To the Graduate Council:

I am submitting herewith a thesis written by Mark J. Bendele entitled "Application of System Cultivation and the Philosophy of Statistical Process Control to a PI Flow Control Loop." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Chemical Engineering.

Charles F. Moore

Charles F. Moore, Major Professor

We have read this thesis
and recommend its acceptance:

Jse Wu Wang

Duane D Bruns

Accepted for the Council:

Lee Minkel

Vice Provost
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Mark J. Bendele

Date August 7, 1989

APPLICATION OF SYSTEM CULTIVATION AND THE PHILOSOPHY OF STATISTICAL PROCESS CONTROL TO A PI FLOW CONTROL LOOP

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Mark J. Bendele

December 1989

DEDICATION

First and foremost, this work is dedicated to the Lord Jesus Christ, who gave me life, to God the Father, who created all things that can be discovered, and to the Holy Spirit, who gives guidance.

I also dedicate this work to my fiancee, Natalia Maria Victoria Camilo Garrido, who has loved and supported me throughout this endeavor. These pages belong to her as much as they do to me.

ACKNOWLEDGEMENTS

I wish to thank my major professor, Dr. C. F. Moore, for his practical guidance and his great patience. I also thank Dr. Tse Wei Wang and Dr. D. D. Bruns.

I am greatly indebted to Dr. James Hackney for his help in the form of technical discussions, computer programs, and the many puns that boosted my morale. Thanks are also due Andy Farell, who shared his experience with system cultivation. I further thank David Canter and Suzanne Roat for their assistance and Vera Williams for her encouragement.

In addition, I would like to thank the Measurement and Control Engineering Center for its opportunities, funding, and support.

ABSTRACT

In recent years, interest in quality, productivity, and competitiveness has led to extensive use of the techniques of statistical process control (SPC). While the philosophy of SPC is very sound, the applications of its traditional techniques are limited in the multivariable environment of continuous chemical processing. Furthermore, traditional SPC is not useful in the analysis of the performance of control systems. However, a new technique, called system cultivation, has emerged to provide a means of systematically sorting through process data to analyze multivariable relationships for continual process improvement. It also provides a means of monitoring the quality of control systems.

This study shows how system cultivation may be used to study the steady state behavior and the dynamic behavior of a PI flow control loop. It shows how the cultivation techniques of time-shifted distribution, hypothesis feedback modeling, and controlled resolution may be used to monitor controller performance, changes in a flow system, and valve hysteresis. An important conclusion is that the philosophies of statistical and conventional process control are compatible and mutually beneficial.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE **PAGE**

NOMENCLATURE

C_o = Overall flow coefficient

e = Error (set point - flow rate)

K_c = Proportional gain of the controller

m = Controller output

m_b = Controller bias

m_I = Integral component of the control signal

m_p = Proportional component of the control signal

Q = Flow rate

T = Sampling time

vp = Valve position

ΔP_o = Overall pressure drop across the flow system

Δx = Change in a variable, x

ρ = Fluid density

CHAPTER 1

INTRODUCTION

Given the challenge of today's competitive international market, companies must increase the quality of their products and also increase productivity and efficiency. Industry's role model for achieving this task is Japan, whose industry owes much of its success to the use of statistical process control (SPC). As managers and engineers attempt to implement SPC methodology in the continuous chemical processing industry, they encounter opposition from the conventional process control community and difficulties due to the limitations of SPC.

The problem with SPC is that its techniques were developed for the discrete parts industry and are not able to cope with the complexity of continuous chemical processes. Nor is SPC able to keep up with the vast, overwhelming amount of process data that is available to today's engineer. But the manager's or engineer's daily question still remains: Are the plant's processes and the control systems that operate them performing properly? If not, how can they be improved? The basic philosophy behind SPC, however, is still good. It only needs a method of implementation for continuous processes and their control systems.

Fortunately, system cultivation has emerged as a viable methodology for continual process improvement. It offers a multivariable framework for SPC and a simple way to organize and sort through huge amounts of process data to discover changes in process relationships. It has been shown to be a powerful tool for analyzing steady-state performance[8,19].

Recently, there has been interest in developing system cultivation as a tool for the analysis of process dynamics and controller performance[10,18]. Given that plant-wide control and information management are on the horizon, it is essential that useful techniques be available to use the wealth of plant-wide information to analyze the performance of control systems at both the regulatory and supervisory levels. The aim of this thesis is to add to that development by applying system cultivation to a PI flow control loop.

CHAPTER 2

BACKGROUND

There is a revolution in the world economy today. The U.S. can no longer dominate the global market by economies of scale and cheap resources, and most of its domestic markets are open to international competition. In order to compete in the world economy, companies must increase productivity and eliminate waste, while supplying a product of better quality. To meet the challenge, managers and engineers have been turning to statistical process control (SPC).

Although SPC originated in North America and Europe, beginning with the work of Shewhart in the late 1920's[27], its greatest benefits have been reaped in Japan. W. E. Deming[5,6,7] took Shewhart's ideas to Japan after World War II and, by using statistical techniques to improve quality and productivity, helped the Japanese to become economic world leaders. Threatened by the Japanese competition, U.S. industry, especially the auto industry, took interest in the statistical techniques. Not only did the auto industry begin to use SPC, but it required its vendors to assure the quality of their products using statistical techniques. The movement has spread from industry to industry and company to company, so that today a large number of companies require statistical charts on products from vendors.

Although it is important to track the quality of raw materials and end products, a more important task is to monitor the performance of the manufacturing process. It is during manufacturing when failure occurs and waste is generated. In the CPI in recent years, engineers have been using SPC during production to achieve better quality and productivity, while decreasing energy consumption and inventories and improving maintenance. Many companies have reported impressive savings and success, and this has intensified the push for SPC[3]. Although SPC has its successful applications, grave difficulties are met in applying it to a continuous chemical processing plant. Perhaps the success of SPC in continuous processing is not so much the result of the use of SPC as it is the result of a new mindset on quality and productivity which has brought together teams of people dedicated to the discovery of solutions for improvement. Before getting into the problems associated with SPC, its methods and techniques will be discussed.

2.1 Methods of Statistical Process Control

There are several basic statistics and problem-solving techniques associated with SPC[3,25]. The application of SPC breaks down into 6 basic segments:

1. Define the problem and set priorities. This can be

done with the aid of a Pareto chart, which is a bar chart that ranks problems according to frequency of occurrence or cost.

2. Determine how a process behaves using run (trend) charts, histograms, and control charts. The basic idea is to determine the target and variability of a process by measuring statistics, such as mean, range, and standard deviation, of key variables.
3. Determine the cause of a behavior using cause and effect diagrams and scatter plots.
4. Find solutions to a problem through teamwork.
5. Implement the solution.
6. Evaluate the solution by collecting data and comparing with previous behavior.

The main tool of SPC is the control chart, a graphical means of analyzing and maintaining variation in a process.

2.1.1 The Shewhart Control Chart

SPC attempts to differentiate between two types of causes of variation in a process: random or common causes and assignable or special causes. Some variations in a process are a mixture of small random effects, which can not be traced and for which little can be done. Other variations are large and identifiable and are due to assignable causes. Shewhart[27] observed that variation due to random causes exhibit a normal distribution, whereas variation due to

assignable causes goes beyond expected random variation and does not follow statistical laws. He plotted on a chart sample means versus time as shown in Figure 2.1. The distribution of the means is expected to be normal about a grand mean and such that 99.73% of the means fall within three standard deviations above and below the grand mean. The grand mean and upper and lower control limits are drawn on the chart. If a sample falls outside the limits then the process is "out of control" and assignable causes of variation are present. A process is "in control" when no samples fall outside the limits. One may also plot range or standard deviation on a control chart. A process is in control when both the average and spread of the process are in control.

Other charts that are available are the cumulative sum (CUSUM) chart and the exponentially weighted moving average (EWMA) chart[15]. Each has its own traits, and one may be more suited for a given application than another. Whatever the brand, a control chart is used to determine whether or not a process variable is on target with expected, normal variation.

2.2 Application of SPC to Continuous Processes

SPC has yielded many benefits, but its application to continuous processes is limited. It is one thing to measure

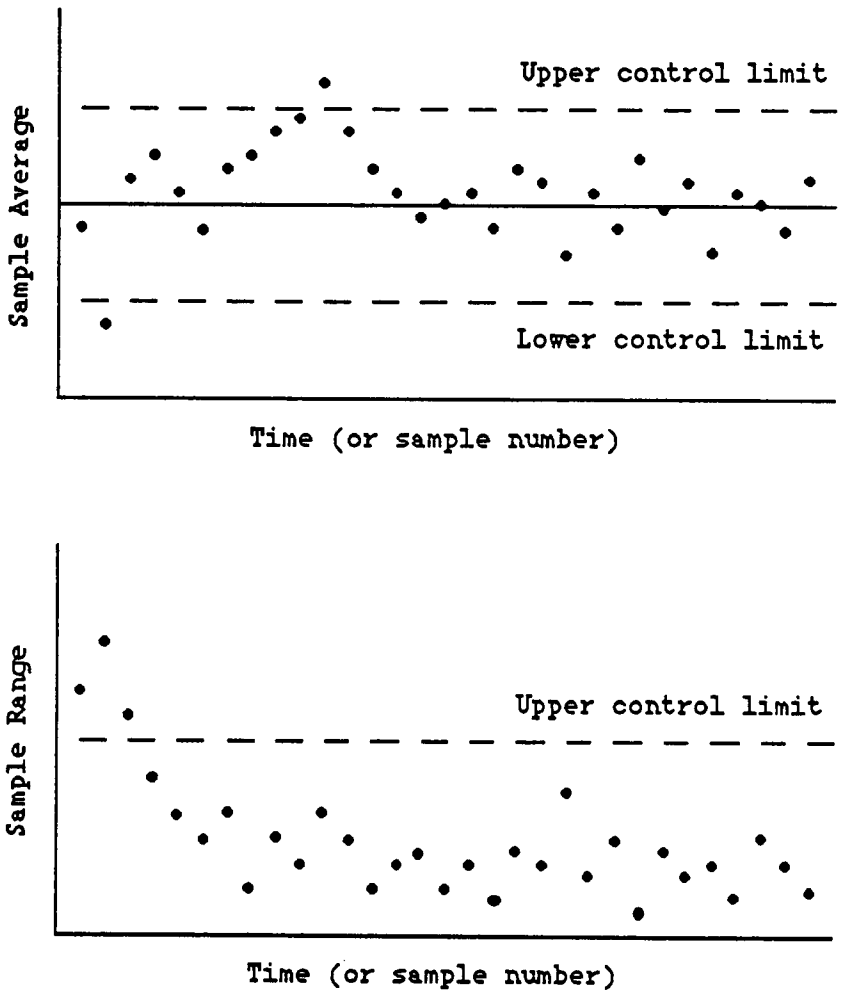


Figure 2.1: Shewhart Control Chart

and control the quality of the end product, but quite another to go upstream into the unit operations and determine the quality of their performance. The limitations of traditional SPC come from the fact that SPC was developed for the parts manufacturing industry, which is much different from the continuous processing industry. These limitations have been discussed by B.C. Moore[20], C.F. Moore[22], and MacGregor[16]. For example, the methods discussed in the previous section assume that observations are independently and normally distributed and that a hypothesis testing procedure is appropriate. These are reasonable assumptions for discrete manufacturing but not necessarily for continuous processing. It is usually not possible to use control charts to monitor variables in continuous processing for reasons discussed below. Furthermore, the perspective of SPC clashes with that of conventional process control, causing enmity between the two. Such antagonism is not necessary, however, as each could benefit from the other.

2.2.1 Limitations of SPC in Continuous Processing

In discrete manufacturing, components are put together in an assembly line or batches are processed and controlled in discrete steps. But continuous processing is composed of numerous, interconnected, unit operations in which chemicals are continuously separated, combined, and reacted to form a wide range of products. Continuous processes have several

properties that make the application of traditional SPC difficult[20,22]:

1. A strong interaction among process variables which makes the determination of cause and effect very difficult.
2. A constantly changing environment that causes uncontrollable changes in variability.
3. Difficulty in measuring productivity and quality, which may be defined, for example, in terms of a complex optimization.
4. An overwhelming quantity of process information.

These properties make the application of the six basic steps of SPC discussed in Section 2.1 difficult and laborious.

2.2.2 A Clash of Perspective

As mentioned by C.F. Moore[24], there is a bit of antagonism in the chemical processing industry between the disciplines of statistical process control (SPC) and conventional process control (CPC). Although both have the same goal, to improve the operation and control of the plant, they have different backgrounds, perspectives, and methodologies: a difference that can lead to confusion, conflict, and competition. Perhaps chemical process control would benefit if the tools and insights of both disciplines could be combined into a symbiosis.

One argument among the two camps concerns the definition of when a process is "in control." For CPC, a process is "in control" when adjustments are made to material and energy balances such that sensors measuring the balances track their target values within acceptable engineering tolerance. For SPC, a process is in control when sampled measurements and changes in the measurements are normally distributed and fall within statistical limits around a mean value, with no care about whether engineering tolerances are exceeded. Thus, a process in control according to SPC may not be in control according to CPC, and visa versa.

Another argument comes from a fundamental difference in focus between the two camps. The focus of CPC is to regulate a process against frequent and large disturbances. The focus of SPC is to eliminate those disturbances. The process is not adjusted until both solid evidence exists that variations in process variables are beyond those statistically expected and the cause of the variations are identified and documented. In CPC, the process is adjusted as soon as possible before the cause of variation is determined, if at all.

While it is certainly a noble cause to attempt to eliminate all disturbances, it is certainly not a realistic ideal. While it may be easy to identify a disturbance (in some cases), it may be difficult or impossible to remove that disturbance, especially in continuous chemical processing.

On the other hand, CPC engineers may be so preoccupied with dynamics and the application of automatic control to reject disturbances, they may install an unnecessary control system to compensate for a removeable disturbance.

There is also a different view on the nature of processes. To the CPC engineer, relationships (cause and effect) in nature are inherently deterministic and can be described by engineering equations describing physical phenomena. The SPC statistician believes there is always a random aspect (or variation) in nature that can be described and predicted only by probabilities and distributions, ie. stochastic models rather than deterministic models.

Furthermore, CPC is concerned with dynamic effects, while SPC is more concerned with static aspects of the process. Understanding dynamics is important to the design and implementation of feedback control loops, but SPC ignores dynamics and assumes and insists upon static operation. While this seems naive on the part of SPC, it must be admitted that the steady-state performance of processes is important and should not be ignored.

Although the two perspectives are quite different they each have their good points to be exploited. It would certainly be beneficial if SPC could be used to monitor and analyze processes and control systems, both of which little may be known or even questioned. What is needed is a framework to achieve that purpose.

CHAPTER 3

SYSTEM CULTIVATION AND A MULTIVARIABLE FRAMEWORK FOR STATISTICAL PROCESS CONTROL

Alwan and Roberts[1] bring out the interesting point that the perspective of traditional SPC may be needlessly limited in applying the concepts of Shewhart. The central concept of SPC is that of the state of statistical control. Shewhart defined a state of control as follows: "a phenomenon will be said to be controlled when through the use of past experience, we can predict, at least within limits, how the phenomenon may be expected to vary in the future. Here it is understood that prediction within limits means that we can state, at least approximately, the probability that the observed phenomenon will fall within the given limits[27]." This definition does not require independent and identically distributed or simple random behavior nor the use of control charts. The basic idea is that we use past experience to predict future behavior within limits.

All of the problems with SPC discussed in the previous section boil down to one thing: the statistical methods employed by SPC assume stationary, independent, and identically distributed random variables. This is a very gross and unrealistic assumption in continuous chemical processing. Furthermore, SPC is not able to track

performance reflected by multivariable relationships[20]. In continuous processing, the multivariable relationships are what are of interest, not the distribution of the variables themselves. Those complex relationships are what govern the behavior of the systems and subsystems in the plant. It is they that must be statistically studied and monitored. The bridge we are looking for is a way to use the philosophy of SPC to monitor those relationships.

3.1 The Multivariable Framework

A multivariable framework for the application of SPC philosophy has been proposed by B.C. Moore[17,19,20]. It provides a simple way to monitor changes in physical relationships and to track down causes for the changes. The basic idea is to sort the process data into physically meaningful categories. Statistics from each category may then be compared with statistics from another data set. For example, it may be known that process variable z(t) is physically related to x(t) and y(t). This may be based on a theoretical model or a heuristic or hypothesis model. A great advantage to this method is that one need not develope mathematical models. The simple knowledge that certain variables are related to others is the only requirement; thus, complex relationships may be analyzed. One may reduce the resolution of the variables x and y (e.g. to integer

numbers from 0 to 31) to form categories. The variable z may be sorted into the categories of x and y, and statistics on z may be calculated. Upon comparing statistics on z from corresponding categories of another data set, one may find unexpected variations that could not be seen by looking at the overall statistics on z alone. Furthermore, statistical changes in subset categories of z reflect changes in the physical relationship described by the variables. One might also order the data by sorting z and y into categories of x and thus observe the relationship between z and y at constant x. The procedures used above are referred to as hypothesis feedback modeling and controlled resolution.

In terms of practical implementation, one may design an "information still"[17], a well designed set of sorting operations and statistical tests. The idea is to "distill" the huge amounts of data into meaningful categories with meaningful order and display the results to an engineer in an easily interpreted form. A still could be defined for a set of relationships to be monitored. One might separate normal from abnormal data and then sort through the data with a number of stills, boiling down the data until the problems are found. One can do this systematically with a decision tree.

3.2 System Cultivation

With the ideas of B.C. Moore, a new form of analysis has emerged called system cultivation[8,10,23]. It allows one to "cultivate" the day-to-day operation of a process or plant by signalling and diagnosing problems and process changes for the continual improvement and optimization of the plant. As suggested by Hackney[11], system cultivation may be considered in a broad sense to include many methods of analysis, such as system identification, fault detection, signal validation, time-series analysis, hypothesis feedback modeling, controlled resolution, statistical process control, and others. Cultivation is not a design methodology.

System cultivation makes the philosophy of SPC applicable to continuous processes by allowing one to compare past data with present data and discover the existence of a problem or "assignable cause". Furthermore, while SPC control charts do not offer a means of discovering the source of a problem, the "information still" of system cultivation allows one to search and sort through huge amounts of data and complex relationships to find the cause.

3.3 The Heat Exchanger Example

A.E. Farell[8] applied the methods of cultivation to a simulated, liquid-liquid, shell and tube heat exchanger with good results. In his example, the outlet temperature of the

hot process stream is controlled by manipulating the chill water flow rate using a PI control algorithm. Based on the steady-state energy balances around the heat exchanger, he proposed the following implicit model:

$$F_{cw} = f(F_{hot}*T_{hot,in}, T_{cw}, T_{amb})$$

where

F_{cw} = flow rate of chill water

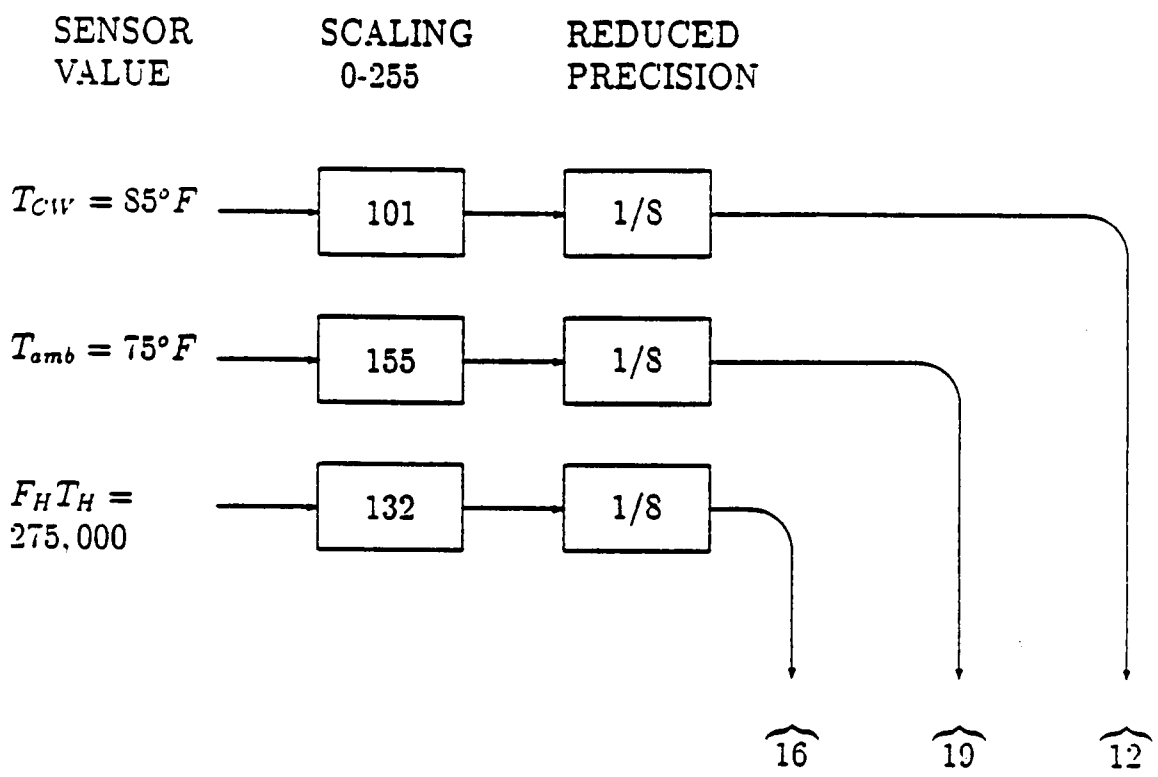
F_{hot} = flow rate of hot stream

$T_{hot,in}$ = inlet temperature of hot stream

T_{cw} = inlet temperature of chill water, and

T_{amb} = ambient temperature.

To organize the data into meaningful categories, he reduced the resolution of the data to 5 bits. Thus, values within a certain small range for a given independent variable are condensed into a 5-bit integer number. He then concatenated the 5-bit integers for the 3 independent variables ($F_{hot}*T_{hot,in}$, T_{cw}, T_{amb}) to define a label for each sampling instant as shown in Figure 3.1. One may imagine the label defining a 3-D box corresponding to a specific set of operating conditions. Values of F_{cw} were then sorted into the appropriate boxes representing the operating conditions from which they were recorded. Statistics on F_{cw}, such as average, standard deviation, and frequency, were calculated for each box. Farell then sorted each box by average, renumbered the boxes with consecutive



LABEL VALUE = 161912

Figure 3.1: Data Conversion

Source: Farell, A.E., "Application of Process Control Cultivation to a Liquid-Liquid Heat Exchanger," M.S. Thesis, The University of Tennessee, Knoxville, August, 1987.

box or crate numbers, and plotted each box vs F_{CW} on what he called an overall plot. Heretofore the overall plot will be referred to as a statistical plot. A summary of the sorting procedure is shown in Figure 3.2, and a statistical plot is shown in Figure 3.3. The plot shown contains two data sets: a normal set, representing a base case or standard operation, and a set with the overall heat transfer coefficient changed by 35%. Each box is plotted with mean and 2-sigma standard deviation.

During the sorting procedure, labels are matched between the two data sets, so that boxes with the same crate number represent the same operating condition. A shift in the plot is clearly visible, indicating a change in the relationships of the energy balance. Also shown in Figure 3.3 is an overall box representing the mean and standard deviation of the entire data set of the base case. One might think of the overall box as the end view of a control chart. The shift in the data could not be seen by this overall view. Also, if the overall box represents a control chart, data from normal operation can fall outside the statistical limits.

3.4 Cultivation of Dynamics

In the previous example, the steady state behavior of the heat exchanger was analyzed. Dynamics would be seen as spread in the data and would contribute to the standard

Data base:

t	x	y	z
1	30.1	24.9	11.7
2	30.0	27.8	10.0
3	30.0	24.7	11.6
4	30.1	27.6	9.9
:	:	:	:

Reduce resolution:

1	2340	11.7
2	2344	10.0
3	2340	11.6
4	2344	9.9
:	:	:

Sort by label:

2340	11.7
2340	11.6
:	:
2344	10.0
2344	9.9
:	:

Summarize:

2340	11.67 ± 0.20
2344	10.01 ± 0.15
:	:

Sort by average and renumber labels:

1	10.01 ± 0.15
2	11.67 ± 0.20
:	:

Figure 3.2: Summary of Sorting Operations

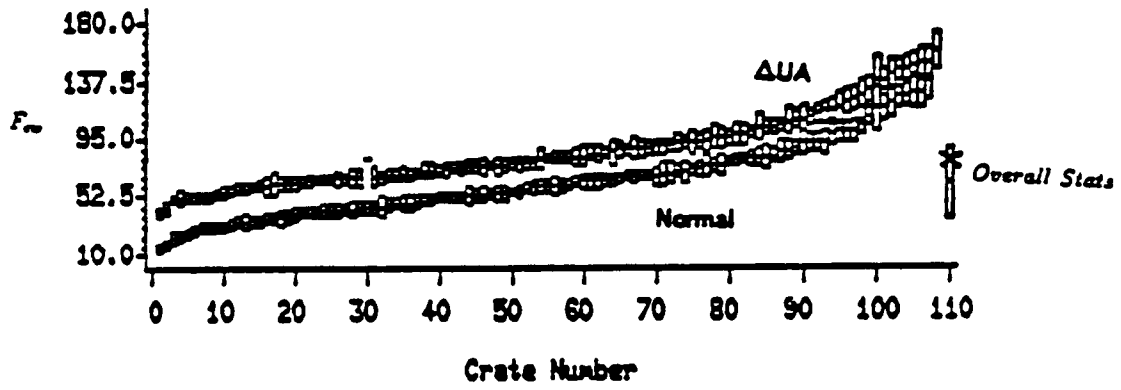


Figure 3.3: Overall Plot

Source: Farell, A.E., "Application of Process Control Cultivation to a Liquid-Liquid Heat Exchanger," M.S. Thesis, The University of Tennessee, Knoxville, August, 1987.

deviation in a category. The dynamics were not of concern. However, if cultivation is to be used to analyze the performance of a control system, dynamics can not be ignored.

B.C. Moore[18] observed that "if you were to observe a production team discussing signals in the process of analyzing some situation, you would hear them point out events concerning movement of a physical variable in time ... They would talk in terms of magnitude and time." He noted that there are two components to a signal: (1) a relatively constant level of fast variation (due to noise and cyclic behavior) and (2) movement at a relatively constant rate with fast variation taken into account.

One may decompose a signal into segments such that adjacent segments differ in either rate or level of fast variation. Each segment or individual will have the form

(class, t_b, dt, dv, r_{fv}, v_m)

where

class = {1, 2, 3}

1 = rest, with fast variation (eg. noise)

2 = downward movement

3 = upward movement

t_b = beginning time

dt = time duration

dv = change in value (non-negative)

r_{fv} = range associated with fast variation

v_m = initial median value

This framework offers another way to sort the data into categories.

Another method for the analysis of dynamics and control systems is the time-shifted distribution (TSD) developed by Hackney[10]. It is an interesting brew of controlled resolution analysis developed by B.C. Moore and cross-correlation analysis. Hackney used the TSD analysis to detect plant-model mismatch in model predictive control in the face of unmeasured disturbances. TSD analysis will be discussed in Chapter 5.

CHAPTER 4

THE PI FLOW CONTROL LOOP

The system to be studied is a proportional-integral (PI) flow control loop[26,28,29,30]. The system is shown in Figure 4.1. The system consists of an equal-percentage control valve, an orifice flow meter, a PI feedback controller, and a section of pipe. The controller controls flow rate at a set point set by an operator or by another controller. Disturbances to the system are changing fluid properties, changing set point, and changing overall pressure drop across the entire flow system. Data collected from the system include set point, flow rate, error, integral error, controller output, valve position, and overall pressure drop.

The flow system was simulated on the VAX cluster at the University of Tennessee using FORTRAN. In the simulation, measured flow rate is compared with set point, a routine calculates controller output, and then another routine calculates valve dynamics, including hysteresis, to get a valve position. The valve position is then fed into a steady-state simulation to calculate the flow rate. Noise, other disturbances, and effects due to acceleration of the fluid are included. The time increment used in the simulation of the valve dynamics and flow is much less than

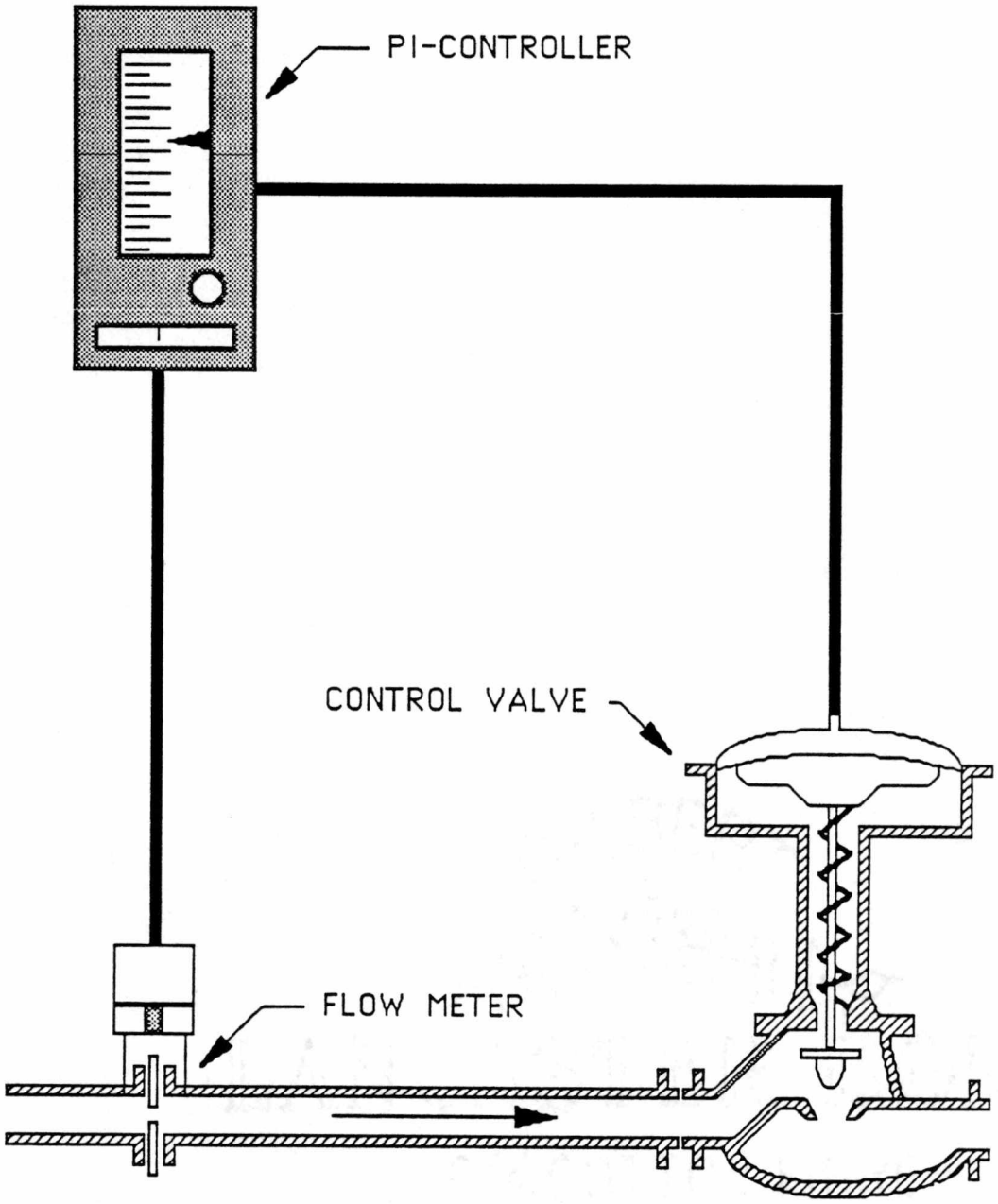


Figure 4.1: PI Flow Control Loop

that used to simulate the controller. Thus, the simulation is one of a continuous process with a discrete controller.

