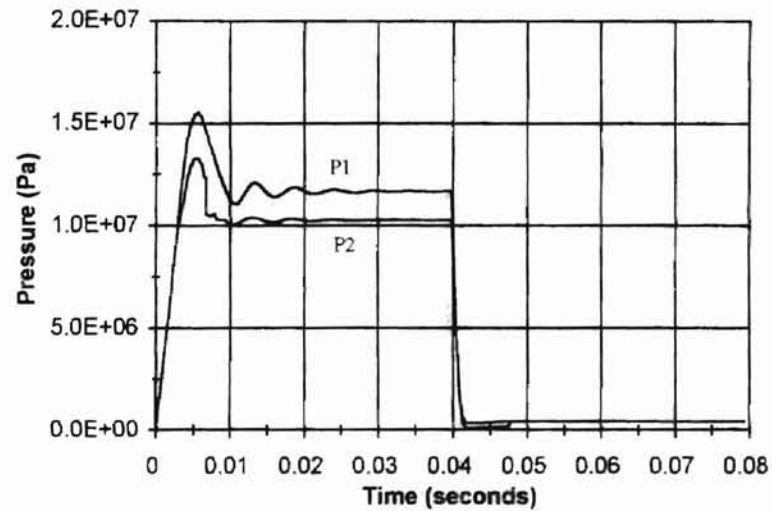


Figure 5.7: Pressure Response for Test #3

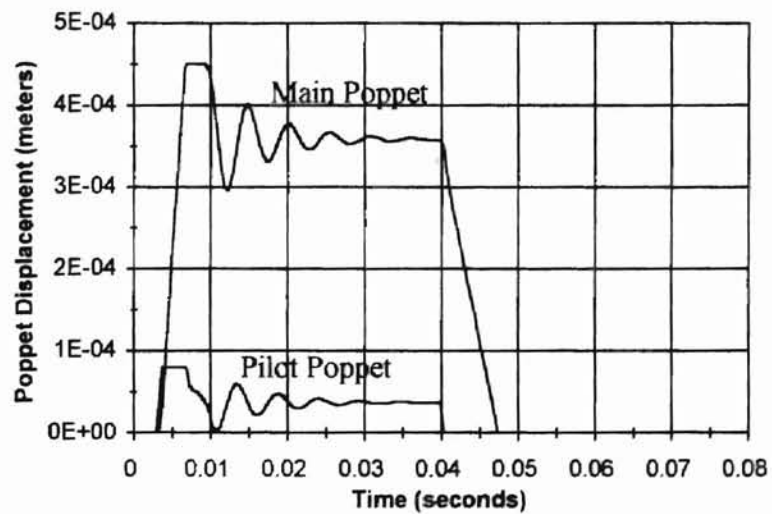


Figure 5.8: Displacement Response for Test #3

Test #4:

Main Poppet Maximum Travel Limit (x_{max}) = 4.75×10^{-4} meters

Pilot Poppet Maximum Travel Limit (y_{max}) = 0.5×10^{-4} meters

Test #4 Solution:

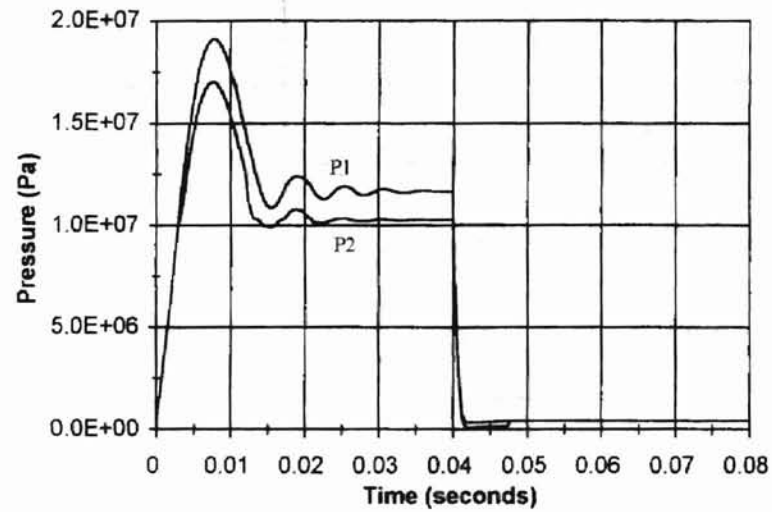


Figure 5.9: Pressure Response for Test #4

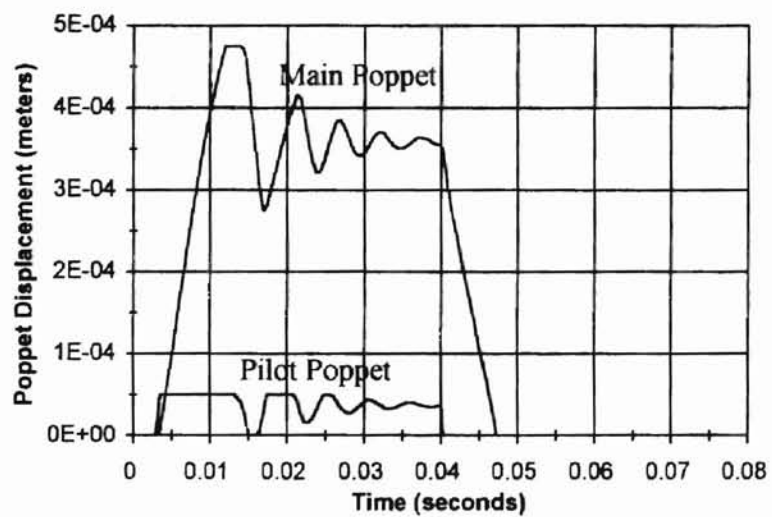


Figure 5.10: Displacement Response for Test #4

Table 5.2: Compiled Test Results for Variations of the Pilot-Operated Relief Valve Problem.

Test Number	Travel Limits (meters) X_{max} Y_{max}	Number Of Restarts	Computation Times (seconds)							
			Nonstiff ODE Solvers			Stiff ODE Solvers				
			Runge Kutta (4,5)	Runge Kutta (2,3)	Adams Bashforth Moulton	Gear's BDFs Max Order =2	Gear's BDFs Max Order =5	Gear's NDFs Max Order =2	Gear's NDFs Max Order =5	Rosenbrock
1	3.0×10^{-3} 0.5×10^{-4}	8	239.9	373.8	346.5	10.9	10.4	10.1	10.0	14.9
2	3.0×10^{-3} 0.8×10^{-4}	8	240.4	335.0	346.8	16.9	13.2	15.2	12.9	24.2
3	4.5×10^{-3} 0.8×10^{-4}	8	264.0	335.0	369.8	21.2	13.7	18.6	13.8	32.6
4	4.75×10^{-4} 0.5×10^{-4}	12	265.0	334.0	360.8	21.8	17.3	19.8	17.4	32.6

Discussion of Computation Time Results

The effectiveness of the various ODE solvers can be evaluated by studying Table 5.2. In the presence of mathematical stiffness, Runge-Kutta (4,5) was the most effective of the conventional nonstiff ODE solvers. The stiff ODE solvers were significantly more efficient. At worst, an eightfold improvement was achieved when using a stiff ODE solver. Processing speeds were over 20 times faster than that of Runge-Kutta (4,5) in some cases. With the aid of cubic splines generated using control points and the Event Switching Algorithm, these computational efficiency gains may be realized. The stiff solvers would have failed due to abrupt changes in their Jacobian matrices if these tools were absent. As such, the effort required to implement these tools is well justified. This is especially true of very large systems which could take days to integrate.

Among the stiff solvers, all of the variations of Gear's method proved to be slightly more efficient than Rosenbrock's method. This condition is problem specific. There exists a set of problems for which Gear's method is ineffective [22]. However, the inherent stability of Rosenbrock's method will lead to a successful solution. The ability of Rosenbrock's method to solve a wider variety of problems often compensates for its slightly longer processing time.

Table 5.2 also reveals some information about the variations of Gear's method. Numerical differentiation formulas of a given order are generally more efficient than their corresponding backward differentiation formulas. In the two cases where the NDFs did not perform better than the BDFs, the computation times are nearly equal. For both BDFs and NDFs, an increase from 2nd order to 5th improved the computation time by as little as 1% or by as much as 35% depending on the problem. Again, stability is the key issue. Differentiation formulas of 2nd order maintain A-stability [22]. However, this stability property is lost if the differentiation formula is above order two. For the pilot-operated relief valve problem, stability is maintained through 5th order.

The Event Switching Algorithm performed eight ODE solver restarts each during test numbers 1, 2 and 3. Despite the equal number of restarts, computation times varied significantly between these three tests for a given stiff ODE solver. These results suggest that restart costs are not the dominant factor in determining processing time. A careful review of the solutions reveals the true cause of the varying computation times. Inspection of Figure 5.4 reveals a very simple dynamic response. Both poppets spend most of the response time against their mechanical stops during Test #1. During Test #2,

the pilot poppet spends only a short amount of time against the upper travel limit (Figure 5.6). The dynamics of Test #2 are slightly more complex than those of Test #1. As a result, slightly more computation time is required. The solution for Test #3 (Figure 5.8) is clearly more oscillatory than either Test #1 or #2. Again, the increased dynamic complexity is accompanied by longer processing times.

