AN IMPROVED CONTROL LOOP

PERFORMANCE MONITOR

By

THOMAS JUDSON WOOTERS

Bachelor of Science in Chemical Engineering

Brigham Young University

Provo, UT

2005

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2008

AN IMPROVED CONTROL LOOP

PERFORMANCE MONITOR

Thesis Approved:

Dr. R. Russell Rhinehart

Thesis Adviser

Dr. Karen A. High

Dr. J. Rob Whiteley

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGEMENTS

I have many to thank whose influence had a great impact upon me while obtaining my education. I would first like to acknowledge my Advisor, Dr R. R. Rhinehart whose insights, support, enthusiasm and gentle prodding helped direct my learning. Before beginning my work, he wrote on a piece of loose paper the words “Do Not Study”. After seeing the concerned look on my face, he acted as though he had miswritten, and corrected the motto as “Do, Not Study”. Most of the insights I gained from course work and research came through application of this motto. Countless times Dr Rhinehart excitedly shared with me fun projects he was working on, illustrating the need to both enjoy what you do and do what you enjoy. I am also grateful to him for the financial support while pursuing my education, teaching classes and performing research. To my graduate committee members Dr K. High and Dr J. R. Whiteley whose enthusiasm for optimization and chemical process control respectively was more than contagious. I am grateful for their time in helping me to complete this work.

I am grateful also to the remainder of the School of Chemical Engineering with whom I interacted daily. Thanks go to Oklahoma State University as a whole for providing me an opportunity to pursue a graduate degree in a community where I felt comfortable and welcome.

I am also grateful to Samuel Owusu whose original research this current work is built upon. His dissertation has been vital to my work. And the ideas and theorems presented therein are clear and easy to follow. I hope to one day meet him and thank him for blazing the path for my work.

Finally, I would not be where I am today if it were not for my family. My wife Vanessa has been supportive through this adventure, lovingly reminding me to get as much out my education as possible. To Joshua who bared through Dad's "work at school", many "meetings" and seeing "people", all meaning not being able to be with him. To William who was born here in Stillwater during my second semester. Every one of his birthdays will remind us of how blessed our time at OSU has been. Thanks also to both my parents Neale and Kathy Wooters, and also to Vanessa's parents Wayne and Tammy Barrett.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This work is based on the doctoral dissertation of Samuel Owusu who graduated with his PhD in May of 2006 from Oklahoma State University. The outcome of his research was a real-time monitor capable of determining the performance of a control loop. The intent of the current work is to build upon the work presented by Owusu. First, the theoretical background is discussed in Chapter 1 followed by an explanation of algorithms developed in the original work and improvements made in this work discussed in Chapter 2. There are two general aspects of the algorithm improvements:

1) More robust and efficient computer program to perform necessary calculations
2) Improve algorithms for assessing controller performance
    a. More correctly calculate the window length for real-time monitoring
    b. Eliminate unsupported condition in determining the Markov Chain model

Chapters 3 and 4 introduce and discuss two experimental apparatuses

1) Simulated first-order plus time delay (FOPTD) system controlled by PI
2) Pilot-plant scale two-phase flow experiment using a cascade strategy to control pressure drop.

Finally, suggested future work and conclusions are discussed in Chapters 5 and 6.

1.1 Literature Review

According to an assessment by Hugo (2000), a typical process plant operates with hundreds of control loops maintained by a limited number of operators, instrument technicians or control engineers. Because tuning these loops is only one out of many human responsibilities, problem controllers often go for long periods without correction. This results in some 66% - 80% of controllers in industry improperly tuned for current process requirements. Mosca (2003) noted that well tuned process control loops contribute to safety, productivity and lower maintenance costs, perhaps in that order of importance. Poor process control may stem from poor controller tuning, instrument malfunction, process nonlinearities or process design. Even the performance of a well tuned controller may, over time, erode due to gradual process changes.

Bartels (2007) noted that the last two decades have brought downsizing to reduce labor costs. In addition, a large number of technicians and control engineers will retire soon leading to even fewer employees to maintain the large number of loops operating in industrial plants. In fact, one “major chemical company that did a demographic analysis … found that one of its largest plants would lose 75% of its operating staff to retirement by the end of the decade.”

With fewer employees, systematic, periodic inspections are being replaced by condition-based inspections. Automated loop performance monitoring facilitates this initiative by raising a flag only when a loop is no longer healthy. In the past 25 years many researchers have attempted to solve this problem.

Harris, *et al.* (1989) presented a benchmark commonly referred to as minimum variance control (MVC) or the Harris Index (HI). This method determines the ideal minimum variance for a particular loop, then compares current real-time variance to the ideal value using Equation 1.1

$$HI = \frac{\text{Current Variance}}{\text{Minimum Achievable Variance}}. \qquad (1.1)$$

The index denotes perfect control when HI = 1 and poor control when HI is large. Minimum Variance Control, while able to indicate loops in oscillation, may incorrectly flag sluggish loops, since their variance is low. In addition, the Harris Index has no absolute meaning. A particular HI value for one loop may be considered good control, while the same value in another loop may be considered poor control. Finally, the process time delay is required to correctly identify the process model used in determining the minimum variance, which may be cumbersome to obtain.

Rhinehart (1995) , Ko and Edgar (1998), Bezergianni *et al*. (2000) and Kadali, *et al*. (2002) have all contributed to solving this problem. All of these techniques, however, fall short of practical application, either because of cumbersome process model identification needs, inadequate statistical consideration or simply the inability to identify all frequency type controller problems. More recently, Li, *et al*. (2003) proposed the use of the chi-squared statistic to compare a reference set of "good" control actuating error run-lengths to run-lengths in a window of real-time control. If a statistically significant process change is noticed, the control is flagged as "poor". The technique uses routine data

harvested from the data historian. Although it worked well, undesirably, the theoretical foundation for Chi-Squared analysis is inexact, and continued research sought a more rigorous approach.

Owusu (2006) proposed a technique to model actuating error run-lengths using a Markov Chains and transition probabilities. The transition probabilities are essentially a measure of run-length termination and are well described by Binomial Statistics. Owusu notes that this idea originates from Avery and Ingalls (2002) in critique of the work performed by Li, *et al*. Owusu's work contained some computational inefficiencies and proposed a number of unsupported criteria for determining the real-time run-length statistical window. This work seeks to tie-up some of these loose ends.

1.2 Objective

The objective of this work is to replicate the work performed by Owusu, then propose improvements to the methods and algorithms developed by Owusu and demonstrate their effectiveness through simulations and experimental work. This first requires a discussion of the theoretical background in Owusu's work discussed in the remainder of Chapter 1.

1.3 Deviation Data

A controller is healthy when performing according to design. Lack of perfect process understanding limits the ability to control a process according to some absolute state of health. This work assumes that the process owner is the ultimate decision maker regarding acceptable or good controller performance.

To be explained in detail later, the approach observes patterns in the actuating error. The process owner chooses a “good” period of control by whatever criteria he or she uses, and the algorithm sets up control limits about statistics that characterize patterns in the “good” process data. Then, if the statistics from subsequent periods are within the control limits, control is reported as “good”. If the control limits are violated, controller health is flagged as “poor”.

Since real-time controller performance is compared against a standard considered “good” control for that particular process, the original data set must be taken during “good” control. These data are most likely to be found soon after a controller is satisfactorily tuned; however the process owner ultimately decides what “good” data are. No *a priori* knowledge of the process is required because the health monitor analyzes “good” process data and uses it as the ideal. The data set absolutely must represent normal operation; so no setpoint changes or other purposeful perturbations are needed outside of those already present in normal operation.

1.4 Markov Chain Model

Imagine a process under automatic control where the actuating error signal read by the controller is defined as the deviation from the setpoint. Throughout this work, the term “error” will refer to the actuating error, unless otherwise specified. Under “good” control, the actuating error signal will remain close to zero more often than not; however, due to various sources of noise, an error value of exactly zero is rarely obtained. What occurs

instead is a constant “bouncing” above and below the setpoint or above and below the zero value. For this study, the standard definition of actuating error or deviation is defined as “setpoint minus controlled variable”, so positive actuating error results from the controlled variable below the setpoint and negative actuating error results from the controlled variable above the setpoint.

Consider monitoring the actuating error signal of some process under feedback control immediately after the controller has been tuned. Consider that the first error sample is positive or below the setpoint; only the sign is necessary, the magnitude is unimportant. The second error sample also happens to be positive resulting in 2 positive samples in a row. The next sample is negative, resulting in the beginning of a run on the negative error side. This hypothetical data set is found in Figure 1.1.