4.1 Simulation of System Components

This section describes each component of the flow loop, giving the mathematical model used to simulate each component. The next section describes how the components of the simulation fit together.

4.1.1 PI Controller

The control action was computed using the position algorithm for the PI control law with provisions for protection against reset windup. The control equation was as follows:

$$m_n = m_b + K_c\left[e_n + \frac{T}{\tau_I}\sum_{i=1}^{n} e_i\right] \qquad (4.1)$$

4.1.2 Dynamics of Pneumatic Control Valve

The dynamics of the control valve were modelled using a first-order equation[30]:

$$\frac{x(s)}{m(s)} = \frac{A/K}{\left(\frac{C}{K}\right)s + 1}$$

where x is valve travel, m is the control signal, A is the area of the actuator diaphram, K is the spring constant, and

C is a damping factor. Dividing by the maximum valve travel, x_{max},

$$\frac{vp(s)}{m(s)} = \frac{K_v}{\tau_v s + 1} \qquad (4.2)$$

where vp is the valve position, K_v is the valve gain in terms of vp versus m, and τ_v is the valve time constant.

An actuator must overcome the static friction between the valve stem and packing, resulting in hysteresis. A force balance on the stem shows that the valve will move when

$$|mA - Kx| > f_s$$

where mA is the force exerted by the actuator, Kx is the force exerted by the spring, and f_s is the force of static friction. Dividing by the spring constant and maximum valve travel, x_{max},

$$\left| \frac{mA}{Kx_{max}} - \frac{x}{x_{max}} \right| > \frac{f_s}{Kx_{max}}$$

If the fraction of hysteresis is defined as f_s/Kx_{max}, and valve position as x/x_{max}, then the valve will move when

$$|mK_v - vp| > \frac{\%hysteresis}{100} \qquad (4.3)$$

In the numerical simulation, a new valve position is calculated each time increment using a first-order difference

equation for the dynamics of the actuator. If the valve has been at rest or is changing its direction of motion, then the inequality in Equation 4.1 is checked to determine whether the valve position will actually change.

4.1.3 The Flow Process

The flow system consists of three basic components, each of which add resistance to flow: 1) control valve, 2) flow meter, 3) piping system, including pipe, pipe fittings, changes in elevation, and additional units, and 4) acceleration of the liquid mass. The flow depends on the overall pressure drop across the system, which is a major disturbance to the process and is input at each time increment in the simulation. At each increment, the flow rate is calculated such that the sum of the pressure drops due to the resistances of the four components is equal to the overall pressure drop.

The flow rate through the valve is given by

$$Q = c_v \sqrt{\frac{\Delta P_v g_c}{\rho}} \qquad (4.4)$$

where

Q = flow rate

ΔP_v = pressure drop across the valve

ρ = density, and

C_v = the valve coefficient.

An equal-percentage valve is used, so that

$$c_v = c_{v_{max}} \alpha^{(vp-1)} \qquad (4.5)$$

where α is the rangeability parameter and $C_{v,max}$ is the valve coefficient of a fully open valve.

The flow rate measured by the orifice meter, as shown in Figure 4.2, is determined by[4]

$$Q = c_o \frac{A_o}{\sqrt{1-\beta^4}} \sqrt{\frac{2\,\Delta P_t\, g_c}{\rho}} \qquad (4.6)$$

where

ΔP_t = the pressure drop measured by the DP transmitter
= $P_1 - P_2$

C_o = the discharge coefficient = 0.61

A_o = the area of the orifice

β = the ratio of orifice diameter to pipe diameter.

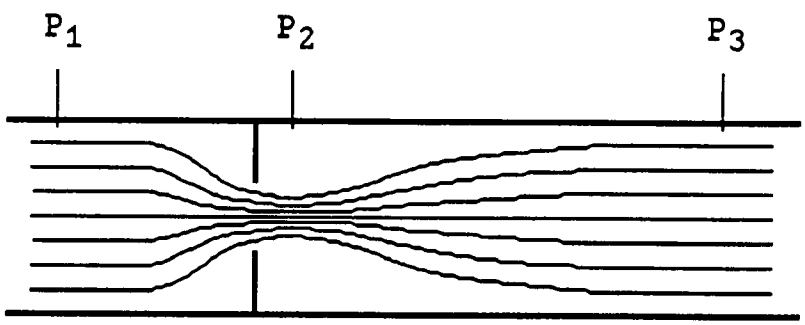


Figure 4.2: Orifice Flow Meter

The pressure loss due to the flow meter, ΔP_m, is the difference between P_1 and P_3. It can be approximated by[31]

$$\frac{P_1 - P_3}{P_1 - P_2} = 1 - \beta^2$$

so that

$$\Delta P_m = (1-\beta^2)\,\Delta P_t . \qquad (4.7)$$

In the simulation, white noise is added to ΔP_m to simulate a noisy flow meter.

It was desired to keep the piping system as simple and general as possible. Resistance due to pipe fittings is included into the simulation as equivalent length of pipe. The changes in elevation or changes in a liquid level in a tank are incorporated into the overall pressure drop. Additional units, such as heat exchangers, can be simulated by

$$Q = c_u \sqrt{\frac{\Delta P_u g_c}{\rho}} \qquad (4.8)$$

where C_u is the discharge coefficient and ΔP_u is the pressure drop across the unit.

The liquid in the piping has an inertia that adds dynamics to the flow system. The inertia was treated as pressure loss or resistance to flow. The acceleration of fluid is given by

$$\frac{\rho L}{g_c}\frac{dv}{dt} = \frac{4\rho L}{g_c \pi D^2}\frac{dQ}{dt} = \Delta P_a$$

where v is the fluid velocity, and ΔP_a is the pressure drop due to acceleration. For a small time increment, Δt, the pressure drop may be approximated by

$$\Delta P_a = \frac{4\rho L}{g_c \pi D^2}\frac{\Delta Q}{\Delta t} \qquad (4.9)$$

4.1.4 Process Disturbances

Disturbances enter the system through changing overall pressure drop, flow meter noise, set point changes, and changing fluid properties. The disturbances in this system are simulated using some form of the ARIMA (autoregressive integrated moving average) models of Box and Jenkins[2,12]. The two forms used in this simulation are the IAR (integrated autoregressive) model, described by the following difference equation,

$$D_t = (1+\Phi)D_{t-1} - \Phi D_{t-2} + a_t$$

and the IMA (integrated moving average) model,

$$D_t = D_{t-1} + a_t - \theta a_{t-1}$$

where D_t is the disturbance at time t, a_t is a normally distributed random variable with mean 0, and Φ and θ are model parameters. If Φ and θ are 0, then the model is a nonstationary random walk. Otherwise, the IAR model is a

smoothed random walk, and the IMA model is a random walk with superimposed white noise.

The above disturbance models are stochastic, but a form of the IAR can be used to model deterministic disturbances. This can be done by only allowing a_t to be nonzero at random times. For $\Phi = 0$, the model results in random steps of random duration. For $\Phi = \exp(-T/\tau)$, where T is the sampling interval and τ is the time constant, the model results in exponential rises of random magnitude and duration. White noise may also be added to these deterministic models. Figures 4.3 through 4.6 show examples of disturbances for flow meter noise, overall pressure drop (random walk and IAR), and set point (deterministic).

4.2 The Design of the Flow System

The flow simulation is summarized in Figure 4.7. The process block simultaneously solves Equations 4.4 through 4.9 for flow rate given valve position and overall pressure drop. The valve position is used in Equation 4.5 to calculate the C_v used in Equation 4.4. ΔP_t may be substituted by ΔP_m by combining Equations 4.6 and 4.7. We now have four equations relating flow rate to each component of pressure drop. These equations are solved by quessing the flow rate such that $\Delta P_o = \Delta P_v + \Delta P_m + \Delta P_u + \Delta P_a$.

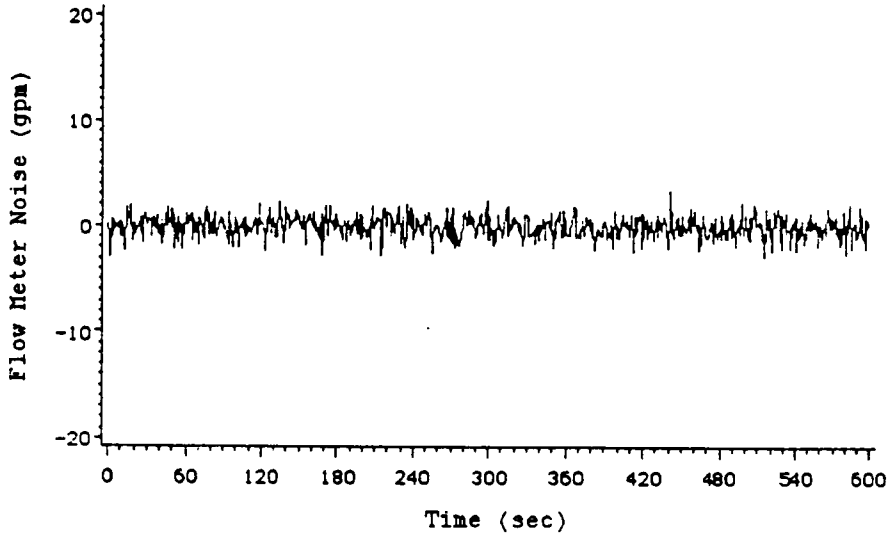


Figure 4.3: Flow Meter Noise

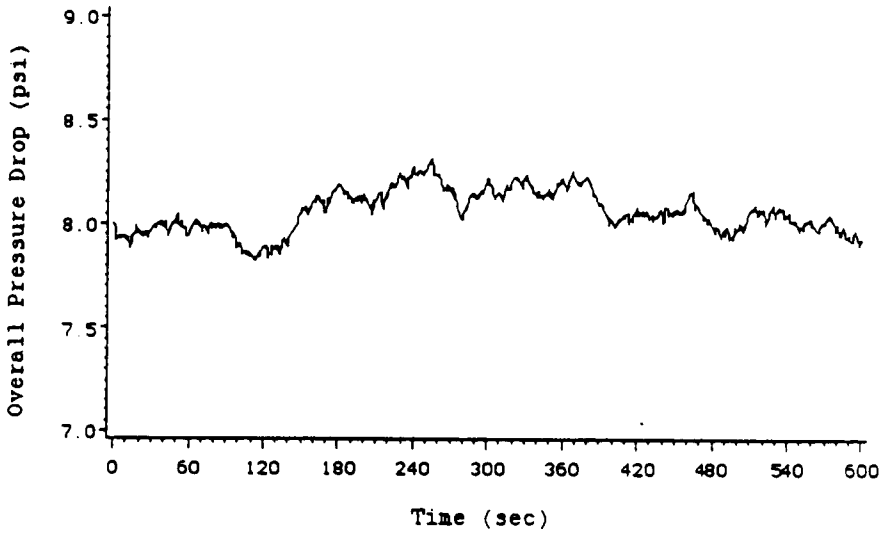


Figure 4.4: Random Walk Disturbance

Figure 4.5: Integrated Autoregressive (IAR) Disturbance

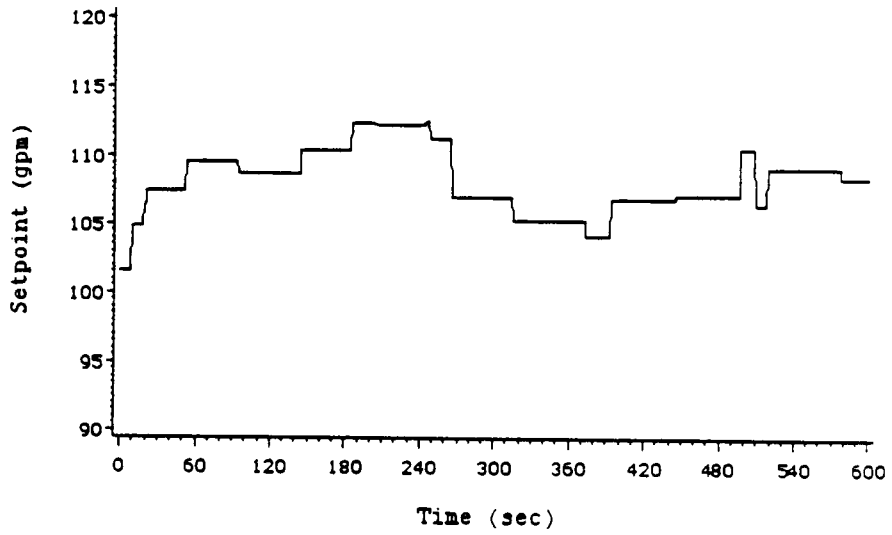


Figure 4.6: Deterministic Disturbance

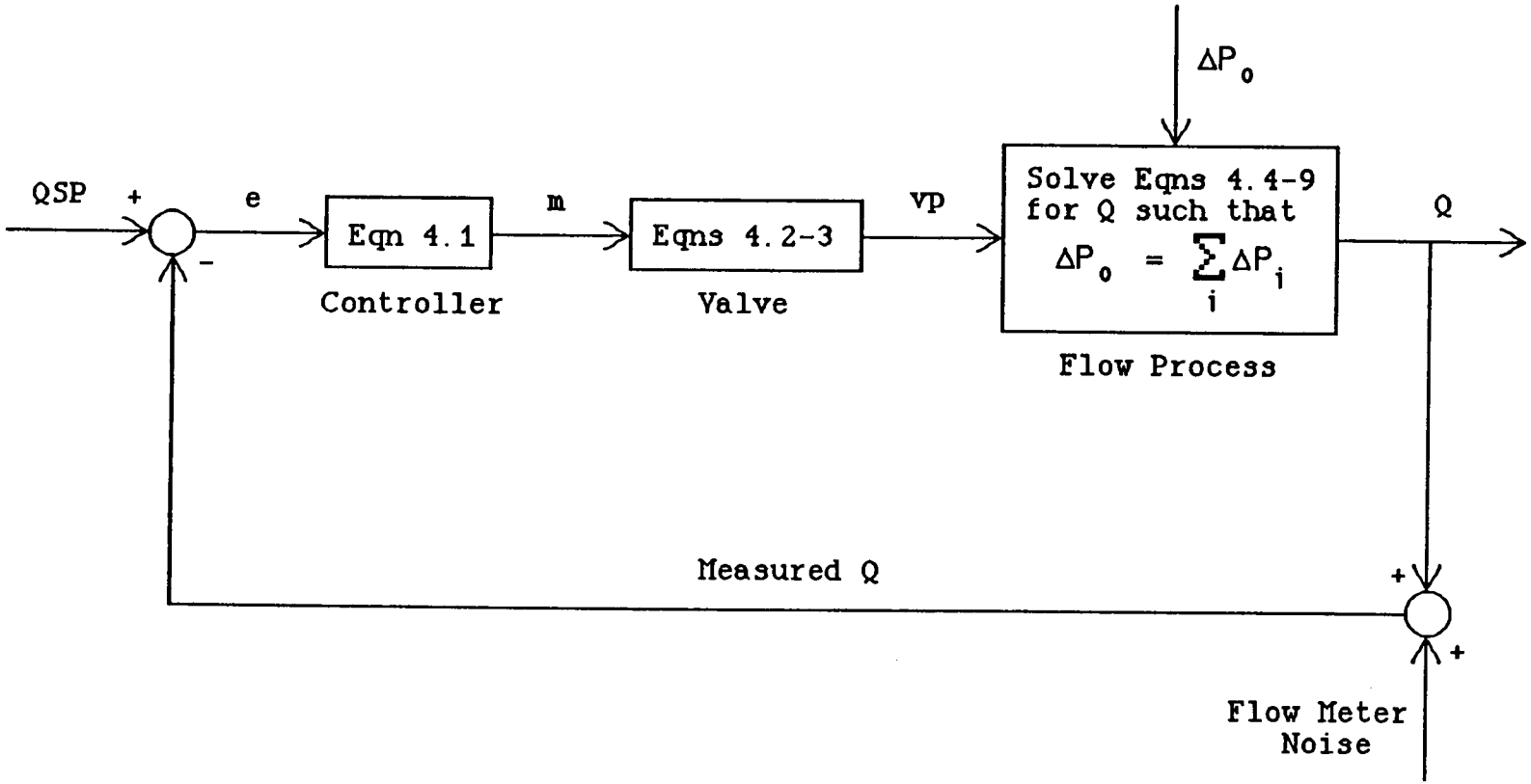


Figure 4.7: Diagram of Flow Loop Simulation

The parameters used in the base case are shown in Table 4.1. The base case is the standard or reference condition representing "normal" operation. In collecting data for the analyses in the next chapter, several simulations were run to mimic problem situations. The simulations were run so that the flow always remained turbulent. Fluid properties were assumed constant throughout a simulation. Figure 4.8 shows the installed valve characteristic of the base case, and Figures 4.9 through 4.11 show the response to a step test. The time increment of the flow simulation was 0.1 second, and the controller had a sampling time of 1.0 second.

Disturbances for flow meter noise, overall pressure drop, and set point are shown in Figures 4.3, 4.4, and 4.6, respectively. The flow meter noise is a gaussian random number with 1.0 standard deviation. Flow meter noise and setpoint changes are input at each sampling instant of the controller. The overall pressure drop is input at each time increment of the flow simulation. The setpoint ranges from 80 to 120 gpm, and the overall pressure drop ranges from 6 to 10 psi.

The controller was tuned to minimize the sum of the integral squared error and the integral of squared controller output for a simulation including all three of the above disturbances. The tuning parameters were refined by trial and error through the use of a step test with only flow meter

Table 4.1: Parameters of the Flow Simulation

Parameter	Value
Valve Characteristic	Equal Percentage
Rangeability, α	50
Valve Flow Coefficient, C_v	10 in^2
Percent Hysteresis	5 %
Gain of Valve Actuator, K_V	0.067 psi^{-1}
Valve Time Constant, τ_V	5 sec
Controller Gain, K_C	0.11
Controller Integral Time, τ_I	16.0 sec
Controller Sampling Time, T	1 sec
Set Point, QSP	100 gpm
Nominal Pressure Drop, ΔP_O	8 psi
Ratio of Pipe to Orifice Diameter, β	0.5
Flow Coefficient of Orifice Meter, C_O	0.61
Density, ρ	62.4 lbm/ft^3
Viscosity, μ	1.0 cp
Pipe Diameter, D	4 in
Pipe Length	150 ft
Pipe Roughness	0.00015 ft
Flow Coefficient of Flow Component, C_u	5 in^2
Time Increment of Flow Simulation, Δt	0.1 sec

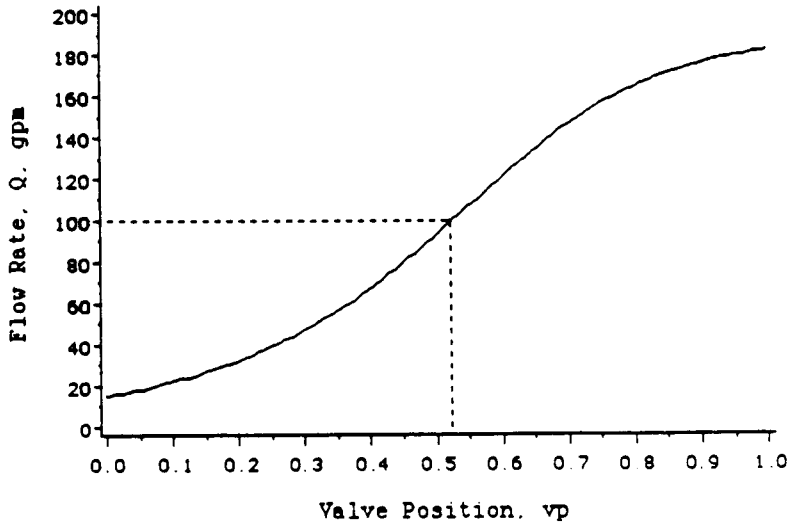


Figure 4.8: Installed Flow Characteristic of the Simulated Control Valve

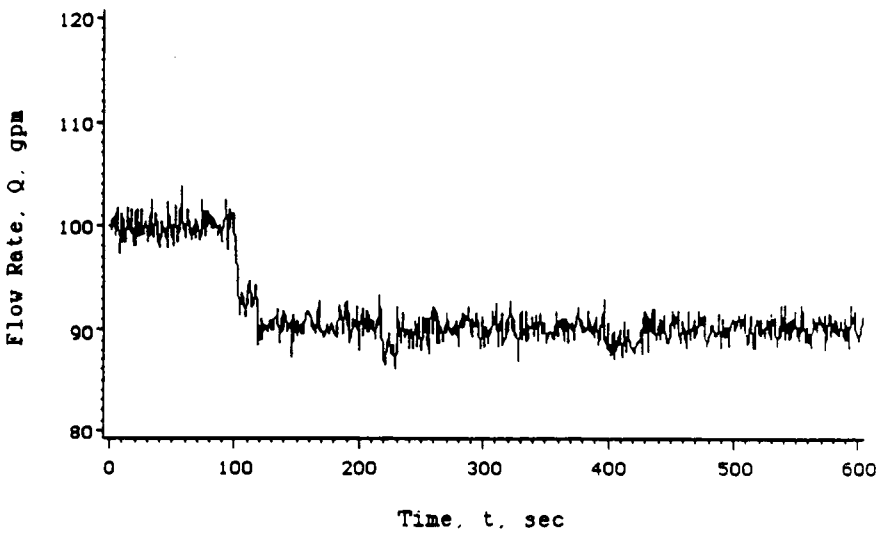


Figure 4.9: Response of Flow Rate to a Step in Set Point

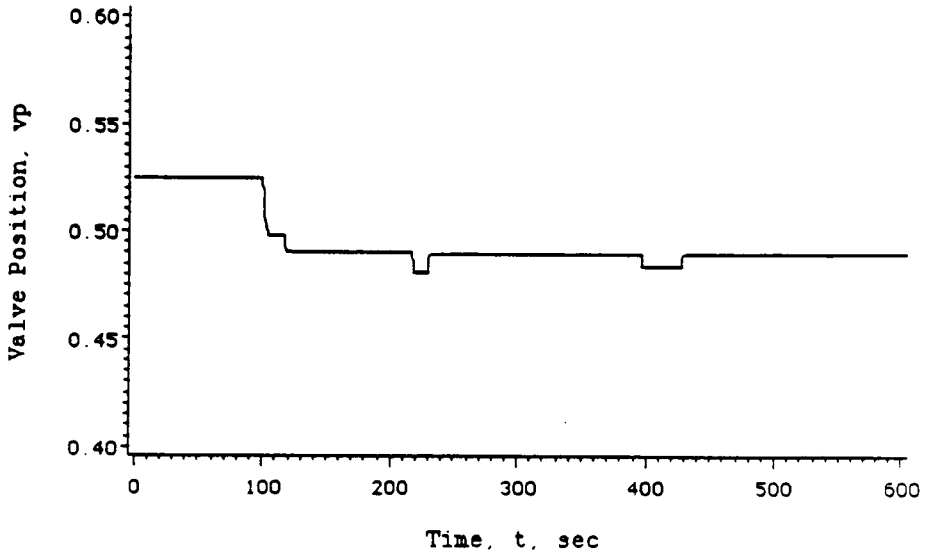


Figure 4.10: Response of the Valve to a Step in Set Point

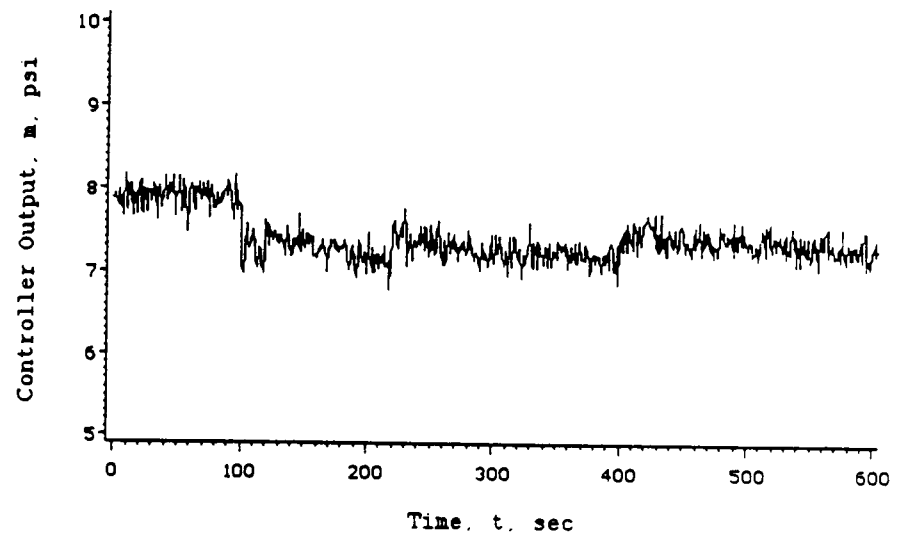


Figure 4.11: Controller Output for a Step in Set Point

noise present. The objective was to get a good response to setpoint and pressure drop changes but little response to flow meter noise.

4.3 Simulation of Problems

As mentioned above, several simulations were performed to generate data for comparison with the base case. Some problems were simulated by changing parameters listed in Table 4.1, and others were simulated by modifying program code. The changes used to simulate various problems are shown in Table 4.2. For example, a plugging condition was simulated by changing the parameter C_u. A stuck sensor was simulated by simply holding its output constant while allowing all other variables to change at will. All changes to simulations were applied from beginning to end, so that no data set contained both problem data and normal data.

Table 4.2: Simulation of Problems

Problem	Change
Plugging	C_u from 5 to 3 in^2
Change in fluid density	ρ from 62.4 to 50 lbm/ft^3
Change in hysteresis	% hysteresis from 5 to 10 %
Plugging combined with change in hysteresis	Both changes above
Flow meter drift	Add a constant 30 gpm to flow signal
Increased flow meter noise	Standard deviation of gaussian random number from 1 to 2 gpm
Stuck valve	% hysteresis from 5 to 100 %
Stuck sensor	Hold sensor output constant while allowing all other variables to change as normal

CHAPTER 5

CULTIVATION ANALYSIS

The cultivation techniques discussed in Chapter 3 are useful in analyzing a PI flow control loop. The techniques may be used to create an information still to sort the data and a decision tree to diagnose problems. Problems that may be addressed are:

1. Sensor Problems
    a. Stuck or drifting flow meter
    b. Stuck valve position sensor
    c. Poor measurement of the overall pressure drop
2. Valve Problems
    a. Stuck valve
    b. Change in valve hysteresis
    c. Change in valve gain due to:
        i. Plugging
        ii. Change in fluid properties
3. Controller Problems
    a. Controller left on manual
    b. Control parameters changed
    c. Poor controller tuning

The data collected from the flow simulation for use in the analysis includes the following variables:

t = time at k

Q = flow rate at k

e = error at k

m = controller output at k-1

vp = valve position at k

ΔP_O = overall pressure drop at k

QSP = set point at k

where k is the sampling instant. This chapter discusses the use of sensor statistics, statistical plots, hypothesis feedback, and controlled resolution to analyze the PI flow control loop.

5.1 Sensor Statistics

Obviously, any analysis of data will be inaccurate if the sensors used to collect the data are not functioning properly. This thesis does not treat in depth the problem of signal validation. However, the detection of sensor failure should be an important part of a cultivation study, since a failure can not only harm the integrity of the cultivation analysis but also harm the performance of the control loop. A helpful tool for discovering obvious sensor failures or other serious faults is the analysis of individual sensors and monitored variables[8]. It is emphasized that sensor statistics are not very useful for detecting problems that are not obvious.

In the PI flow control loop, the only sensor that affects loop performance is the flow meter. The recorded

variables ΔP_o and vp do not affect performance, but their failure could affect the cultivation analysis. The calculated variables e and m can be helpful in differentiating between possible sensor failures.