Test numbers 3 and 4 may be compared to examine the effects of restarting the various stiff ODE solvers. Although dynamically similar, Test #4 requires four more restarts than Test #3. The 2nd order BDFs for Gear's method required only a 2.8% increase in processing time to accommodate the four additional restarts. Similarly, 2nd order NDFs required 6.5%. However, the 5th order BDFs and NDFs required over 26% more processing time to accomplish the same task. The costs of restarting the low order differentiation formulas for Gear's method are clearly much lower than those of the high order formulas. By comparison, Rosenbrock's method required no significant computation time to perform the additional restarts. This fortunate result is inherent to the method itself. Rosenbrock's method is a single-step solver which requires no historical information. As a result, the first step is virtually the same as any other step. Gear's method is a multistep solver which requires historical information to complete a step. When the solver is started, no historical information is available and special measures must be introduced to start the solver.

Error Analysis

In an effort to investigate the reliability of the stiff solvers in terms of accuracy, one final test was performed. Test #4 was repeated using Runge-Kutta (4,5) and a step

size limit of 10^{-7} seconds. This step size limitation is several orders of magnitude smaller than the minimum step size produced by the adaptive step size control algorithm with no size restrictions. Because accuracy is directly related to step size, the simulation performed at 10^{-7} was considered to be "exact" for all practical purposes and served as a baseline for comparison. The results from the inlet pressure (P_1) response of Test #4 using the various ODE solvers were compared to the results obtained using Runge-Kutta (4,5) with a step size restriction of 10^{-7} second. The results were refined to provide state values at increments of 10^{-4} second. At each increment, the pressure difference between the results from the solver of interest and the "exact" solution was calculated. This difference was used to determine error percentage as follows:

$$\text{Percent Error} = \frac{|P_{\text{solver}} - P_{\text{exact}}|}{P_{\text{exact}}} \times 100 \% \quad (5.16)$$

Where, $P_{\text{solver}} =$ Inlet Pressure (P_1) as computed by the ODE solver of interest with no step size restriction.

$P_{\text{exact}} =$ Inlet Pressure (P_1) as computed by Runge-Kutta (4,5) with a step size restriction of 10^{-7} second.

The results of this error analysis, in terms of maximum error and average error, are compiled in Table 5.3.

Table 5.3: Error Analysis Results for Inlet Pressure Response of Test #4

Numerical Integration Method	Maximum Error	Average Error
Runge-Kutta (4,5)	0.0294 %	0.0024 %
Adams-Bashforth-Moulton	0.1930 %	0.0077 %
Gear's:BDFs Max. Order =2	0.3932 %	0.0696 %
Gear's:BDFs Max. Order =5	0.3539 %	0.0522 %
Gear's:NDFs Max. Order =2	1.0272 %	0.1124 %
Gear's:NDFs Max. Order =5	0.1709 %	0.0434 %
Rosenbrock's Method	0.0756 %	0.0140 %

A graphical depiction of the error analysis results is contained in Figure 5.11.

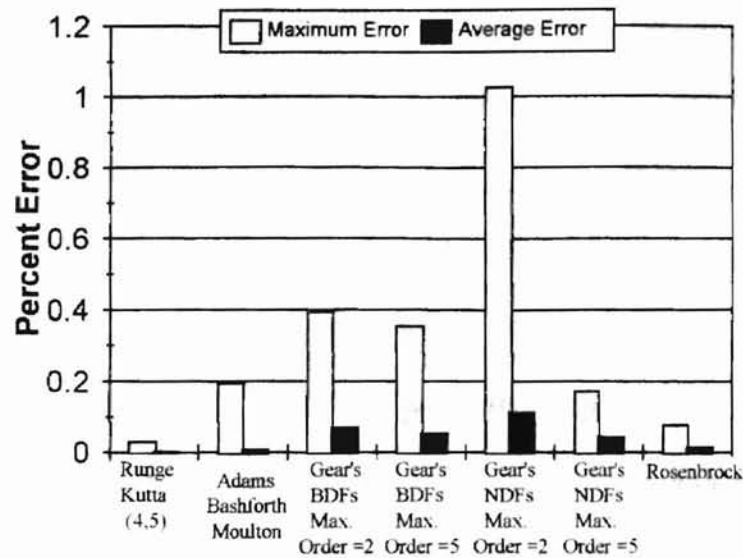


Figure 5.11: Comparison of Error Analysis Results

Discussion of Error Analysis Results

The worst case error was slightly over 1%. Accuracy of this magnitude is more than adequate for hydraulic pressure calculation. Runge-Kutta (4,5) was the most accurate of all the solvers but this accuracy is achieved at the expense of processing time. The Rosenbrock method provides accuracy comparable to Runge-Kutta (4,5) without excessive processing time. Among the various forms of Gear's Method, Numerical Differentiation Formulas (NDFs) with a 5th order limit were the most accurate. This form of Gear's method is also the most efficient in terms of processing time making it a powerful integration method for stiff hydraulic systems. Ultimately, stability will determine the most suitable integration method because adequate accuracy may be achieved using any of the stiff ODE solvers proposed in this study.

Summary

The optimum ODE solver for hydraulic systems is problem specific. However, Rosenbrock's method would be the most useful to the average hydraulic system designer. Although not as computationally expeditious as Gear's Method, Rosenbrock's method is considerably more efficient than the non-stiff solvers. The Rosenbrock method is also conceptually easier to understand than Gear's method and its robust stability properties facilitate the solution of a wider variety of problems. In addition, the restart costs of Rosenbrock's method are insignificant. This characteristics makes it compatible with the Event Switching Algorithm. For systems with a large number of components, these restart cost could become excessive if Gear's method were used.

CHAPTER 6 *Simulation of Hydraulic Systems*

EXPERIMENTAL VERIFICATION

Introduction

The ultimate goal of computer modeling and simulation of hydraulic systems is to determine the dynamic response of “real-world” hardware. This response information is invaluable to hydraulic system designers. In the computer environment, the effect of design changes on the dynamic response can be determined without the financial consequences of complicated “breadboard” testing. However, computer simulation can not completely replace laboratory tests. Computer analysis results are only as accurate as the methods used to simulate the physical system. As a result, verification of computer simulation results must be performed at some point in the design process. In this spirit, the dynamic response of an actual direct-acting relief valve was simulated using the modeling and integration techniques present in this work and then compared to available test data.

Test Component

The hydraulic component under consideration is a direct-acting relief valve. A typical direct-acting relief valve is represented in Figure 6.1. The flow through the valve remains at zero until the force acting on the poppet is sufficient to lift the poppet from its seat. Increasing pressure acting on the poppet area at Port P creates this force. As the poppet lifts from the seat, a flow path from Port P to Port T is created and the pressure relieved. A relief valve is typically used to limit hydraulic system pressure. Relief valves

protect other hydraulic system components from accidental overpressurization and prevent injuries related to overpressurization.

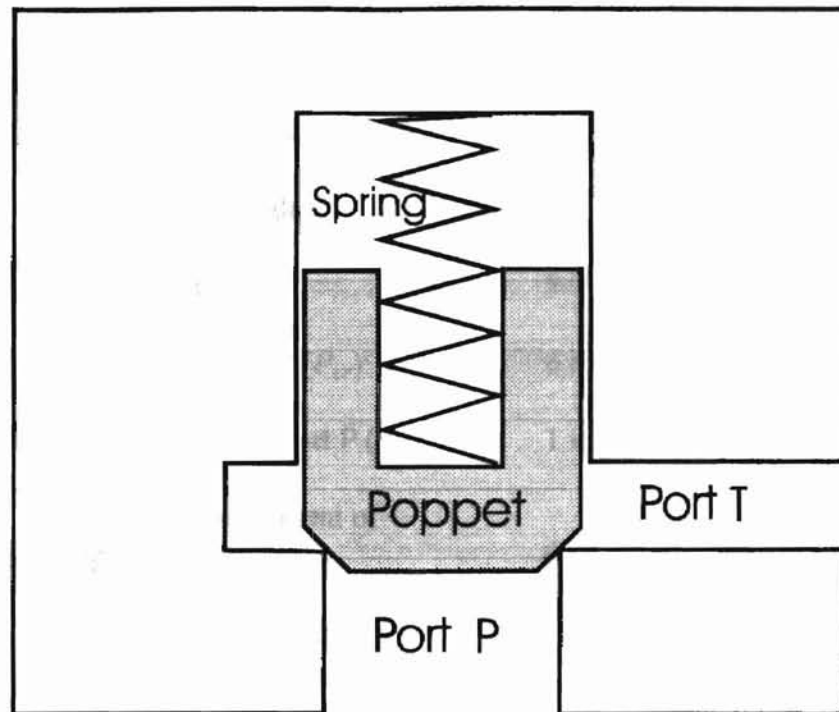


Figure 6.1: Typical Direct-Acting Pressure Relief Valve

The dynamic performance of a relief valve is of paramount importance for obvious safety reasons. As a result, performance testing of relief valves is common and data from such tests is readily available. The availability of such test data makes the direct-acting relief valve a perfect candidate for experimental verification of computer simulation results.