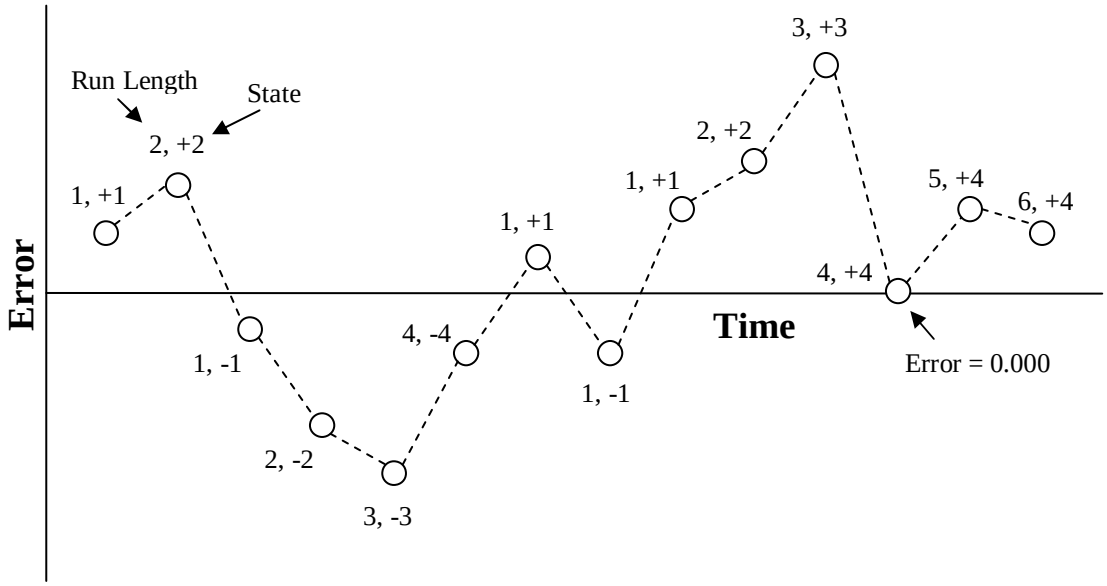

Figure 1.1: Hypothetical Error Signal

Each actuating error data point is labeled by the number of samples on a particular side of zero error (run length) that this data point represents and the state in which this data point would be found. The third sample labeled (1,-1) begins a run of 4 samples in a row below the zero error axis, followed by a single sample above and then below. The final run in the hypothetical actuating error signal found in Figure 1.1 contains 6 samples in a row in a “plus” run. The third to last point marked (4, +4) has a value of 0.000. Since it is not a zero crossing it is included in the run sign of the previous data. The last run has six sequential ‘+’ data points. Although the run length value rises to +6, the state remains at +4, the maximum value illustrated. This example run helps explain the use of states in a Markov Chain to describe run lengths.

A Markov Chain is a set of linked locations that may be visited only in a particular order. Each location is designated by a state number and represents a certain run length. The Markov Chain for this application is illustrated in Figure 1.2. States are identified as ovals and the arrows show permissible transitions. In this application, the condition represented by each state has only two possible states it may next visit. For example, from State +2, either the +3 or -1 State may be visited. Visiting the +3 State requires a continued run of positive actuating error and visiting the -1 State requires a zero crossing. The chain only maintains information on where to go next, it does not matter from whence the current state came. This “memoryless” property enables the Markov Chain to predict the probability of a future state condition knowing only the current state condition. For the Markov Chains used in this work, the state is the run length on either side of the zero error axis. An infinite Markov Chain would allow infinite runs on either

side of the zero error axis; however, this is not practical, so a limit is set. This limit is determined by an algorithm internal to this method, which will be explained later.

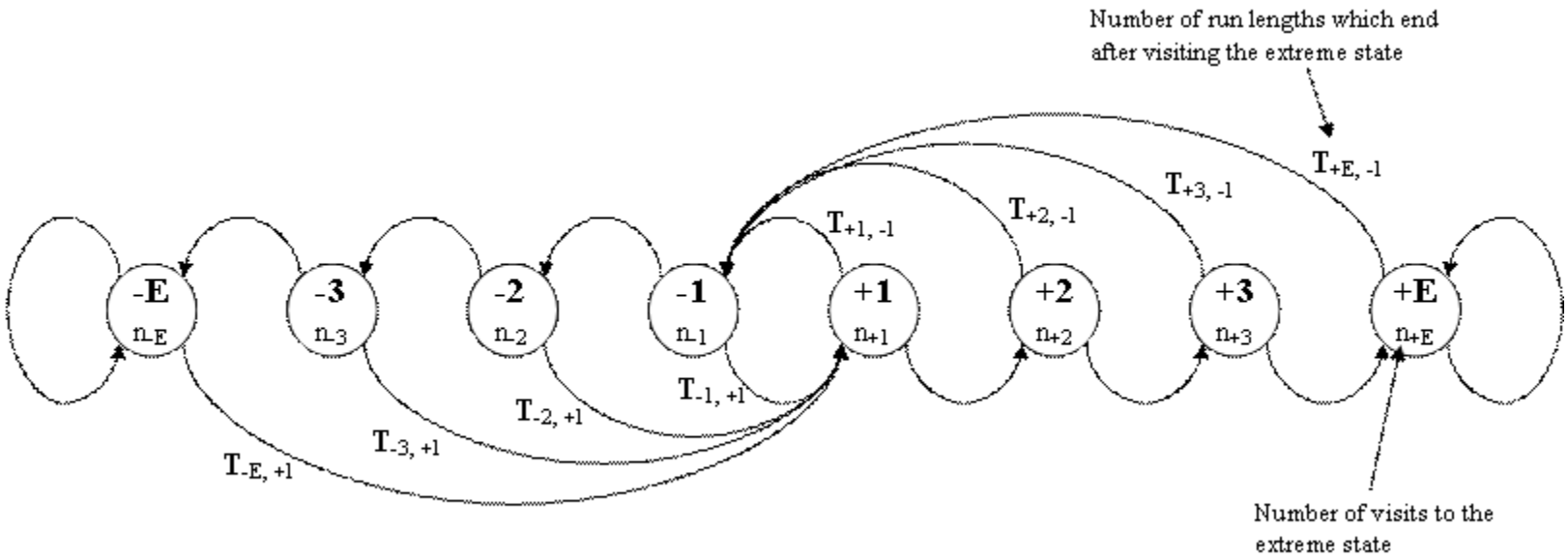

Figure 1.2: Markov Chain

The chain in Figure 1.2 is also labeled so as to see how it can extend to any length of chain needed. The +/- E State is the extreme state, or final state that may be visited. This chain has 4 positive and 4 negative states, so +E may be labeled +4 and –E may be labeled -4. All conditions represented by any state have two possible choices to move, either to the next state in the increasing direction or to the first state of the opposite sign. The one exception to this rule is the extreme State in which the condition may either "revisit" the extreme State or visit the first State with the opposite sign. This explains the final run of samples in Figure 1.1; there are 3 positive error samples, followed by a 4^{th} sample equal to zero error, then 2 more positive error samples. An actuating error of exactly zero is not considered a zero crossing. After the 4^{th} sample above zero error, each subsequent positive error sample revisits the extreme State of +4. Table 1.1 provides the final counts of the samples and the visited states in Figure 1.1. The term sample is used to denote the number of times a data point visits a particular state.

Table 1.1: Number of Visits to Each State

State	-4	-3	-2	-1	+1	+2	+3	+4
Samples	1	1	1	2	3	2	1	3

1.5 Transition Probability

Let i represent the +/- visited state value, then the zero crossing transition probability P_i is defined as the probability for the subsequent sample to make a zero crossing and visit the first State of the opposite sign (+/- 1). On the other hand, the probability that the next

sample will remain on the same side of the zero error axis and continue a run is $1-P_i$. To calculate the transition probability for all states, a few terms are defined. Let n_i denote the number of samples which have visited a State i and let $T_{i,j}$ denote the number of samples that leave State i and go to State j. For the positive states Equation (1.2)

$$P_{+i} = \frac{T_{+i,-1}}{n_{+i}}, \tag{1.2}$$

and for negative states Equation (1.3)

$$P_{-i} = \frac{T_{-i,+1}}{n_{-i}} \tag{1.3}$$

The transition probability for each state will be unique for any system under automatic control. Furthermore, if the state transition probabilities are known for a system under "good" control, the future real-time actuating error may be compared against the "good" set to determine controller health. A statistical test will now be defined by which the two sets, "good control data" and the future set "real-time data" are compared.

Continuing with the example from Figure 1.1, the transition probabilities are presented in Table 1.2.

Table 1.2: Zero Crossing Transition Probabilities For Data in Figure 1.1

State	-4	-3	-2	-1	+1	+2	+3	+4
Samples	1.00	0.00	0.00	0.50	0.33	0.50	0.00	0.00

The limited set of "good" control data set which Figure 1.1 represents contains only 14 samples from which the transition probabilities in Table 1.2 are calculated. This introduces the question of how many samples should exist in the "good" control data set to provide meaningful transition probabilities. Given a larger data set, the transition probability limits should converge to values representative of the process under control. One solution to this question is provided in Chapter 2.

1.6 Binomial Distribution

A binomial experiment exhibits five properties listed below:

1. The number of trials 'n' is fixed.
2. Each experimental unit results in only two possible outcomes. Of the two characteristic events or outcomes, the one of interest is often referred to as success and the other failure.
3. The probability of success on each trial, denoted as *p*, remains constant.
4. The outcome for any one experimental unit is independent of the outcome for any other experiment unit.
5. The random variable *x*, counts the number of "successes" in n trials

In the controlled process the actuating error signal fits the description of a binomial process. 1) The number of trials may be fixed by setting a specified observation window and counting the visits to one state. 2) The transition probability predicts two outcomes; either a subsequent sample will transition across the zero axis or it will continue it's run to a higher state. 3) Furthermore, the transition probability is a constant value calculated

from a good data set. 5) The random variable x is assigned the number of run-lengths which make a zero crossing (defined as a success).

The fourth requirement is most closely satisfied by a process controlled such that the actuating error signal contains only measurement error or noise. This would represent a process where each sampling is nearly independent of its predecessor. However, a degree of autocorrelation exists in many automated systems, because the feedback loop provides process output information back to the controller. Therefore, current samples are influenced by past samples. This requirement is admittedly one of the violations of the idealizations behind the method described in this work and a suitable solution was not found by Owusu and is not obtained in the current work. It is believed however, that in the effort to find both an academically and practically satisfying method to monitor controller health, the binomial distribution best describes state transition probability.

The probability of getting exactly x successes from n trials is given by the following binomial distribution Function (1.4) for all $n \geq x \geq 0$ where $n \geq 1$

$$P_x(x, p, n) = \binom{n}{x} p^x (1-p)^{n-x}, \tag{1.4}$$

where p is the transition probability and n is the fixed number of samples which visited a particular state. Because the binomial distribution is discrete, the cumulative Function (1.5) is the summation of each discrete probability from x to n

$$P_{cum,x}(x, p, n) = \sum_{x=0}^{n} \binom{n}{x} p^x (1-p)^{n-x}. \tag{1.5}$$

One important property of the binomial distribution is its first moment, also called the mean or expectation. This is reported in Equation (1.6) here without derivation

$$E(x) = np\,. \tag{1.6}$$

This property will be used to successfully calculate the binomial distribution while avoiding computer overflow errors often associated with calculating factorials.