Figures 5.1 through 5.4 show sensor statistics comparing normal data to data involving a stuck flow meter, a stuck valve position sensor, a stuck valve, and a controller left on manual. The figures show normalized average and standard deviation for each variable over an entire simulation. The actual values for average and standard deviation are not important here, since these would change with normal operation. Two data sets showing the same average and standard deviation for a variable on the figure indicates that the variable looks normal in each case. The signs to look for are obvious changes in character. For example, in both the case of a stuck valve and a stuck valve position sensor the variable vp will have a very small or zero standard deviation. On a strip chart of this variable, one would see a straight line, while the other variables may show normal movement. These two problems can be distinguished by the error, e, which will have a mean of zero for a stuck valve position sensor and a non-zero mean for a stuck valve. However, if the standard deviation of the controller output, m, is zero, the problem is that the controller is set on manual. The stuck flow meter will cause Q to have a very

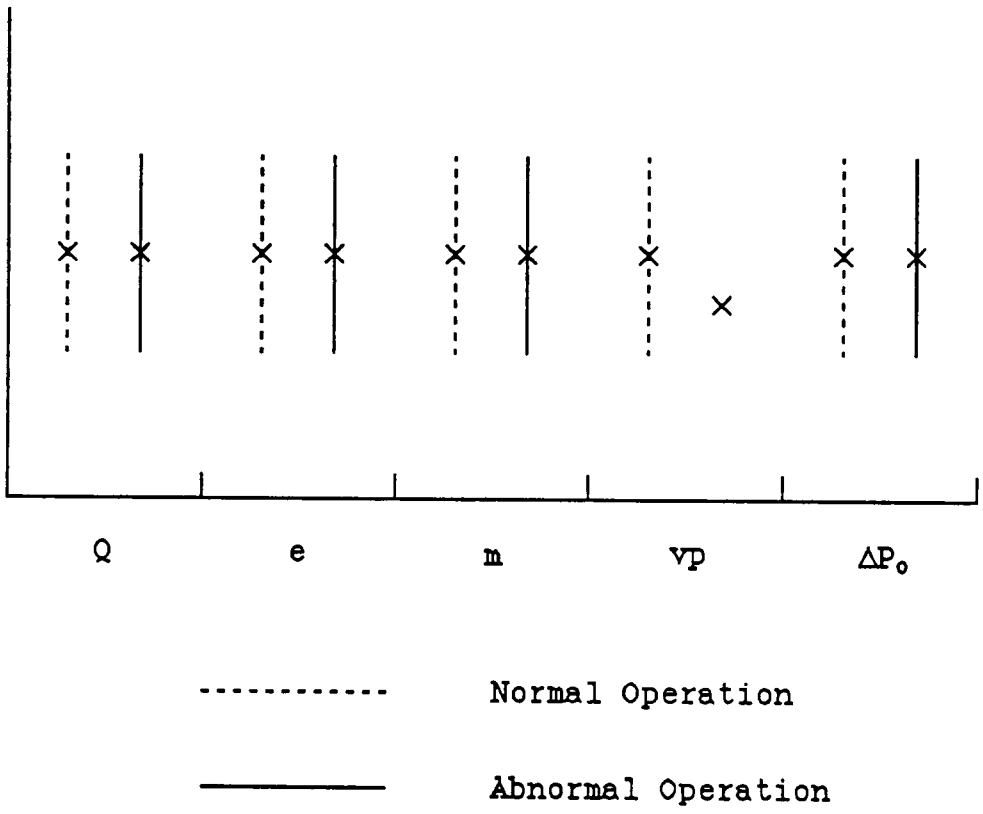


Figure 5.1: Sensor Statistics for a Stuck Valve Position Sensor

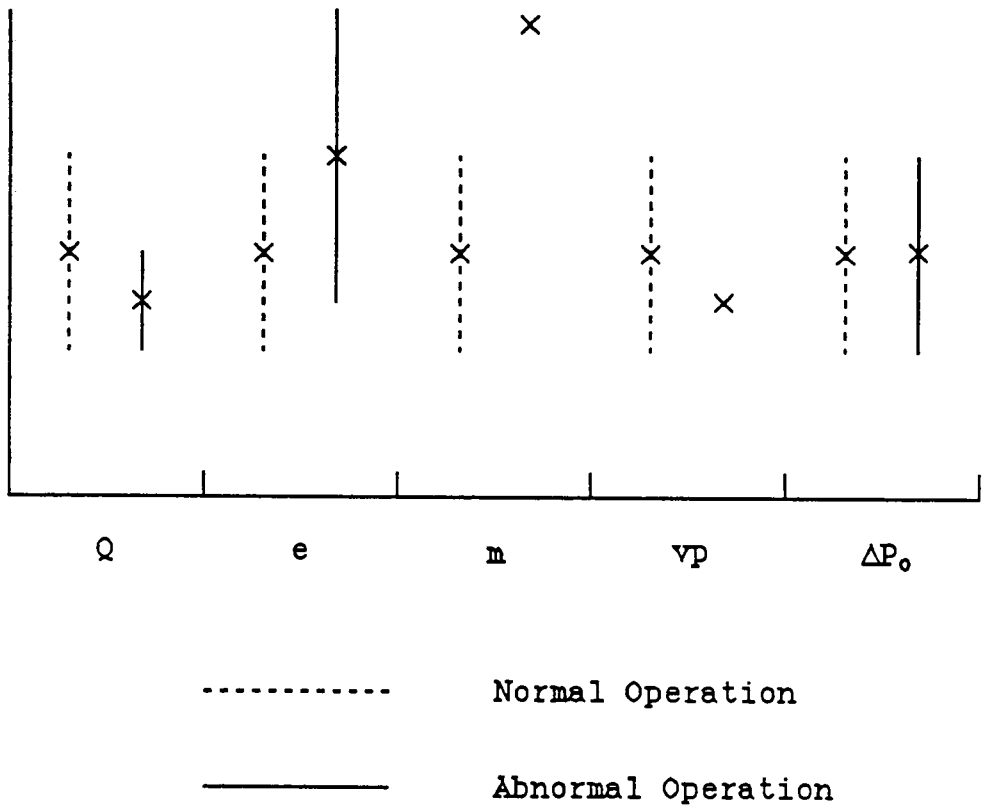


Figure 5.2: Sensor Statistics for a Stuck Valve

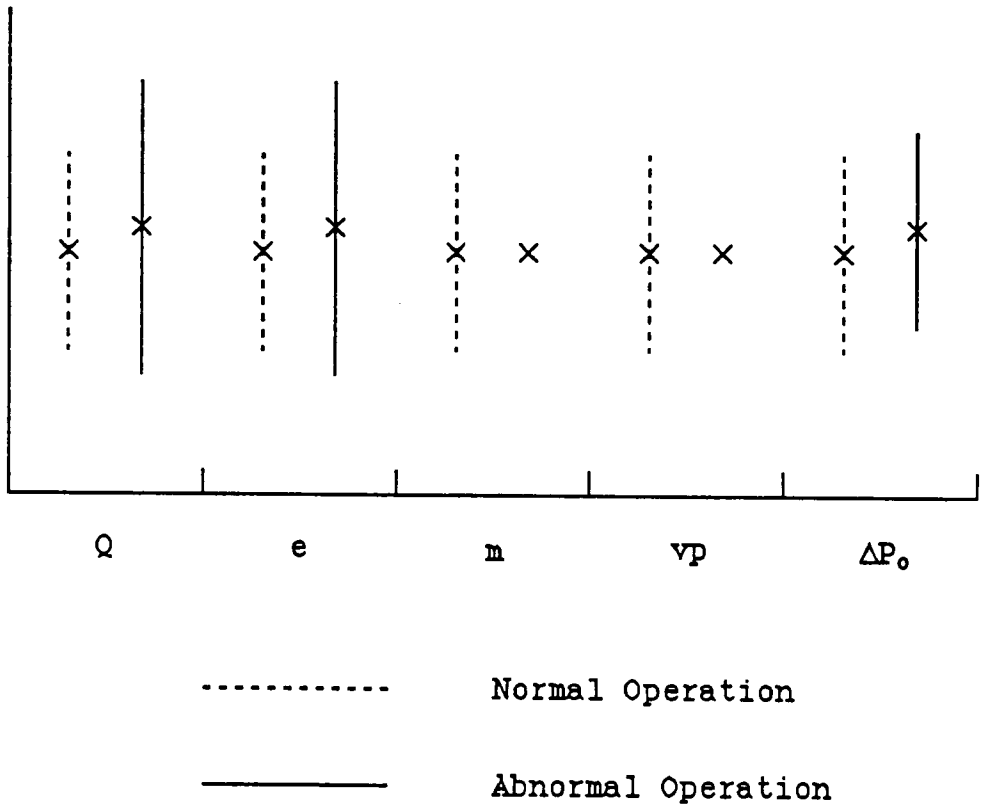


Figure 5.3: Sensor Statistics for a Controller Left on Manual

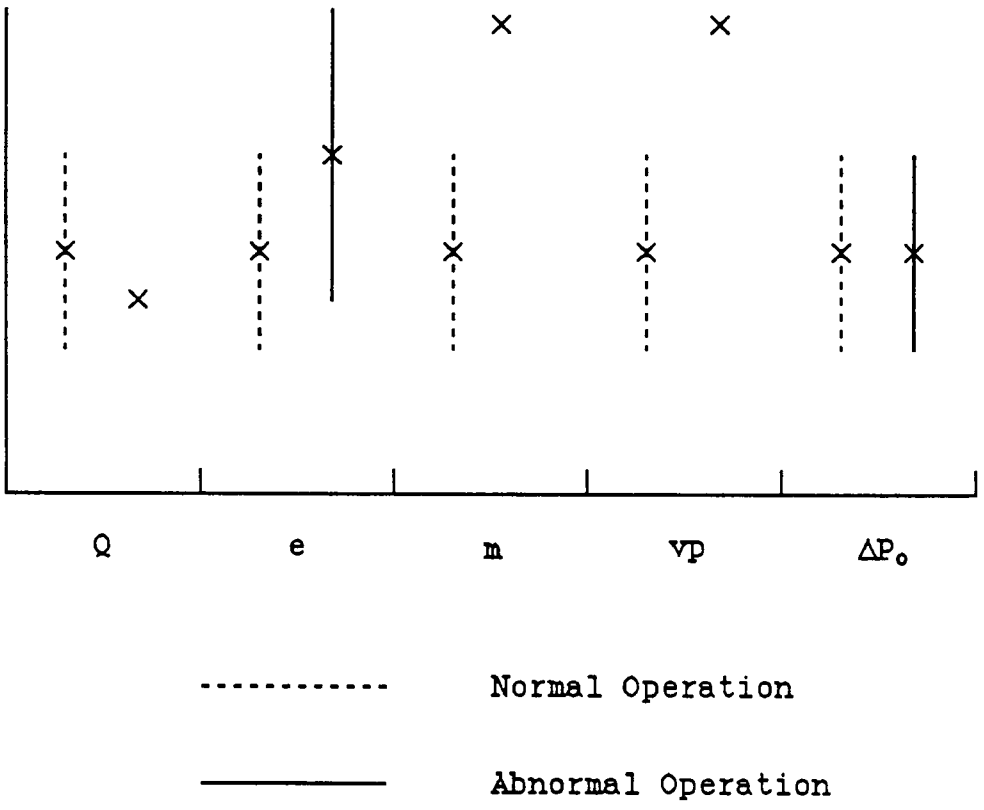


Figure 5.4: Sensor Statistics for a Stuck Flow Meter

small or zero standard deviation and e to have a non-zero mean. Furthermore, the valve may be fully open or fully closed. The problem of detecting a poor measurement of the overall pressure drop, ΔP_O, is more difficult and will be discussed again later. For an obvious failure of ΔP_O, one may notice that its standard deviation is very small or zero, while the other statistics are normal. Table 5.1 summarizes the analysis of sensor statistics.

5.2 Gain Change

Changes in a process can result in changes in process gain. The change in gain may in turn affect the control performance. In a flow process, a change in gain may signal a significant problem, such as plugging. The valve gain may be expressed in terms of flow rate versus valve position, $\Delta Q/\Delta vp$, or in terms of flow rate versus controller output, $\Delta Q/\Delta m$. The former is more useful than the latter, which involves valve dynamics.

To analyze valve gain, one must consider the overall flow system and not just the valve. The pressure drop across each component of the flow system is given by[26]

$$\Delta P_i = \rho\left(\frac{Q}{C_i}\right)^2$$

Table 5.1: Summary of Analysis of Sensor Statistics

Problem	Q	e	m	vp	ΔP_o
Stuck vp sensor	Normal	Normal (0 mean)	Normal	zero σ	Normal
Stuck valve	σ corresponds with ΔP_o	Non-zero mean	High σ or Saturation	zero σ	Normal
Controller on manual	σ corresponds with ΔP_o	Non-zero mean	zero σ	zero σ	Normal
Stuck flow meter	zero σ	Non-zero mean	High σ or Saturation	High σ or Saturation	Normal

where C_i is the flow coefficient of the component. The sum of the pressure drops for each component equals the overall pressure drop or driving force across the entire flow system. If changes in elevation are incorporated into the overall pressure drop, then the overall pressure drop at steady state is given by

$$\Delta P_o = \frac{\rho}{g_c}\left[\frac{1}{c^2} + \frac{1}{c_v^2}\right]Q^2$$

where

$$\frac{1}{c^2} = \sum_i \frac{1}{c_i^2} \quad \text{and} \quad c_v = c_{v_{max}}\,\alpha^{(vp-1)}$$

Rearranging,

$$Q = c_o\sqrt{\frac{\Delta P_o\, g_c}{\rho}}$$

where

$$c_o = \sqrt{\frac{1}{\frac{1}{c^2} + \frac{1}{c_v^2}}}$$

Thus, the gain is a function of the valve position, density, and overall pressure drop. The gain is also affected by the coefficient, c, which can be changed by

plugging. Figure 5.5 shows the flow characteristics for a base case, a case with a plug, a case with a change in density, and a case of flow meter drift. These curves were generated by steady state calculations. Figures 5.6 and 5.7 show the valve gain versus valve position and flow rate for each of the cases. It is obvious that if a gain change has occurred, then a different valve position will be required to achieve the same flow rate. This phenomenon may appear as a shift on a statistical plot.

For constant density, ρ, and flow coefficient, c, the flow rate may be described by the following implicit model:

$$Q = f\left(vp, \sqrt{\Delta P_o}\right)$$

Although equations where derived above to show the nature of the flow system, such equations are not necessary for the following analysis. One only needs to know that certain variables are related to others, as shown in the implicit model above. The power of this method is not as obvious in this example as it would be for the case of a very complex system for which there is no rigorous mathematical model.

Given the implicit model, data may now be sorted according to the procedure described in Sections 3.1 and 3.3. Sorting into categories of vp and $\sqrt{\Delta P_o}$, and then sorting the categories by the mean of Q, one may obtain the statistical plot shown in Figure 5.8. A shift is visible on

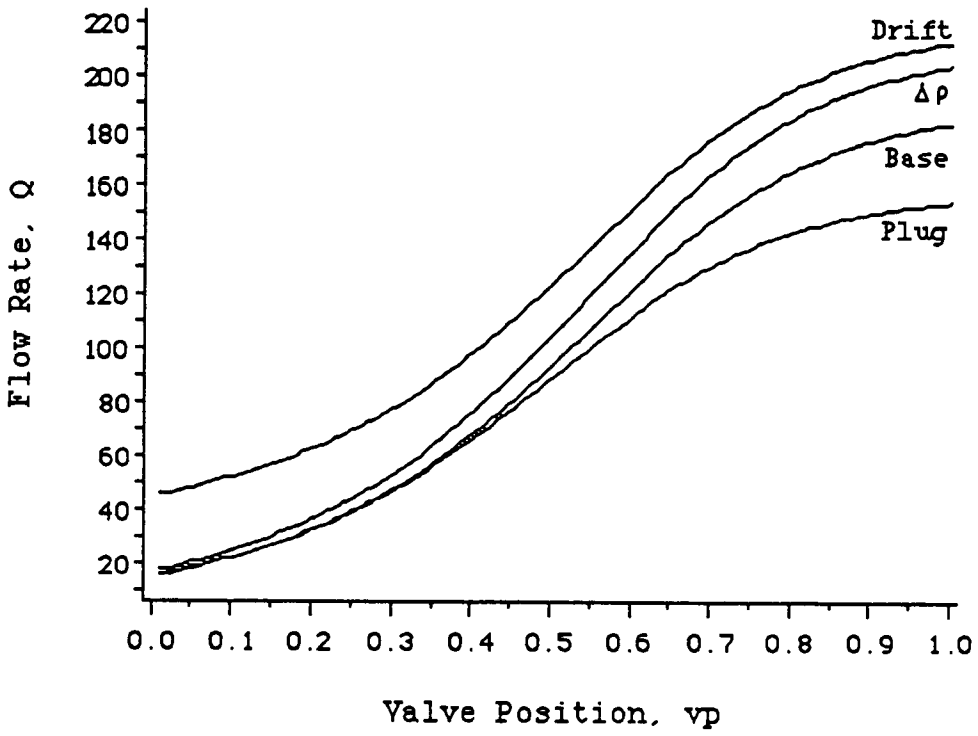
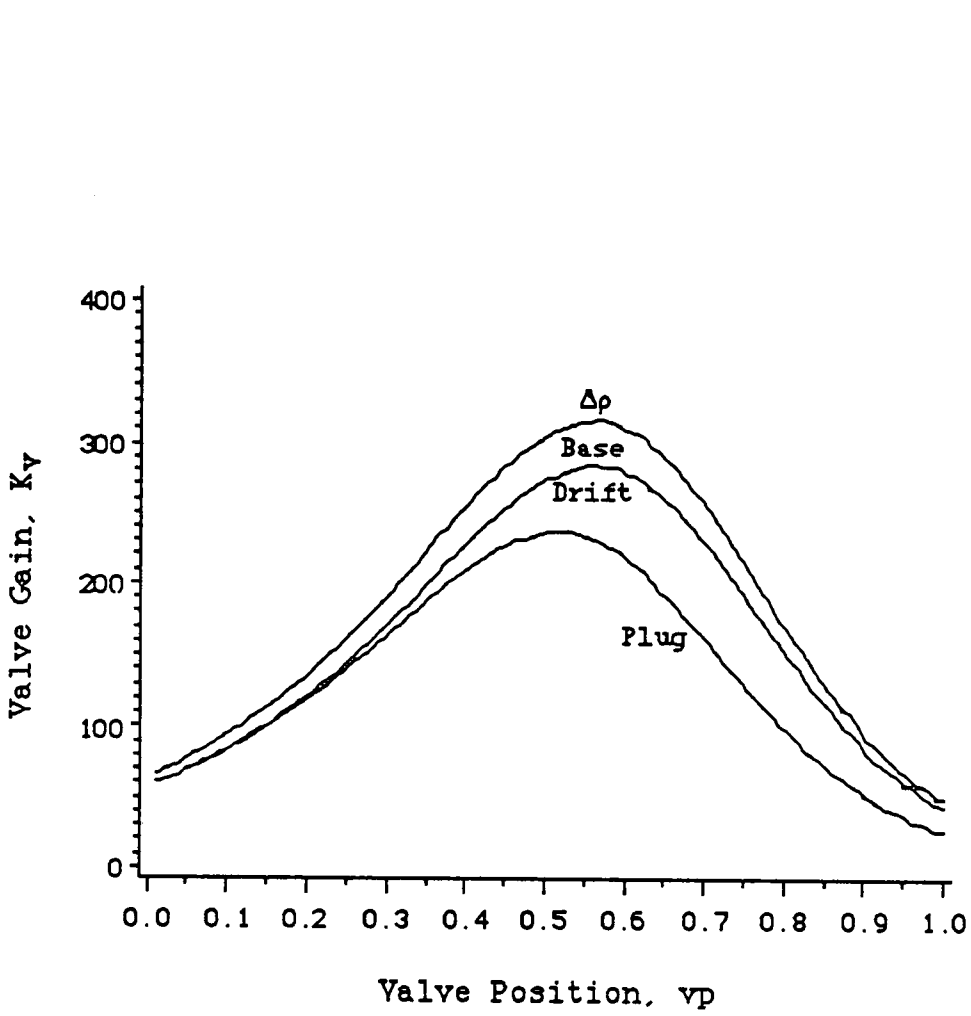


Figure 5.5: Flow Characteristics for Various Process Changes



Note: Base and Drift are on the same curve

Figure 5.6: Valve Gain versus Valve Position for Various Process Changes

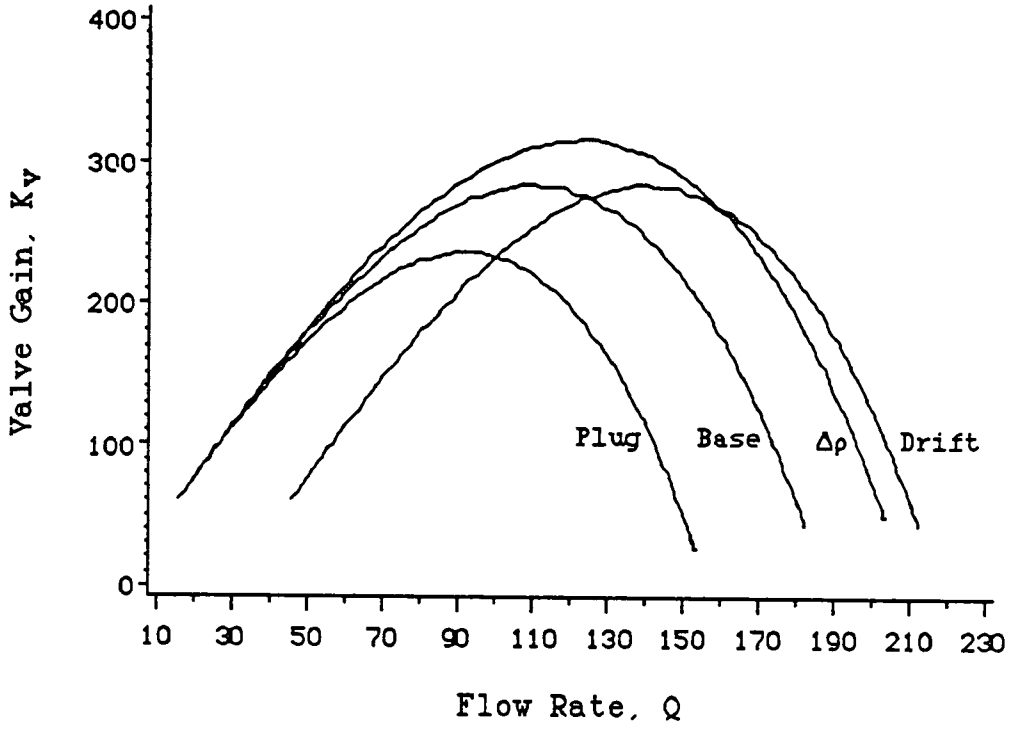


Figure 5.7: Valve Gain Versus Flow Rate for Various Process Changes

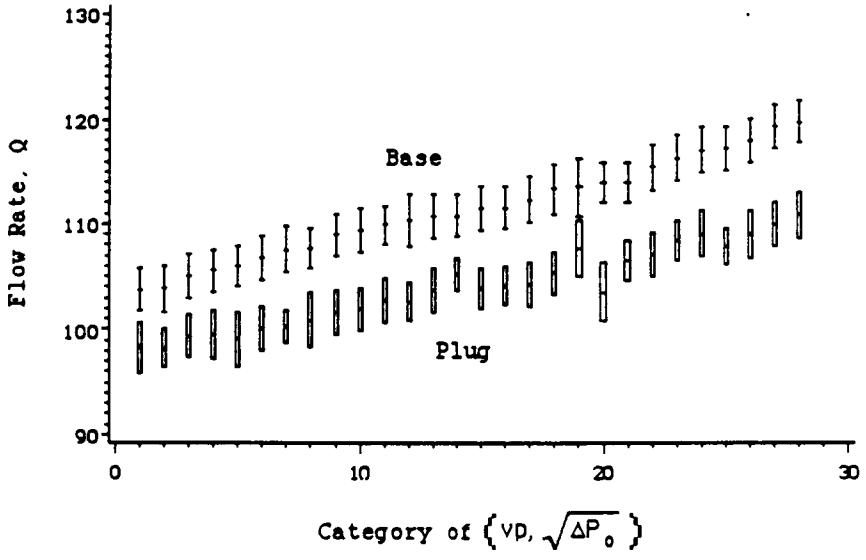


Figure 5.8: Statistical Plot of Q Showing a Change in gain

the plot, indicating a change in gain. However, since the range of valve position is different between the two cases, there are few boxes to compare. In the closed loop simulation vp is a dependent variable and Q is an independent variable as the set point of Q is set to range from 80 to 120 gpm. It is better to rearrange the implicit model,

$$vp = f\left(Q, \sqrt{\Delta P_o}\right)$$

to obtain the statistical plot shown in Figure 5.9. More boxes will have the same label, since more boxes will have the same Q than they will have the same vp. This plot compares the base case to a case with a plug. The shift indicates a decrease in valve gain. Note that the separation increases from left to right. This phenomenon is also evident in Figure 5.5, which shows the deviation of the flow characteristic increasing with valve position. Figure 5.10 shows a statistical plot for the case of a decrease in fluid density, which results in an increase in gain. Flow meter drift also gives a shift as shown in Figure 5.11.

Farell[8] characterized the statistical plots by calculating the average difference in means, the average standard deviation, the fractional overlap, and separation. Fractional overlap is the fraction of the total number of categories in the test case which "overlap" the base case. Separation is given by

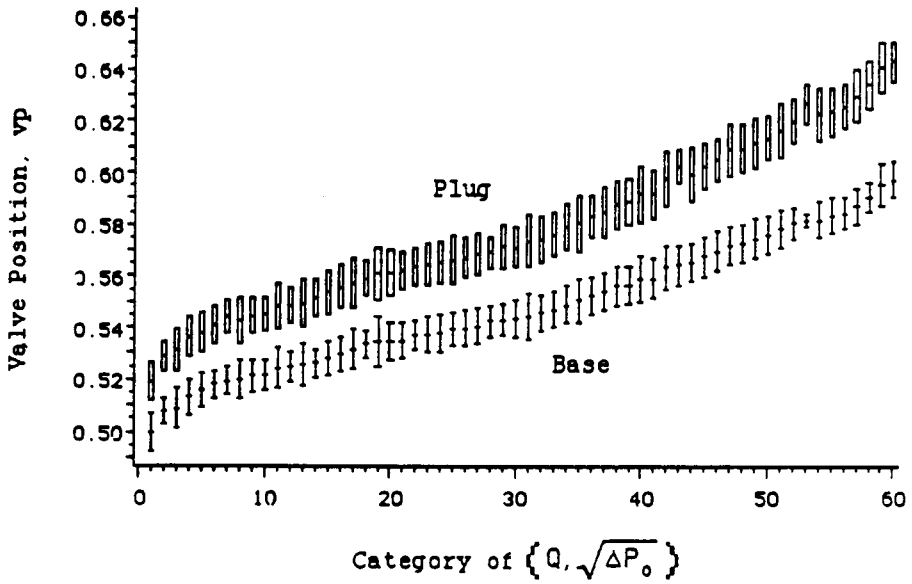


Figure 5.9: Statistical Plot of vp Showing a Change in Gain Due to a Plug

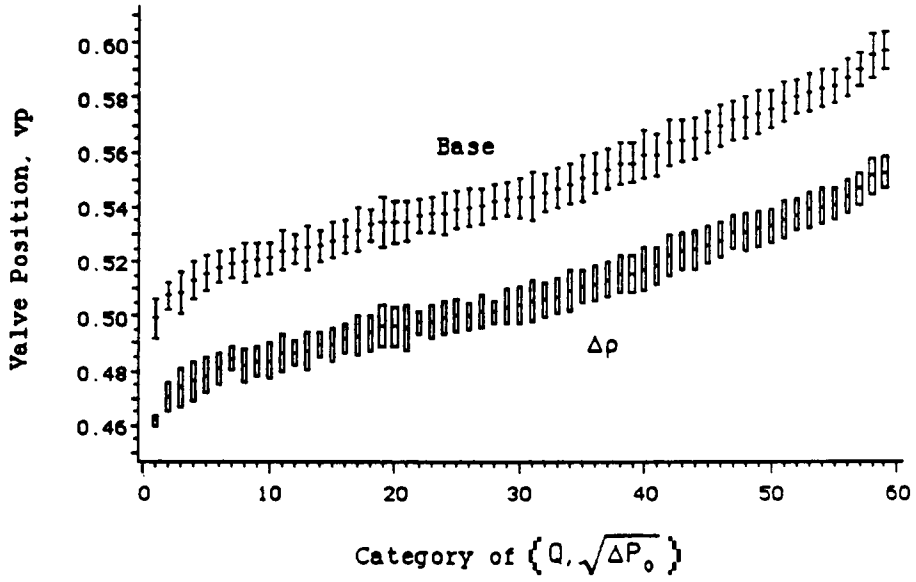


Figure 5.10: Statistical Plot of vp Showing a Change in Fluid Density

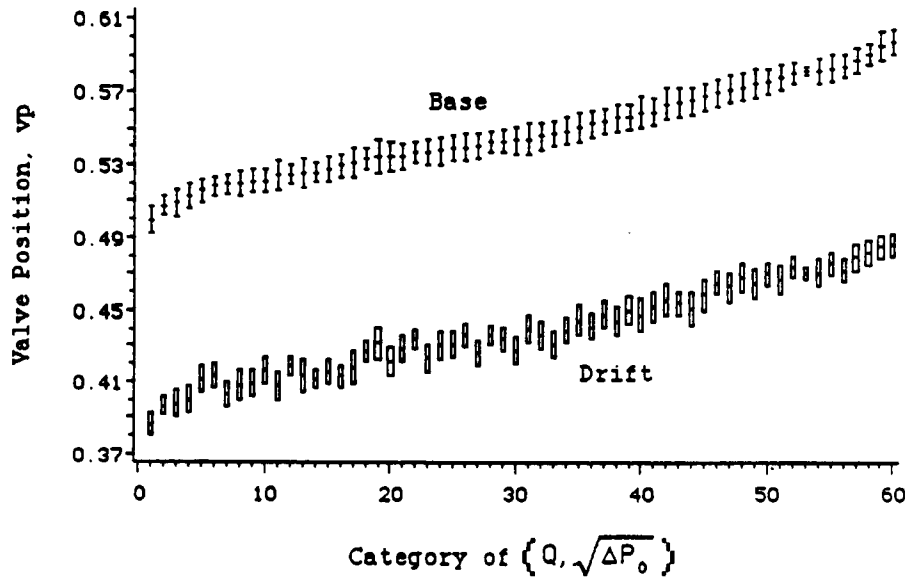


Figure 5.11: Statistical Plot of vp Showing Flow Meter Drift

$$\text{Separation} = \frac{1}{n}\sum_{i=1}^{n}\frac{M_{new_i} - M_{norm_i}}{s_{new_i} + s_{norm_i}}$$

where

M_{new_i} = mean value in category i in new (test) data

M_{norm_i} = mean value in category i in normal (base) data

s_{new_i} = standard deviation of new data in category i

s_{norm_i} = standard deviation of normal data in category i

Table 5.2 shows the characteristic statistics for the plots shown in Figures 5.9 to 5.11.

If a pressure drop measurement is taken across a section of the flow system that does not contain a valve, a simpler analysis may be used. One may simply calculate

$$c_o\sqrt{\frac{g_c}{\rho}} = \frac{Q}{\sqrt{\Delta P_o}} = \text{constant}$$

A change in the constant would indicate a plug, for example, in that section of the flow system.

The constant may also be calculated in the presence of a valve through the use of a relationship plot of Q versus $\sqrt{\Delta P_o}$ at constant vp, as shown in Figure 5.12. The slope of each line is $c_o\sqrt{g_c/\rho}$ and is calculated by least squares regression, forcing the y-intercept to the origin. The slopes are shown in Table 5.3. In this case, experience has

Table 5.2: Characteristics of Statistical Plots

Statistic	Plug	Change in Fluid Density	Flow Meter Drift
Separation	2.01	-3.05	-7.94
Fractional Overlap	0.0	0.0	0.0
Average Difference in Means	0.030	-0.040	-0.109
Avg. Diff. in stand. deviation	0.004	-0.002	0.0

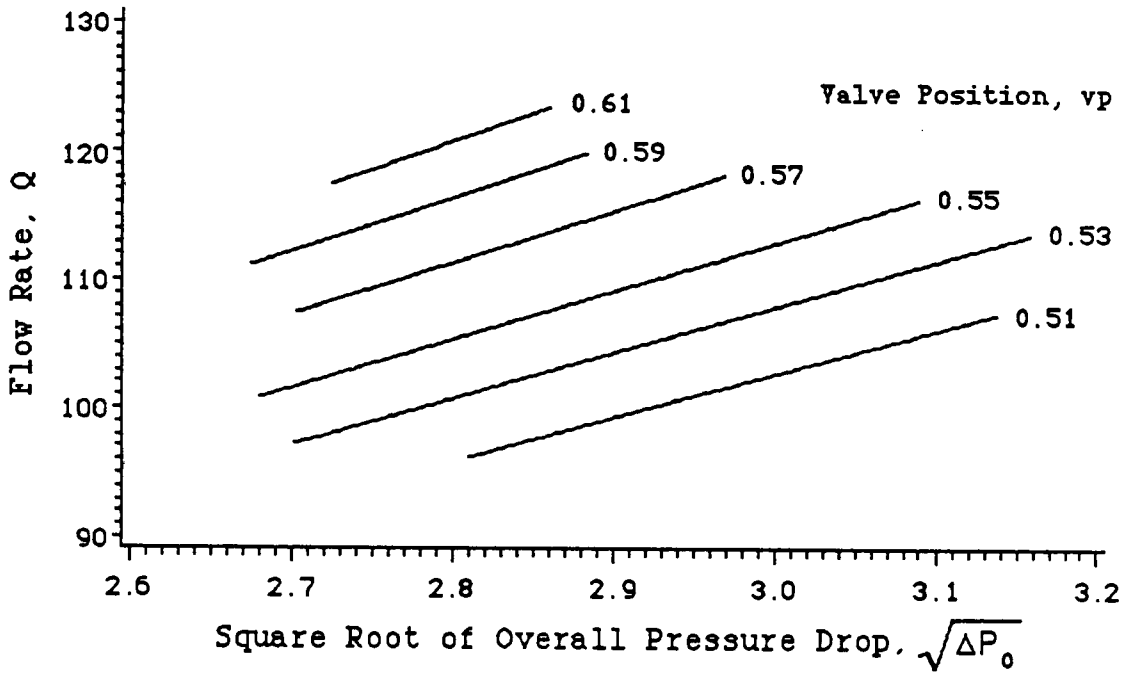


Figure 5.12: Relationship between Q and ΔP_o at Constant vp

Table 5.3: Statistics for Relationship Plot of Q vs $\sqrt{\Delta P_o}$ at Constant vp

vp	Slope	s(slope)	Correlation, r^2
0.50 - 0.52	34.23	0.03	0.720
0.52 - 0.54	36.00	0.02	0.786
0.54 - 0.56	37.65	0.02	0.748
0.56 - 0.58	39.79	0.03	0.502
0.58 - 0.60	41.59	0.03	0.544
0.60 - 0.62	43.10	0.06	0.579

shown that the results are not accurate or reproducible. A more reliable method is to construct a statistical plot relating vp to $Q/\sqrt{\Delta P_o}$. The plot may be constructed with either variable as the independent variable. Examples are shown in tabular form in Tables 5.4 and 5.5.

It should be noted that one must take care to choose the correct resolution in reducing the data set. If the resolution is too large, the data in a category may be poorly distributed, leading to poor results. A very small resolution may not give a desired reduction of data or may be influenced by noise.

5.3 Hysteresis

The detection of hysteresis requires some thought on how to sort the data. The first thought one might have is to plot vp versus m in order to trace the dead band. However, as shown by the scatter plot in Figure 5.13, the width of the dead band is blurred by dynamics. One can not use the deviation variables Δm and Δvp, because the noise on Δvp will cover the effects of hysteresis. But, with an understanding of the nature of hysteresis, one may arrive at a sorted data set that will be useful.