In this case, data from a relief valve test performed at FES Incorporated in Stillwater, Oklahoma was used. The test unit was a commercially available direct-acting

relief valve. Testing was performed with MIL-H-5606 hydraulic fluid at 38 Celsius. As a result of laboratory work, the following valve characteristics were identified:

Poppet Mass (m):	0.0031 kg
Spring Rate (k):	105076 N/m
Damping Coefficient (b):	35 N/(m/sec)
Poppet Half-Angle (α):	30°
Seat Diameter (d):	7.75×10^{-3} m
Cracking Pressure (P_{cr}):	6.897×10^6 Pa
Fluid Volume at Port P (v):	1×10^{-4} m ³
Discharge Coefficient of Poppet/Seat (C_d):	0.61
Bulk Modulus of Fluid (β):	1.8×10^9 N/m ²
Density of Fluid (ρ):	845 kg/m ³

Test System

For testing purposes, the test unit was installed in the hydraulic system depicted schematically in Figure 6.2. Shifting the directional control valve with an electrical

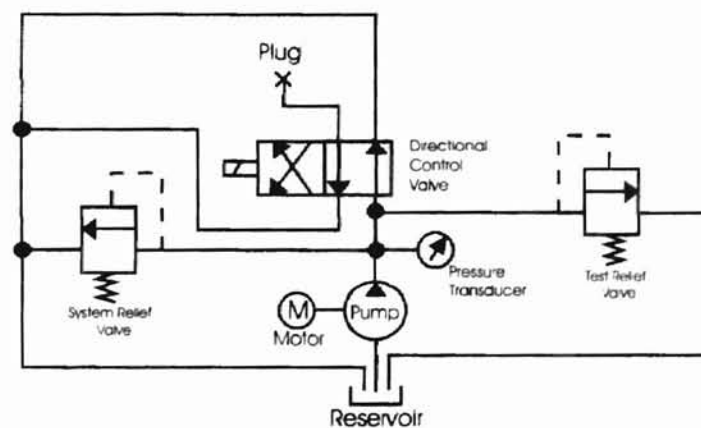


Figure 6.2: Direct-Acting Relief Valve Test Circuit

step input sends a flow rate (Q_m) of $6.305 \times 10^{-4} \text{ m}^3/\text{sec}$ to the test valve. However, a step input flow rate does not occur when the directional control valve is shifted. Fully developed flow occurs over a finite period of time. In an effort to account for this condition, a 15 millisecond ramp to full flow was utilized for Q_m .

Mathematical Models and Differential Equations

The mathematical model for variable-orifice flow and the differential equations for the direct-acting relief valve were developed in the same manner as those for the pilot-operated relief valve in Chapter 5 of this study. The resulting differential equations are as follows:

$$\dot{x}_1 = \frac{\beta}{v} \left(Q_m - \pi d \sin(\alpha) x_2 Q_2 - \frac{d^2 \pi}{4} x_3 \right) \quad (6.1)$$

$$\dot{x}_2 = x_3 \quad (6.2)$$

$$\dot{x}_3 = \frac{1}{m} \left(\frac{d^2 \pi}{4} x_1 - \frac{d^2 \pi}{4} P_{cr} - b x_3 - k x_2 - \pi C_d d \sin(2\alpha) x_1 x_2 \right) \quad (6.3)$$

Where,

x_1 = Pressure at Port P

x_2 = Poppet Displacement

x_3 = Poppet Velocity

Q_2 = Flow Rate Across Poppet Seat Arrangement

Results of Computer Analysis

A cubic spline based orifice model was used for the poppet/seat arrangement and the Event Switching Algorithm was employed for boundary handling. Numerical

integration of the differential equations was performed using Gear's Method for Stiff ODEs. Numerical Differentiation Formulas with a maximum order of 5 were utilized.

For a relief valve, the parameter of interest is the pressure at Port P. The dynamic response for this pressure, as predicted by employing the methods outlined in this work, is displayed in Figure 6.3. The following information can be gleaned from this plot:

Peak Pressure: 1.513×10^7 Pa

Peak Time: 0.0068 seconds

Steady State Pressure: 8.23×10^6 Pa

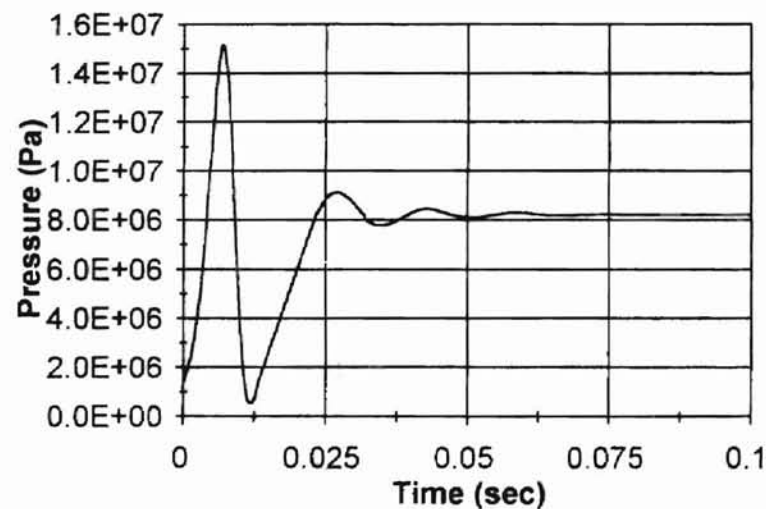


Figure 6.3: Simulation Results for Direct-Acting Relief Valve

Test Results

The dynamic response from laboratory testing is displayed in Figure 6.4. Again, the performance characteristics may be obtained from the plot:

Peak Pressure: 1.552×10^7 Pa
Peak Time: 0.008 seconds
Steady State Pressure (Average): 8.15×10^6 Pa

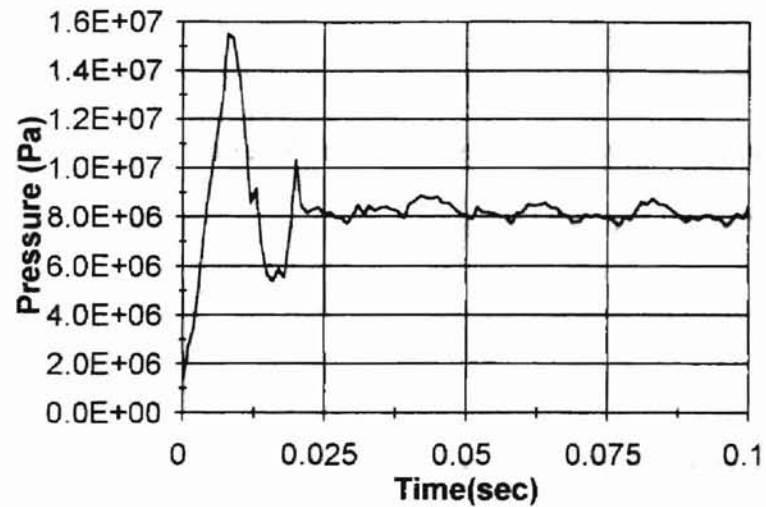


Figure 6.4: Test Results for Direct-Acting Relief Valve

The computer simulation result is superimposed over the measured dynamic response in Figure 6.5.

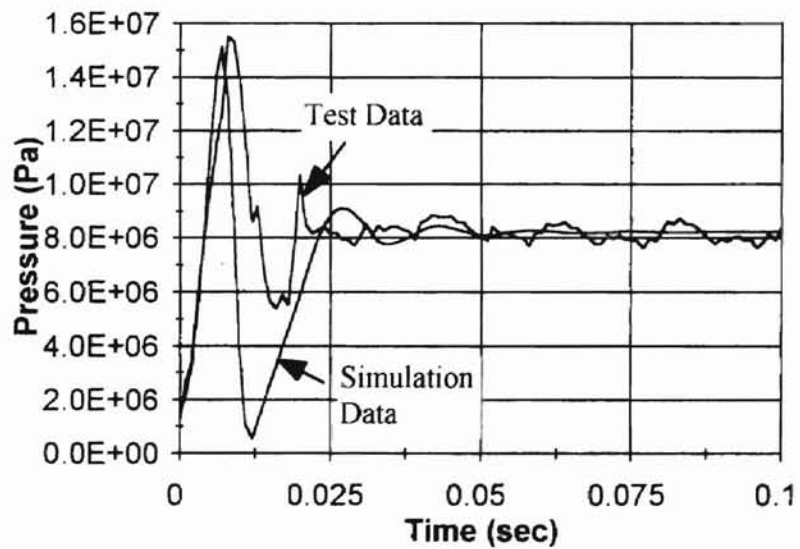


Figure 6.5: Simulation and Test Results Superimposed

Discussion of Results

A comparison of the test results to the computer simulation results reveals excellent correlation in terms of peak pressure. The difference between the two values is less than 3%. Peak pressure is the most important characteristic of relief valve performance. Pressure spikes are often responsible for hydraulic system damage. The predicted steady-state pressure value also closely matches the experimental data. Although steady-state analysis may be performed without numerical integration, the steady-state value is a good measure of correlation between computer-generated results and empirical results. Comparing the peak times reveals some difference in terms of response time. The predicted response is about 0.0017 seconds faster. This slight disparity is due the way the system was modeled. The dynamics of the pressure transducer were not considered in the system model. As a result, the measured response lags the predicted response. A more complex model could be developed to account for instrumentation dynamics if greater response time accuracy is desired.

CONCLUSION AND RECOMMENDATIONS

Conclusion

Hydraulic systems are generally considered to be mathematically stiff. This stiffness is the direct result of widely varying time constants within a single system. During numerical integration of a stiff set of ordinary differential equations, mathematical stiffness forces conventional ODE solvers to use very small time steps to maintain stability. These small time steps lead to a considerable amount of computational effort and processing time.

In an effort to reduce processing time, ODE solvers designed specifically for stiff problems have been devised. In addition, adaptive step size routines have been developed to improve computational efficiency. Hydraulic systems often contain discontinuities which cause both of these advancements to fail or become inefficient. Stiff ODE solvers use a Jacobian matrix to perpetuate a solution. Abrupt changes to the Jacobian matrix due to discontinuities will cause the stiff solver to fail. Similarly, abrupt changes will cause adaptive step size algorithms to "hunt" around a discontinuity until the step size is reduced enough to traverse the problem area. This hunting process is excessively time consuming.