The binomial distribution is discrete, meaning that x can never contain a decimal part. For instance, it would not make sense to flip a coin 11.5 times. The coin is either flipped 11 times or 12 times. This results in a distribution similar to the one found in Figure 1.3. The number of samples is 50 (n = 50) and the probability is 0.5 (p = 0.5), chosen to correspond to the calculated transition probability for State +2 from the examples in Figure 1.1.

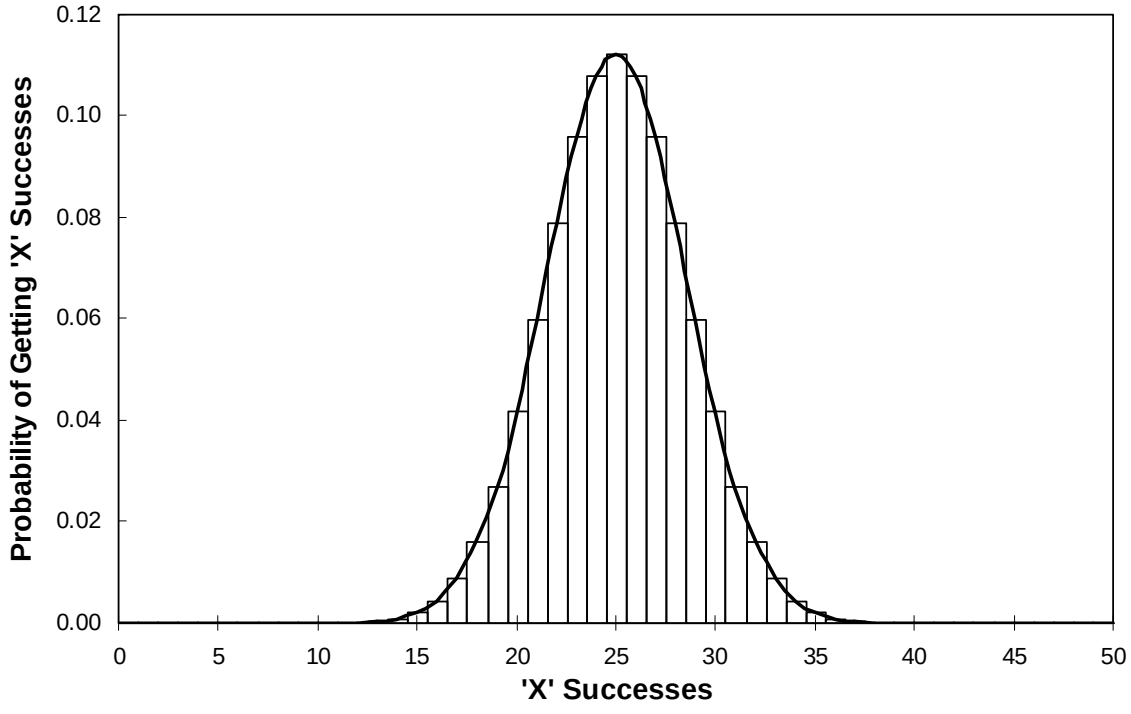

Figure 1.3: Binomial Distribution (Total Samples in set = 50, Probability = 0.5)

The continuous line is for convenience only, and is meant to touch the top middle of every column. The remaining plots in this section will maintain only the continuous line and not the columns as it will be more convenient in describing the concepts.

1.7 Hypothesis Testing

In hypothesis testing, two contradictory statements are proposed. Often, one statement is favored, and it is rejected only if sample information is sufficient to do so. If not rejected, it is "accepted". The meaning of "accepted" is not "proven" or "found true", but instead means that there was not sufficient evidence to reject. The initially favored statement is called the null hypothesis or H_0 and the other statement the alternate hypothesis or H_a.

The decision to accept or reject the null hypothesis is based on confidence limits wherein some decision error is allowed. Let these limits be called α (probability of a Type-I error) and β (probability of a Type-II error). The two error types and their relation to the decision to reject the null hypothesis are show in Table 1.3.

Table 1.3: Hypothesis Testing Decision/Situation Grid

Situation→ **Decision** ↓	**H_0 is true**	**H_0 is False**
Accept H_0	$1-\alpha$ (Correct decision)	β (Incorrect Decision)
Reject H_0	α (Incorrect Decision)	$1-\beta$ (Correct decision)

1.7.1 Analysis of Type I Errors

To describe hypothesis testing using the binomial distribution, consider a case of 50 samples (n = 50). For a transition probability of 0.5 (p = 0.5) which results in an expected or mean value of 25 successes. Figure 1.4 shows this distribution with a hypothetical Type I error rate (α) is the sum of the two tail Type I error rates (α_T) in the shaded tail areas resulting from an upper and lower control limit of 30 and 20 samples respectively.

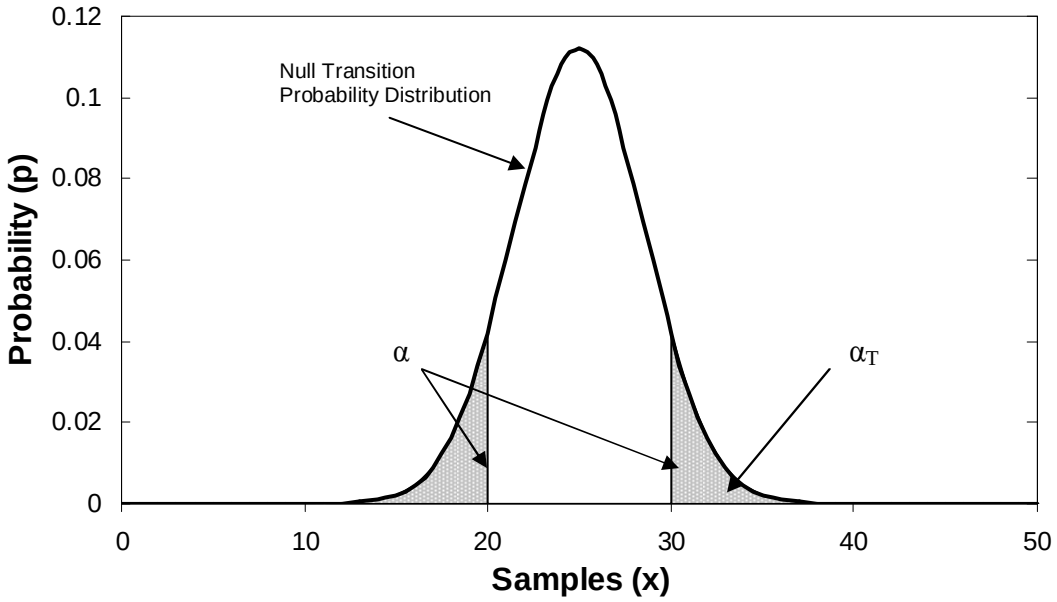

Figure 1.4: Binomial Distribution (Total Samples in set = 50, P_{null} = 0.5, α_T = 0.10)

If x is chosen to be 25 with limits +/-5 so that 20 is the lower limit and 30 is the upper limit Type I error rate (α) for both tails combine for an approximate probability of 0.20 or 20% of x successes found in the two tails. Each tail Type I error rate (α_T) then contains approximately 10% of the x successes. This means that there is a probability of 0.20 that the null hypothesis will be rejected when it should be accepted. The analog example often

cited when describing Type I errors is that of a court of law. In this case, with a Type I error of $\alpha = 0.20$, there is a 20% chance of incorrectly finding someone guilty.

1.7.2 Analysis of Type II Errors

After the limits are determined, the test requires an alternate hypothesis, H_a, with which to compare. Let λ designate a change of probability from the nominal probability. For an increased alternate probability use Equation (1.7)

$$P_{upper} = P_{null} + \lambda(1 - P_{null}) \tag{1.7}$$

and to find the decreased alternate probability alternate hypothesis use Equation (1.8)

$$P_{lower} = \lambda(1 - P_{null}). \tag{1.8}$$

These equations allow the user to specify a particular amount of change that they want to analyze for Type II errors. In Figure 1.5, the nominal distribution is now plotted along side an alternate hypothesis whose mean probability has increased. The alternate mean probability is 0.64 ($P_{upper} = 0.64$) making the mean 32 samples. The total number of samples ($n = 50$) remain the same. Beta (β) is calculated by summing the area under the alternate distribution found within the upper control limit (UCL) found during Type I Error analysis.