Hysteresis does not come into play when the valve stem is moving but, instead, when it is not moving. When the valve stops, an observer could determine the hysteresis by

Table 5.4: Statistics Relating $Q/\sqrt{\Delta P_o}$ to vp

vp	Mean of $Q/\sqrt{\Delta P_o}$	Stand. Dev.
0.50 - 0.52	33.90	0.32
0.52 - 0.54	35.86	0.36
0.54 - 0.56	37.88	0.35
0.56 - 0.58	39.80	0.36
0.58 - 0.60	41.85	0.37
0.60 - 0.62	43.25	0.36

Table 5.5: Statistics Relating vp to $Q/\sqrt{\Delta P_o}$

$Q/\sqrt{\Delta P_o}$	Mean of vp	Stand. Dev.
32 - 34	0.500	0.004
34 - 36	0.521	0.004
36 - 38	0.542	0.005
38 - 40	0.562	0.004
40 - 42	0.581	0.005
42 - 44	0.602	0.004

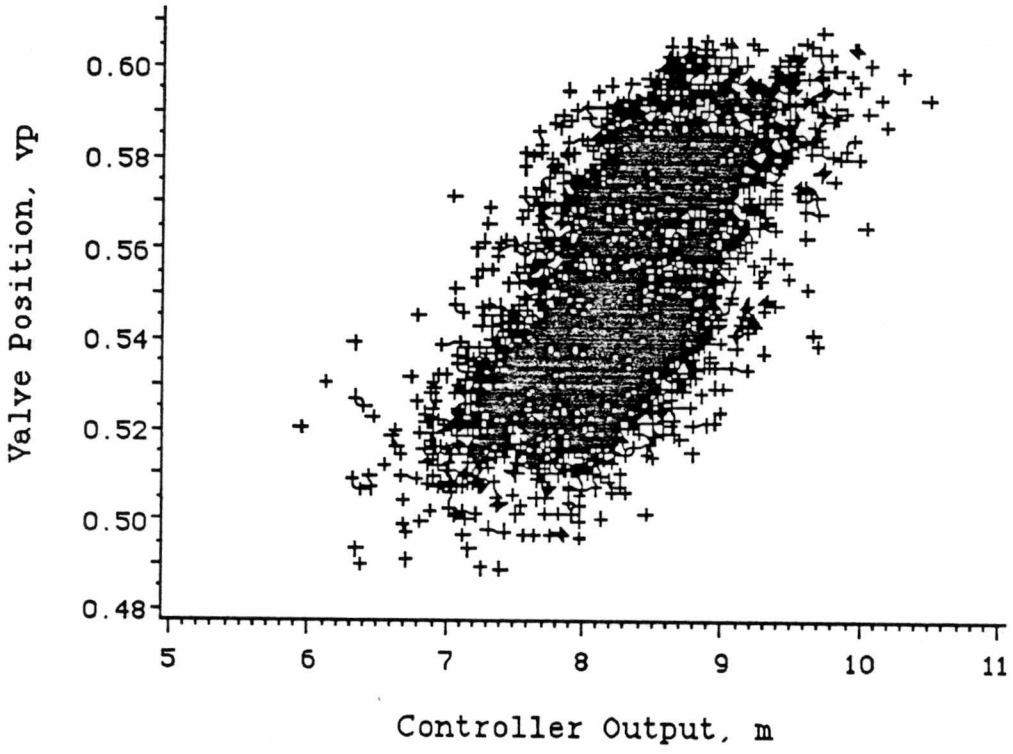


Figure 5.13: Scatter Plot of Valve Hysteresis

observing and recording the controller output required to get the valve moving again.

Since the controller is digital, the controller output that causes the valve to move may exceed that actually required. Thus, the observer should record the valve position and controller output at the sampling instant before the instant at which the valve was noticed to be moving.

If the controller gain is low, the recorded controller outputs will be very close to those required to move the valve, and the data will fall along two lines as shown in Figure 5.14. These lines represent the boundaries of the dead band. The lines will be far apart for a large hysteresis and close for a small hysteresis. To detect a change in hysteresis, one may look for a change in the range of m for a given vp. For a larger controller gain, many points will fall between the lines, as shown in Figure 5.15. However, with the same hysteresis, the same boundaries will be evident. If the observer had recorded the controller outputs that actually caused the valve to move, many points would have fallen outside the lines and blurred the boundary of the dead band. Figure 5.16 shows a case of an increase in hysteresis.

In practice, the observer is a data acquisition system that records data at every sampling instant. This means the cultivation routine must sort out the data corresponding to

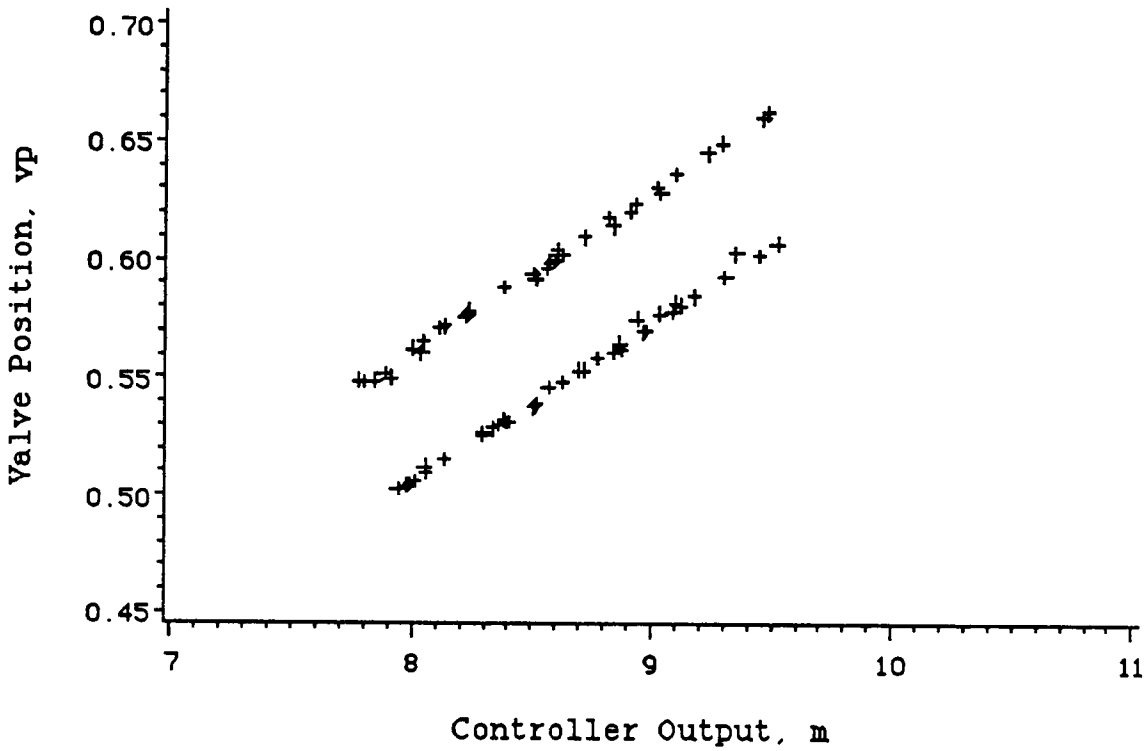


Figure 5.14: Hysteresis Plot for Low Controller Gain

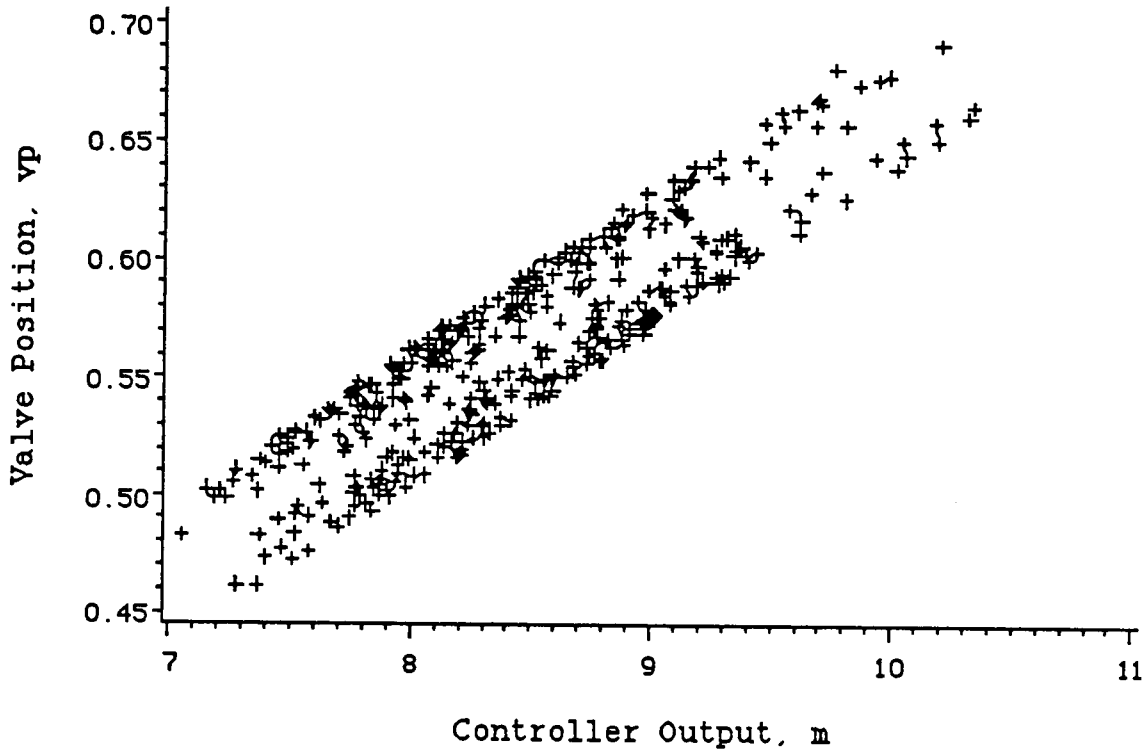


Figure 5.15: Hysteresis Plot for Normal Controller Gain

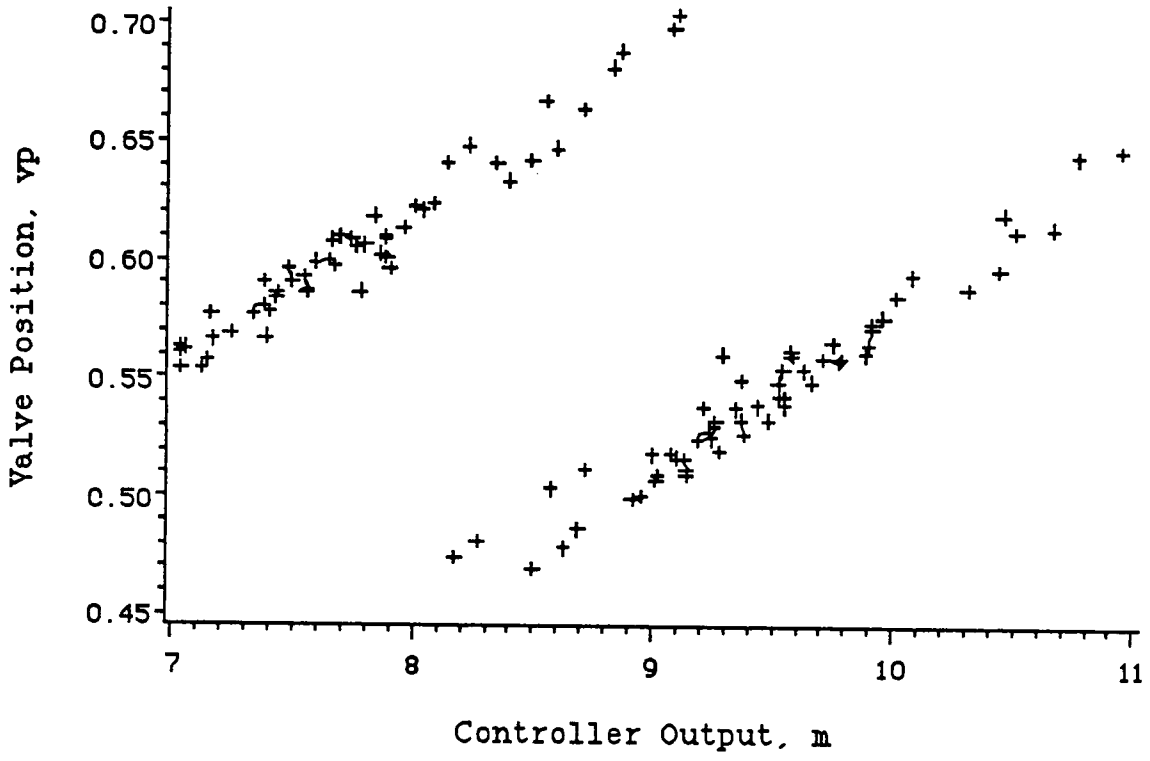


Figure 5.16: Hysteresis Plot for a Large Amount of Hysteresis

the valve at rest. In determining a state of rest the sorter can not search for a Δvp of zero. It can however, search for data such that $|\Delta vp| > \varepsilon$, where ε is greater than the magnitude of noise. An ε that is smaller than the magnitude of noise will not cause any trouble. Any noise that is plotted will fall about the center line between the boundaries of the deadband, as shown in Figure 5.17.

The cultivation routine that may be used to perform the above operation was discussed in Section 3.3. In the above operation, one is really sorting the data into class 1: signals at rest with fast variation. The range of fast variation, r_{fv}, is associated with the noise of the sensor measuring valve position.

The result of an analysis of hysteresis is shown in Figure 5.18. The plot shows base data collected from a valve with 3% hysteresis and test data collected from a valve with 10% hysteresis. The range of controller output is much larger for the test data than the base data. A change in hysteresis is clear.

It is not necessary to look for a change in the time constant of the valve as a possible process problem, because a change in hysteresis is accompanied by a change in the time constant. Both are a result of friction between stem and packing: hysteresis from static friction and the time constant from dynamic friction.

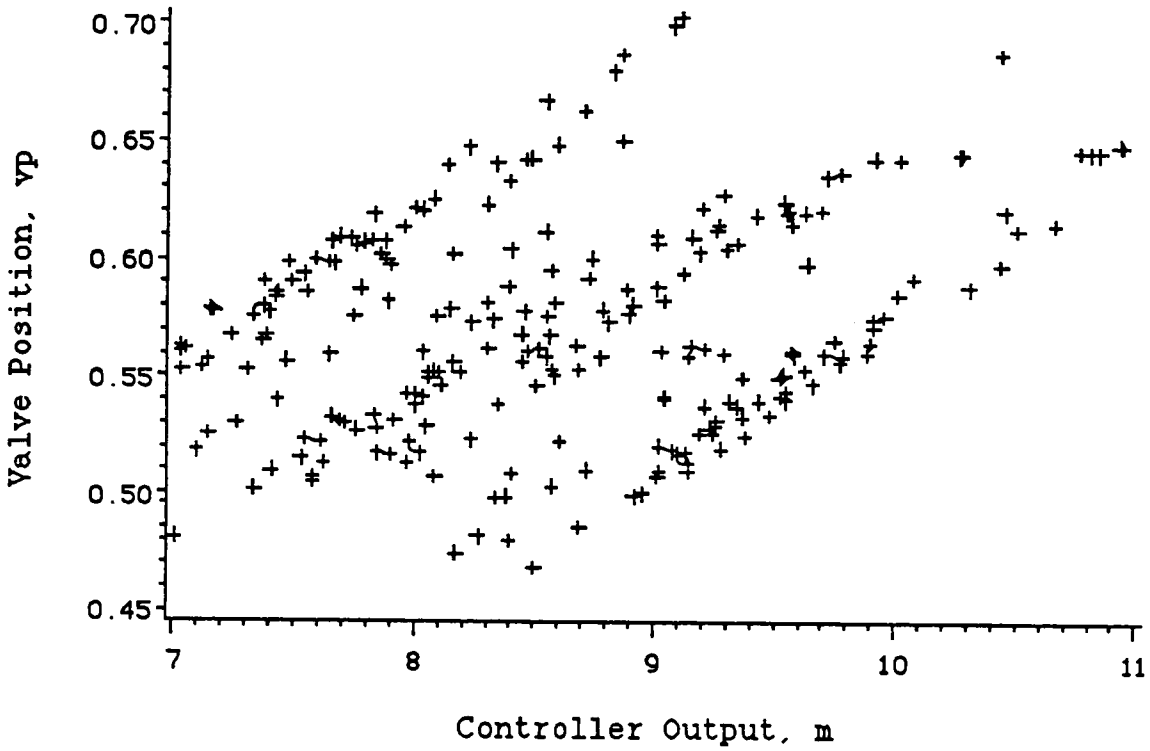


Figure 5.17: Effect of Valve Position Noise on Hysteresis Plot

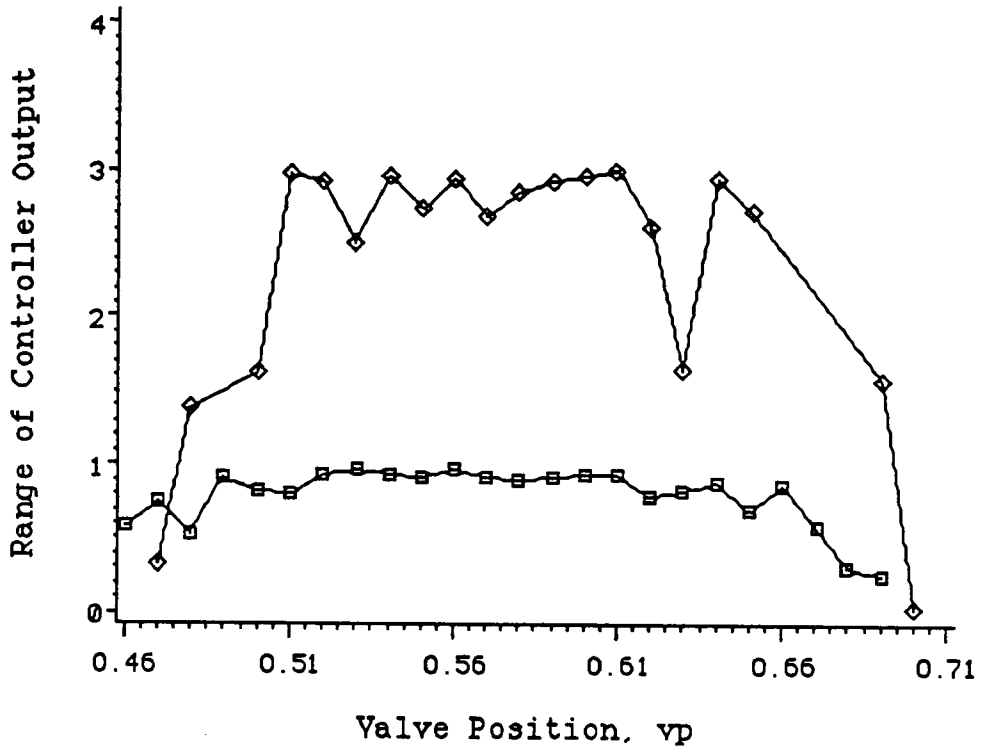


Figure 5.18: Result of a Change in Hysteresis

There is another use for the data sorted in the above fashion for detecting hysteresis. That is, if we record the midrange of m for each box of vp, we can construct the linear relationship between m and vp. If this relationship changes, then a problem exists in the transmission of the control signal, m, to the valve. In collecting the data, boxes containing a range of m that is not within a small deviation from the maximum range of m should be discarded. This will insure a good value of midrange.

5.4 Control Performance

Perhaps a more important analysis is an analysis of the control performance. There are a huge number of flow loops in a plant, most of which are likely not well tuned. There are too many to bother with, and only the loops that are causing serious problems may be retuned. A loop may not even be retuned following valve maintenance. Furthermore, if one wanted to retune a loop, how would he know how well the loop is tuned or in which direction to tune, ie. more conservative or less conservative? How would one use plant data to make the decision? Most methods require the use of a known disturbance, which is not available in plant data. An alternative method involves the time-shifted distribution (TSD) plot developed by Hackney[10]. There is no general

theory for the TSD analysis. Its development is heuristic as shown in the discussion that follows.

The basic idea behind the TSD plot is simple. If there is an error, ie. the process output is not at set point, the controller should be moving the process in a direction that will reduce the error; otherwise, the controller is contributing to or causing error. This is meant in a general sense, bearing in mind that a controller designed for an underdamped response will by nature move in the wrong direction during overshoot.

Consider the closed-loop step response of a first-order system with a PI controller as shown in Figure 5.19. Note that the controller output steps up initially and then steps down to steady state, while the process moves up to steady state. A phase plot of the change in controller output versus error, as shown in Figure 5.20, gives a different perspective.

Separating the control signal into the integral (m_I) and proportional (m_p) modes, one obtains

$$m_I = m_b + \frac{K_c}{\tau_I}\int e\,dt \quad \text{and} \quad m_p = m_b + K_c e$$

As shown in Figure 5.21, for an overdamped system, the integral mode rises without overshoot. Figure 5.22 shows a phase plot of the change in the integral mode (Δm_I) of the

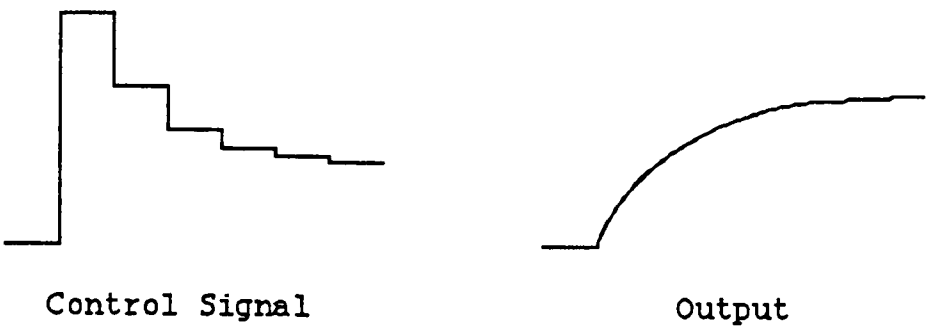


Figure 5.19: Closed-loop Step Response

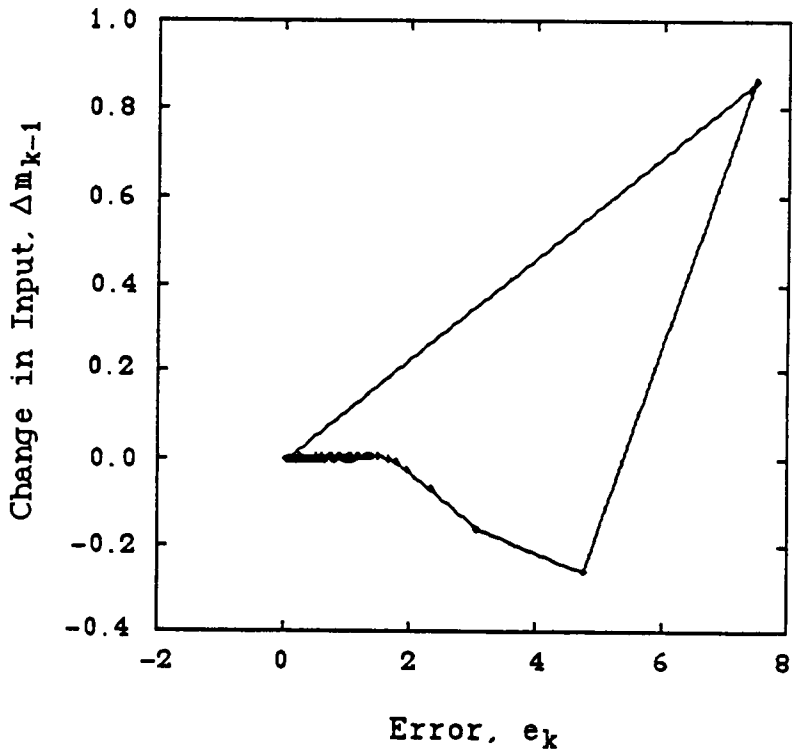


Figure 5.20: Change in Input Versus Error

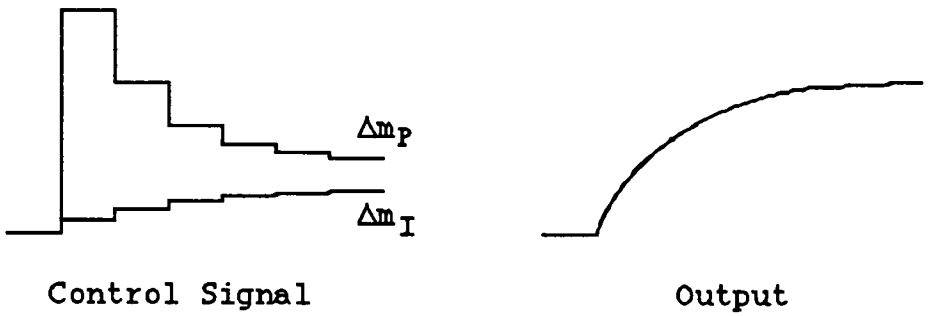


Figure 5.21: Closed-loop Step Response Showing Proportional and Integral Control Action

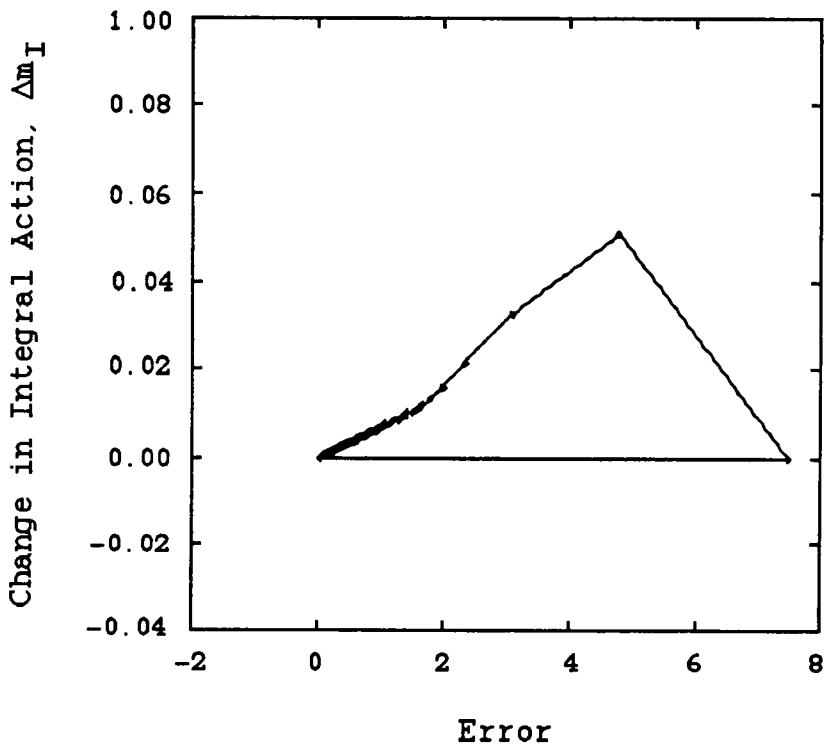


Figure 5.22: Change in Integral Action Versus Error

control signal versus error. Note that all of the data falls into the first quadrant. Had the step change been negative, the data would have fallen into the third quadrant. For an underdamped system, the plot spirals into the origin.

If the controller is well tuned, data on the phase plot will fall mostly into quadrants 1 and 3. Figure 5.23 shows a scatter plot of such data, representing a simulation of the PI flow control loop with a well-tuned controller. The data falling into quadrants 2 and 4 is mainly the result of flow meter noise. A poorly tuned controller causes much data to fall into quadrants 2 and 4, as shown in Figure 5.24. The data for this plot was generated by a simulation with a high controller gain. These plots may be characterized by determining the percentage of the total number of points that fall in quadrants 1 and 3.

In the analysis above, we have established the distribution of data between the positive (1 and 3) and negative (2 and 4) quadrants for only one sampling step beyond the controller action. To obtain more information, such as dead time or sluggishness, we must consider the effect of a control action on the process versus time. By shifting the data, plotting Δm_{k-1} versus e_{k+n}, where n is a positive integer, one would be analyzing the effect of Δm on the output versus time. Error is used instead of the output in order to accommodate the case of a changing set point. A

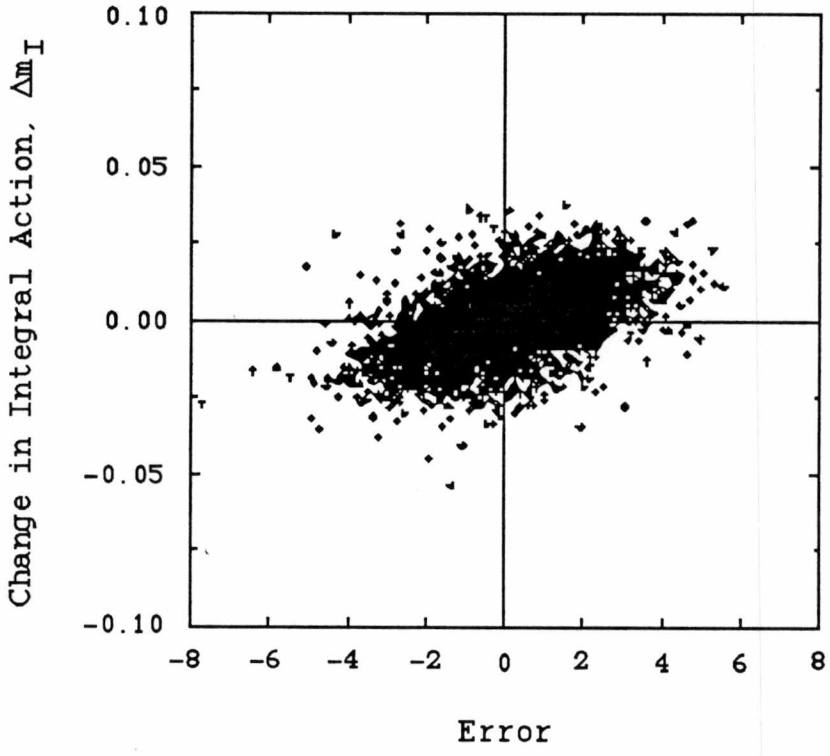


Figure 5.23: Scatter Plot of Δm_I Versus Error for Good Control

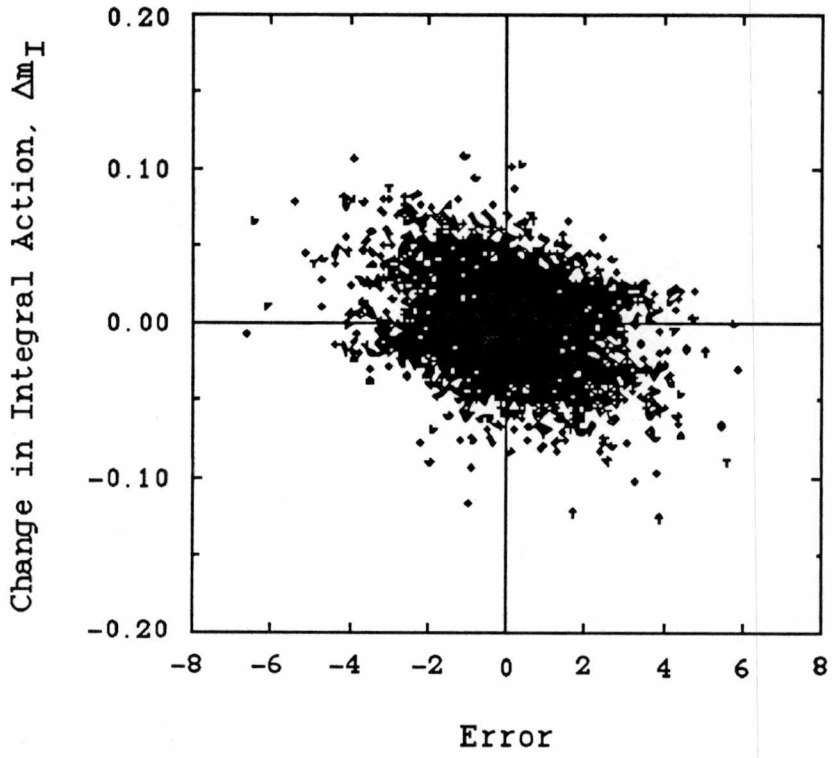


Figure 5.24: Scatter Plot of Δm_I Versus Error for Poor Control

time-shifted distribution (TSD) plot, as shown in Figure 5.25, is a plot of the percentage of data points in quadrants 1 and 3 versus the shift, n. Hackney mentions that the TSD plot resembles cross correlation because of the shift in time.