In order to solve the problems caused by discontinuities, it is necessary to identify the types of discontinuities found in hydraulic systems. Discontinuities may exist in mathematical models or they may be encountered during the integration process. Model discontinuities typically result from slope discontinuities in the mathematical model.

The lack of mathematical smoothness causes abrupt changes in the Jacobian matrix and results in solver failure. Discontinuities encountered during numerical integration have the same disastrous effects. This type of discontinuity is termed a *state event*. In hydraulics, state events occur when mechanical devices encounter travel limits. For example, the differential equations describing a system will change if an actuator reaches a mechanical stop.

In this study, two powerful techniques have been developed to overcome the problems associated with these discontinuities. Cubic splines generated with a four-point method were created to smooth mathematical models. In addition, an *Event Switching Algorithm* (ESA) was developed to provide seamless integration over state events.

The four-point cubic spline technique provides a means to join two discontinuous functions. This technique involves positioning a control point in such close proximity to the true end point of a curve that the difference is physically insignificant. However, the relative position of the control point and the true end point determines the shape of the resulting curve. Control points may be used to force the slope continuity of a cubic spline bridging a point of discontinuity. This new technique has proven to be both versatile and effective. In addition to bridging discontinuities, cubic splines may be implemented to eliminate the infinite stiffness associated with infinite slopes.

The ESA provides a means to transcend state event discontinuities. Successful integration is accomplished by restarting the ODE solver when a state event is encountered. This restarting procedure eliminates abrupt changes to the Jacobian matrix. State events are defined by discontinuity functions. The exact time of a state event

occurs when a discontinuity function is equal to zero. For hydraulic systems, these discontinuity functions are based on the forces applied to a mass and on its displacement. After each step, the appropriate discontinuity functions are evaluated. If a state event is encountered (as signaled by a sign change), an event location routine is invoked to determine the precise time of the state event. Subsequently, the ODE solver is restarted with the new set of differential equations. This technique has proven effective in the successful integration of systems containing actuators or valve components which encounter physical travel limits.

With the aid of the cubic spline modeling tools and the ESA, hydraulic systems are compatible with ODE solvers specifically designed for stiff differential equations. Of these solvers, Gear's method and Rosenbrock's method are predominant. Gear's method has greater potential for computational efficiency. Tests performed in this study reveal processing times over 20 times faster than conventional (nonstiff) ODE solvers. While not as efficient as Gear's method, Rosenbrock's method has superior stability properties. As a result, Rosenbrock's method can solve a wider variety of problems.

Suggestions for Further Study

The value of curves developed by the four-point method and of the ESA have been demonstrated in this study. Significant gains in processing speed were realized by implementing these techniques. However, further study would refine and improve these techniques. A list of topics for further study is presented below:

- 1) The shape of the cubic spline used to bridge a model discontinuity may effect processing speed. Curves approaching sharp corners would likely slow the ODE solver

significantly. However, these sharper curves would more accurately represent the original discontinuous model. A study of processing speed versus model accuracy could give insight into this relationship.

2) The ESA uses an event location scheme to pinpoint the time at which a state event occurs. Several methods for event location are available. Among these methods are bisection, false position, secant, linear interpolation, inverse quadratic interpolation, and the Illinois method [18]. A comparison study of these methods may reveal advantages in terms of computational efficiency or stability. In this effort, the Illinois method was used exclusively.

3) If the ESA is applied to multiple objects in a single system, it is possible for more than one state variable to cross its threshold value during a single time step. An event location scheme designed to handle this condition would increase the versatility of the ESA.

Works Cited

- [1] Ellman, Asko, Robert Piche', and Matti Vilenius. "Integration of Numerically Stiff Fluid Power Circuit Models Using an L-Stable Runge-Kutta Method." Paper presented at *7th Bath Intl. Fluid Power Workshop*, September 1994.
- [2] Ellman, A.U. "Proposals for Utilizing Theoretical and Experimental Methods in Modelling Two-Way Cartridge Valve Circuits." *Acta Polytechnica* ME101. 1992 pp. 2-57.
- [3] Bowns, D. E., S. K. Dugdale, and S. P. Tomlinson. "Progress Towards a General Purpose Hydraulic System Simulation Language." *Proc. 6th Intl. Fluid Power Symposium*, 1981, pp.115-131.
- [4] Bowns, D. E., and L. M. Wang. "The Digital Computation of Pressures in Hydraulic Pipes with Small Volume Using an Iterative Technique." *Proc. Instn. Mech. Engrs.*, 1990, Part C, Vol. 204, pp. 29-36.
- [5] Ellman, A., and R. Piche'. "Numerical Integration of Fluid Power Circuit Models Using Two-Stage Semi-Implicit Runge-Kutta Methods." *Proc. Instn. Mech. Engrs.*, 1994, Part C, Vol. 208, pp. 167-175.
- [6] Krus, P. "The Simulation of Fluid Power Systems with Complex Load Dynamics." *International Journal of Modelling and Simulation*, 1986, Vol. 6, No. 2, pp. 52-57.
- [7] Krus, P.,and J. O. Palmberg. "Simulation of Fluid Power System in Time and Frequency Domains." *Proc. 7th Intl. Fluid Power Symposium*, 1986, pp. 73-79.
- [8] Bowns, D. E., R. E. Dorey, and S. P. Tomlinson. "Computer Simulation Techniques for the Dynamic Performance Assessment of Fluid Power Systems." *Proc. 7th Intl. Fluid Power Symposium*, 1986, pp.81-88.
- [9] Ellman, A. U., and M. J. Vilenius. "Methods for Simulating the Steady-State and Dynamic Behavior of Two-Way Cartridge Valve Circuits." *SAE J. of Commercial Vehicles*, 1990, Vol. 99, No. 2, pp. 384-393.
- [10] Carver, M. B., and S. R. MacEwen. "Numerical Analysis of a System Described by Implicitly-Defined Ordinary Differential Equations Containing Numerous Discontinuities." *Appl. Math. Modelling*, December 1978, Vol. 2, pp. 280-286.
- [11] Ellison, D. "Efficient Automatic Integration of Ordinary Differential Equations with Discontinuities." *Mathematics and Computers in Simulation*, 1981, Vol. XXIII, pp. 12-20.

- [12] Caney, K. "Integration Across Discontinuities in Ordinary Differential Equations Using Gear's Method." *University of Bath, School of Engineering Report 478*, 1979.
- [13] Berzins, M., and A. J. Preston. "Algorithms for Location of Discontinuities in Dynamic Simulation Problems." *Computers Chem. Engng*, 1991, Vol. 15, No. 10, pp. 701-713.
- [14] Crosbie, R. E., and J. L. Hay. "Digital Techniques for the Simulation of Discontinuities." *Proc. of the 1974 Summer Computer Simulation Conference*, 1974, pp. 87-91.
- [15] Carver, M. B. "Efficient Integration Over Discontinuities in Ordinary Differential Equation Simulations." *Mathematics and Computers in Simulation*, 1978, Vol. XX, pp. 190-196.
- [16] O'Regan, P. G. "Step Size Adjustment at Discontinuities for Fourth Order Runge-Kutta Methods." *The Computer Journal*, 1970, Vol. 13, No. 4, pp. 401-404.
- [17] Chaplin, R.I., R.E. Crosbie, and J. L. Hay. "Integration Routines for Systems with Discontinuities." *The Computer Journal*, 1974, Vol. 17, No. 3, pp. 275-278.
- [18] Moler, C. "Are We There Yet." *Matlab News and Notes, Simulink 2 Special Edition*, 1997, pp. 16-17.
- [19] Gerald, C. F., and P. O. Wheatley. *Applied Numerical Analysis*. 5th ed. Reading, MA: Addison-Wesley 1994, pp. 233-244.
- [20] Fitch, E. C., and I. T. Hong. *Hydraulic Component Design and Selection*. Stillwater, OK: Bardyne 1997. pp. 37-42.
- [21] Lapidus, L., and W. E. Schiesser. *Numerical Methods for Differential Systems: Recent Developments in Algorithms, Software, and Applications*, New York: Academic Press 1976, pp. 97-124.
- [22] *Using Matlab*. Natick, MA: The Mathworks, Inc. 1997. pp. 8-1 to 8-52.
- [23] Reichelt, M. W., and L. F. Shampine. "The Matlab ODE Suite." *SIAM J. Sci. Comput.*, January 1997, Vol. 18, No. 1, pp. 1-22.

Appendix A

Procedure for Generating Cubic Splines to Interpolate Between Four Points

In matrix form, three cubic spline polynomials connecting four points may be generated as follows [19].