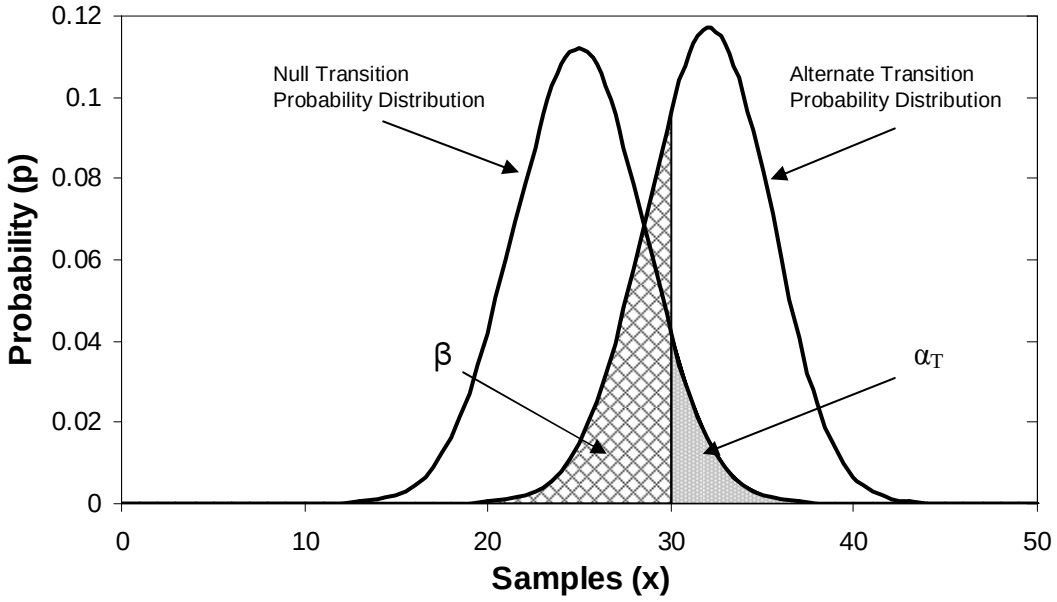

Figure 1.5: Binomial Distribution (Total samples in set = 50,

$P_{null} = 0.5$, $\alpha_T = 0.10$, $P_{upper} = 0.64$, $\beta = 0.33$)

In this case, beta (β) is the cross-hatched region below 30 samples found under the alternate probability which corresponds to a cumulative probability of approximately 0.33. Beta (β) is the probability of accepting the null hypothesis when in fact it should be rejected. The analog in a court of law would be failing to find a defendant guilty who truly is guilty. For the example found in Figure 1.5, there exists a 33% chance of failing to find a defendant guilty who should be found guilty. Again, alpha (α_T) for each tail individually is approximately 0.10.

1.8 Real-Time Window Size

The reason for doing both the Type-I and Type-II test is to find the number of samples that fulfill a user provided maximum for both α and β. As the number of samples

increases, the overlapping area decreases between the null and alternate hypothesis. Figure 1.6 illustrates this idea by modifying the example from Figure 1.5 to contain a total sample set n = 100. The tail alpha (α_T) is kept the same as above ($\alpha_T = 0.10$). The mean number of successes x for the null probability is still 50 and new limits of LCL = 42 and UCL = 58 found from an $\alpha_T = 0.10$.

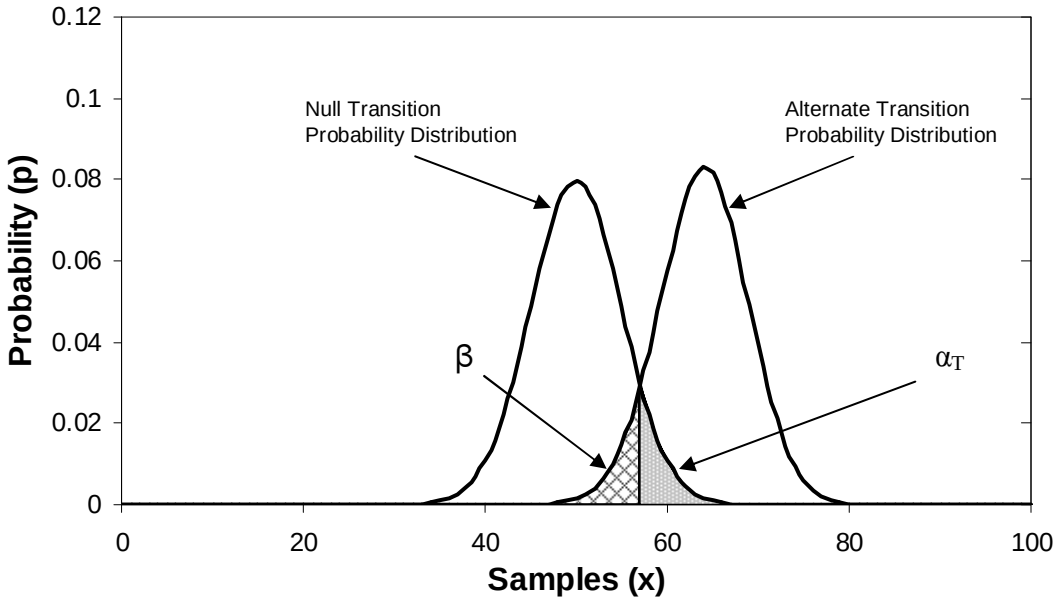

Figure 1.6: Binomial distribution (Samples = 50, $P_{null} = 0.5$, $\alpha_T = 0.10$, $P_{upper} = 0.64$, $\beta = 0.09$)

Notice that, while the Type I test error remained the same, a Type I error tail (α_T) of 0.10, the increased number of samples resulted in much lower overlapping area. The Type II test then resulted in an error (β) of 0.09, much smaller then the previous 0.33 when there were 50 total samples in the set. The strategy then, is to allow the user the choice for Type I and Type II error rates, then determine how many samples are required to satisfy

these conditions. The analogy in a court of law is that an increased quantity of evidence provides improves greater confidence in making a guilty or not guilty decision

For the example mean null probability of 0.50 found in Figure 1.6, if the user had previously specified a Type I error rate (α) of 0.10 and a Type II error rate (β) of 0.10, a total set of 100 samples would be required to accept or reject the null hypothesis. The lower control limit (LCL) would be 42 samples and the upper control limit (UCL) would be 58 samples. Another way to view this is that if the transition probability for State +2 remains within the limits 42/100 = 0.42 and 58/100 = 0.58 during real-time monitoring, then the null hypothesis is accepted, meaning no change in mean transition probability occurred. On the other hand, if the transition probability drops below 0.42 or moves above 0.58 during real-time monitoring, then the null hypothesis is rejected. These decisions carry with them the user specified 10% chance ($\alpha = 0.10$, $\beta = 0.10$) of an incorrect decision in either case.

It was mentioned that the null transition probability chosen for these examples is 0.50 corresponding to the calculated transition probability for State +2 only in Figure 1.1. This alludes to the fact that this technique calculates the number of samples needed for each state to comply with the user specified Type I error rate (α) and Type II error rate (β). Obviously, the example will not work when the transition probability is 1.00 (as it is for State -4) or 0.00 (as it is for States -3, -2 and +3) since it would be impossible to violate one of the two limits 1.00 or 0.00. In addition, since each state may have different transition probabilities, the number of samples required to meet the Type I and Type II

tests may be different per state. When the total number of samples, n, is calculated for each state based on the null hypothesis (transition probability for each state) a total number of samples required may be determined then used during real-time analysis. This total number of samples for all states is called the total statistical window. Chapter 2 discusses the technique used to calculate the total statistical window length (total required number of samples).

CHAPTER II

ALGORITHM IMPROVEMENTS

This chapter presents updates to the work reported by Owusu. The first update involves improving the computational robustness and efficiency of calculating the binomial distribution. The second update details two algorithm changes, one to more correctly calculate the window length for real-time monitoring and another to eliminate an unsupported condition in determining the Markov Chain model.

2.1 Binomial Distribution Calculation Improvements

The method used in the original work to calculate binomial distributions produced overflow run-time errors under certain situations. In the following section, improvements are detailed which eliminate overflow concerns and lower computational requirements.

2.1.1 Calculating Binomial Distribution – Owusu

The original algorithm used by Owusu to solve for discrete binomial probabilities takes a piece wise approach. The probability of getting x number of successes (transition probability) from n trials (total visits to a particular state) where the mean probability (null transition probability) is p is again reported as

$$P_x(x, p, n) = \binom{n}{x} p^x (1-p)^{n-x} \tag{2.1}$$

where

$$\binom{n}{x} = \frac{n!}{x!(n-x)!} \tag{2.2}$$

also called the binomial coefficient (2.2). Owusu simplifies the binomial coefficient as the ratio U over L where U is n to n-x+1 (whole integers counting down) and L is x to 1 (also whole integers counting down) shown here

$$\binom{n}{x} = \frac{\prod_{U=n}^{n-x+1} U}{\prod_{L=x}^{1} L}. \tag{2.3}$$

It is now possible to split the terms into pieces to avoid memory overflow issues that are common to calculating factorials. For example, expression (2.4) determines the binomial coefficient terms that would come from 5 successes in a total of 10 trials where the mean probability is 0.4. The ratio of number U by L will be:

$$\binom{n}{x} = \frac{10 \cdot 9 \cdot 8 \cdot 7 \cdot 6}{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = \frac{10}{5} \cdot \frac{9}{4} \cdot \frac{8}{3} \cdot \frac{7}{2} \cdot \frac{6}{1} = 2.00 \cdot 2.25 \cdot 2.67 \cdot 3.50 \cdot 6.00 \tag{2.4}$$

These terms are successively combined with p^x and $(1-p)^{n-x}$. Owusu's algorithm first multiplies the $(1-p)^{n-x}$ with each successive term from the binomial coefficient series, then finally multiplies p^x to obtain the solution. Using the example from above

$$(1-p)^{n-x} = (1-0.4)^{10-5} = 0.078 \tag{2.5}$$

and

$$p^x = 0.4^5 = 0.010. \tag{2.6}$$

Now multiply each term of the binomial coefficients with 0.078 to obtain 19.596. The final step is to multiply 19.596 by 0.010 to arrive at the binomial probability 0.201. This is the probability that 5 successes would result from a total sample of 10 tries where the mean probability is 0.4. The chart in Figure 2.1 summarizes these steps.