With the TSD plot one can analyze controller performance and determine characteristics of the control response. Figure 5.26 shows a case with a conservative controller, and Figure 5.27 shows a case with a very aggressive controller. These plots were generated from simulations of the PI flow control loop. Note that the plot resembles a second-order, closed-loop step response. The connection is not clear, but one may be able to determine whether the closed-loop response is underdamped or overdamped. In any case, it is clear from the TSD plot that the controller in Figure 5.27 performs differently than the one in Figure 5.26. Figure 5.28 shows the case of a high controller gain. In this case the controller is responding to flow meter noise and destablizing the system. A plot beginning at a value below 50% indicates a serious problem. This means that for a majority of the time, the controller is moving in the wrong direction.

One means of characterizing the TSD plot is to locate the crossover points where the plot crosses the 50% line. The first crossover moves right as the controller action becomes more conservative. One may also find the extrema

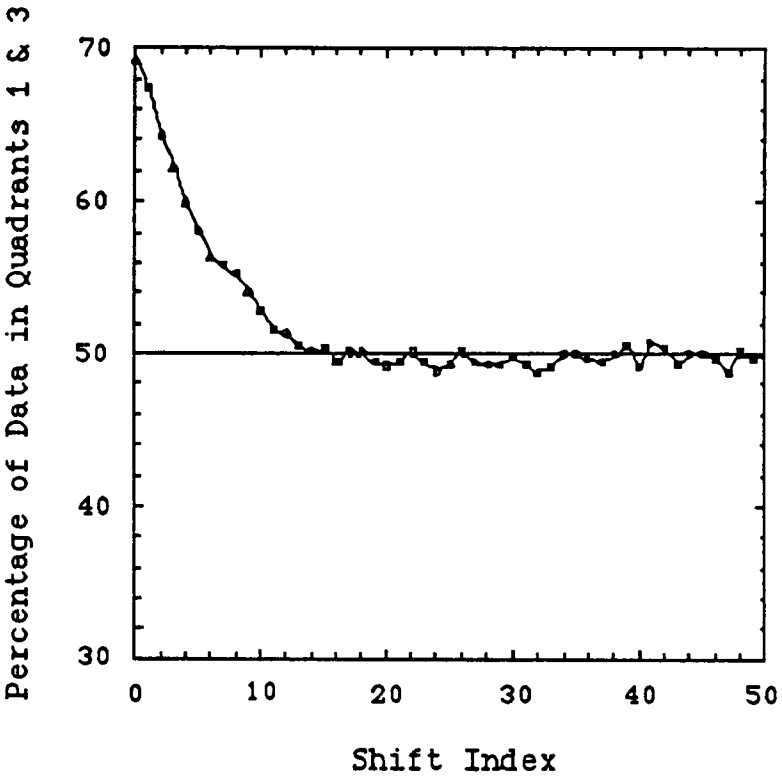


Figure 5.25: TSD Plot

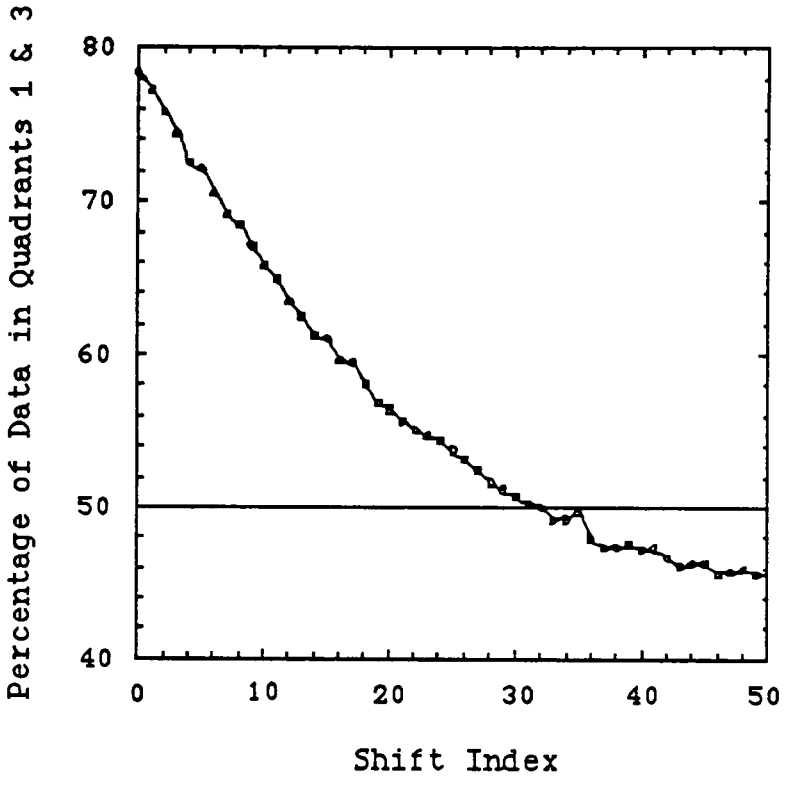


Figure 5.26: TSD Plot for a Conservative Controller

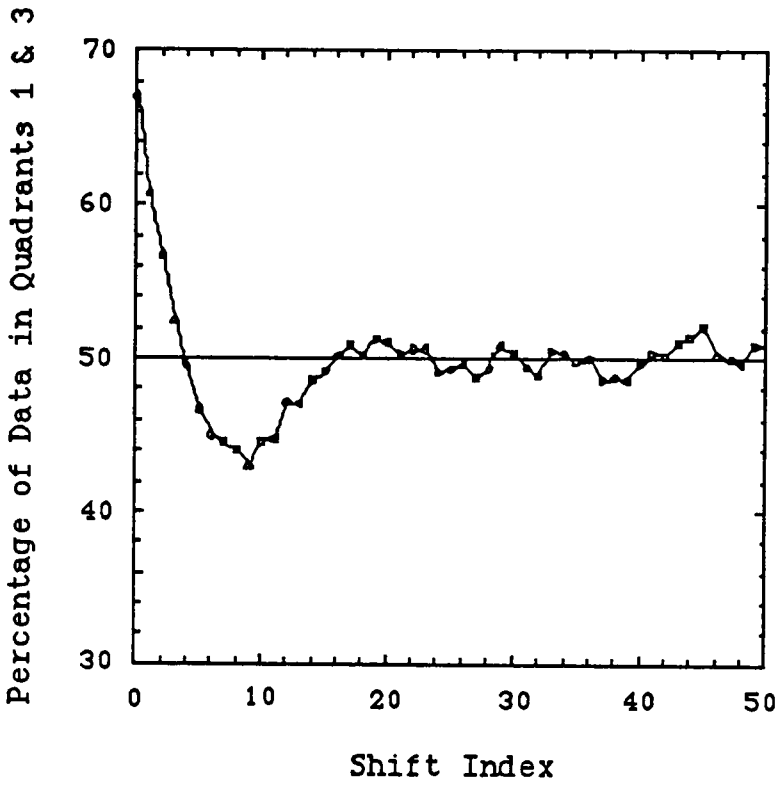


Figure 5.27: TSD Plot for an Active Controller

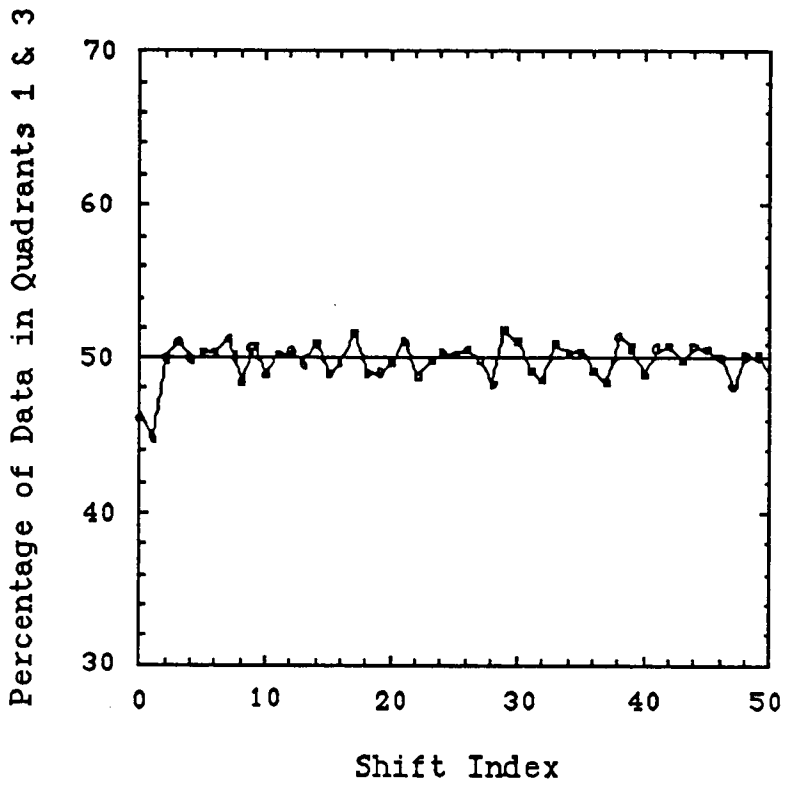


Figure 5.28: TSD Plot for a Controller Reacting to Noise

between crossovers and integrate between the curve and the 50% line between crossovers. Such an analysis is shown in Table 5.6.

5.5 The Decision Tree

A decision tree that may be used for the analysis of the PI flow control loop is shown in Figure 5.29. The tree offers a logical progression through the analysis techniques discussed in this chapter. The top of the tree uses sensor statistics to search for obvious failures, such as stuck sensors, a stuck valve, and a controller left on manual. These are problems that are easy to detect but can render the proceeding analysis useless. This part of the tree was derived from Table 5.1 on page 49.

Next, a TSD plot is analyzed to determine whether or not the control performance has changed. If not, the control is normal. If there is a change, it is characterized as more active if the crossover moves left, more conservative if the crossover moves right, and unstable if the first integral is negative. There is also a check for a change in the controller parameters K_C and τ_I. The tree then splits into two parallel branches to determine the cause of the change.

One branch checks a statistical plot of vp versus Q and ΔP_O. If separation has occurred, then there is a gain change due to plugging, a change in fluid properties, or flow meter

Table 5.6: Characteristics of a TSD Plot

Crossover	Integral	Extrema	Location of Extrema
3.84	28.05	66.99	0
15.90	-45.78	42.92	9
23.40	4.72	51.26	19
28.45	-3.54	48.77	27
30.40	0.90	50.81	29
32.70	-1.39	48.89	32
34.58	0.59	50.48	33

% of points in quadrants 1 & 3 at zero shift: 66.991

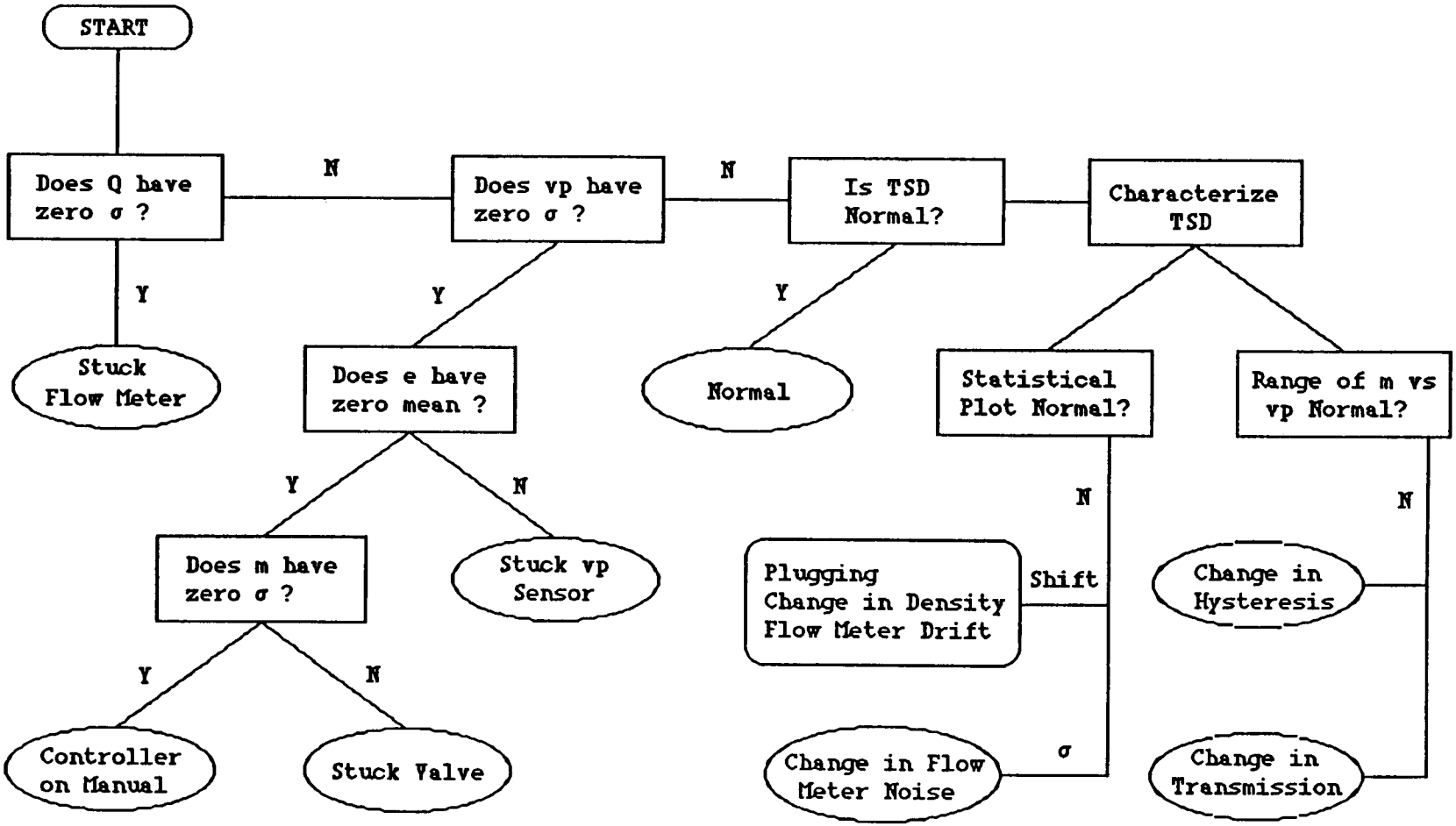


Figure 5.29: Decision Tree

drift. A higher standard deviation may indicate either higher sensor noise for Q, vp, or ΔP_o or increased dynamics. If the control has become more active or unstable and no other problems are detected in the analysis, then the flow meter noise has changed.

The other branch employs the methods of Section 5.3. It checks the range of m versus vp to detect a change in hysteresis. It also checks the midrange of m versus vp to examine the transmission of m to the valve.

An alternative arrangement of the tree would be to analyze the statistical plot at all times. In this way, sensor problems not picked up by the TSD plot may be detected.

The existence of multiple problems does not cause a serious problem in this application. If more than one sensor or element experiences serious failure, only one failure will be diagnosed by the decision tree. The other failures would become apparent in subsequent analyses after the malfunction is repaired. A change in gain and a change in hysteresis may be detected simultaneously, since the decision tree branches into parallel sequences.

The existence of normal and abnormal operation in a data set may cause some difficulty, especially in determining gain change. The TSD analysis would see the abnormality, although not in its full affect. The analysis of hysteresis would not

be affected as long as the hysteresis is increasing. Since the analysis looks for a maximum range of m, the maximum hysteresis would be found.

CHAPTER 6

RESULTS

The analysis techniques of system cultivation as discussed in Chapter 5 were applied to the PI flow control loop. Several simulations under various process conditions were performed to test the decision tree, beginning with the time-shifted distribution (TSD) plots.

Comparison of the base case, as shown in Figure 6.1 and Table 6.1, with a plugging condition, as shown in Figure 6.2 and Table 6.2, shows that the first crossover has moved right with an increase in the first integral. This indicates that the control has become more sluggish. Such would be expected, since a plug decreases the gain of the valve. Step changes were simulated under both conditions in order to show that the TSD plot does relate to an actual change in the control action. Plots of these simulations, shown in Figure 6.3 and Figures 4.9 to 4.11 on pages 37 and 38, show the increase in sluggishness.

Another observation of the TSD plot is that the integrals after the first remain small, while the extrema hug the 50-percent line. In that region, the plot is simply showing noise. In fact, whenever the extrema do not stray from the 50-percent line by more than about 3 percent, no matter how large the integral, the plot is considered to be

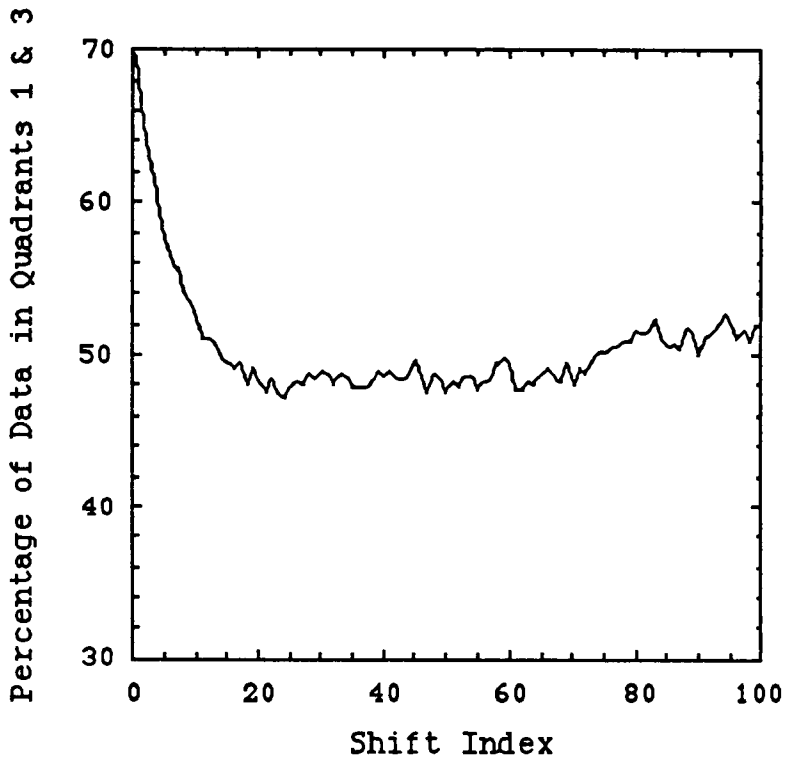


Figure 6.1: TSD Plot of Base Case

Table 6.1: Characteristics of the TSD Plot for the Base Case

Crossover	Integral	Extrema	Location of Extrema
13.75	91.66	69.71	0
73.70	-89.95	47.24	24
>99.00		52.58	94

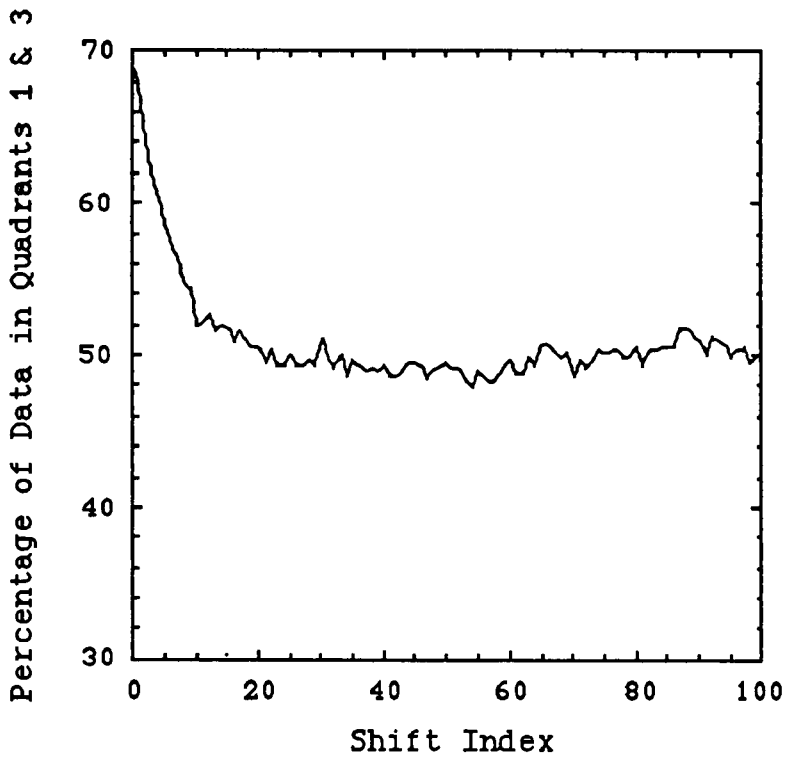


Figure 6.2: TSD Plot for the Case of Plugging

Table 6.2: Characteristics of the TSD Plot for the Case of Plugging

Crossover	Integral	Extrema	Location of Extrema
20.45	108.37	68.71	0
21.58	-0.32	49.43	21
22.40	0.17	50.42	22
29.39	-3.53	49.27	26
30.71	0.70	51.07	30
64.48	-32.18	47.91	54
67.48	1.29	50.67	65
68.52	-0.10	49.80	68
69.12	0.05	50.18	69

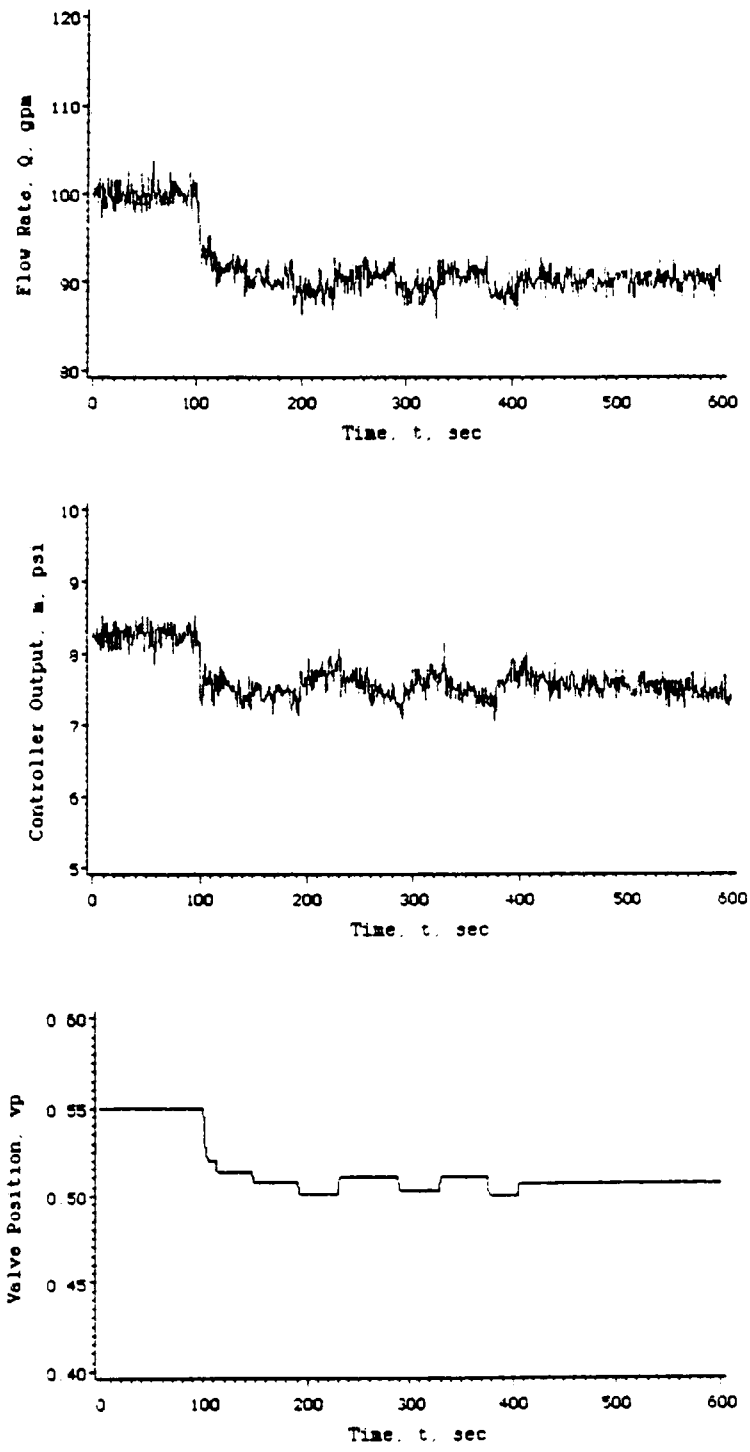


Figure 6.3: Step Test for the Case of Plugging

in a region of noise from which no more information may be obtained.

This brings up an important point in the use of TSD plots: the data must contain plenty of dynamic action plus process noise and/or oscillation. Consider the text book case of the overdamped step response with no noise, as shown in Figures 5.21 and 5.22 on page 77. A TSD plot of such a system would show a straight line at 100 percent. There is nothing to pull the plot down to the 50-percent line. Obviously, no information could be obtained from such a plot.

Comparing a case of increased hysteresis with the base case again shows increased sluggishness, as shown in Figure 6.3, Table 6.5, and Figure 6.4. But, in addition, the first and second integrals have become large, indicates that the control response has become more oscillatory. Comparison of the step response bears this observation as well. The case of plugging combined with hysteresis shows an even more pronounced shift in the TSD plot and more oscillation in the step response, as shown in Figure 6.6, Table 6.4, and Figure 6.7.

A decrease in fluid density would increase the gain of the process and thus make the control action more aggressive. The comparison of TSD plots in Figure 6.8, Table 6.5, and Figure 6.9 show that this is indeed the case. It shows a more active control with an increase in oscillation.

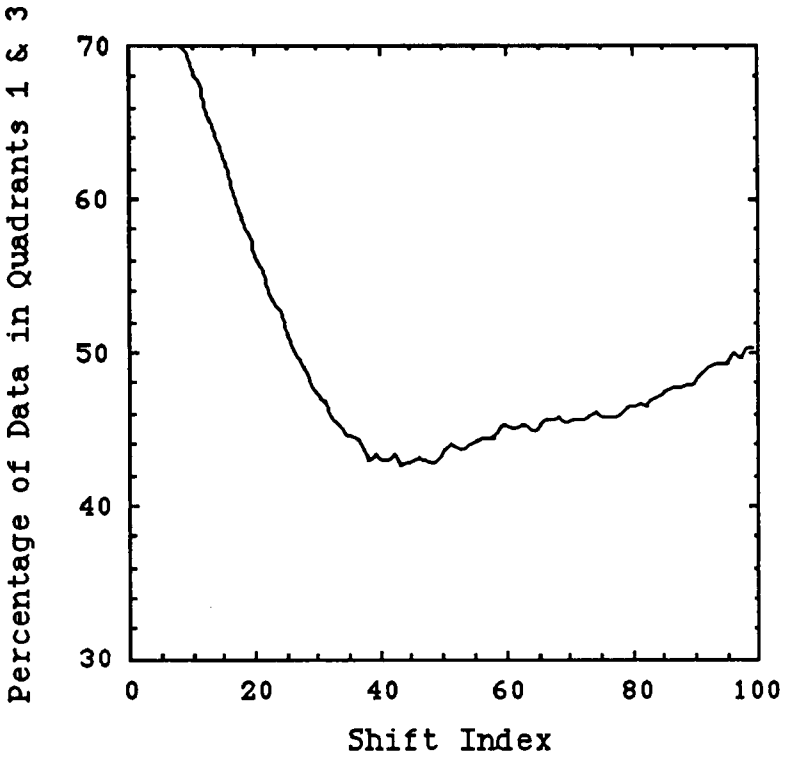


Figure 6.4: TSD Plot for the Case of Increased Hysteresis

Table 6.3: Characteristics of the TSD Plot for the Case of Increased Hysteresis

Crossover	Integral	Extrema	Location of Extrema
26.22	406.43	86.39	0
97.51	-308.70	42.52	43
>99.00			

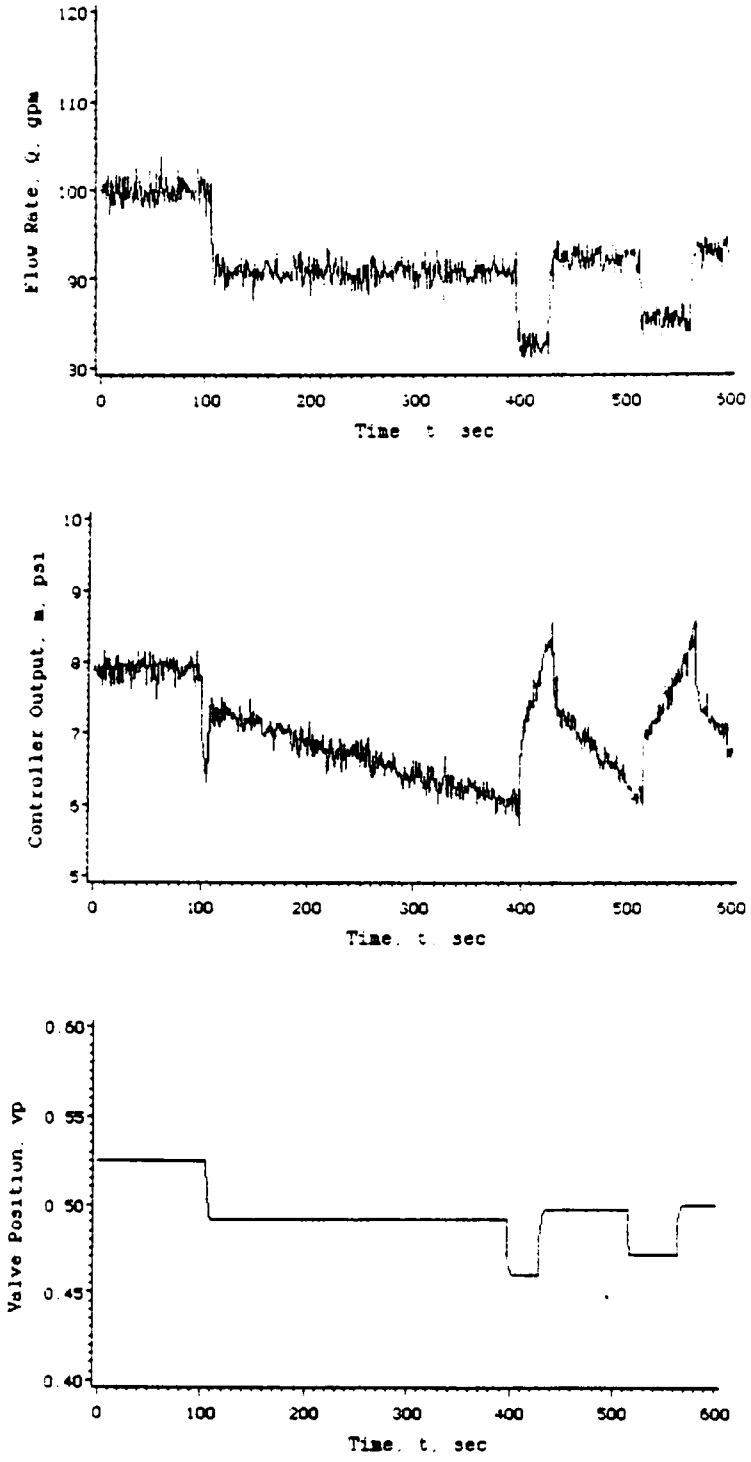


Figure 6.5: Step Test for the Case of Increased Hysteresis

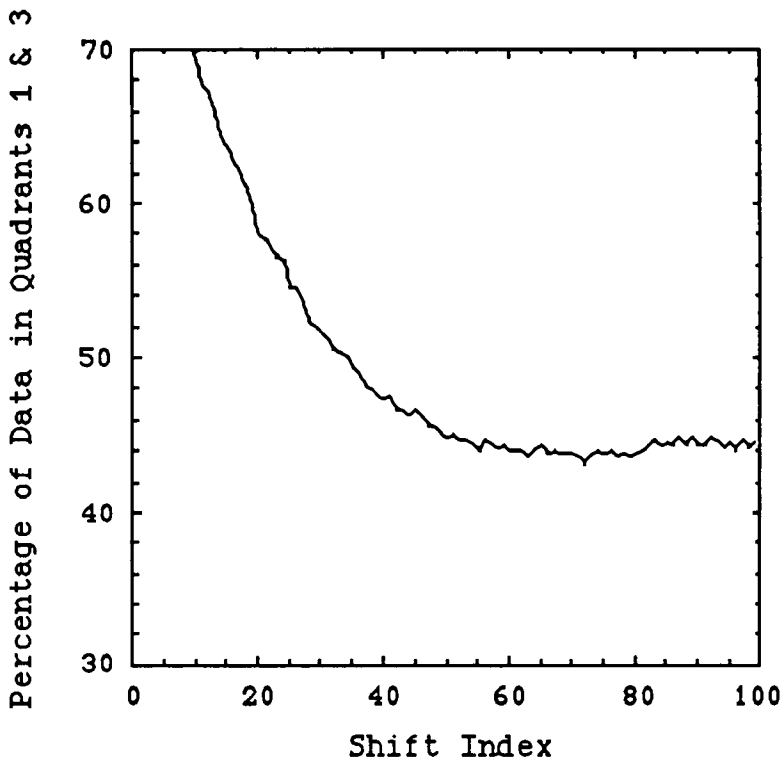


Figure 6.6: TSD Plot for the Case of Increased Hysteresis Combined with Plugging

Table 6.4: Characteristics of the TSD Plot for the Case of Increased Hysteresis Combined with Plugging