- 1) Define the four points in the points in Cartesian coordinates:

$$\begin{aligned} \text{point 1} &= (x_1, y_1) \\ \text{point 2} &= (x_2, y_2) \\ \text{point 3} &= (x_3, y_3) \\ \text{point 4} &= (x_4, y_4) \end{aligned} \quad (\text{A.1})$$

- 2) Create X and Y matrices:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \quad (\text{A.2})$$

- 3) Generate H matrix:

$$H = \begin{bmatrix} x_2 - x_1 \\ x_3 - x_2 \\ x_4 - x_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \quad (\text{A.3})$$

- 4) Create the RHS (right hand side) matrix:

$$\text{RHS} = \begin{bmatrix} 0 \\ 6 \left(\frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1} \right) \\ 6 \left(\frac{y_4 - y_3}{h_3} - \frac{y_3 - y_2}{h_2} \right) \\ 0 \end{bmatrix} \quad (\text{A.4})$$

- 5) Build the A matrix:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ h_1 & 2(h_1 + h_2) & h_2 & 0 \\ 0 & h_2 & 2(h_2 + h_3) & h_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.5})$$

- 6) Solve set of equations to get S matrix:

$$S = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix} = A^{-1} \cdot \text{RHS} \quad (\text{A.6})$$

- 7) The three cubic spline equations are:

Between point 1 and point 2: (A.7)

$$g_1 = (x-x_1)^3 \left(\frac{S_2 - S_1}{6h_1} \right) + (x-x_1)^2 \left(\frac{S_1}{2} \right) + (x-x_1) \left(\frac{y_2 - y_1}{h_1} - \frac{2h_1 S_1 + h_1 S_2}{6} \right) + y_1$$

Between point 2 and point 3: (A.8)

$$g_2 = (x-x_2)^3 \left(\frac{S_3 - S_2}{6h_2} \right) + (x-x_2)^2 \left(\frac{S_2}{2} \right) + (x-x_2) \left(\frac{y_3 - y_2}{h_2} - \frac{2h_2 S_2 + h_2 S_3}{6} \right) + y_2$$

Between point 3 and point 4: (A.9)

$$g_3 = (x-x_3)^3 \left(\frac{S_4 - S_3}{6h_3} \right) + (x-x_3)^2 \left(\frac{S_3}{2} \right) + (x-x_3) \left(\frac{y_4 - y_3}{h_3} - \frac{2h_3 S_3 + h_3 S_4}{6} \right) + y_3$$

Appendix B

Experimental Determination of the Pressure and Flow Relationship of an Orifice

Test System and Equipment

The hydraulic system used for this test is depicted in Figure B-1. In addition to the hydraulic system, a stop watch and graduated cylinder were required.

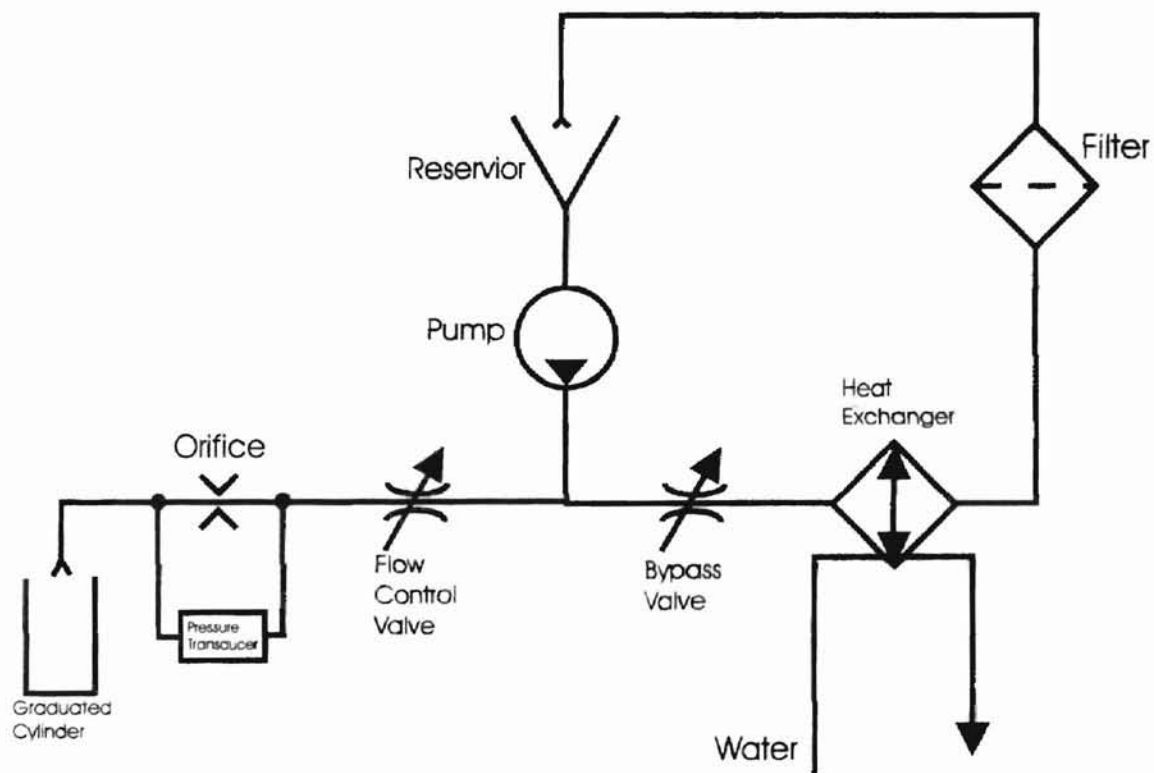


Figure B-1: Hydraulic System for Orifice Testing

Test Conditions

Test Fluid: SAE 10W Oil

Fluid Temperature: 73 °F

Orifice Dimensions: Diameter = 0.012 inches Thickness = 0.010 inches

Test Procedure

- 1) The test system was filled with the specified test fluid.
- 2) The bypass valve was opened.
- 3) The pump drive motor was started.
- 4) Water flow rate through heat exchanger was adjusted to achieve the desired fluid temperature.
- 5) The bypass valve and flow control valve were adjusted to obtain incrementally larger pressure differences across the orifice.
- 6) At each increment, the flow rate through the orifice was measured and recorded using a graduated cylinder and a stop watch.

Test Results**Table B-1: Test Results for 0.012 Diameter Orifice**

Pressure Difference (psid)	Flow Rate (in³/sec.)
0.0	0.0000
2.5	0.0069
5.0	0.0135
7.5	0.0192
10.0	0.0244
12.5	0.0288
15.0	0.0335
17.5	0.0374
20.0	0.0412
22.5	0.0439
25.0	0.0474
30.0	0.0539
35.0	0.0593
40.0	0.0643
45.0	0.0685
50.0	0.0743
55.0	0.0801
60.0	0.0839
65.0	0.0893
70.0	0.0936
75.0	0.0970
80.0	0.0997
85.0	0.1024
90.0	0.1055
95.0	0.1084
100.0	0.1114
105.0	0.1141
110.0	0.1168
115.0	0.1196
120.0	0.1223

Appendix C

MATLAB Script File for Example 3.1


```

%File Name: sp8ex.m
%File Description:
%   MATLAB Script file used to generate cubic spline for
%   orifice model based on experimental data (Example 3.1).
%   0.012 orifice; SAE 10W @ 73 F

%%DECLARE EMPTY MATRICES FOR PLOTTING PURPOSES
xv=[];
xp=[];
xp1=[];
xp2=[];
gx=[];
gx1=[];
gx2=[];

%%INITIALIZE VARIABLE CONSTANTS
n=4; %number of points for cubic spline
dia=0.012; %orifice diameter
area=((dia^2)*pi)/4; %orifice flow area
ro=8.171e-5; %fluid density (lbf-sec^2/in^4)
v=.1623; %kinematic viscosity of fluid (in^2/sec)
cd=.63; %orifice discharge coefficient
cf=3.85;%conversion factor - in^3/sec to GPM

%% DEPARTING SLOPE AT ORIGIN
slope=.00220

%% GENERATE CONTROL POINT AT ORIGIN
%% (x1,y1) = True end point
%% (x2,y2) = Control point

x1=0;
y1=0;
epsx=1;
x2=x1+epsx*eps;
y2=slope*x2;
while((y2-y1)<eps)
    x2=x1+epsx*eps;
    y2=slope*x2;
    epsx=epsx*10;
end

%% GENERATE CONTROL POINT AT 85psid
%% (x3,y3) = True end point
%% (x4,y4) = Control point

x3=85; % Pressure difference at second end point
y3=cd*area*sqrt(2/ro)*sqrt(x3); %flow rate at 85 psid
epsx=1;
x4=x3+epsx*eps;
y4=cd*area*sqrt(2/ro)*sqrt(x4);
while((y4-y3)<eps)
    x4=x3+epsx*eps;
    y4=cd*area*sqrt(2/ro)*sqrt(x4);
    epsx=epsx*10;
end

%% RESULTING 4 POINTS FOR CUBIC SPLINES
x=[x1;x2;x3;x4]
y=[y1;y2;y3;y4]

```

```

%%Generate H Matrix
for i=1:1:(n-1);
    k(i)=x(i+1)-x(i);
end
h=k';

%% Generate RHS Matrix
rhs(1,1)=0;
rhs(n,1)=0;

for i=2:1:(n-1);
    rhs(i,1)=6*((y(i+1,1)-y(i,1))/h(i,1)-(y(i,1)-y(i-1,1))/h(i-1,1));
end

%% Generate A matrix
a(1,1)=1;
a(n,n)=1;

for i=2:1:(n-1)
    a(i,i-1)=h(i-1,1);
    a(i,i+1)=h(i,1);
    a(i,i)=2*(h(i-1,1)+h(i,1));
end

%% Solve System to get S Matrix
s=inv(a)*rhs;

%% Use S values to calculate coefficients of cubic splines
for i=1:1:(n-1)
    interval=i;
    cubeco(i)=(s(i+1,1)-s(i,1))/(6*h(i,1));
    squareco(i)=s(i,1)/2;
    linco(i)=((y(i+1,1)-y(i,1))/h(i,1)-
        (2*h(i,1)*s(i,1)+h(i,1)*s(i+1,1))/6;
end

%% Generate points for plotting
%% Only the middle interval is important
incr=2.5; %pressure increment for plot points
i=1; %% generate plot points for cubic (xp,gx)
for xv=x(2,1):incr:x(3,1)
    xp(i,1)=xv;
    gx(i,1)=(((xv-x(2,1))^3)*cubeco(2)+((xv-x(2,1))^2)*squareco(2)+(xv-
        x(2,1))*linco(2)+y(2,1));
    i=i+1;
end

%% Generate plot points for Turbulent flow Equation (xpl,gx1)
xv=eps:incr:90;
xpl=xv;
gx1=area*(cd*sqrt(2/ro)*(xv).^0.5);

% Plot the original test data
express=[0 2.5 5 7.5 10 12.5 15 17.5 20 22.5 25 30 35 40 45 50 55 60
    65 70 75 80 85];
exq=cf*[0 1.8e-3 3.5e-3 4.98e-3 6.35e-3 7.48e-3 8.7e-3 9.71e-3 1.07e-
    2 1.14e-2 1.23e-2 1.40e-2 1.54e-2 1.67e-2 1.78e-2 1.93e-2
    2.08e-2 2.18e-2 2.32e-2 2.43e-2 2.52e-2 2.59e-2 2.66e-2];
plot(express,exq,'b');
hold