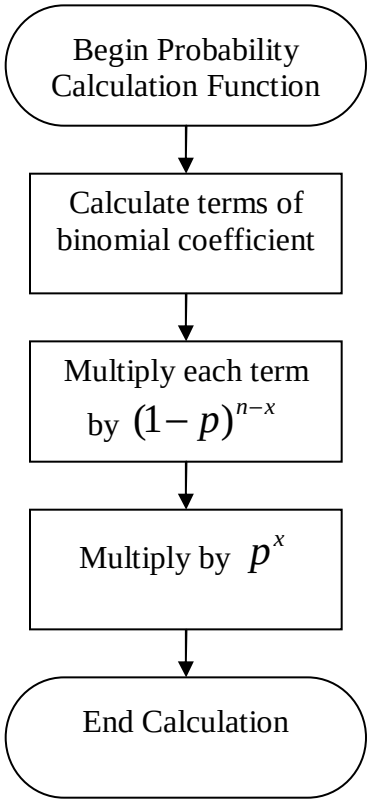

Figure 2.1: Owusu Algorithm to Calculate Binomial Distribution

Once the probability has been obtained for each discrete number of successes where x = 0, 1, 2, 3, …, n then the complete binomial distribution curve is known for a particular mean probability (p) and number of samples (n).

There are two advantages to using this algorithm:

1) Memory overflow is usually avoided because the binomial coefficient is stored as one single value.
2) The operation minimizes total number of multiplication operations to x+2, where x is the number of successes.

2.1.2 Calculating Binomial Distribution – Wooters

The method presented contains one main disadvantage; it is still subject to memory overflow issues. The main problem arises when either p^x or $(1-p)^{n-x}$ is too small for a double precision variable type and is therefore truncated to zero. For instance, when the binomial distribution mean is 0.500, the probability of finding 900 successes in 2000 total attempts is 8.00 E-8, but the Owusu method will find the probability equal to 0. This occurs because $(1-p)^{n-x} = 0.500^{1100} = \text{too small for double precision}$. In this work, a similar concept is used; except terms are split into smaller pieces to allow for flexibility in calculating the discrete probability with fewer risks of overflow.

Again consider the previous example where n = 10, x = 5 and the mean probability is 0.4. First, calculate the five terms of the binomial coefficient, reported below as Equations (2.7) through (2.9) for convenience

$$\binom{n}{x} = 2.00 \cdot 2.25 \cdot 2.67 \cdot 3.50 \cdot 6.00 \tag{2.7}$$

Also, each term in p^x or $(1-p)^{n-x}$ can be broken up into smaller pieces namely,

$$p^x = \prod_{i=1}^{x} p = \prod_{i=1}^{5} 0.4 = 0.4 \cdot 0.4 \cdot 0.4 \cdot 0.4 \cdot 0.4 \tag{2.8}$$

and

$$(1-p)^{n-x} = \prod_{i=1}^{n-x} (1-p) = \prod_{i=1}^{10-5} (1-0.4) = 0.6 \cdot 0.6 \cdot 0.6 \cdot 0.6 \cdot 0.6 . \tag{2.9}$$

The product of each sequential term is then calculated using the following relationship, where "max" is the maximum number of terms found between the three sets.

$$\prod_{i=1}^{\max} \binom{n}{x}_i \cdot \left(p^x\right)_i \cdot \left((1-p)^{n-x}\right)_i \tag{2.10}$$

Whichever of the three sets contains the most terms is said to be the base set. In this case, each set contains 5 terms; however, if the x successes were 4, the $\binom{n}{x}_i$ and $\left(p^x\right)_i$ sets would contain only 4 terms while the $\left((1-p)^{n-x}\right)_i$ set would contain 6 terms. To resolve this situation, the algorithm creates 2 two additional terms of unity in each of the sets containing only 4 terms. Now each set would contain 6 terms. Since this example already contains 5 terms in each set Equations 2.11 through 2.15 report sample calculations.

$$2.00 \cdot 0.4 \cdot 0.6 = 0.480 \tag{2.11}$$

$$2.25 \cdot 0.4 \cdot 0.6 = 0.540 \tag{2.12}$$

$$2.00 \cdot 0.4 \cdot 0.6 = 0.640 \tag{2.13}$$

$$2.00 \cdot 0.4 \cdot 0.6 = 0.840 \tag{2.14}$$

$$2.00 \cdot 0.4 \cdot 0.6 = 1.440 \tag{2.15}$$

As long as $x \geq n - x$ the final set of terms (Eq. 2.11 through 2.15 above) are sorted and only x number of calculations are required. This is also the same number of calculations required by the original Owusu algorithm. If, however, $x < n - x$, the terms will only be sorted up to the xth term, after which the terms will be filled with values from p^x and $(1-p)^{n-x}$ which when multiplied together will always be less than unity and only pseudo-sorted. A better decision can be made on which terms to multiply together because the list is, at worst, pseudo-sorted. It is undesirable for the result of any terms products to be too small or too big from a memory overflow standpoint, so pick numbers to satisfy this condition. The algorithm used in this work initializes the probability at unity, then looks at the first and last terms in the final set (Eq 2.11 – 2.15) and multiplies the current calculated probability by whichever term keeps it closer to 0.5. An index is set at the beginning and end of the final set and as a number is used in the product, the indices move closer together. Equations 2.16 through 2.21 detail this procedure.

$$\text{Probability} = 1.000 \tag{2.16}$$

$$\text{Probability} = \text{Probability} \cdot \text{FinalTerm}_1 = 1.000 \cdot 0.480 \quad = 0.480 \tag{2.17}$$

$$\text{Probability} = \text{Probability} \cdot \text{FinalTerm}_5 = 0.480 \cdot 1.440 \quad = 0.691 \tag{2.18}$$

$$\text{Probability} = \text{Probability} \cdot \text{FinalTerm}_2 = 0.691 \cdot 0.540 \quad = 0.373 \tag{2.19}$$

$$\text{Probability} = \text{Probability} \cdot \text{FinalTerm}_4 = 0.373 \cdot 0.840 \quad = 0.314 \tag{2.20}$$

$$\text{Probability} = \text{Probability} \cdot \text{FinalTerm}_3 = 0.314 \cdot 0.640 \quad = 0.201 \tag{2.21}$$

This algorithm finally arrives at the same calculated probability as the original Owusu algorithm, however, now with added flexibility to avoid nearly all overflow. In fact, the

largest and smallest numbers required by the original Owusu algorithm was 19.596 and 0.010 respectively. The largest and smallest numbers required for storage suggested by this current work is 1.440 and 0.201 remaining much closer to the final answer. Figure 2.2 reports the algorithm.

The only remaining chance of overflow is when the calculated probability for a given number of x successes is actually smaller than the value storable within a double precision variable; however this is resolved using the next modification. In addition, there exists one disadvantage still between the two algorithms, the Owusu algorithm requires fewer operations when $x < n - x$. This is overcome by one final algorithm modification.

Instead of calculating all of the probabilities for each possible number of x successes as the original Owusu, et al, algorithm does, only those x successes whose probability lie above a specified threshold of 1E-16 are calculated. Since the binomial probability is unknown beforehand instead of starting to calculate the distribution from an extreme where a small binomial probability is expected, start at the x success which represents the binomial mean.

For all cases studied in this work, the mean or expected number of x successes of the binomial distribution $E(x) = np$ is also the mode of the samples. This means that the probability associated with np successes will be the highest of the entire set of possible x successes. Instead of beginning at x = 0 and continuing to calculate the probability of each x success until reaching x = n, start by calculating np successes and then calculate

above and below this number of successes until finding the x successes whose probability is below the threshold 1E-16, then assume all binomial probabilities beyond this number of successes is 0.