Crossover	Integral	Extrema	Location of Extrema
34.10	457.92	84.33	0
>99.00		43.07	72

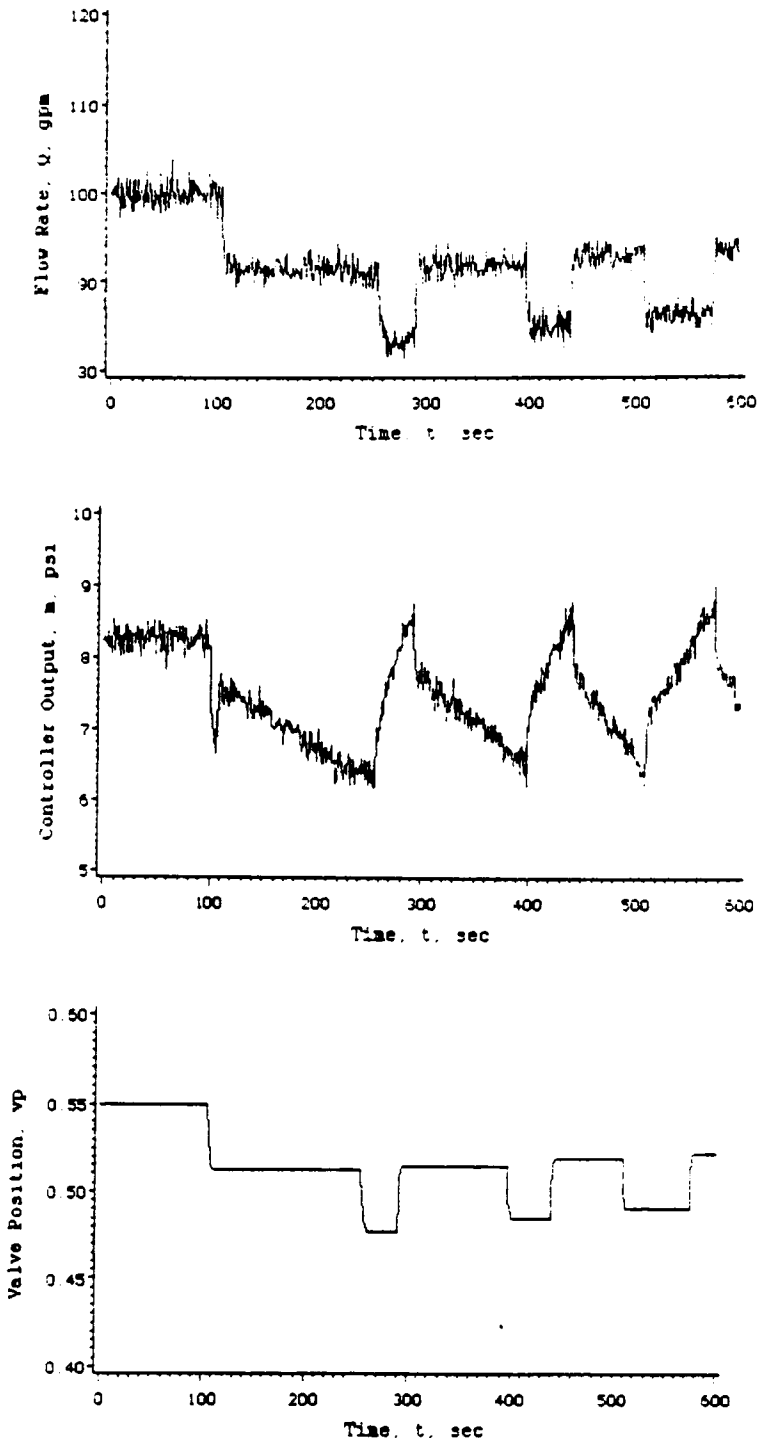


Figure 6.7: Step Test for the Case of Increased Hysteresis Combined with Plugging

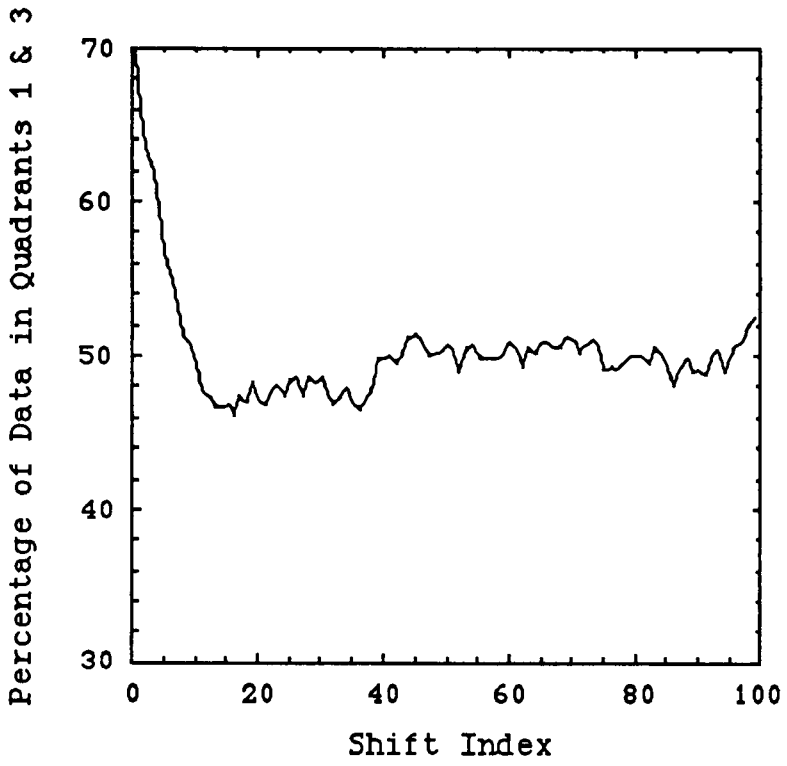


Figure 6.8: TSD Plot for the Case of a Decrease in Fluid Density

Table 6.5: Characteristics of the TSD Plot for the Case of a Decrease in Fluid Density

Crossover	Integral	Extrema	Location of Extrema
9.44	76.75	69.82	0
43.06	-74.25	46.11	16
46.90	3.44	51.41	45
47.46	-0.02	49.92	47
51.18	1.07	50.66	50
52.63	-0.75	48.97	52
54.69	0.94	50.63	54
58.54	-0.72	49.72	55
61.13	1.03	50.81	60

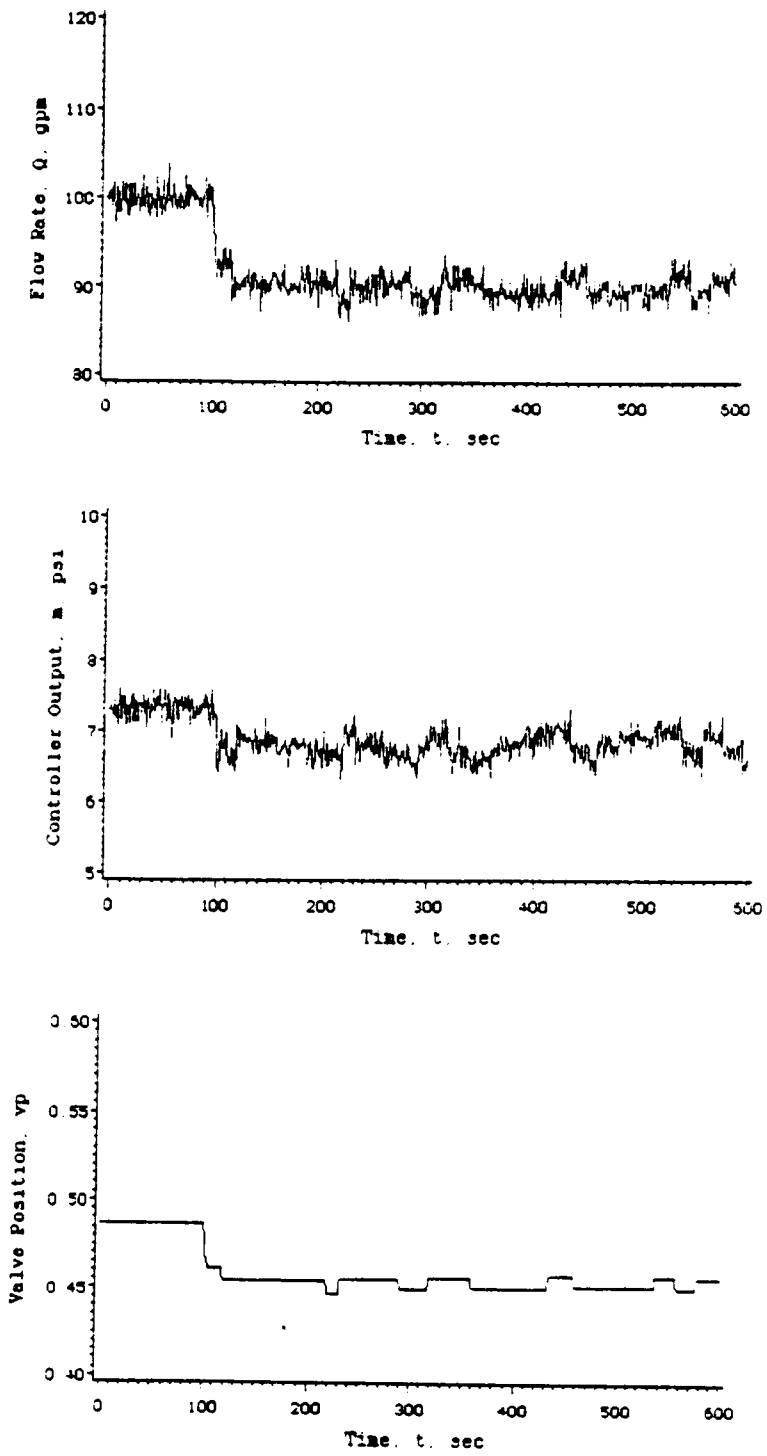


Figure 6.9: Step Test for the Case of a Decrease in Fluid Density

Figure 6.10, Table 6.6, and Figure 6.11 show the case of an increase in flow meter noise. In this case, the crossover has moved far left, while the integrals remain small, and the extrema hug the 50-percent line. Noise has increased in the system, and the controller is overreacting to the noise, as shown in the step response.

One very important consideration with TSD plots is the effect of disturbances. If a change in the disturbance has a significant effect on the TSD plot, then the analysis might not be of much value. Figure 6.12 compares TSD plots of the base case with those of cases involving an altered disturbance. The altered disturbance is the one which changes the set point. The disturbance used in the base case is a series of step changes as shown in Figure 4.7. The altered disturbances are shown in Figure 6.13. They include a doubled gain, the addition of white noise, and a change to exponential steps. Note that the TSD plots do not show a significant change in the first crossover. In fact, the plots look almost identical until they approach the 50% line, after which they enter the region of noise.

If, however, the disturbance type changes significantly, there may be an effect on the TSD plots as shown in Figure 6.14. The altered disturbances, shown in Figure 6.15, include a random walk and an integrated autoregressive

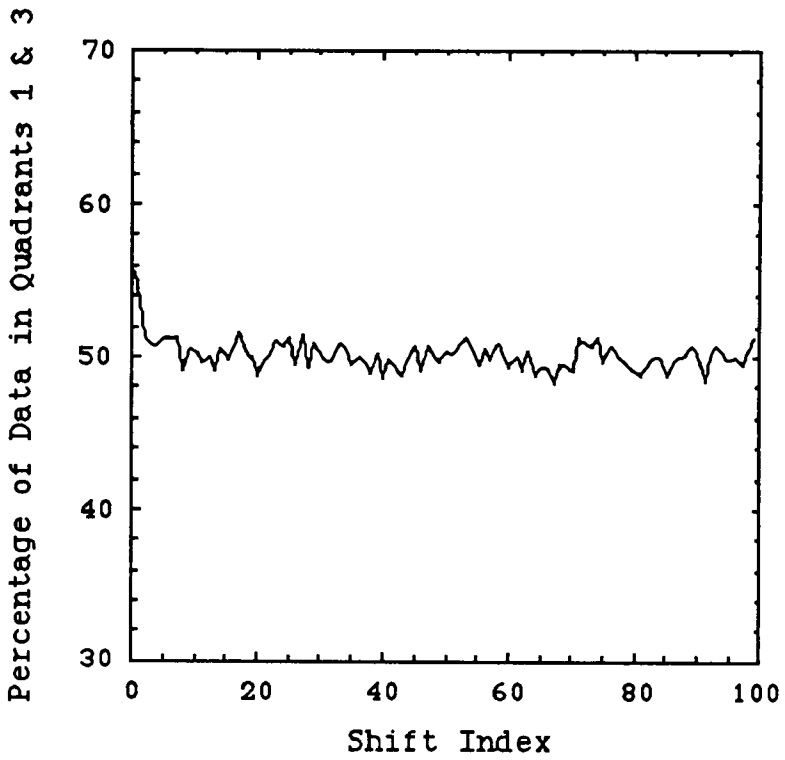


Figure 6.10: TSD Plot for the Case of an Increase in Flow Meter Noise

Table 6.6: Characteristics of the TSD Plot for the Case of an Increase in Flow Meter Noise

Crossover	Integral	Extrema	Location of Extrema
7.58	12.42	55.65	0
8.60	-0.42	49.18	8
10.47	0.63	50.56	9
13.67	-1.10	49.14	13
14.63	0.21	50.43	14
15.38	-0.09	49.75	15
19.02	2.09	51.62	17
21.56	-1.39	48.84	20
25.69	2.82	51.23	25

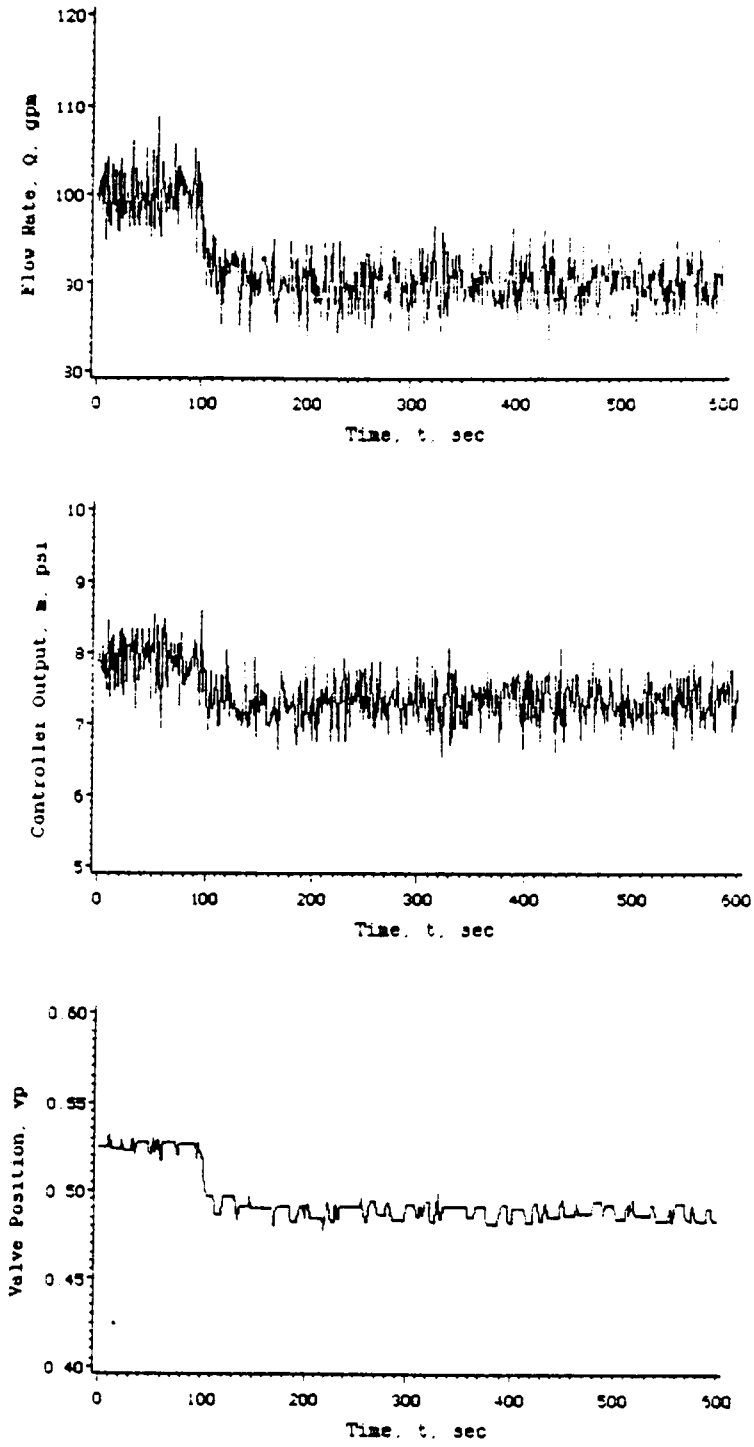


Figure 6.11: Step Test for the Case of an Increase in Flow Meter Noise

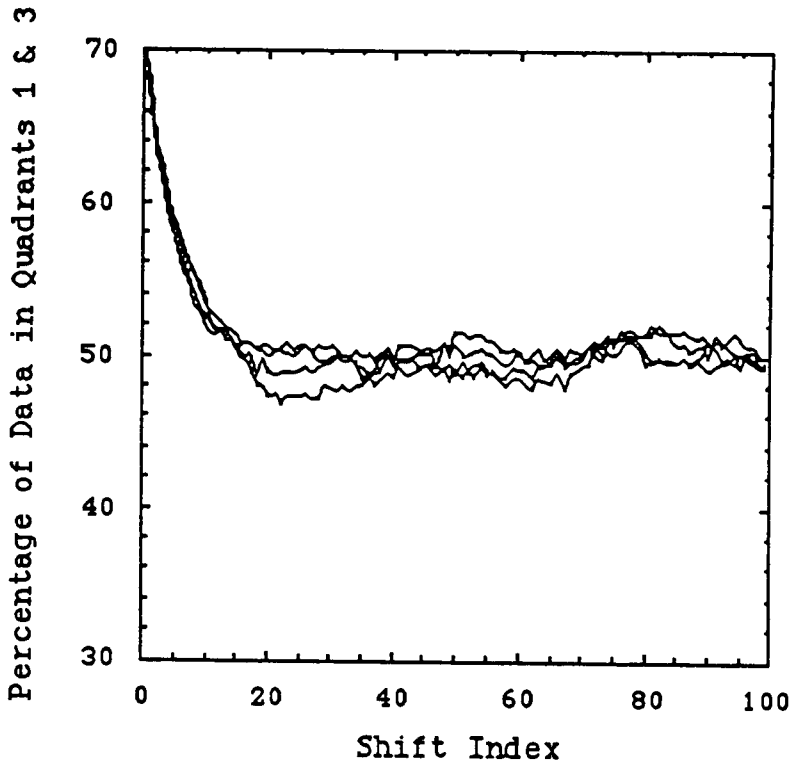


Figure 6.12: TSD Comparison for Different Disturbances

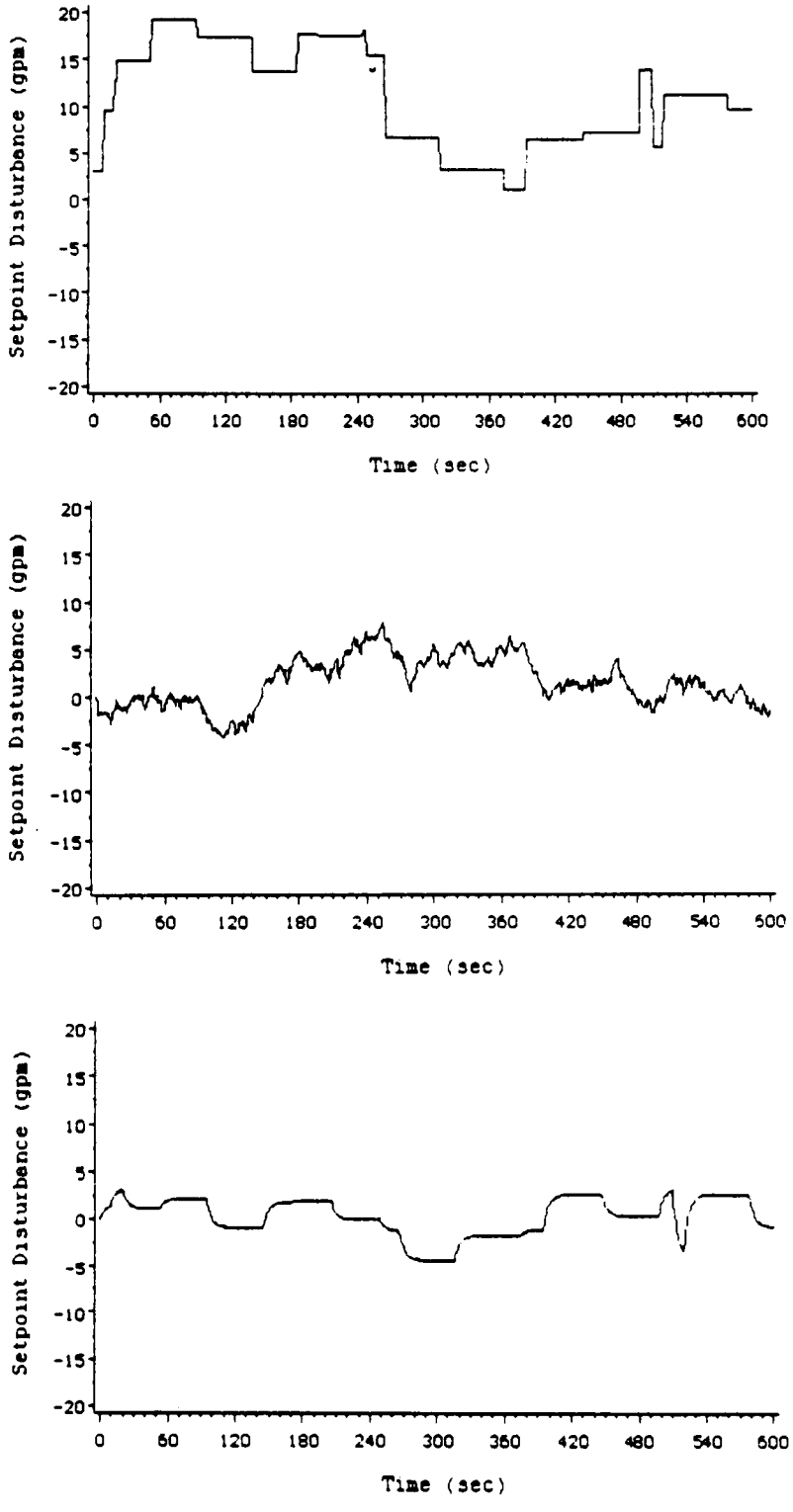


Figure 6.13: Examples of Altered Disturbances

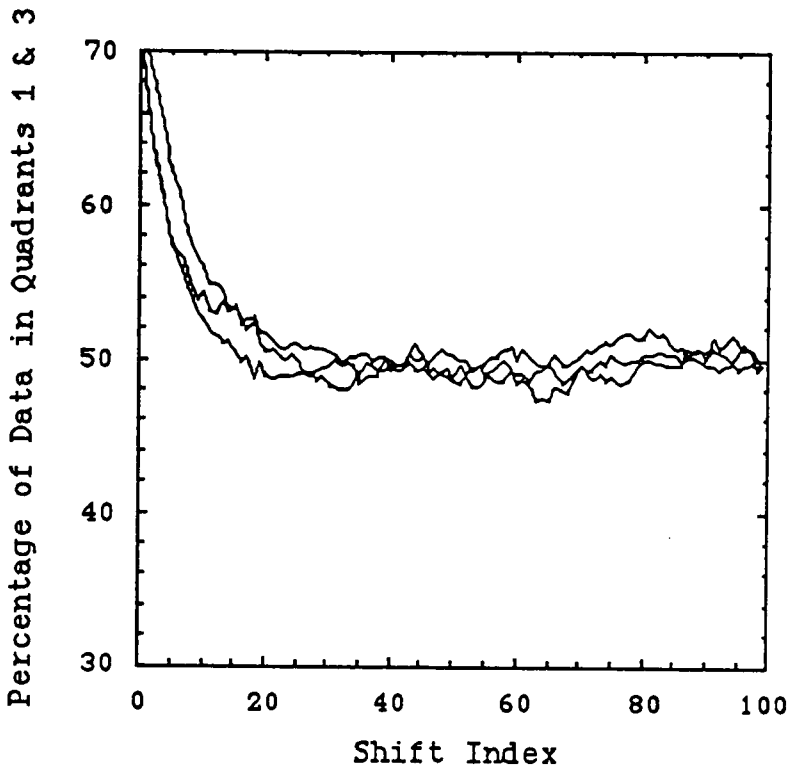


Figure 6.14: TSD Comparison for Disturbances of Significantly Different Types

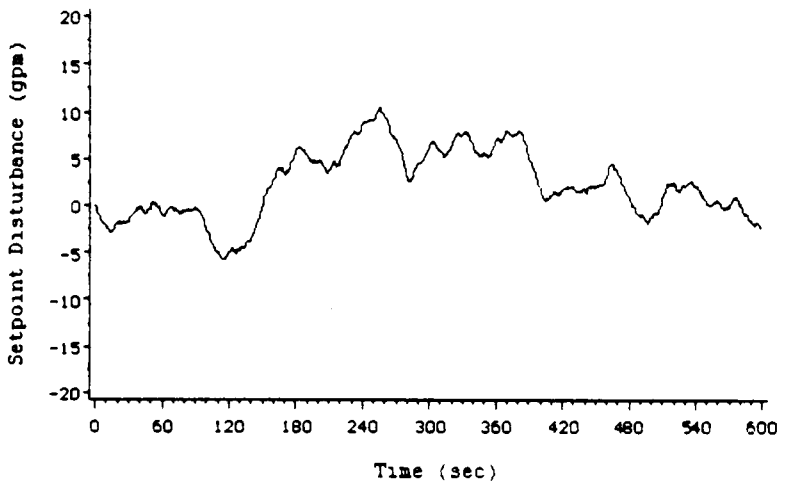


Figure 6.15: Examples of Altered Disturbances of Significantly Different Type

disturbance. Fortunately, it is not expected that a disturbance would undergo a radical change in type.

Since the decision tree branches into two parallel paths from the TSD analysis, one to detect changes in hysteresis and the other to detect gain changes, the existence of one condition should not affect the detection of the other. This is indeed the case. The analysis of hysteresis for various cases is shown in Table 6.7. Note that the maximum range of m remained at about one for all of the cases that had no change in hysteresis. The two cases that involve 10 percent hysteresis both show a maximum range of m of three, even the case that also has a plug.

Table 6.7: Results of Analysis of Hysteresis

Case	Maximum Range of m
Base	0.96
10% Hysteresis	3.08
Plugging	0.94
Plugging with 10% Hysteresis	3.00
Decrease in Fluid Density	0.98
Increase in Flow Meter Noise	0.98

There are several factors to keep in mind regarding analysis of hysteresis. First, the accuracy of the analysis is largely affected by the number of data points sorted and labelled as "undergoing hysteresis." The number of points thus sorted decreases with a decrease in closed-loop gain. The resolution of vp might be increased in order to collect more data per category, but a large resolution blurs the boundary of the deadband. Of course, one could simply collect more data to exceed a minimum number of points per category.

Table 6.8 shows that gain changes caused by plugging can be detected by a shift in the relationship between vp and $Q/\sqrt{\Delta P_o}$. Note that the shift is the same for two cases with to same valve gain but different hysteresis. This shows that the analysis is independent of the hysteresis. Table 6.9 shows the cases of a decrease in fluid density and flow meter drift.

Unfortunately, one could not distinguish the two causes of gain change (plugging and a change in density) and flow meter drift without further knowledge. This knowledge may be obtained from the cultivation analysis of an adjoining system. Suppose, for example, that cultivation of a material or energy balance suggested a problem with an incoming flow, a flow coming from the system being studied. Plugging may then be eliminated, since the control system would have

Table 6.8: Results of Analysis of Gain Change

$Q/\sqrt{\Delta P_o}$	vp(st.dev.) for: Base Case	Plugging	Change in Hysteresis	Change in Hysteresis with Plugging
32-34	0.500(0.004)	0.520(0.005)	0.500(0.005)	0.521(0.005)
34-36	0.521(0.004)	0.545(0.005)	0.522(0.004)	0.545(0.004)
36-38	0.542(0.005)	0.569(0.005)	0.542(0.005)	0.569(0.004)
38-40	0.562(0.004)	0.595(0.005)	0.562(0.005)	0.595(0.005)
40-42	0.581(0.005)	0.622(0.005)	0.582(0.004)	0.622(0.005)
42-44	0.602(0.004)	0.648(0.005)	0.601(0.004)	0.648(0.004)
Separation		3.41	0.02	3.69
Average Difference in Means		0.032	0.000	0.032
Average Difference in St. Dev.		0.001	0.000	0.000

Table 6.9: More Results of Analysis of Gain Change

$Q/\sqrt{\Delta P_o}$	vp(st.dev.) for: Base Case	Change in Density	Increase in Flow Meter Noise	Flow Meter Drift
32-34	0.500(0.004)	0.464(0.004)	0.504(0.007)	0.389(0.005)
34-36	0.521(0.004)	0.484(0.004)	0.523(0.008)	0.411(0.005)
36-38	0.542(0.005)	0.502(0.004)	0.541(0.007)	0.431(0.006)
38-40	0.562(0.004)	0.521(0.004)	0.562(0.008)	0.453(0.005)
40-42	0.581(0.005)	0.539(0.004)	0.581(0.007)	0.473(0.004)
42-44	0.602(0.004)	0.557(0.004)	0.597(0.006)	0.491(0.004)
Separation		-4.83	-0.01	-12.01
Average Difference in Means		-0.040	0.000	-0.110
Average Difference in St. Dev.		0.000	0.003	0.000

compensated to keep the flow at set point. If a change in fluid density could also be ruled out, then the problem with the flow system would be flow meter drift.