```

```
plot(express,exq,'+');

plot(xp,gx,'r') %% plot cubic spline model
plot(xp1,gx1,'g') %%plot turbulent flow eq
plot(x3,y3,'o') %% plot transition point

xlabel('Pressure (psid)')
ylabel('Flow Rate (cu.in/sec)')

%% print resulting cubic spline coefficients to screen
coeffs=[cubeco(2);squareco(2);linco(2)]
```

Appendix D

MATLAB Script Files for Example 3.2

```

%File Name: staticrv3.m
%File Description:
%   MATLAB Script file used to generate
%   the first of two cubic splines for
%   the static relief valve model of
%   Example 3.2

%%DECLARE EMPTY MATRICES FOR PLOTTING PURPOSES
xv=[];
xp=[];
xp1=[];
xp2=[];
gx=[];
gx1=[];
gx2=[];

%%INITIALIZE VARIABLE CONSTANTS
n=4; %number of points for cubic spline
cd=.61; %orifice discharge coefficient
Pcr=800; %cracking pressure(psid)
Pmax=850; %Maximum pressure (psid)
Qmax=45.045; %Maximum flow rate (cu. in. /sec)
Kv=Qmax/sqrt(Pmax); %lumped constant for turb. flow EQ.
slope=Qmax/(Pmax-Pcr); %slope (m) of linear (spring) portion of model
offset=Qmax-slope*Pmax; %offset (b) of straight line equation

%% GENERATE CONTROL POINT AT 1st End point
%% (x1,y1) = True end point
%% (x2,y2) = Control point

x1=720; %Desired endpoint pressure
y1=0; %flow rate corresponding to x1
epsx=1;
x2=x1+epsx*eps;
y2=0;
while((x2-x1)<eps)
    x2=x1+epsx*eps;
    epsx=epsx*10;
end

%% GENERATE CONTROL POINT AT 2nd End point
%% (x1,y1) = True end point
%% (x2,y2) = Control point

x3=825; %Desired endpoint pressure
y3=slope*x3+offset; %flow rate corresponding to x3
epsx=1;
x4=x3+epsx*eps;
y4=slope*x4+offset;

while((y4-y3)<eps)
    x4=x3+epsx*eps;
    y4=slope*x4+offset;
    epsx=epsx*10;
end

%% RESULTING 4 POINTS FOR CUBIC SPLINES
x=[x1;x2;x3;x4]
y=[y1;y2;y3;y4]

```

```

%%Generate H Matrix
for i=1:1:(n-1);
    k(i)=x(i+1)-x(i);
end
h=k';

%% Generate RHS Matrix
rhs(1,1)=0;
rhs(n,1)=0;

for i=2:1:(n-1);
    rhs(i,1)=6*((y(i+1,1)-y(i,1))/h(i,1)-(y(i,1)-y(i-1,1))/h(i-1,1));
end

%% Generate A matrix
a(1,1)=1;
a(n,n)=1;

for i=2:1:(n-1)
    a(i,i-1)=h(i-1,1);
    a(i,i+1)=h(i,1);
    a(i,i)=2*(h(i-1,1)+h(i,1));
end

%% Solve System to get S Matrix
s=inv(a)*rhs;

%% Use S values to calculate coefficients of cubic splines
for i=1:1:(n-1)
    interval=i;
    cubeco(i)=(s(i+1,1)-s(i,1))/(6*h(i,1));
    squareco(i)=s(i,1)/2;
    linco(i)=((y(i+1,1)-y(i,1))/h(i,1))-
(2*h(i,1)*s(i,1)+h(i,1)*s(i+1,1))/6;
end

%% Generate points for plotting
%% Only the middle interval is important
incr=(x(n,1)-x(1,1))/100; %pressure increment for plot points
i=1; %% generate plot points for cubic (xp,gx)
for xv=x(2,1):incr:x(3,1)
    xp(i,1)=xv;
    gx(i,1)=(((xv-x(2,1))^3)*cubeco(2)+((xv-x(2,1))^2)*squareco(2)+(xv-
x(2,1))*linco(2)+y(2,1));
    i=i+1;
end

%% Generate plot points for straight line
%% portion of original model
xv=0:incr:Pmax+incr;
xp1=xv;
gx1=slope*xv+offset;

%% Generate plot points for turb flow Eq.
%% portion of original model
xv=0:incr:1.4*Pmax;
xp2=xv;
gx2=Kv*xp2.^0.5;

```

```
plot(xp,gx,'r') %% plot cubic spline
axis([0 1.4*Pmax 0 1.4*Qmax])
hold
plot(xp2,gx2,'b') %% plot turbulent flow eq
plot(xp1,gx1,'g') %% plot linear portion

%% print resulting cubic spline coefficients to screen
coeffs=[cubeco(2);squareco(2);linco(2)]
```

```

%File Name: staticrv2.m
%File Description:
%   MATLAB Script file used to generate
%   the second of two cubic splines for
%   the static relief valve model of
%   Example 3.2

%%DECLARE EMPTY MATRICES FOR PLOTTING PURPOSES
xv=[];
xp=[];
xp1=[];
xp2=[];
gx=[];
gx1=[];
gx2=[];

%%INITIALIZE VARIABLE CONSTANTS
n=4; %number of points for cubic spline
cd=.61; %orifice discharge coefficient
Pcr=800; %cracking pressure (psid)
Pmax=850; %Maximum pressure (psid)
Qmax=45.045; %Maximum flow rate (cu. in. /sec)
Kv=Qmax/sqrt(Pmax); %lumped constant for turb. flow EQ.
slope=Qmax/(Pmax-Pcr); %slope (m) of linear (spring) portion of model
offset=Qmax-slope*Pmax; %offset (b) of straight line equation

%% GENERATE CONTROL AT 1st End point
%% (x1,y1) = True end point
%% (x2,y2) = Control point

x1=840; %Desired endpoint pressure
y1=slope*x1+offset; %flow rate corresponding to x1
epsx=1;
x2=x1+epsx*eps;
y2=slope*x2+offset;
while((y2-y1)<eps)
    x2=x1+epsx*eps;
    y2=slope*x2+offset;
    epsx=epsx*10;
end

%% GENERATE CONTROL AT 2nd End point
%% (x1,y1) = True end point
%% (x2,y2) = Control point

x3=900; %Desired endpoint pressure
y3=Kv*sqrt(x3); %flow rate corresponding to x3
epsx=1;
x4=x3+epsx*eps;
y4=Kv*sqrt(x4);
while((y4-y3)<eps)
    x4=x3+epsx*eps;
    y4=Kv*sqrt(x4);
    epsx=epsx*10;
end

%% RESULTING 4 POINTS FOR CUBIC SPLINES
x=[x1;x2;x3;x4]
y=[y1;y2;y3;y4]

```



```

%%Generate H Matrix
for i=1:1:(n-1);
    k(i)=x(i+1)-x(i);
end
h=k';

%% Generate RHS Matrix
rhs(1,1)=0;
rhs(n,1)=0;

for i=2:1:(n-1);
    rhs(i,1)=6*((y(i+1,1)-y(i,1))/h(i,1)-(y(i,1)-y(i-1,1))/h(i-1,1));
end

%% Generate A matrix
a(1,1)=1;
a(n,n)=1;

for i=2:1:(n-1)
    a(i,i-1)=h(i-1,1);
    a(i,i+1)=h(i,1);
    a(i,i)=2*(h(i-1,1)+h(i,1));
end

%% Solve System to get S Matrix
s=inv(a)*rhs;

%% Use S values to calculate coefficients of cubic splines
for i=1:1:(n-1)
    interval=i;
    cubeco(i)=(s(i+1,1)-s(i,1))/(6*h(i,1));
    squareco(i)=s(i,1)/2;
    linco(i)=(y(i+1,1)-y(i,1))/h(i,1)-
    (2*h(i,1)*s(i,1)+h(i,1)*s(i+1,1))/6;
end

%% Generate points for plotting
%% Only the middle interval is important
incr=(x(n,1)-x(1,1))/20; %pressure increment for plot points
i=1; %% generate plot points for cubic (xp,gx)
for xv=x(2,1):incr:x(3,1)
    xp(i,1)=xv;
    gx(i,1)=(((xv-x(2,1))^3)*cubeco(2)+((xv-x(2,1))^2)*squareco(2)+(xv-
x(2,1))*linco(2)+y(2,1));
    i=i+1;
end

%% Generate plot points for straight line
%% portion of original model
xv=0:incr:Pmax+incr;
xp1=xv;
gx1=slope*xv+offset;

%% Generate plot points for turb flow Eq.
%% portion of original model
xv=Pmax:incr:1.4*Pmax;
xp2=xv;
gx2=Kv*xp2.^5;