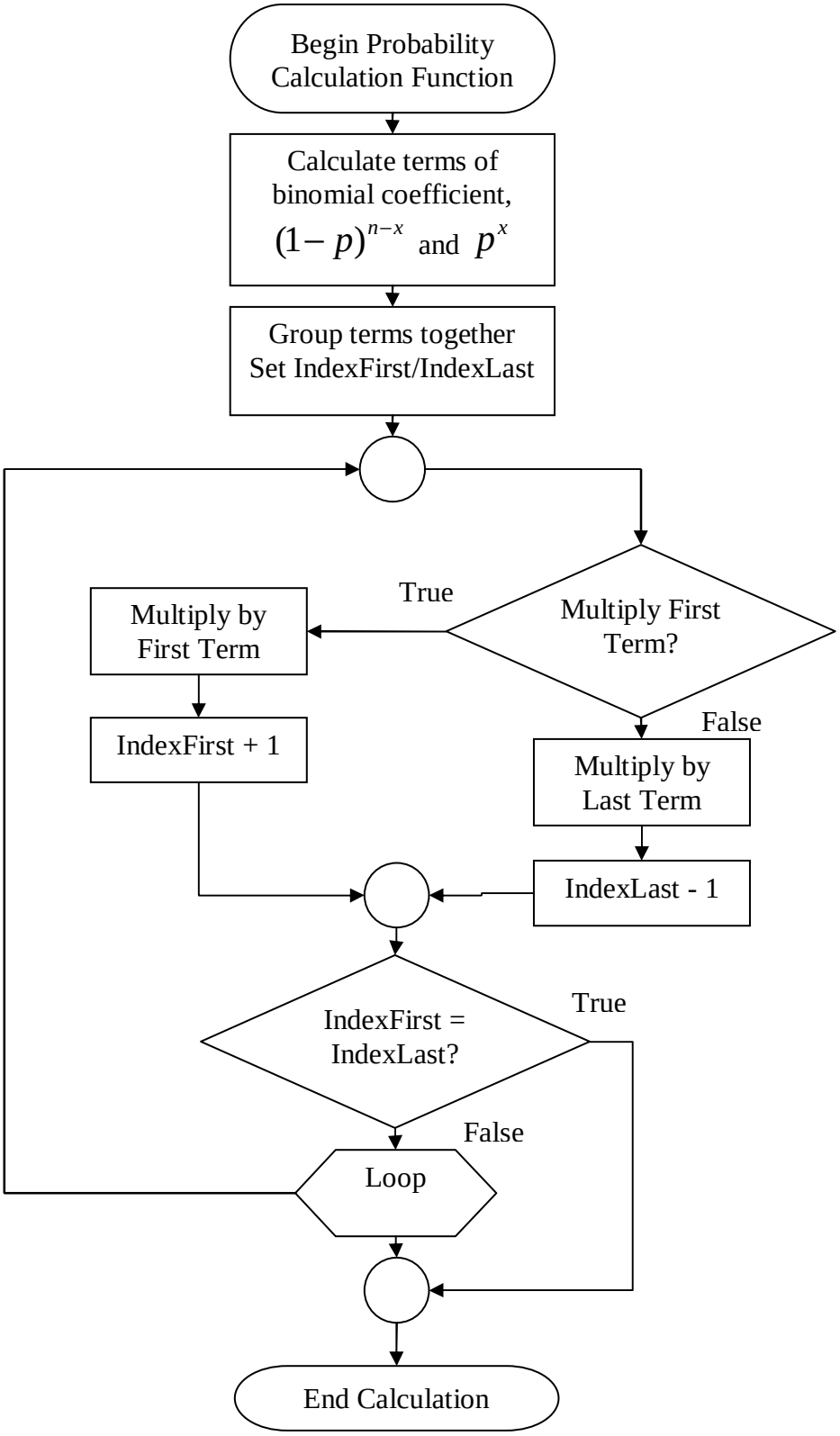

Figure 2.2: Wooters Algorithm to Calculate Binomial Distribution

The example in this section has been using an n of 10 samples and a mean probability of 0.4. No x success exists with a probability less than the threshold 1E-16. Instead an alternate example where n = 100 samples and a mean probability of 0.5 is used to illustrate this idea. The mean or expected number of samples is 50, which has a calculated probability of 0.0796. From there, move both above and below the expected mean of 50 to find the first x success number that has a calculated probability below the threshold 1E-16. The calculated probability for 10 and 90 successes is found to be 1.37E-17. The algorithm stops here and sets the probability for any number of successes between 0 and 10 and between 90 and 100 to be equal to 0. Figure 2.3 reports the example and shows thresholds below which the probability is set equal to 0.

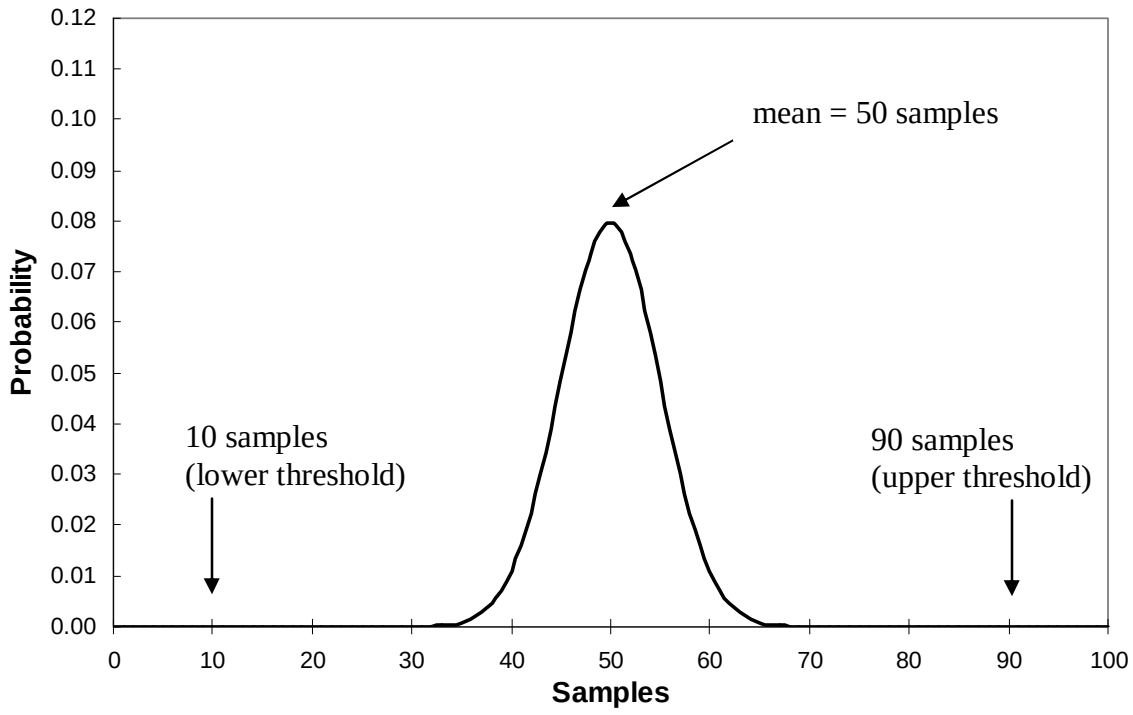

Figure 2.3: Binomial Probability Curve with Mean, Lower and Upper Threshold for Practical Calculation (n = 100, p = 0.5)

This is obviously not explicitly correct, but it is more than sufficient for the requirements of the method this work is developing.

2.2 Window Length and Model Determination Algorithm Improvements

Below is found a discussion on sampling ratio and number of states to provide background on the window length and how to determine the Markov Chain model. Following this introduction is a discussion of the original algorithm proposed by Owusu. Then, two improvements are discussed which deal with the window length and Markov Chain model determination.

2.2.1 Window Length and Model Determination Background

2.2.1.1 Sampling Ratio

Two variables are necessary to sufficiently model the run-length distribution in the controlled process actuating errors, namely the sampling ratio and number of states to be used in the model. The sampling ratio is the ratio of the number of data samplings in the data acquisition and control system (DACS) to the number used for the health monitor. The ratio is initialized at unity, which is equivalent to the controller's sampling frequency. A sampling ratio of three would mean that the health monitor sampled once for every three controller samples. As the sampling ratio increases the effects of persistence and autocorrelation mechanisms are lost, and the signal appears more like random noise. Figure 24 shows a process rejecting disturbances using a PI controller.

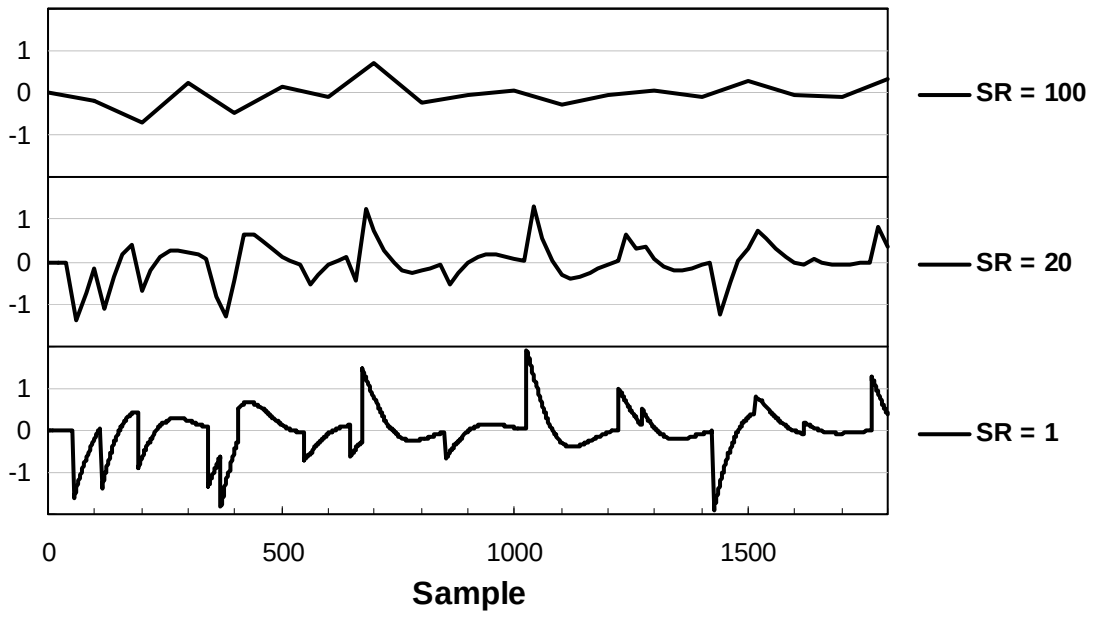

Figure 2.4: Sampling Process Output at Various Sampling Ratios

A sampling ratio of unity matches the data acquisition and control frequency; however, as the sampling ratio increases less autocorrelation from feedback control is present in the sampled signal. Finally, at a sampling ratio of 100, there is essentially no autocorrelation between samples. This is not necessary representative of all systems; the amount of autocorrelation retained in the sampled signal is system dependent, meaning that a sampling ratio does not need to be 20 or 100 to attenuate autocorrelation.

This affect is useful to the current work. The higher the sampling ratio, the more independent each health monitor sample is from its previous sample. This means that the transition probability for each state approaches 0.5 since truly, each actuating error is just as likely to be positive as negative. In general, the further from 0.5 the transition probability is, the closer to either 0.0 or 1.0 that a transition probability limit will be as

determined by the Type I test. The transition probability of 0.0 means that all samples which visit the current state will move to the next higher state. This is the same as saying that no zero crossing can occur from that state. The transition probability of 1.0 means that all samples which visit the current state will next go to the +/-1 state which signifies a zero crossing every time. These are not limits, since a transition probability cannot be either below 0.0 nor above 1.0; therefore, these transition probabilities are avoided. The means whereby extreme transition limits are avoided by Owusu is by forcing the nominal transition probabilities to between 0.25 and 0.75 for every state. This keeps the limits calculated from the Type I step away from 0.0 and 1.0.

2.2.1.2 Number of States

In addition, the Markov chain requires that a specified number of states be fixed. The health monitor is initialized with 8 states (4 for each of the "+" and "-" runs). The health monitor must be initialized with more than 2 states, and all systems under study in this work require more than 6 total states, therefore, the monitor is initialized with 8 states to improve program execution efficiency. Figure 2.5 shows the example of an 8 state Markov chain where the +/-4 state is called the extreme state. For a system whose actuating errors are close to 0.5 the number of samples which visits each higher state (run length) should be half the lower state.