If the TSD plot shows an increase in noise, and there is no shift in the statistical plot, but the statistical plot shows an increase in standard deviation, then the problem is increased flow meter noise. This is shown by comparison of Tables 6.6 and 6.9. An increase in standard deviation alone would not be enough to diagnose increased noise, since the added variation could be caused by a more active disturbance.

In the absence of a shift in the TSD plot, the analysis of gain change may still be performed in order to detect some sensor problems. These sensors include ΔP_o and vp, which have no effect on the control action. An increase in standard deviation would indicate increased noise from ΔP_o or vp. A shift in the statistical plot with no change in the midrange, obtained by relating m versus vp, would indicate a drift in ΔP_o. A shift with the change in midrange indicates drift in vp. Table 6.10 shows data for a statistical plot, and Figure 6.16 shows the change in midrange.

6.1 Discussion of Results

Much of the results have already been discussed, but there are remaining points to explore. First, consider the method used in the analysis of gain change. The results were

Table 6.10: Analysis of Gain Change for Sensor Problems

$Q/\sqrt{\Delta P_o}$	vp(st.dev.) for: Base Case	Drift in $\sqrt{\Delta P_o}$	Drift in vp
32-34	0.500(0.004)	0.510(0.004)	0.549(0.004)
34-36	0.521(0.004)	0.531(0.004)	0.571(0.005)
36-38	0.542(0.005)	0.553(0.004)	0.592(0.005)
38-40	0.562(0.004)	0.574(0.005)	0.612(0.004)
40-42	0.581(0.005)	0.596(0.005)	0.632(0.005)
42-44	0.602(0.004)	0.618(0.005)	0.652(0.005)
Separation		1.39	5.60
Average Difference in Means		0.012	0.050
Average Difference in St. Dev.		0.000	0.000

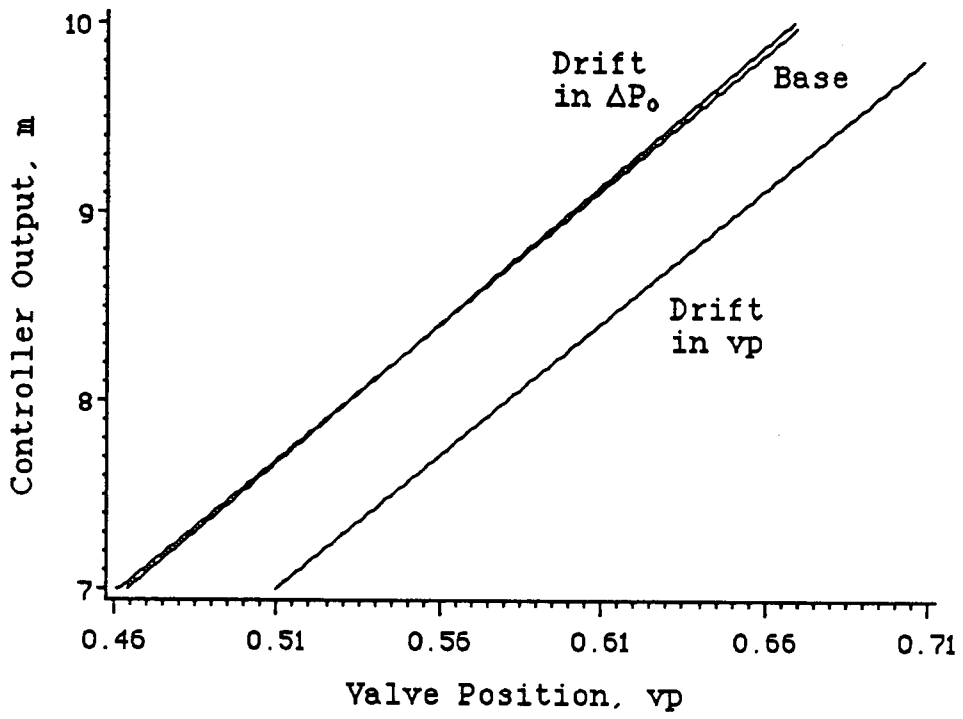


Figure 6.16: Change in Midrange Caused by Sensor Problems

compared in tabular form along with some overall statistics, such as separation and average difference in means. Given the non-linear nature of the process, the overall statistics may not be very useful. For example, the difference in means increases with valve position in the cases shown but could decrease with valve position under different operating conditions. For the same reason it does not make sense to linearly regress the means on the statistical plots. The best method to use would be pattern recognition. Farell[9] is doing work in this regard.

Next, let us revisit the discussion in Section 2.2.2 concerning the clash between conventional process control (CPC) and statistical process control (SPC). In applying cultivation analysis, the focus is taken away from the variation of variables themselves and, instead, placed on the variation of relationships between variables. In this way, the conventional control system is allowed to keep the controlled variables within engineering tolerance, while system cultivation is used to keep process relationships "in control." Cultivation is able to reveal the changes in a process that are masked by the compensating action of a control system. The control system need not be put in manual for the purpose of doing a statistical study. In fact, a change in the behavior of the control system can tip off the need for a process analysis, as shown by the use of the TSD

plot in analyzing the PI flow control loop. The TSD analysis separates normal disturbance rejection from the dynamic effects to reveal unexpected changes. The result is that one may reduce system variability (ie. variability of process relationships) and achieve a more consistent operation, since problems will be detected long before the control performance becomes unbearable enough to cause notice. The reduction of variability, after all, is a goal of SPC.

One final observation is that the analyses above have followed the philosophy of SPC without the use of control charts, run charts, histograms, etc. In keeping with the philosophy discussed in Chapter 3, the reduced data set from the base case in the form of statistical plots, TSD plots, etc., serves as the "past experience" with which "we can predict ... how the phenomenon may be expected to vary in the future." Unexpected variance from the base case indicates abnormal behavior. Traditional SPC methodology uses a control chart as a representation of the base case. System cultivation offers the added advantage of a means of sorting through various relationships to determine the cause of the abnormal behavior. Furthermore, the decision tree proposed in Chapter 5 satisfactorily provides a logical progression through those relationships to solve problems with a PI flow control loop.

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

The following conclusions may be drawn from the application of cultivation techniques (hypothesis feedback modeling, controlled resolution, and time-shifted distribution) and the philosophy of statistical process control to a PI flow control loop.

1. Cultivation techniques are useful in detecting problems or changes in a flow system, including changes in valve gain and hysteresis.
2. The cultivation technique of time-shifted distribution (TSD) analysis can be used to monitor the control performance of a PI flow control loop. Furthermore, the effect of changes in the performance of a control system can be separated from disturbances.
3. Conventional and statistical process control are compatible.
    a. By means of cultivation techniques, the philosophy of SPC may be used to monitor the quality of a control system and to detect process problems masked by the compensating action of the control system.
    b. Changes in the performance of a control system may signal process problems.

This last conclusion may seem obvious, but an operator's first inclination often is to assume that a problem resides in the control system and not the process.

Finally, the following recommendations are suggested for future work.

1. While the time-shifted distribution analysis has been applied to model predictive control and PI flow control, it would be useful to study its usefulness with more conventional, advanced control, such as feedforward and cascade control.
2. Given the movement from regulatory control through layers of control systems to plant-wide control, it would be useful to determine the utility of system cultivation in analyzing the relationships and interaction between control loops and processes.

LIST OF REFERENCES

LIST OF REFERENCES

1. Alwan, L.C., and H.V. Roberts, "Time-Series Modeling for Statistical Process Control," Journal of Business and Economic Statistics, 6, 1, January, 1988, p.87-95.

2. Box, G.E.P., and G.M. Jenkins, Time Series Analysis, Forecasting and Control, Holden-Day, San Francisco, 1970.

3. Chowdhury, J., "Quality Control Moves Upstream in CPI Plants," Chemical Engineering, 93, 8, April 28,1986, p.19-23.

4. De Nevers, N., Fluid Mechanics, Addison-Wesley, 1977.

5. Deming, W.E., Quality, Productivity, and Competitive Position, MIT Center for Advance Engineering Study, Cambridge, MA, 1982.

6. Deming, W.E., "Some Principles of the Shewhart Methods of Quality Control," Mechanical Engineering, March, 1944.

7. Deming, W.E., "What Happened in Japan," Industrial Quality Control, August, 1967.

8. Farell, A.E., "Application of Process Control Cultivation to a Liquid-Liquid Heat Exchanger," M.S. Thesis, The University of Tennessee, Knoxville, August, 1987.

9. Farell, A.E., "An Application of Neural Networks to System Cultivation Methods," a report submitted to the Measurement and Control Engineering Center, The University of Tennessee, Knoxville, 1988.

10. Hackney, J.E., "Cultivation of Model Predictive Control Systems," Ph.D. Dissertation, The University of Tennessee, Knoxville, May, 1989.

11. Hackney, J.E., personal communication, 1988.

12. Harris, T.J. and J.F. MacGregor, "Design of Multivariable Linear-Quadratic Controllers Using Transfer Functions," AICHE Journal, 33, 9, September, 1987, p.1481-1495.

13. Himmelblau, D.M., Fault Detection and Diagnosis in Chemical and Petrochemical Processes, Elsevier Scientific Publishing Company, Amsterdam, 1978.

14. Himmelblau, D.M., Process Analysis by Statistical Methods, John Wiley and Sons, Inc., New York, 1970.

15. Hunter, J.S., "The Exponentially Weighted Moving Average," Journal of Quality Technology, 18, 4, October, 1986.

16. MacGregor, J.F., "On-Line Statistical Process Control," Chemical Engineering Progress, 84, 10, October, 1988, p.21-31.

17. Moore, B.C., "An Academic Spy in the Real World of Industrial Computer Control: A Candid Report," Technical Report No. 8602, 'T Lab, P.O. Box 427, Plaquemine, LA 70765, March,1986.

18. Moore, B.C., "Analysis of Process Dynamics using Computer Memory and Controlled Resolution," 1988 American Control Conference, March 8, 1984.

19. Moore, B.C., "Hypothesis Feedback Models for Multivariable Systems," The Dow Chemical Company, Plaquemine, LA, October, 1986. (Submitted to IEEE Transactions on Automatic Control).

20. Moore, B.C., "Multivariable Challenges in the Petrochemical Industry," Technical Report No. 8601, 'T Lab, P.O. Box 427, Plaquemine, LA 70765, February, 1986.

21. Moore, B.C., "Systematic Attention to Detail by Production Organizations for Sustained Improvement," The Dow Chemical Company, Plaquemine, LA, October 1987. (Presented at the Second National Symposium on Statistical Process Control, Tempe, AZ, November 9-11, 1987).

22. Moore, C.F., "Multivariable Framework for Statistical Quality Control," The University of Tennessee, Knoxville, 1986.

23. Moore, C.F., "Process and Control System Cultivation: A New Direction in Process Control Research," February, 1987, Seminar presented at The University of Tennessee, Knoxville.

24. Moore, C.F., "What and Who is in control? A Process Control Perspective on Statistical Process Control," The Shell Process Control Workshop II, December, 1988.

25. Oakland, J.S., Statistical Process Control, John Wiley and Sons, New York, 1986.

26. Roffel, B. and J.E. Rijnsdorp, Process Dynamics, Control and Protection, Ann Arbor Science, 1982.

27. Shewhart, W.A., Economic Control of Quality of Manufactured Product, D. Van Nostrand Co., Inc., New York, 1931. (Republished in 1981 by the American Society for Quality Control)

28. Shinskey, F.G., Process Control Systems, 2nd ed., McGraw Hill, 1979.

29. Smith, C.A., and A.B. Corripio, Principles and Practice of Automatic Process Control, John Wiley and Sons, 1985.

30. Stephanopoulos, G., Chemical Process Control: An Introduction to Theory and Practice, Prentice-Hall, 1984.

31. _____ , Fluid Meters: Their Theory and Application, 5th ed., American Society of Mechanical Engineers, New York, 1959.