```

```
plot(xp,gx,'r') %% plot cubic spline
axis([0 1.4*Pmax 0 1.4*Qmax])
hold
plot(xp2,gx2,'b') %% plot turbulent flow eq
plot(xp1,gx1,'g') %% plot linear portion

%% print resulting cubic spline coefficients to screen
coeffs=[cubeco(2);squareco(2);linco(2)]
```

Appendix E

MATLAB Script Files for Example 4.1

```

%File Name: cyl5dr.m
%File Description:
%   MATLAB script file used for Example 4.1
%   This file is the driver for cyl5.m

%Initialize time(t) and state variable (x) matrices
t=[];
x=[];

%% set ODE solver options
options=odeset('Events','on','Abstol',1e-7,'RelTol',1e-4,'MaxOrder',3);

start=cputime; %% record time of day at which computation is started
tfinal=.75; %% time to end simulation
tspan=0:tfinal/500:tfinal; % time span with refinement
y0=[0 0 0 0 0 0]; % initial conditions
mlstatic=0; % Initial state flag value
tel=0; % Initialize termination time

%% Core Integration Loop
while(tel < tfinal) %% while termination time is less than final time
    tspan=tel:(length(tel)):tfinal/500:tfinal; %% time span for current
    section
    %% Call Desired ODE solver
    %% Upon termination solver returns:
    %%     time step values and corresponding state values (ti, xi)
    %%     termination time step value(tel)
    %%     state values at termination (yei)
    %%     flag to identify which event caused termination (iei)
    [ti xi tel yei iei]=ode15s('cyl5',tspan,y0,options,mlstatic);
    t=[t' ti']; %% save combined results (t and x)
    x=[x' xi'];

    iecheck=isempty(iei); %% if no event is detected, iecheck =0;

    if(iecheck==0) %% if no termination due to event (end simulation)
        %% tfinal has been reached

        %% If state B is reached, reset initial conditions
        %% and state flag value
        if(iei==1)
            yei(length(tel),6)=0;
            y0=yei(length(tel),:);
            mlstatic=0;
        end

        %% If State A is reached, reset initial conditions
        %% and state flag value
        if(iei==2)
            yei(length(tel),6)=0;
            y0=yei(length(tel),:);
            mlstatic=-1;
        end

        %% If State C is reached, reset initial conditions
        %% and state flag value
        if(iei==3)
            yei(length(tel),6)=0;
            y0=yei(length(tel),:);
            mlstatic=1;
        end
    end
end

```

```
end % end iecheck
```

```
end % end while loop
```

```
stop=cputime; %% record ending time of day
```

```
elapsed=stop-start %% calculate elapsed processing time
```

```

%% File Name: cyl5.m
%% File Description:
%% This file contains the differential equations
%% for integration and event location info.
%% for Example 4.1

function [out1,out2,out3]=cyl5(t,x,flag,m1static) %%declare function
                                                name

if nargin < 3 | isempty(flag)

beta=150000; %% bulk modulus
vol=10; %% fluid line volumes
Qin=38.5; %% pump discharge flow
Pcr=800; %% relief valve cracking pressure
Pmax=850; %% max pressure for relief valve model
Qmax=45.05; %% max flow for relief valve model
Abore=((4^2)*pi)/4; %% cylinder bore area
Arod=Abore-(((2.5^2)*pi)/4); %% cylinder rod end area
m=9.98965; %%lumped load mass
k=700; %% load spring rate
b=200; %% load damping ratio
Ddc=0.27842; %%Effective directional control valve diameter
Dor=0.11277; %% orifice diameter
Adc=(Ddc^2*pi)/4; %% directional control valve flow area
Aor=(Dor^2*pi)/4; %% orifice flow area
ylimit=2; %% cylinder stroke limit

%% transition pressure values orifice models
ptdcl=1.240278210965916e+000;
ptdc2=ptdcl;
ptfor=7.560191764662377e0;

%% Implement various orifice flow models and provide for reverse flow
if abs(x(1)-x(2))>ptdcl;
    temp1=(.61*Adc*sqrt(2/7.95e-5))*sqrt(abs(x(1)-x(2)));
else (ptdcl>=abs(x(1)-x(2)));
    temp1=(9.350e-001*(abs(x(1)-x(2)))^3-4.649e+000*(abs(x(1)-
        x(2)))^2+9.617e+000*(abs(x(1)-x(2))));
    end
if x(1)<0;
    u1=-temp1;
else
    u1=temp1;
end

if abs(x(3)-x(4))>ptfor;
    temp2=(.61*Aor*sqrt(2/7.95e-5))*sqrt(abs(x(3)-x(4)));
else (ptfor>=abs(x(3)-x(4)));
    temp2=(2.031e-3*(abs(x(3)-x(4)))^3-5.339e-2*((x(3)-x(4)))^2+6.390e-
        1*(abs(x(3)-x(4))));
    end
if (x(3)-x(4))<0;
    uor=-temp2;
else
    uor=temp2;
end

if (abs(x(4))>ptdc2);
    temp3=(.61*Adc*sqrt(2/7.95e-5))*sqrt(abs(x(4)));
else (ptdc2>=abs(x(4)));

```

```

    temp3=(9.350e-001*(abs(x(4)))^3-
        4.649e+000*(abs(x(4)))^2+9.617e+000*(abs(x(4))));
end
if x(4)<0;
    u2=-temp3;
else
    u2=temp3;
end

%% Calculate various flow rates
Qdc1=u1;
Qdc2=u2;
Qor=uor;
Qbore=x(6)*Abore;
Qrod=x(6)*Arod;

%% Relief valve model
if(x(1)<=720)
    Qrv=0;
end
if(x(1)>720 & x(1)<=825)
    Qrv=6.440e-006*(x(1)-720)^3+(1.367e-003)*(x(1)-720)^2+1.943e-
        016*(x(1)-720);
end
if(x(1)>825 & x(1)<=840)
    slope=Qmax/(Pmax-Pcr);
    offset=Qmax-slope*Pmax;
    Qrv=5.2061e-005*(x(1)-840)^3+(-8.5917e-003)*(x(1)-840)^2+5.0000e-
        001*(x(1)-840)+(slope*840+offset);
end
if(x(1)>840 & x(1)<900)
    slope=Qmax/(Pmax-Pcr);
    offset=Qmax-slope*Pmax;
    Qrv=5.2061e-005*(x(1)-840)^3+(-8.5917e-003)*(x(1)-840)^2+5.0000e-
        001*(x(1)-840)+(slope*840+offset);
end
if(x(1)>900)
    Qrv=1.545*sqrt(x(1));
end

%% Differential Equations
dx=zeros(6,1);
dx(1)=(beta/vol)*(Qin-Qrv-Qdc1);
dx(2)=(beta/vol+Abore*x(5))*(Qdc1-Qbore);
dx(3)=(beta/vol+Arod*(ylimit-x(5)))*(Qrod-Qor);
dx(4)=(beta/vol)*(Qor-Qdc2);

if(mlstatic==-1 | mlstatic==1) %% if mass is pinned against stop
dx(5)=0;
dx(6)=0;
else
    %% Call cyl5a - contains equations of motion for lumped load
    dxtemp=cyl5a(t,x);
    dx(5)=dxtemp(1);
    dx(6)=dxtemp(2);
end

out1=dx; %% send function evaluations to solver

else

```

```
switch(flag)

case 'events' %% event location info.
    ylimit=2; %% travel limit

    %% Discontinuity functions for lumped load
    if(mlstatic==0)
        ymin=x(5);
        ymax=ylimit-x(5);
        sumforces=0;
    end
    if(mlstatic~=0)
        ymin=0;
        ymax=0;
        dxtemp=cyl5a(t,x); % call to equations of motion for load
                        % for use as discontinuity function
        sumforces=mlstatic*dxtemp(2); %% multiply by state flag
    end

    %% send discontinuity functions to event locater
    out1=[sumforces;ymin;ymax];
    out2=[1;1;1]; %% any zero crossing is terminal
    out3=[-1;-1;-1]; %% terminate when crossing from pos. to neg.

end
end
```



```
% File Name: cyl5a.m
% File Description:
%   Contains equations of motion for lumped load
%   for access as function evaluations or
%   as discontinuity functions
function [xdot]=cyl5a(t,x)

%% Physical parameters
Abore=((4^2)*pi)/4;
Arod=Abore-(((2.5^2)*pi)/4);
m=9.98965;
k=700;
b=200;

%% Equations of motion
xdot(1)=x(6);
xdot(2)=(1/m)*(Abore*x(2)-Arod*x(3)-k*x(5)-b*x(6));
```