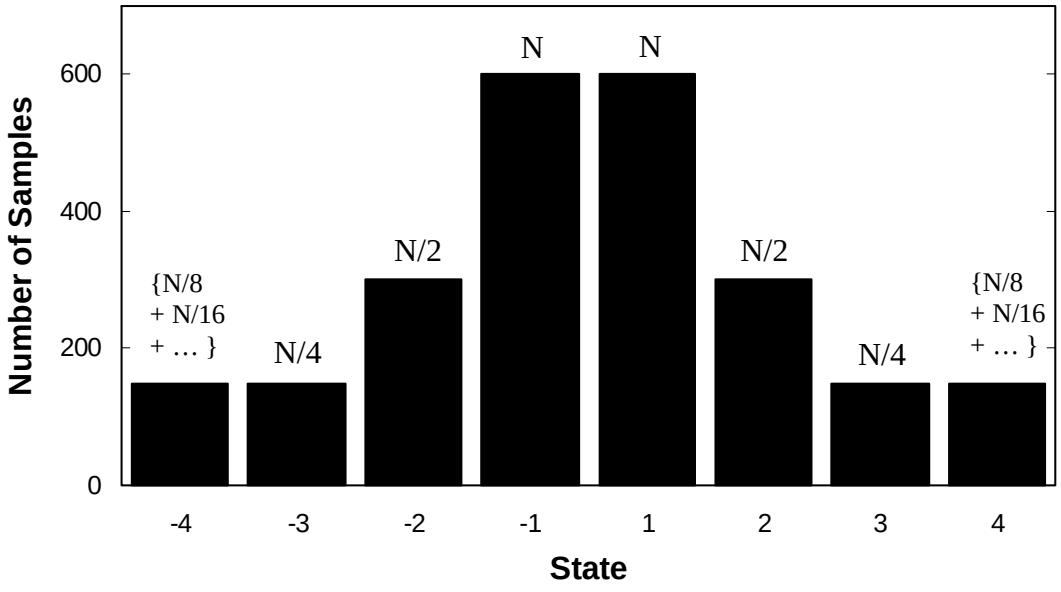

Figure 2.5: Example Sample Distribution for a Markov Chain of 8 States

The transition probabilities will be nearly 0.5 for all states in a perfectly controlled system. If N represents the number of samples found in the +/-1 states, and each higher state has half its predecessor, the extreme states for a Markov chain of 8 samples would each have N/4 samples. This is calculated from the infinite series {N/8 + N/16 + N/32+…}. If the extreme states each contained N/4 samples, this results in 12.5% of all samples residing in the extreme states. Again, this is assuming the case where the Markov chain contains 8 states and the transition probabilities for each state is 0.5. Owusu proposes that approximately 10% of the sample distribution should be found in the extreme states. This value is used to determine the number of states to be used in the chain.

In determining the model framework, there are two assumptions made based on the preceding discussion about sampling ratio and number of states.

1. Transition probabilities for all states must be between 0.25 and 0.75
2. The extremes state should not contain more than 10% of the sample distribution

Neither of these two assumptions are very well supported; however, specific values are required to determine the Markov chain model.

2.2.2 Window Estimation Algorithm – Owusu

In the chemical process industry, it is often not possible or practical to tune loops to their singular optimal control. Long time delays, manipulated variable limitations and the consideration of downstream affects can prevent a loop from being perfectly controlled. What is optimum for one loop might be wrong for another loop where slower change is preferred. Therefore, the employee who maintains a particular loop or set of loops is the judge as to whether a loop is well tuned or not. The health monitor requires good data from which to judge future controller action. This data is most likely to be obtained immediately after a loop is retuned. The set of good data should contain only normal process behaviors, meaning that there is no reason to exaggerate controller performance by making unnecessary step changes during this period. The health monitor is able to measure the controller performance run length distribution from normal process data because the controller is continually acting, even to simple noise.

After the desired good data period is determined, the user specifies the desired Type I error (α) and Type II error (β) which requires yet a third variable (λ). The third variable

(λ) represents the change in mean transition probability that the user would like to distinguish from the null mean. The health monitor initializes the sampling ratio (SR) at unity and the total number of states at 8.

The health monitor analyzes the entire set of data representing good control. All run lengths are counted and, at least initially, every run length of more than 4 samples is put into the extreme bins of +/-4. During the run length counting phase, an additional set of 8 bins are used to keep track of how many times a run makes a zero crossing after visiting one of the 8 total states. From the total run length count and the zero crossing count, the null transition probability is determined for positive runs by

$$P_{+i} = \frac{T_{+i,-1}}{T_{+i}}, \tag{2.22}$$

and for negative runs by

$$P_{-i} = \frac{T_{-i,+1}}{T_{-i}}. \tag{2.23}$$

If any P_{+i} or P_{-i} is outside of the bounds 0.25 to 0.75, then the sampling ratio is incremented by 1. The entire set of data representing good control is once again analyzed and transition probabilities recalculated. This is repeated until all transition probabilities are between 0.25 and 0.75.

Next, the percentage of samples which visited the extreme states is calculated, and if it is found to be more than 10%, the number of states is increased by 2, one extra state for

each of the “+” and “-” runs. The entire data set is reanalyzed and checked against the transition probability bounds of 0.25 and 0.75 and percent of samples in the extreme states. This continues until the assumption 1 and 2 are met. Figure 2.6 outlines the steps necessary to determining the correct sampling ratio and number of states.

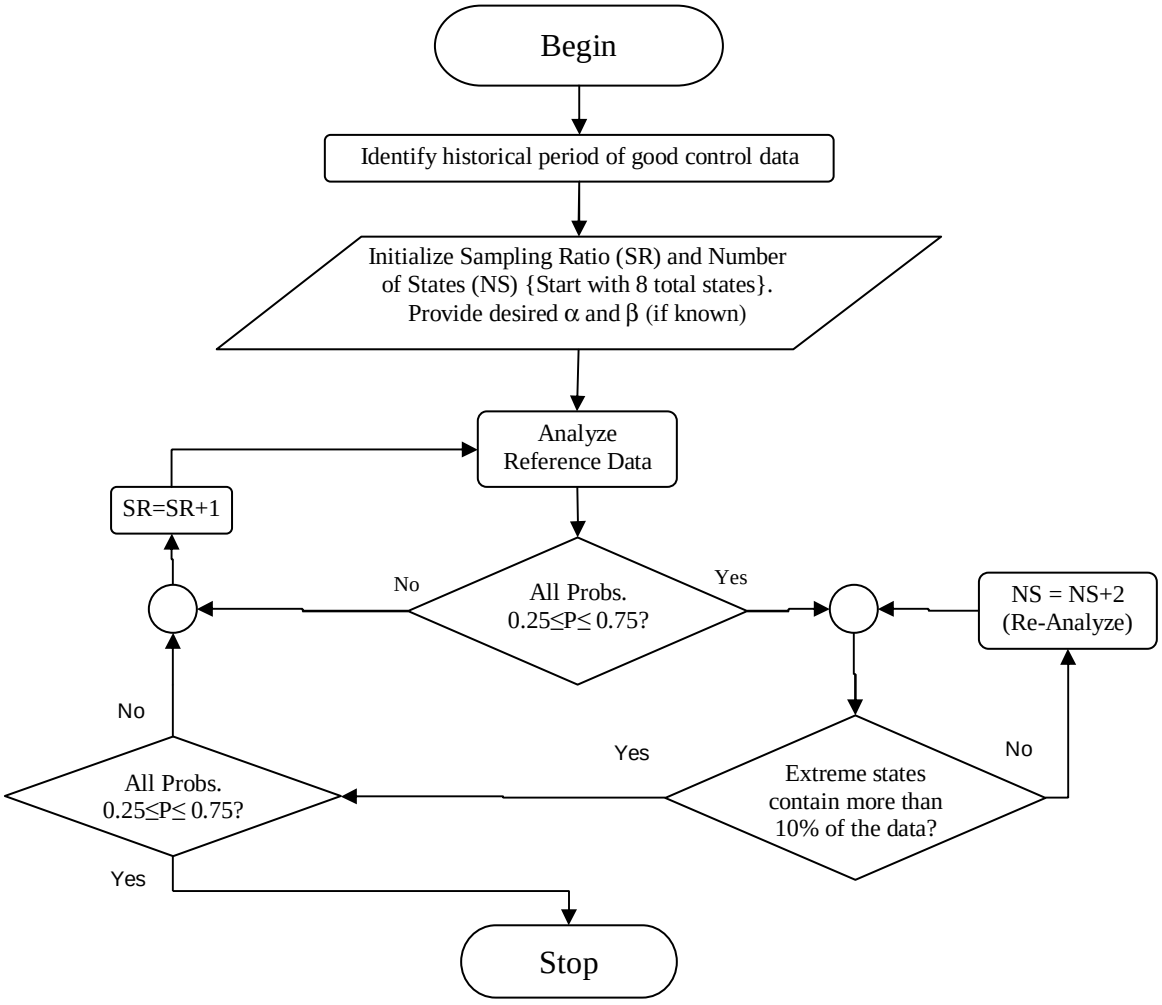

Figure 2.6: Owusu Window Estimation Algorithm

Now that the model has been determined, a window length, to be used in real-time monitoring is needed. This window length must provide sufficient data to be statistically accurate. The health monitor then finds the state on both the “+” and “-” side which

contains the fewest samples. These two states will serve as a basis state for which the Type II test is performed. The Type II test will determine the required number of samples needed to visit these two states to satisfy the statistical requirements provided by the user.