APPENDIX

```
C*******************************************************************************
C
C       PROGRAM NAME:  FLOW
C       PROGRAMMER:  Mark Bendele
C       PURPOSE:  This program simulates a flow control loop.  The system
C                 consists of a length of pipe (including fittings in
C                 equivalent length of pipe), an orifice flowmeter, a
C                 control valve, and a PI-controller.
C
C*******************************************************************************
C
C       LIST OF PARAMETERS
C
C       ALPHA         = rangeability parameter of equal-percentage valve
C       ATIME_SP      = time of next change in disturbance (set point)
C       ATIME_DPO     = time of next change in disturbance (pressure drop)
C       BETA          = orifice diameter/pipe diameter
C       BOUND_LO_DPO  = low limit of pressure drop disturbance
C       BOUND_HI_DPO  = high limit of pressure drop disturbance
C       BOUND_LO_SP   = low limit of set point disturbance
C       BOUND_HI_SP   = high limit of set point disturbance
C       CO            = orifice coefficient
C       CU            = flow coefficient of an addition flow component
C       CV            = valve coefficient
C       D             = pipe diameter
C       D_SP          = set point disturbance
C       D_SP1         = old set point disturbance
C       D_DPO         = pressure drop disturbance
C       D_DPO1        = old pressure drop disturbance
C       DENS          = fluid density
C       DP            = pressure drop across * (O=overall,M=flowmeter,P=pipe,
C                                               T=dp transmitter on flowmeter,
C                                               U=additional unit)
C       DT            = time interval, "delta t"
C       DTIME_SP      = maximum time between disturbances (set point)
C       DTIME_DPO     = max time between disturbances (pressure drop)
C       DQ            = incremental step in Q during incremental search
C       IE            = integral error
C       ISEED         = seed for random number generator
C       E             = error
C       E1            = previous error
C       EPSILON       = convergence criterion
C       F             = Fanning friction factor
C       FITTINGS_L    = equivalent length of pipe fittings
C       GEOM          = geometric factor for orifice meter calculation
C       H_PERCENT     = % hysteresis
C       H             = hysteresis in terms of controller output
C       K_DPO         = gain for step or exponential disturbance
C       K_SP          = gain for step or exponential disturbance
C       KC            = controller proportional gain
C       KV            = gain of control valve
C       LL            = lower limit of controller output
C       M             = controller output after hysteresis
C       M1            = controller output of previous time step after hysteresis
C       M             = controller output
C       PHI           = IAR model parameter
C       PI            = 3.14159
C       PIPE_L        = pipe length
C       Q             = flowrate
C       QMEAS         = flowrate measured by flowmeter
C       RE            = Reynold's number
C       ROUGH         = surface roughness of inside pipe wall
```

```
C       RR              = relative roughness = ROUGH/D
C       SIGMA_DPO       = std dev of random no. (press. drop disturbance)
C       SIGMA_SP        = std dev of random no. (set point disturbance)
C       SIGMA_Q         = std dev of random no. (flow meter noise)
C       SP              =
C       T               = time, seconds
C       THETA_DPO       = (IMA model parameter)
C       THETA_SP        = (IMA model parameter)
C       TI              = controller integral time constant
C       TIME_TO_SAMPLE  = time at which controller will take action
C       TOTAL_L         = pipe length plus equivalent length of fittings
C       TSAMP           = controller sampling time
C       TV              = time constant of control valve
C       TYPE_DPO        = disturbance type of overall pressure drop
C       TYPE_SP         = disturbance type of set point disturbance
C       UL              = upper limit of controller output
C       VISC            = fluid viscosity
C       VP              = valve position
C       VP1             = valve position of previous time step
C       Alphanumeric:
C       VALVCHAR        = inherent valve characteristic
C       Logical:
C       LINEAR          = indicates a linear valve characterisic when .TRUE.
C       EQPERC          = indicates an equal-percentage valve when .TRUE.
C       INCRMNT         = when .TRUE., numerical method is incremental search
C       BISECT          = when .TRUE., numerical method is bisection
C       FLAG            = logical flag
C
C       CONVERSION FACTORS
C          -- multiply by *_TO_* to convert from * => *
C       CFPS_TO_GPM         => cubic feet per sec to gallons per min
C       CP_TO_LBM_PER_FT_S  => centipoise to lb-mass per foot sec
C       GC                  => "G sub C", F=ma/GC
C       FT_TO_IN            => feet to inches
C       SQFT_TO_SQIN        => square feet to square inches
C
C*******************************************************************************

        CHARACTER*1 VALVCHAR, TYPE_SP, TYPE_DPO
        CHARACTER*80 INFILE, OUTFILE
        REAL IE, KC, KV, LL, M, M1, K, MR, K_SP, K_DPO
        LOGICAL BISECT, EQPERC, INCRMNT, LINEAR, FLAG

        TYPE *,' ENTER INPUT FILENAME'
        ACCEPT 10030, INFILE
        TYPE *,' ENTER OUTPUT FILENAME'
        ACCEPT 10030, OUTFILE
10030   FORMAT(A80)
        OPEN(UNIT=1,STATUS='OLD',NAME=INFILE)
        OPEN(UNIT=10,STATUS='NEW',NAME=OUTFILE,FORM='UNFORMATTED')

C   INPUT

        READ(1,10010)VALVCHAR
10010   FORMAT(12X,A1)
        IF(VALVCHAR.EQ.'E' .OR. VALVCHAR.EQ.'e') THEN
             LINEAR = .FALSE.
             EQPERC = .TRUE.
        ELSE IF(VALVCHAR.EQ.'L' .OR. VALVCHAR.EQ.'l') THEN
             LINEAR = .TRUE.
             EQPERC = .FALSE.
        ELSE
```

```
            TYPE *,'UNKNOWN VALVE CHARACTERISTIC'
            STOP
          END IF
          READ(1,10020) ALPHA,
         2              CV,H_PERCENT,KV,TV,
         2              KC,TI,TSAMP,QSP,LL,UL,
         2              DPO,
         2              BETA,CO,HQ,
         2              DENS,VISC,
         2              D,PIPE_L,FITTINGS_L,ROUGH,
         2              CU,
         2              DQ,DT,EPSILON,TSTP,
         2              TYPE_SP,SIGMA_SP,PHI_SP,THETA_SP,K_SP,
         2              ATIME_SP,DTIME_SP,BOUND_LO_SP,BOUND_HI_SP,
         2              TYPE_DPO,SIGMA_DPO,PHI_DPO,THETA_DPO,K_DPO,
         2              ATIME_DPO,DTIME_DPO,BOUND_LO_DPO,BOUND_HI_DPO,
         2              SIGMA_Q,
         2              ISEED
10020    FORMAT(26(12X,F14.9,/),12X,A1,/,8(12X,F14.9,/),12X,A1,/,
         2           9(12X,F14.9,/),12X,I11)

C    INITIALIZE

          IF (ISEED.GT.0) ISEED = -ISEED

          CFPS_TO_GPM = 448.83
          CP_TO_LBM_PER_FT_S = 6.7197E-04
          FT_TO_IN = 12.
          GC = 32.174
          PI = 3.14159
          SQFT_TO_SQIN = 144.

          CV = CV/SQFT_TO_SQIN
          CU = CU/SQFT_TO_SQIN
          D = D/FT_TO_IN
          DQ = DQ/CFPS_TO_GPM
          GEOM = PI*D*D*BETA*BETA/4./SQRT(1.-BETA**4)
          RR = ROUGH/D
          VISC = VISC*CP_TO_LBM_PER_FT_S
          TOTAL_L = PIPE_L + FITTINGS_L

          Q = QSP/CFPS_TO_GPM
          RE = 4.*Q*DENS/(PI*VISC*D)
          F = 0.00138*(1.+(2.E+04*RR+1.E+06/RE)**(1./3.))     ! Moody Equation
          DPP = 6.998141E-04*F*TOTAL_L*DENS*Q*Q/D**5
          DPT = DENS*Q*Q/(2.*CO*CO*GEOM*GEOM)/GC/SQFT_TO_SQIN
          DPM = DPT*(1-BETA*BETA)
          DPU = DENS*Q*Q/(CU*CU)/GC/SQFT_TO_SQIN
          DPV = DPO - DPM - DPP - DPU
          VP = Q/CV/SQRT(DPV*GC*SQFT_TO_SQIN/DENS)
          IF (EQPERC) VP = 1. + ALOG(VP)/ALOG(ALPHA)
          VP1 = VP
          M = VP/KV
          IF (M.LT.LL .OR. M.GT.UL) THEN
               WRITE(6,20010)
20010     FORMAT(' LIMITS ON CONTROLLER OUTPUT EXCEEDED DURING ',
         2           'INITIALIZATION')
               STOP
          END IF
          M1 = M
          MR = 8.433434
          E = 0.
```

```
          E1 = 0.
          IE = TI/KC*(M-MR)
          JDIR = 0
          LDIR = 0
          H = H_PERCENT/100.
          T = 0.
          TIME_TO_SAMPLE = 0.
          Q = Q*CFPS_TO_GPM
          QMEAS = Q
          SP = QSP
          DPO1 = DPO
          D_SP = 0.
          D_SP1 = 0.
          Q_1 = Q

C   PI-CONTROLLER
C      -- using the velocity equation

200       IF(T.GE.TIME_TO_SAMPLE) THEN
               WRITE(10,ERR=999) T,QMEAS,E,IE,M,VP,SQRT(DPO),QSP
               CALL DISTURBANCE (T,D_SP,TYPE_SP,SIGMA_SP,PHI_SP,
          2                      THETA_SP,K_SP,ISEED,A_SP,D_SP1,
          2                      ATIME_SP,DTIME_SP)
               CALL BOUNDS (D_SP,D_SP1,BOUND_LO_SP,BOUND_HI_SP)
               QSP = SP + D_SP
               E1 = E
               E = QSP - QMEAS
               IE1 = IE
               IE = IE + E*TSAMP
               IF(M.EQ.UL .AND. IE.GT.IE1) IE=IE1
               IF(M.EQ.LL .AND. IE.LT.IE1) IE=IE1
               M = MR + KC*E + KC*IE/TI
               IF(M.GT.UL) M = UL
               IF(M.LT.LL) M = LL
               TIME_TO_SAMPLE = TIME_TO_SAMPLE + TSAMP
          END IF

C   VALVE DYNAMICS
C      -- first-order model

          A = EXP(-DT/TV)
          VP = KV*(1.-A)*M1 + A*VP1
          M1 = M

C   HYSTERESIS

          IF(VP-VP1.LT.0) THEN
               IDIR = -1
          ELSE IF(VP-VP1.EQ.0) THEN
               IDIR = 0
          ELSE
               IDIR = 1
          END IF

          IF(IDIR.NE.0 .AND. IDIR.EQ.JDIR) THEN
               VP1 = VP
          ELSE IF(ABS(M*KV-VP1).GE.H) THEN
               VP1 = VP
               JDIR = IDIR
          ELSE
               VP = VP1
               JDIR = 0
```

```
        END IF

C   DISTURBANCE
C      -- create a disturbance in the overall pressure drop across the flow
C         system

        CALL DISTURBANCE (T,DPO,TYPE_DPO,SIGMA_DPO,PHI_DPO,
     2                    THETA_DPO,K_DPO,ISEED,A_DPO,DPO1,
     2                    ATIME_DPO,DTIME_DPO)
        CALL BOUNDS (DPO,DPO1,BOUND_LO_DPO,BOUND_HI_DPO)

C   PROCESS
C      This section calculates the flowrate through a pipe fitted with an
C      orifice meter and a control valve, given valve position.

C   If the valve is linear and closed, call the flowrate zero
C   and skip this section.

100     IF(LINEAR .AND. VP.EQ.0.) THEN
            DPM = 0.
            DPP = 0.
            DPU = 0.
            DPV = DPO
            Q = 0.
            GOTO 20
        END IF

C   Use incremental search and bisection to converge on the
C   the overall pressure drop.  Guess Q to make DPP+DPM+DPV+DPU=DPO.

        INCRMNT = .TRUE.
        BISECT = .FALSE.
        Q = Q/CFPS_TO_GPM
        Q_OLD = Q
        Q1 = -1.
        Q2 = -1.
40      IF(BISECT) Q = (Q1+Q2)/2.
        RE = 4.*Q*DENS/(PI*VISC*D)
        IF(RE.GT.0.) THEN
            F = 0.00138*(1.+(2.E+04*RR+1.E+06/RE)**(1./3.)) !Moody Equation
        ELSE
            F = 0.
        END IF
        DPP = 6.998141E-04*F*TOTAL_L*DENS*Q*Q/D**5
        DPT = DENS*Q*Q/(2.*CO*CO*GEOM*GEOM)/GC/SQFT_TO_SQIN
        DPM = DPT*(1-BETA*BETA)
        DPU = DENS*Q*Q/(CU*CU)/GC/SQFT_TO_SQIN
        IF(EQPERC) DPV = DENS*Q*Q/(CV*CV*ALPHA**(2.*VP-2.))/GC/SQFT_TO_SQIN
        IF(LINEAR) DPV = DENS*Q*Q/(CV*CV*VP*VP)/GC/SQFT_TO_SQIN
        DPA = 4.*DENS*PIPE_L*(Q-Q_OLD)/(PI*D*D*DT)/GC/SQFT_TO_SQIN
        FDP = DPP+DPM+DPV+DPU+DPA-DPO
        IF(ABS(FDP).LE.EPSILON) GOTO 20
        IF(INCRMNT) THEN
            IF(FDP.LT.0) THEN
                FDP1 = FDP
                Q1 = Q
                Q = Q + DQ
            ELSE
                FDP2 = FDP
                Q2 = Q
                Q = Q - DQ
            END IF
```

```
                IF(Q1.GT.0. .AND. Q2.GT.0.) THEN
                        INCRMNT = .FALSE.
                        BISECT = .TRUE.
                END IF
                GOTO 40
           END IF
           IF(FDP*FDP1) 10,20,30
10         Q2 = Q
           FDP2 = FDP
           GOTO 40
30         Q1 = Q
           FDP1 = FDP
           GOTO 40
20         Q = Q*CFPS_TO_GPM
           CONTINUE

C    Hysteresis on the flowmeter
C       -- serves no useful purpose

           IF (HQ.GT.0.) THEN
                IF(Q-Q_1.LT.0) THEN
                        KDIR = -1
                ELSE IF(Q-Q_1.EQ.0) THEN
                        KDIR = 0
                ELSE
                        KDIR = 1
                END IF
                IF(KDIR.NE.0 .AND. KDIR.EQ.LDIR) THEN
                        Q_1 = Q
                        FLAG = .TRUE.
                ELSE IF(ABS(Q-Q_1).GE.HQ) THEN
                        Q_1 = Q
                        LDIR = KDIR
                        FLAG = .TRUE.
                ELSE
                        LDIR = 0
                        FLAG = .FALSE.
                END IF
           ELSE
                FLAG = .TRUE.
           END IF

C    This is the square root extractor, which converts the pressure
C    drop across the orifice meter to flowrate for use by the
C    controller.

           IF(FLAG) THEN
              QMEAS = 0.61*GEOM*SQRT(2.*DPT/DENS*GC*SQFT_TO_SQIN)*CFPS_TO_GPM
              Q_NOISE = SIGMA_Q*GAUSS(ISEED)
              QMEAS = QMEAS + Q_NOISE
           END IF

C    TERMINATION

           T = T + DT
           IF(T.GE.TSTP) STOP
           GOTO 200

999        TYPE *,' ERROR DURING WRITE'
           STOP

           END
```

```
C*******************************************************************************
C
C       PROGRAM NAME:  CULT.FOR
C       PROGRAMMER:  MARK J. BENDELE
C       DATE :  1 MARCH 1988
C       PURPOSE:  TO PERFORM SYSTEM CULTIVATION ACCORDING TO THE
C                 INSTRUCTIONS GIVEN IN A PARAMETER FILE
C                 PROGRAM, CULT_DESIGN.FOR
C       NOTE:  THIS MAIN PROGRAM ONLY ACQUIRES VIRTUAL MEMORY.  SEE
C              SUBROUTINE CULTIVATION FOR DOCUMENTATION.
C       SUBROUTINES:  ASCII_CHECK, CULTIVATION
C
C*******************************************************************************

          CHARACTER*1 CHECK
          CHARACTER*2 CHECK2
          CHARACTER*50 PARFILE, INFILE, OUTFILE, DUMMY
          character*400       aline
          INTEGER STATUS,
        2         VM_SIZE(16),
        2         VM_ADDR(16),
        2         VM_DESC(2),
        2         LIB$GET_VM,
        2         LIB$FREE_VM

          logical aflag, flag

C  Read name of input data file and number of varibles in a record
C  from the parameter file.

          OPEN(UNIT=1,STATUS='OLD',FILE='PARFILE.CLT',READONLY,IOSTAT=IOS)
          IF (IOS.GT.0) THEN
               PARFILE = ' '
          ELSE
               READ(1,10010) PARFILE
10010   FORMAT(1X,A50)
               CLOSE(UNIT=1)
          END IF
          TYPE *,' Enter <CR> to keep old parameter file'
          WRITE(*,20010) PARFILE
20010   FORMAT(1X,'Parameter file is ',a50)
30        WRITE(*,20020)
20020   FORMAT (1X,'Enter parameter file:  ',$)
          READ (*,10020) DUMMY
10020   FORMAT (a50)
          IF (DUMMY(1:1).NE.' ') PARFILE = DUMMY
          OPEN(UNIT=10,STATUS='OLD',FILE=PARFILE,DEFAULTFILE='.PAR',READONLY,
        2         IOSTAT=IOS)
          IF (IOS.GT.0) THEN
               GOTO 30
          ELSE
               OPEN(UNIT=2,STATUS='NEW',FILE='PARFILE.CLT')
               WRITE(2,20030) PARFILE
20030          FORMAT(1X,A50)
               CLOSE(UNIT=2)
          END IF
          READ(10,10010) INFILE, OUTFILE
          READ(10,*) NVAR
          OPEN (UNIT=12,FILE=INFILE,DEFAULTFILE='.DAT',
        2         STATUS='OLD',READONLY)
          call ascii_check(12,aflag)
          if (.not. aflag) then
```

```
              close(12)
              open (unit=12,file=infile,defaultfile='.dat',
     2             status='old',form='unformatted',
     2         readonly)
            end if

C  Determine length (number of records) of data file.

            NREC = 0
            if (aflag) then
                 TYPE *,' ASCII data file used.'
10               READ(12,502,END=20) CHECK2
                 IF (CHECK2(1:1).EQ.' ' .OR.
     2               CHECK2(1:1).EQ.'      ') THEN
                     IF (CHECK2(2:2).EQ.'T' .OR.
     2                   CHECK2(2:2).EQ.'t' .OR.
     2                   CHECK2(2:2).EQ.'D' .OR.
     2                   CHECK2(2:2).EQ.'d') THEN
                         GOTO 10
                     ELSE
                         NREC = NREC+1
                     END IF
                 END IF
                 GOTO 10
            else
                 type *,' BINARY data file used.'
                 nrec   = 0
                 flag   = .true.
                 do while(flag)
                     read(12,iostat=status,end=20) aline
                     if (aline(1:5).ne.' time' .and.
     2                   aline(1:5).ne.' TIME' ) then
                            nrec   = nrec+1
                     end if
                 end do
            end if

20          REWIND 12
            WRITE(6,510) NREC
500         FORMAT(A1)
502         FORMAT(A2)
510         FORMAT(1X,I10,1X,'RECORDS IN DATA FILE')

C  Determine sizes of blocks of virtual memory required.

            VM_SIZE(1) = NVAR*NREC*4
            VM_SIZE(2) = NVAR*NREC*4
            VM_SIZE(3) = NREC*4
            VM_SIZE(4) = NREC*4
            VM_SIZE(5) = NREC*4
            VM_SIZE(6) = NVAR*4
            VM_SIZE(7) = NVAR*4
            VM_SIZE(8) = NVAR*4
            VM_SIZE(9) = NVAR*4
            VM_SIZE(10) = NVAR*4
            VM_SIZE(11) = NVAR*4
            VM_SIZE(12) = NVAR*24
            VM_SIZE(13) = NREC*4
            VM_SIZE(14) = NREC*4
            VM_SIZE(15) = NREC*4
            VM_SIZE(16) = NVAR*4
```

```
C  Acquire virtual memory.

         DO I=1,16
              STATUS = LIB$GET_VM (VM_SIZE(I),VM_ADDR(I))
              IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))
         END DO
         VM_DESC(1) = VM_SIZE(12)
         VM_DESC(2) = VM_ADDR(12)

C  Start cultivating.

         CALL CULTIVATION ( %VAL(VM_ADDR(1)),
        2                   %VAL(VM_ADDR(2)),
        2                   %VAL(VM_ADDR(3)),
        2                   %VAL(VM_ADDR(4)),
        2                   %VAL(VM_ADDR(5)),
        2                   %VAL(VM_ADDR(6)),
        2                   %VAL(VM_ADDR(7)),
        2                   %VAL(VM_ADDR(8)),
        2                   %VAL(VM_ADDR(9)),
        2                   %VAL(VM_ADDR(10)),
        2                   %VAL(VM_ADDR(11)),
        2                   VM_DESC,
        2                   %VAL(VM_ADDR(13)),
        2                   %VAL(VM_ADDR(14)),
        2                   %VAL(VM_ADDR(15)),
        2                   %VAL(VM_ADDR(16)),
        2                   NREC,
        2                   NVAR,
        2                   OUTFILE,
        2                   AFLAG )

         STOP
         END
```

```
C*************************************************************************
C
C       SUBROUTINE NAME:  CULTIVATION
C       PROGRAMMER:  MARK J. BENDELE
C       DATE:  1 MARCH 1988
C       PURPOSE:  Perform cultivation analysis using sorting operations and
C                 controlled resolution.  Sorting is performed according to
C                 the hypothesis feedback model given in a parameter file.
C
C       VARIABLES:
C           AFLAG       = logical indicating whether data file is in ASCII
C                         or binary form
C           BEGIN       = vector containing locations of beginning of
C                         collections (boxes)
C           BOX_PLOT    = logical indicating whether box plots are desired
C           ICOUNT      = counter
C           COUNT       = vector containing number of data points
C                         (observations) in a collection (box)
C           GRID_FILE   = logical indicating whether it is desirable to
C                         output data (containing real values of independent
C                         variables) to a file
C           IDATA       = array containing integer data of desired resolution
C                         converted from real plant data
C           INTERPOLATE = logical indicating whether interpolation is desired
C           IX          = vector containing integer values of independent
C                         variables in collection (box) being examined
C           MEAN        = mean
C           MINSV       = minimum singular value used for SVD in interpolation
C                         routine
C           MINREC      = minimum number of records needed to justify the
C                         existence of a collection (box)
C           NAME        = vector containing names of the variables
C           NBOX        = number of boxes (collections)
C           NCASE       = number of cases
C           NCOUNT      = a count
C           NDV         = number of dependent variables
C           NIV         = number of independent variables
C           NREC        = number of records in plant data
C           NVAR        = number of variables
C           OUTFILE     = name of output file without extension
C           POINT       = index of data points in a collection (box) that are
C                         not bad data (marked as -1)
C           RANGE       = number that determines resolution of integer data,
C                         e.g. RANGE = 31 means 8-bit resolution.  RANGE may
C                         be any number greater than zero (0)
C           RELATIONSHIP_PLOT = logical indicating whether relationship plots
C                               are desired
C           RDATA       = array containing real plant data
C           SCALE       = vector of scaling factors for each variable used to
C                         convert from real data to integer data of desired
C                         resolution
C           SENSOR_STATS = logical indicating whether sensor statistics are
C                          desired
C           SPAN        = vector containing span of each variable from lowest
C                         value to highest value
C           STDEV       = standard deviation
C           VM_ADDR     = virtual memory address
C           VM_SIZE     = virtual memory size
C           WK          = work vector
C           XKEY        = index of independent variables
C           Y           = vector containing real values of the dependent
C                         variable in a collection
```

```
C          YKEY          = index of dependent variables
C          ZERO          = vector containing lowest bound of each variable
C                          used in converting data from real to integer
C
C       SUBROUTINES:  CHAR_STRING_LENGTH
C                     COLLECT
C                     INTERP
C                     REGRESS
C
C*******************************************************************************

        SUBROUTINE CULTIVATION ( RDATA,
     2                           IDATA,
     2                           INDEX,
     2                           BEGIN,
     2                           COUNT,
     2                           ZERO,
     2                           SPAN,
     2                           RANGE,
     2                           SCALE,
     2                           XKEY,
     2                           YKEY,
     2                           NAME,
     2                           WK,
     2                           Y,
     2                           POINT,
     2                           IX,
     2                           NREC,
     2                           NVAR,
     2                           OUTFILE,
     2                           AFLAG )

        REAL*4 RDATA(NREC,NVAR),
     2         ZERO(NVAR),
     2         SPAN(NVAR),
     2         RANGE(NVAR),
     2         SCALE(NVAR),
     2         WK(NREC),
     2         Y(NREC),
     2         MEAN,
     2         MINSV

        INTEGER*4 IDATA(NREC,NVAR),
     2            INDEX(NREC),
     2            BEGIN(NREC),
     2            COUNT(NREC),
     2            XKEY(NVAR),
     2            YKEY(NVAR),
     2            POINT(NREC),
     2            IX(NVAR),
     2            VM_SIZE(4),
     2            VM_ADDR(4),
     2            LIB$GET_VM,
     2            LIB$FREE_VM,
     2            STATUS

        CHARACTER*24 NAME(NVAR)
        CHARACTER*50 OUTFILE
        CHARACTER*400 ALINE
        CHARACTER*103 GRID_NAME
        CHARACTER*1 NUMBER_1
        CHARACTER*2 NUMBER_2
```

```
      LOGICAL INTERPOLATE,
     2        SENSOR_STATS,
     2        BOX_PLOT,
     2        GRID_FILE,
     2        RELATIONSHIP_PLOT,
     2        IFLAG,
     2        PERUSE,
     2        AFLAG

C  Read in parameters from parameter file.   Read plant data to be
C  cultivated, and store in the array RDATA(i,j).  Note that a record is
C  referenced by i, and a specific variable can be referenced by j.  Units
C  10 and 12 were opened in the main program.

      READ(10,10010) (NAME(I),ZERO(I),SPAN(I),RANGE(I),I=1,NVAR)
10010 FORMAT(1X,A20,5X,G15.7,5X,G15.7,5X,F3.0)
      READ(10,*) MINREC,
     2           INTERPOLATE,SENSOR_STATS,BOX_PLOT,GRID_FILE,
     2           NCASE
      IF (MINREC.LT.1) MINREC = 1
      IFLAG = .TRUE.
      RELATIONSHIP_PLOT = .FALSE.
      IF(BOX_PLOT) OPEN(UNIT=101,STATUS='NEW',FILE=OUTFILE,
     2                  DEFAULTFILE='.BP')

      if (aflag) then
           PERUSE        = .TRUE.
           I = 1
           DO WHILE (PERUSE)
              READ(12,90010,END=12) ALINE
90010       FORMAT(A400)
              IF (ALINE(1:1).EQ.' ' .OR. ALINE(1:1).EQ.'      ') THEN
                 IF (ALINE(2:2).NE.'D' .AND. ALINE(2:2).NE.'d' .AND.
     2               ALINE(2:2).NE.'T' .AND. ALINE(2:2).NE.'t') THEN
                     READ(ALINE,*) (RDATA(I,J),J=1,NVAR)
                     I = I+1
                 END IF
              END IF
           END DO

      else
           read (12,iostat=status) aline(1:2)
           IF (ALINE(2:2).NE.'D' .AND. ALINE(2:2).NE.'d' .AND.
     2         ALINE(2:2).NE.'T' .AND. ALINE(2:2).NE.'t') THEN
                rewind(12)
           end if
           do i=1,nrec
                read(12,end=12) (rdata(i,j),j=1,nvar)
           end do
      end if

12    CONTINUE

C  Convert real data to integer data, reducing to the desired resolution.
C  Data out of bounds or unusable is marked as -1.  IX and IBAD being used
C  to count bad data.

      DO I=1,NVAR
           SCALE(I) = RANGE(I)/SPAN(I)
      END DO
```

```
          IBAD = 0
          DO I=1,NVAR
               IX(I) = 0
          END DO
          DO IREC=1,NREC
               DO IVAR=1,NVAR
                  IF(RDATA(IREC,IVAR).GE.ZERO(IVAR) .AND.
        2          RDATA(IREC,IVAR).LE.ZERO(IVAR)+SPAN(IVAR)) THEN
                  IDATA(IREC,IVAR)=(RDATA(IREC,IVAR)-ZERO(IVAR))*SCALE(IVAR)
                 ELSE
                    IDATA(IREC,IVAR) = -1
                    IBAD = IBAD + 1
                     IX(IVAR) = IX(IVAR) + 1
                  END IF
               END DO
          END DO
          WRITE(6,600) IBAD, NREC*NVAR
          WRITE(6,601) (IX(I),I=1,NVAR)
600       FORMAT(1X,I7,1X,'OF',1X,I7,1X,'POINTS ARE OUT OF BOUNDS')
601       FORMAT(1X,'BAD COUNT FOR EACH VARIABLE BEGINNING WITH THE FIRST:',/,
        2         1X,<NVAR>(I7,2X))

C  Compute sensor statistics (mean and standard deviation) and write
C  to a file.

          IF(SENSOR_STATS) THEN
             OPEN(UNIT=100,STATUS='NEW',FILE=OUTFILE,DEFAULTFILE='.SS')
             DO J=1,NVAR
                ICOUNT = 0
                DO I=1,NREC
                   IF(IDATA(I,J).GE.0) THEN
                      ICOUNT = ICOUNT + 1
                      WK(ICOUNT) = RDATA(I,J)
                   END IF
                END DO
                CALL STATS(%VAL(%LOC(WK(1))),NREC,ICOUNT,MEAN,STDEV)
                WRITE(100,605) NAME(J), MEAN, STDEV
605             FORMAT(1X,A24,2X,G13.6,2X,G13.6)
             END DO
             CLOSE(UNIT=100)
          END IF

C  Perform cultivation one case at a time, one case being a set of
C  hypotheses with the same independent variables:
C     y(1) = f(x(1),x(2),...,x(niv))
C     y(2) = f(x(1),x(2),...,x(niv))
C      .
C      .
C    y(ndv) = f(x(1),x(2),...,x(niv))

          DO ICASE=1,NCASE

C  Read a set of hypotheses, (y(1),...,y(ndv)) = f(x(1),...,x(niv))

             READ(10,*) NIV,(XKEY(I),I=1,NIV),
        2               NDV,(YKEY(I),I=1,NDV)

C  Calculate the minimum singular value for the SVD in SUBROUTINE INTERP

             IF(INTERPOLATE) THEN
                MINSV = 1./SCALE(XKEY(1))
                DO I=2,NIV
```

```
               TEST = 1./SCALE(XKEY(I))
               IF(TEST.LT.MINSV) MINSV = TEST
            END DO
         END IF

C  This next routine collects records containing the same independent
C  variables together using a sort routine.  Variables are sorted in the
C  order listed in XKEY.  The subroutine COLLECT returns an INDEX of sorted
C  data rather than actually sorting IDATA.  It also returns the number of
C  (NBOX), the location of (BEGIN), the number of elements in (COUNT), and
C  the maximum number of elements in each collection.
C  NOTE:  When you call COLLECT, you don't have to pay for it.

         CALL COLLECT ( %VAL(%LOC(IDATA(1,1))),
     2                  XKEY,
     2                  %VAL(%LOC(WK(1))),
     2                  NREC,
     2                  NVAR,
     2                  NIV,
     2                  %VAL(%LOC(INDEX(1))),
     2                  %VAL(%LOC(BEGIN(1))),
     2                  %VAL(%LOC(COUNT(1))),
     2                  NBOX )

C  Begin analyzing the dependent variables in each collection.

         DO IDV=1,NDV

            IBAD = 0
            IF(BOX_PLOT) WRITE(101,2002)
            IF(GRID_FILE) THEN
               GRID_NAME = ' '
               CALL CHAR_STRING_LENGTH(OUTFILE,ILENGTH)
               GRID_NAME(1:ILENGTH) = OUTFILE
               IPOS = ILENGTH+1
               GRID_NAME(IPOS:IPOS) = '_'
               IPOS = IPOS+1
               CALL CHAR_STRING_LENGTH(NAME(YKEY(IDV)),ILENGTH)
               GRID_NAME(IPOS:IPOS+ILENGTH-1) = NAME(YKEY(IDV))
               DO I=IPOS,IPOS+ILENGTH-1
                  IF (GRID_NAME(IPOS:IPOS).EQ.' ')
     2               GRID_NAME(IPOS:IPOS) = '_'
               END DO
               IPOS = IPOS + ILENGTH
               GRID_NAME(IPOS:IPOS) = '_'
               IPOS = IPOS + 1
               IF (ICASE.LT.10) THEN
                  WRITE(NUMBER_1,'(I1)') ICASE
                  GRID_NAME(IPOS:IPOS) = NUMBER_1
               ELSE
                  WRITE(NUMBER_2,'(I2)') ICASE
                  GRID_NAME(IPOS:IPOS+1) = NUMBER_2
               END IF
               OPEN (UNIT=102,STATUS='NEW',FILE=GRID_NAME,
     2               DEFAULTFILE='.GRID',RECL=132)
            END IF
            DO IBOX=1,NBOX

               IBEGIN = BEGIN(IBOX)
               NCOUNT = COUNT(IBOX)

C  Find the locations of all dependent variables not marked as -1 and store
```

```
C  in POINT.  Fill IX and Y.

                  ICOUNT = 0
                  DO I=1,NCOUNT
                     IREC = I + IBEGIN - 1
                     IF(IDATA(INDEX(IREC),YKEY(IDV)).GE.0) THEN
                        ICOUNT = ICOUNT + 1
                        POINT(ICOUNT) = INDEX(IREC)
                     END IF
                  END DO

                  DO I=1,NIV
                     IX(I) = IDATA(POINT(1),XKEY(I))
                  END DO

                  DO I=1,ICOUNT
                     Y(I) = RDATA(POINT(I),YKEY(IDV))
                  END DO

C  Statistics and interpolation have less meaning when the number of
C  samples is small, in which case the two operations will be skipped.

                  IF(ICOUNT.LT.MINREC) THEN
                     IBAD = IBAD + 1
                     GOTO 100
                  END IF

C  Interpolate each data point to the midpoint of the collection.

                  IF (INTERPOLATE) THEN
                     VM_SIZE(1) = ICOUNT*(NIV+1)*4
                     VM_SIZE(2) = 3*(NIV+1)*4
                     VM_SIZE(3) = NIV+1*4
                     VM_SIZE(4) = NIV*4
                     DO I=1,4
                        STATUS = LIB$GET_VM (VM_SIZE(I),VM_ADDR(I))
                        IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))
                     END DO
                     CALL INTERP ( %VAL(%LOC(Y(1))),
       2                           %VAL(%LOC(RDATA(1,1))),
       2                           %VAL(%LOC(POINT(1))),
       2                           ICOUNT,
       2                           IX,
       2                           XKEY,
       2                           YKEY(IDV),
       2                           SCALE,
       2                           ZERO,
       2                           %VAL(VM_ADDR(1)),
       2                           %VAL(VM_ADDR(2)),
       2                           %VAL(VM_ADDR(3)),
       2                           %VAL(VM_ADDR(4)),
       2                           MINSV,
       2                           NREC,
       2                           NVAR,
       2                           NIV )

                     DO I=1,4
                        STATUS = LIB$FREE_VM (VM_SIZE(I),VM_ADDR(I))
                        IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))
                     END DO
                  END IF
```

```
C  Calculate the mean and standard deviation of the dependent
C  variable in the collection, and write to a file.

                  IF(BOX_PLOT) THEN
                     CALL STATS(Y,NREC,ICOUNT,MEAN,STDEV)
                     WRITE(101,2001) (IX(I),I=1,NIV), MEAN, STDEV, ICOUNT
2001                 FORMAT(1X,<NIV>(I5,1X),G13.6,1X,G13.6,1X,I5)
                  END IF

C  Output data to *.GRID file.  This file contains, for each box or collection,
C  real midpoints of each independent variable, the mean and standard deviation
C  of the dependent variable, and the frequency (no. of points) in the box.
C  The midpoints are stored in WK.

                  IF (GRID_FILE) THEN
                     CALL STATS(Y,NREC,ICOUNT,MEAN,STDEV)
                     DO K=1,NIV
                        WK(K) = (REAL(IX(K))+0.5)/SCALE(XKEY(K))
       2                       +ZERO(XKEY(K))
                     END DO
                  WRITE(102,2003) (WK(I),I=1,NIV),MEAN,STDEV,ICOUNT
2003              FORMAT(<NIV+2>(1X,G13.6),1X,I6)
               END IF

100            CONTINUE

            END DO

            CLOSE (102)
            WRITE(6,602) IBAD, NBOX
602         FORMAT(1X,I5,' OUT OF ',I5,' BOXES DELETED')

         END DO

         END DO

C  Relationship Plots

         READ (10,*) NREL
         IF (NREL.GT.0) OPEN(UNIT=103,STATUS='NEW',FILE=OUTFILE,
       2                     DEFAULTFILE='.RP')

         DO IREL=1,NREL
              READ (10,*) NIV,(XKEY(I),I=1,NIV),NDV,YKEY(1)
              WRITE (103,2002)
2002          FORMAT (1X,'DATA')

C  Convert real data to integer data, using the new precisions for XKEY.
C  Data out of bounds or unusable is marked as -1.  IX and IBAD being used
C  to count bad data.

              DO I=1,NVAR
                     WK(I) = RANGE(I)
              END DO
              IF (NIV.GT.1) READ(10,*) (RANGE(XKEY(I)),I=1,NIV-1)
              DO I=1,NVAR
                     SCALE(I) = RANGE(I)/SPAN(I)
                     IX(I) = 0
              END DO
              IBAD = 0
              DO IREC = 1,NREC
                 DO IVAR=1,NVAR
```

```
                IF(RDATA(IREC,IVAR).GE.ZERO(IVAR) .AND.
     2             RDATA(IREC,IVAR).LE.ZERO(IVAR)+SPAN(IVAR)) THEN
                   IDATA(IREC,IVAR) = (RDATA(IREC,IVAR)-ZERO(IVAR))
     2                                *SCALE(IVAR)
                ELSE
                   IDATA(IREC,IVAR) = -1
                   IBAD = IBAD + 1
                   IX(IVAR) = IX(IVAR) + 1
                END IF
             END DO
          END DO
          DO I=1,NVAR
                RANGE(I) = WK(I)
          END DO

C  Sort data into collections, one for each line on the relationship plot.

          CALL COLLECT ( %VAL(%LOC(IDATA(1,1))),
     2                   XKEY,
     2                   %VAL(%LOC(WK(1))),
     2                   NREC,
     2                   NVAR,
     2                   NIV-1,
     2                   %VAL(%LOC(INDEX(1))),
     2                   %VAL(%LOC(BEGIN(1))),
     2                   %VAL(%LOC(COUNT(1))),
     2                   NBOX )

C  Find the good data and put it in WK for the independent variable and
C  Y for the dependent variable.

          DO IBOX=1,NBOX
                IBEGIN = BEGIN(IBOX)
                NCOUNT = COUNT(IBOX)
                ICOUNT = 0
                DO I=1,NCOUNT
                      IREC = I + IBEGIN - 1
                      IF(IDATA(INDEX(IREC),YKEY(1)).GE.0 .AND.
     2                   IDATA(INDEX(IREC),XKEY(NIV)).GE.0) THEN
                           ICOUNT = ICOUNT + 1
                           POINT(ICOUNT) = INDEX(IREC)
                      END IF
                END DO
                DO I=1,ICOUNT
                      WK(I) = RDATA(POINT(I),XKEY(NIV))
                      Y(I) = RDATA(POINT(I),YKEY(1))
                END DO

C  Do the linear regression for the data in a collection and output to a
C  file.

                IF (ICOUNT.GE.MINREC) THEN
                   vm_size(1) = 2*icount*4
                   vm_size(2) = icount*4
                   vm_size(3) = 3*icount*4
                   do i=1,3
                      status = lib$get_vm (vm_size(i),vm_addr(i))
                      if(.not. status) call lib$signal(%val(status))
                   end do
                   call regress ( %val(vm_addr(1)),
     2                                %val(vm_addr(2)),
     2                             %val(vm_addr(3)),
```

```
     2                                    WK,
     2                                    Y,
     2                                    NREC,
     2                                    ICOUNT,
     2                                    SLOPE,
     2                                    YCEPT,
     2                                    RSQUARE )
                      do i=1,3
                         status = lib$free_vm (vm_size(i),vm_addr(i))
                         if(.not. status) call lib$signal(%val(status))
                      end do

                      XLO = WK(1)
                      XHI = WK(1)
                      DO I=2,ICOUNT
                           IF (WK(I).LT.XLO) XLO=WK(I)
                           IF (WK(I).GT.XHI) XHI=WK(I)
                      END DO
                      IF (NIV.GT.1) WRITE (103,*)
     2                              (IDATA(POINT(1),XKEY(I)),I=1,NIV-1)
                      WRITE (103,*) SLOPE,YCEPT,RSQUARE
                      WRITE (103,*) XLO,SLOPE*XLO+YCEPT,
     2                              XHI,SLOPE*XHI+YCEPT
                   END IF

              END DO
          END DO

C  The End

          CLOSE (101,IOSTAT=STATUS)
          CLOSE (102,IOSTAT=STATUS)
          CLOSE (103,IOSTAT=STATUS)

          RETURN
          END
```

```
C*************************************************************************
C
C       SUBROUTINE NAME:  COLLECT
C       PROGRAMMER:  Mark J. Bendele
C       DATE:  2 March 1988
C       PURPOSE:  To sort data into collections (or boxes).
C       NOTES:  This routine is called by SUBROUTINE CULTIVATION in order
C               to collect dependent variables into boxes defined by
C               independent variables.  More explanation is given within
C               the code.
C
C       VARIABLES:  IDATA = array containing integer data of desired
C                           resolution converted from real plant data
C                   XKEY  = index of independent variables
C                   WK    = work vector for sort routine, ISORT
C                   NREC  = number of records in data base
C                   NVAR  = number of variables in data base
C                   NIV   = number of independent variables
C                   INDEX = index of records in data base -- sorted by ISORT
C                   BEGIN = vector containing locations of beginnings of
C                           collections (boxes)
C                   COUNT = vector containing number of data points
C                           (observations) in a collection (box)
C                   NBOX  = number of boxes (collections)
C
C       SUBROUTINES:  ISORT
C
C*************************************************************************

        SUBROUTINE COLLECT ( IDATA,
     2                       XKEY,
     2                       WK,
     2                       NREC,
     2                       NVAR,
     2                       NIV,
     2                       INDEX,
     2                       BEGIN,
     2                       COUNT,
     2                       NBOX )

        INTEGER*4 IDATA(NREC,NVAR),
     2            XKEY(NVAR),
     2            WK(NREC),
     2            INDEX(NREC),
     2            BEGIN(NREC),
     2            COUNT(NREC)

       LOGICAL IFLAG

C  This sort operation puts the data into collections such that the
C  independent variables in each collection are constant.  The independent
C  variables are sorted in the order listed in XKEY.  The array IDATA is not
C  actually sorted;  instead, an INDEX of IDATA is sorted.

         CALL ISORT(IDATA,WK,NREC,NVAR,NREC,XKEY,NIV,INDEX,1)

C  Identify each collection by ...

         IREC = 1
         IBOX = 0
         DO WHILE (IREC.LE.NREC)
```

```
C  (1) skipping over bad data (marked as -1) to find a collection,

               IFLAG = .TRUE.
               K = 1
               DO WHILE (IFLAG .AND. K.LE.NIV)
                      IF(IDATA(INDEX(IREC),XKEY(K)).LT.0) IFLAG = .FALSE.
                      K = K + 1
               END DO

C  (2) counting the number of elements in the collection,

               IF (IFLAG) THEN
                IBEGIN = IREC
                ICOUNT = 1
                DO WHILE(IFLAG .AND. IREC.LT.NREC)
                   K = 1
                   DO WHILE(IFLAG .AND. K.LE.NIV)
                      IF(IDATA(INDEX(IREC),XKEY(K)).NE.
     2                   IDATA(INDEX(IREC+1),XKEY(K))) IFLAG = .FALSE.
                      K = K + 1
                   END DO
                   IF(IFLAG) THEN
                      ICOUNT = ICOUNT + 1
                      IREC = IREC + 1
                   END IF
                END DO

C  and (3) storing the location of each collection in BEGIN and the number
C  of elements in the collection in COUNT.

                IBOX = IBOX + 1
                BEGIN(IBOX) = IBEGIN
                COUNT(IBOX) = ICOUNT
               END IF
               IREC = IREC + 1
          END DO
          NBOX = IBOX

          RETURN
          END
```

```
C*************************************************************************
C
C       SUBROUTINE NAME:  INTERP
C       PROGRAMMER:  Mark J. Bendele
C       DATE:  3 March 1988
C       PURPOSE:  Given data for one dependent variable versus several
C                 independent variables, interpolate each data point to the
C                 midpoint of the independent variables.
C       NOTE:  This routine calls a subroutine SVDRS to do a singular value
C              decomposition.  SVDRS is a quadratic programming routine
C              linked to this program through a library.
C
C       VARIABLES:  Y       = vector containing real values of the
C                             dependent variable in a collection
C                   RDATA   = array containing real data base
C                   POINT   = index of data points in collection that are
C                             not bad data (marked as -1)
C                   ICOUNT  = number of data points in collection
C                   IX      = vector containing integer values of the
C                             independent variables in the collection
C                   XKEY    = index of independent variables
C                   YKEY    = index of dependent variables
C                   SCALE   = vector of scales for each variable used to
C                             convert real data to integer data of desired
C                             resolution
C                   ZERO    = vector of low values of variables in data base
C                   A       = array of independent variables for each data
C                             point (used in least squares formulation)
C                   S       = vector of singular values
C                   X       = vector of slopes in linear model of colleciton
C                   XMID    = vector identifying midpoint of collection
C                   MINSV   = minimum singular value
C                   NREC    = number of records in data base
C                   NVAR    = number of variables in data base
C                   NIV     = number of independent variables
C
C*************************************************************************

      SUBROUTINE INTERP ( Y,
     2                    RDATA,
     2                    POINT,
     2                    ICOUNT,
     2                    IX,
     2                    XKEY,
     2                    YKEY,
     2                    SCALE,
     2                    ZERO,
     2                    A,
     2                    S,
     2                    X,
     2                    XMID,
     2                    MINSV,
     2                    NREC,
     2                    NVAR,
     2                    NIV )

      REAL*4 Y(NREC),
     2       RDATA(NREC,NVAR),
     2       SCALE(NVAR),
     2       ZERO(NVAR),
     2       A(ICOUNT,NIV+1),
     2       S(3*(NIV+1)),
```

```
     2        X(NIV+1),
     2        XMID(NIV),
     2        MINSV

      INTEGER*4 POINT(NREC),
     2          IX(NVAR),
     2          XKEY(NVAR),
     2          YKEY

C  Perform a linear regression on the data in a collection using the
C  singular value decomposition.  Fill the matrix A with the values of the
C  independent variables corresponding to the values of the dependent
C  variable, which are put in the vector Y.  We will then attempt to solve
C  the least squares problem X = A+ * Y, where A+ is the pseudo-inverse:
C  A+ = V*S+*UT

      DO I=1,ICOUNT
           DO J=1,NIV
                A(I,J) = RDATA(POINT(I),XKEY(J))
           END DO
           A(I,NIV+1) = 1.
      END DO

C  Compute the pseudo-inverse.  SVDRS returns A as V, Y as UT*Y, and S as a
C  vector containing the singular values.

      CALL SVDRS(A,ICOUNT,ICOUNT,NIV+1,Y,NREC,1,S)

C  Invert S.  S => S+

      DO I=1,NIV+1
           IF(S(I).GE.MINSV) THEN
                S(I) = 1./S(I)
           ELSE
                S(I) = 0.
           END IF
      END DO

C  Multiply V*S+

      DO I=1,NIV+1
           DO J=1,NIV+1
                A(I,J) = A(I,J)*S(J)
           END DO
      END DO

C  Compute X = (V*S+) * (UT*Y)

      DO I=1,NIV+1
           X(I) = 0.
           DO J=1,NIV+1
                X(I) = A(I,J)*Y(J) + X(I)
           END DO
      END DO

C  Find the midpoint of each collection; that is, if the independent variables
C  in the collection were converted back to their original real values, what
C  would be the middle of the range of each variable.

      DO K=1,NIV
           XMID(K) = (REAL(IX(K))+0.5)/SCALE(XKEY(K))+ZERO(XKEY(K))
      END DO
```

```
C  With the linear model for the collection, calculate the value of the
C  dependent variable at the midpoint.

      YMID = 0.
      DO I=1,NIV
           YMID = YMID + XMID(I)*X(I)
      END DO
      YMID = YMID + X(NIV+1)

C  Interpolate each data point to the middle of the collection.  YCAL is
C  a value of Y projected onto the object described by the linear model.

      DO I=1,ICOUNT
           YCAL = 0.
           DO K=1,NIV
                YCAL = YCAL + RDATA(POINT(I),XKEY(K))*X(K)
           END DO
           YCAL = YCAL + X(NIV+1)
           Y(I) = YMID + RDATA(POINT(I),YKEY) - YCAL
      END DO

C  Let's blow this taco stand.

      RETURN
      END
```

```
*    Program Name:  REGRESS
*    Purpose:  To perform a linear regression and output slope,
*              standard deviation, and r-squared.
*    Programmers:  J. Hackney, M. Bendele
*    Date:  3-August-1988
*

      subroutine regress (a,b,s,x,y,ldim,icount,slope,ycept,rsquared)

      dimension a(icount,2), b(icount), x(ldim), y(ldim), x_soln(2),
     2          s(icount,3)

      do i=1,icount
           a(i,1) = x(i)
           a(i,2) = 1.
           b(i) = y(i)
      end do

      tol = 0.
      call psolve2 (a,icount,icount,2,b,icount,x_soln,2,s,irank,tol)
      slope = x_soln(1)
      ycept = x_soln(2)

      do i=1,icount
           b(i) = slope*x(i) + ycept
      end do
      call mean (b,icount,icount,bmean)
      call mean (y,ldim,icount,ymean)
      call rsquare (b,y,icount,ldim,icount,bmean,ymean,rsquared)

      return
      end
```

```
C*************************************************************************
C
C       SUBROUTINE NAME:  STATS
C       PROGRAMMER:  MARK J. BENDELE
C       DATE:  2 MARCH 1988
C       PURPOSE:  TO CALCULATE THE MEAN AND STANDARD DEVIATION OF A
C                 SET OF DATA
C       NOTE:  WHEN GIVEN ONLY ONE DATA POINT, STATS RETURNS A STANDARD
C              DEVIATION OF ZERO.
C
C       VARIABLES:  X     = vector containing set of data
C                   IDIM  = dimension of vector
C                   N     = number of elements in vector
C                   MEAN  = mean
C                   STDEV = standard deviaiton
C
C*************************************************************************

          SUBROUTINE STATS(X,IDIM,N,MEAN,STDEV)

          REAL X(IDIM), MEAN

C  Calculate the mean.

          SUM = 0.
          DO I=1,N
               SUM = SUM + X(I)
          END DO
          MEAN = SUM/REAL(N)

C  Calculate the standard deviation.

          SUMSQ = 0.
          DO I=1,N
               SUMSQ = (X(I)-MEAN)**2 + SUMSQ
          END DO
          IF(N.GT.1) THEN
               STDEV = SQRT(SUMSQ/REAL(N-1))
          ELSE
               STDEV = 0.
          END IF

          RETURN
          END
```

VITA

Mark James Bendele was born in Arlington Heights, Illinois on May 12, 1961. As the son of a U.S. Army officer, he has lived in many places. He considers himself a Texan, since he has spent most of his life in Texas, and since his family resides there. He attended Texas A&M University, where he received a Bachelor of Science degree in Chemical Engineering in December of 1983. Currently, he is employed by International Paper Company as a process control engineer.