Appendix F

MATLAB Script Files for Pilot-Operated Relief Valve
Case Study from Chapter 5

```

%File Name: relv8dr.m
%File Description:
%   MATLAB script file used for Chapter 5 Case Study.
%   This file is the driver for relv8.m

%Initialize time(t) and state variable (x) matrices
t=[];
x=[];

%% Set ODE Solver options
options=odeset('Events','on','RelTol',1e-3,'AbsTol',1e-
              6,'BDF','off','MaxOrder',5);

start=cputime; %% record time of day at which computation is started
tfinal=.08; %% time to end simulation
tspan=0:.0001:tfinal; % time span with refinement
y0=[0 0 0 0 0 0]; %initial conditions
m2static=-1; % Initial state flag value for pilot poppet
mlstatic=-1; % Initial state flag value for main poppet
tel=0;

%% Core Integration Loop
while(tel < tfinal)%% while termination time is less than final time
tspan=tel(length(tel)).:0.0005:tfinal;%% time span for current section
%% Call Desired ODE solver
%% Upon termination solver returns:
%%   time step values and corresponding state values (t1, x1)
%%   termination time step value (tel)
%%   state values at termination (yel)
%%   flag to identify which event caused termination (iel)
[t1 x1 tel yel iel]=ode15s('relv8',tspan,y0,options,mlstatic,m2static);
t=[t' t1']; %% save combined results (t and x)
x=[x' x1'];
iecheck=isempty(iel); %% if no event is detected, iecheck =0;

if(iecheck==0) %% if no termination due to event (end simulation)
%%   tfinal has been reached

%% If state B is reached by either poppet, reset initial conditions
%% and state flag value
if(iel==1)
    yel(length(tel),6)=0;
    y0=yel(length(tel),:);
    m2static=0;
end

if(iel==2)
    yel(length(tel),4)=0;
    y0=yel(length(tel),:);
    mlstatic=0;
end

%% If state A is reached by either poppet, reset initial conditions
%% and state flag value
if(iel==3)
    yel(length(tel),6)=0;
    y0=yel(length(tel),:);
    m2static=-1;
end
end

```

```
if(iel==5)
    yel(length(tel),4)=0;
    y0=yel(length(tel),:);
    m1static=-1;
end

%% If state C is reached by either poppet, reset initial conditions
and state flag value
if(iel==4)
    yel(length(tel),6)=0;
    y0=yel(length(tel),:);
    m2static=1;
end

if(iel==6)
    yel(length(tel),4)=0;
    y0=yel(length(tel),:);
    m1static=1;
end

end % end ieccheck

end % end while loop

stop=cputime; %% record ending time of day
elapsed=stop-start % calculate elapsed processing time
```

```

%% File Name: relv8.m
%% File Description:
%% This file contains the differential equations
%% for integration and event location info.
%% for Chap. 5 Case Study.

function [out1,out2,out3]=relv8(t,x,flag,m1static,m2static) %*declare
                                                    function name

if nargin < 3 | isempty(flag)

%% Initialize Constants
Qin=1e-3;
D2=.005;
A2=(D2^2*pi)/4;
D1=.017;
Dp=1.0219*D1;
A1=(D1^2*pi)/4;
Ap=(Dp^2*pi)/4;
Dor=.001;
Aor=(Dor^2*pi)/4;
alpha1=pi/6;
alpha2=pi/6;
rho=845;
vis=14.3e-6;
Ddc=0.008;
Adc=((Ddc^2)*pi)/4;
V1=3e-4;
V2=1e-7;
beta=1.03e9;

%% transition pressure values orifice models
ptf1=1.808e3;
ptf2=2.090e4;
ptfor=5.224e5;
ptfdc=8.163e3;

%% Implement various orifice flow models and provide for reverse flow
if abs(x(1))>ptf1;
    temp1=0.61*sqrt(2/rho)*sqrt(abs(x(1)));
else (ptf1>=abs(x(1)));
    temp1=(1/A1)*(1.543e-14*(abs(x(1)))^3-9.959e-11*(abs(x(1)))^2+2.881e-
        7*(abs(x(1))));
end
if x(1)<0;
    ul=-temp1;
else
    ul=temp1;
end

if abs(x(1)-x(2))>ptfor;
    temp2=0.61*sqrt(2/rho)*sqrt(abs(x(1)-x(2)));
else (ptfor>=abs(x(1)-x(2)));
    temp2=(1/Aor)*(3.759e-23*(abs(x(1)-x(2)))^3-7.014e-17*(abs(x(1)-
        x(2)))^2+5.863e-11*(abs(x(1)-x(2))));
end
if (x(1)-x(2))<0;
    uor=-temp2;
else
    uor=temp2;
end
end

```

```

if abs(x(2))>ptf2;
    temp3=0.61*sqrt(2/rho)*sqrt(abs(x(2)));
else (ptf2>=abs(x(2)));
    temp3=(1/A2)*(2.937e-18*(abs(x(2)))^3-2.192e-13*(abs(x(2)))^2+7.329e-
        9*(abs(x(2))));
end
if x(2)<0;
    u2=-temp3;
else
    u2=temp3;
end

if abs(x(1))>ptfdc;
    temp4=0.61*sqrt(2/rho)*sqrt(abs(x(1)));
else (ptfdc>=abs(x(1)));
    temp4=(1/Adc)*(7.884e-17*(abs(x(1)))^3-2.298e-
        12*(abs(x(1)))^2+3.002e-8*(abs(x(1))));
end

if x(1)<0;
    udc=-temp4;
else
    udc=temp4;
end

%% Differential Equations
dx=zeros(6,1);

if(t<0.04)%% Direction control valve open to relief valve
dx(1)=(beta/V1)*(Qin-pi*D1*sin(alpha)*u1*x(3)-(Aor)*uor-A1*x(4));
else %% Directional control valve shifted to bypass relief valve
dx(1)=(beta/V1)*(Qin-Adc*udc-pi*D1*sin(alpha)*u1*x(3)-(Aor)*uor-
    A1*x(4));
end
dx(2)=(beta/V2)*(Aor*uor+(Ap)*x(4)-pi*D2*sin(alpha2)*(x(5))*u2-
    (A2)*x(6));

if(m1static~=0) %% if main poppet is pinned against stop
dx(3)=0;
dx(4)=0;
else
%% Call relv8a - contains equations of motion for main poppet
dxtemp=relv8a(t,x);
dx(3)=dxtemp(1);
dx(4)=dxtemp(2);
end

if(m2static~=0) %% if pilot poppet is pinned against stop
dx(5)=0;
dx(6)=0;
else
%% Call relv8b - contains equations of motion for pilot poppet
dxtemp=relv8b(t,x);
dx(5)=dxtemp(1);
dx(6)=dxtemp(2);
end

out1=dx; %% send function evaluations to solver

```

```

else

switch(flag)

case 'events'%% event location info.

xlimit=4.75e-4; %% travel limit for main poppet
ylimit=.5e-4; %% travel limit for pilot poppet

%% Discontinuity functions for pilot poppet
if(m2static~=0)
    sftemp=relv8b(t,x); % call to equations of motion for pilot poppet
                        % for use as discontinuity function
    sumforces1=m2static*sftemp(2); %% multiply by state flag
    ymax=0;
    ymin=0;
end

if(m2static==0)
    sumforces1=0;
    ymin=x(5);
    ymax=(ylimit)-x(5);
end

%% Discontinuity functions for main poppet
if(m1static~=0)
    sftemp=relv8a(t,x);% call to equations of motion for main poppet
                    % for use as discontinuity function
    sumforces2=m1static*sftemp(2); %% multiply by state flag
    xmax=0;
    xmin=0;
end

if(m1static==0)
    sumforces2=0;
    xmin=x(3);
    xmax=(xlimit)-x(3);
end

%% send discontinuity functions to event locater
out1=[sumforces1;sumforces2;ymin;ymax;xmin;xmax];
out2=[1;1;1;1;1;1]; %% any zero crossing is terminal
out3=[-1;-1;-1;-1;-1;-1]; %% terminate when crossing from pos. to
                                neg.

end
end

```

```

% File Name: relv8a.m
% File Description:
%   Contains equations of motion for main poppet
%   for access as function evaluations or
%   as discontinuity functions

```

```
function [xdot]=rel8a(t,x)
```

```
%% Physical parameters
```

```

ks1=5000;
damp1=1000;
D1=.017;
Dp=1.0219*D1;
A1=(D1^2*pi)/4;
Ap=(Dp^2*pi)/4;
alpha=pi/6;
m1=.045;
Pre1=100;

```

```
%% Equations of motion
```

```

xdot(1)=x(4);
xdot(2)=(1/m1)*((A1)*x(1)-(Ap)*x(2)-Pre1-damp1*x(4)-ks1*x(3)-
               (0.61*pi*D1*sin(2*alpha))*x(1)*x(3));

```

```

+ File Name: relv8b.m
+ File Description:
+   Contains equations of motion for pilot poppet
+   for access as function evaluations or
+   as discontinuity functions

```

```
function [xdot]=rel8b(t,x)
```

```
%% Physical parameters
```

```

ks2=50000;
damp2=50;
Pcr=100e5;
D2=.005;
A2=(D2^2*pi)/4;
alpha2=pi/6;
m2=.02;

```

```
%% Equations of motion
```

```

xdot(1)=x(6);
xdot(2)=(1/m2)*(A2*x(2)-Pcr*A2-damp2*x(6)-ks2*x(5)-
               0.61*pi*D2*sin(2*alpha2)*x(2)*x(5));

```


2
VITA

Craig Edward Wenzel

Candidate for the Degree of

Master of Science

Thesis: AN INVESTIGATION INTO THE EFFICIENT NUMERICAL INTEGRATION
OF STIFF HYDRAULIC SYSTEMS CONTAINING DISCONTINUITIES

Major Field: Mechanical Engineering

Biographical:

Education: Graduated from Wisconsin Lutheran High School, Milwaukee, Wisconsin in May 1985; received Bachelor of Science degree in Mechanical Engineering from the Milwaukee School of Engineering, Milwaukee, Wisconsin in May 1990. Completed the requirements for the Master of Science degree with a major in Mechanical Engineering at Oklahoma State University in December 1999.

Experience: Employed as a Design Engineer at Oshkosh Truck Corporation, Oshkosh, Wisconsin from May 1990 to August 1993. Employed as a Project Engineer at Apitech Automotive, Butler, Wisconsin from August 1993 to January 1995. Employed by FES/Bardyne Incorporated, Stillwater, Oklahoma as a Project Engineer in August 1995. Currently serving at FES/Bardyne as a Technical Support Engineer.