The user has provided α and β; now called α_T and β_T meaning they are the Type I and Type II errors for the total Markov chain. Limits are calculated around each state therefore there are N_s (number of states) possible places where a limit might be violated. Let α_k and β_k be the Type I and Type II error associated with each individual state. If each state is independent, which has already been stated to not be completely true, then the product of the α_k for each state must be equivalent to the total α_k. This is also true for β_k. If the transition probabilities for each state are independent then the following is provided as the relationship between α_k and α_T

$$\alpha_k = 1 - \sqrt[N_s]{1-\alpha_T}\ , \tag{2.24}$$

and the relationship between β_k and β_T

$$\beta_k = 1 - \sqrt[N_s]{1-\beta_T}\ . \tag{2.25}$$

This is just an approximation since states are not independent.

The health monitor initializes each state with 5 samples (n) and calculates the null binomial distribution, a high transition probability change and a low transition probability change (using the supplied λ). Since the null binomial distribution has two tails, α_k is halved. In each of the following equations "n" is the total number of trials (or samples)

and x is a particular success (or sample # within the set of n). The following equation is used to calculate the high limit (x_H)

$$1-\frac{\alpha_k}{2}=\sum_{x=0}^{x_H}\binom{n}{x}P_o^{\,x}(1-P_o)^{n-x} \tag{2.26}$$

and the following is used to calculate the low limit (x_L)

$$\frac{\alpha_k}{2}=\sum_{x=0}^{x_L}\binom{n}{x}P_o^{\,x}(1-P_o)^{n-x}\ . \tag{2.27}$$

After the health monitor calculates the high and low limits based on α_k, then β_k is found for the upper and lower transition probability change using

$$\beta_k=\sum_{x=0}^{x_H}\binom{n}{x}P_u^{\,x}(1-P_u)^{n-x} \tag{2.28}$$

for the high alternate hypothesis and

$$1-\beta_k=\sum_{x=0}^{x_L}\binom{n}{x}P_l^{\,x}(1-P_l)^{n-x} \tag{2.29}$$

for the upper alternate hypothesis. The Type II error statistic provided by the user is compared β_k with the number of samples "n". If the calculated β_k meets the user specification, then "n" is the number of samples which must visit this state, if not, then n is incremented by 1 until this requirement is met.

When the number of samples required to visit each of the two base cases (one for "+" side, one for "-" side) is determined then the total statistical window can be determined. The samples which visit the base case states must have come from a lower state and will next either visit a higher state (longer run) or will move to +/-1 (zero-crossing). Using the

base cases, the number of samples which must visit the remainder of the states can be determined. For all but the extreme state use Equation (2.30)

$$n_{+i+1} = n_{+i}\left(1 - P_{+i}\right) \tag{2.30}$$

and rearranged the following is also useful as Equation (2.31)

$$n_{+i-1} = \frac{n_{+i}}{\left(1 - P_{+i-1}\right)}. \tag{2.31}$$

Simply stated, this means that the runs that do not make a zero-crossing will visit the next state. These equations are also true for the negative side as Equation (2.32)

$$n_{-i-1} = n_{-i}\left(1 - P_{-i}\right) \tag{2.32}$$

and Equation (2.33)

$$n_{-i+1} = \frac{n_{-i}}{\left(1 - P_{-i+1}\right)}. \tag{2.33}$$

For the extreme states a different equation is used in Equation (2.34)

$$n_{+E} = \frac{n_{+E-1}\left(1 - P_{+E-1}\right)}{P_{+E}} \tag{2.34}$$

and rearranged as Equation (2.35)

$$n_{+E-1} = \frac{n_{+E}P_{+E}}{\left(1 - P_{+E-1}\right)}. \tag{2.35}$$

Also, on the negative side is Equation (2.36)

$$n_{-E} = \frac{n_{-E+1}\left(1 - P_{-E+1}\right)}{P_{-E}} \tag{2.36}$$

and also rearranged as Equation (2.37)

$$n_{-E+1} = \frac{n_{-E} P_{-E}}{\left(1 - P_{-E+1}\right)}. \tag{2.37}$$

By utilizing these equations, the base case defines each half of the Markov chain by expected number of samples.

The transition probability limits for the base cases are calculated in the previous step. To calculate the limits for the other states, the health monitor solves Equation (2.38)

$$1 - \frac{\alpha_k}{2} = \sum_{x=0}^{x_H} \binom{n}{x} P_o^x (1 - P_o)^{n-x} \tag{2.38}$$

for the high limit where x_H is the unknown and Equation (2.39)

$$\frac{\alpha_k}{2} = \sum_{x=0}^{x_L} \binom{n}{x} P_o^x (1 - P_o)^{n-x} \tag{2.39}$$

for the lower limit where x_L is the unknown and n is the number of samples expected to visit this state.

With the limits now calculated and the expected sample visits known for all states, the total statistical window is determined. The statistical window is simply the sum of the expected samples over all states. Finally, the user provided settling time is added to the statistical window. The complete window (statistical window + settling time) is used in the real-time monitoring to flag the process when a transition probability limit has been violated. Figure 2.7 outlines the final steps used after the Markov chain model is

determined by finding the sampling ratio and settling time and up to the calculation of the complete window.

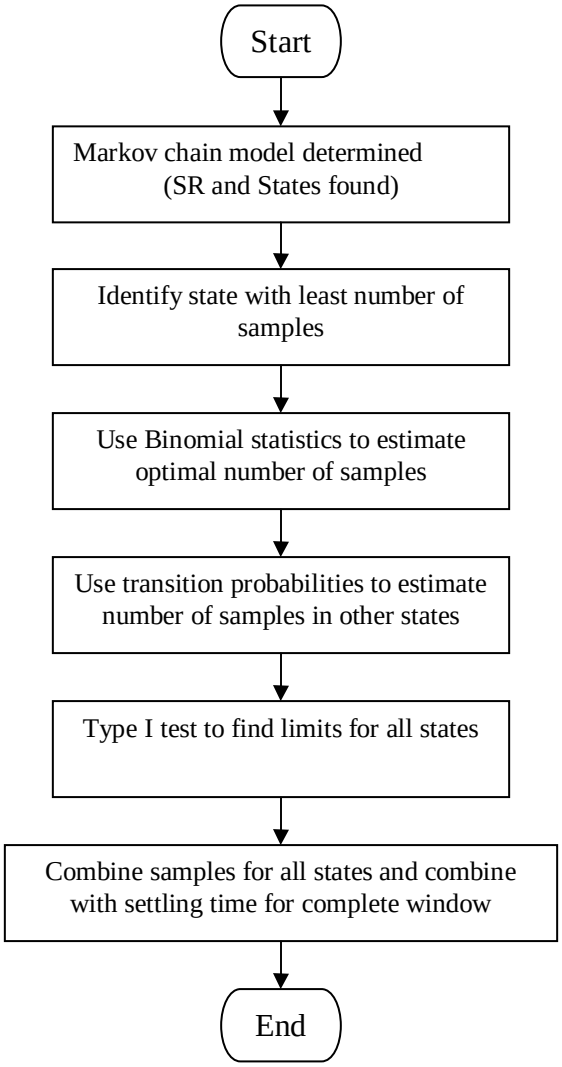

Figure 2.7: Steps from Markov Chain to Complete Window Length

An example is useful in conveying the steps as outlined in Figures 6 and 7. Consider the case of a secondary controller whose setpoints are determined by the primary controller. Figure 2.8 shows just such a system which is well tuned and acting in “good” control.

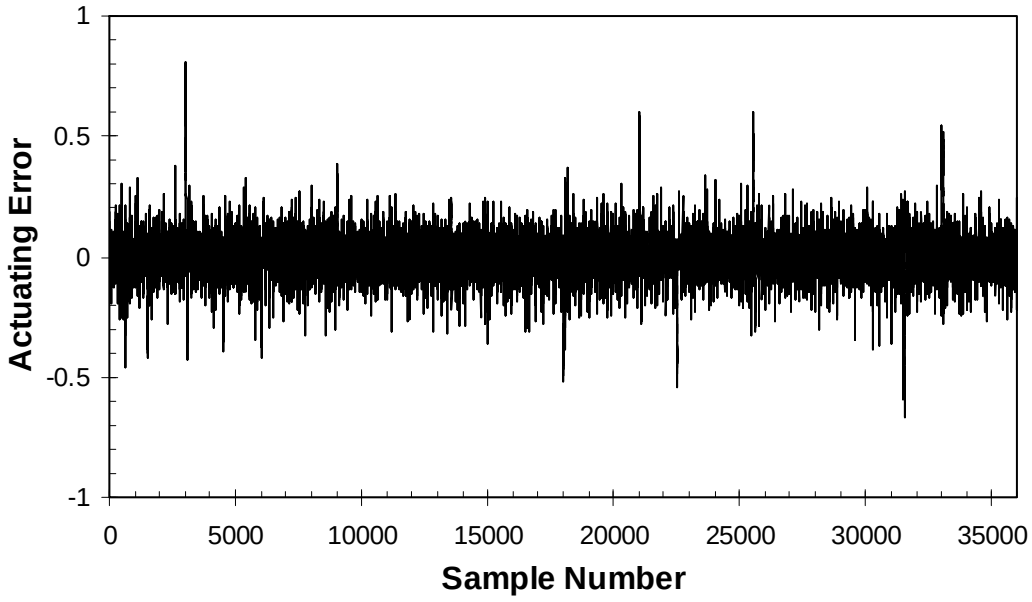

Figure 2.8: Example Actuating Error from a Process Under "Good" Control

The user provides the 36,000 "good" control data points to the health monitor. The user specifies $\alpha = 0.10$, $\beta = 0.10$, $\lambda = 0.90$ and settling time = 20 samples (although real-time testing will not be performed in this example). The health monitor initializes the sampling ratio at unity and number of states at 8. The following series of figures show the progression as sampling ratio and number of states is increased. The transition probabilities (line) pertain to the left-axis and the number of samples (bars) pertains to the right-axis.

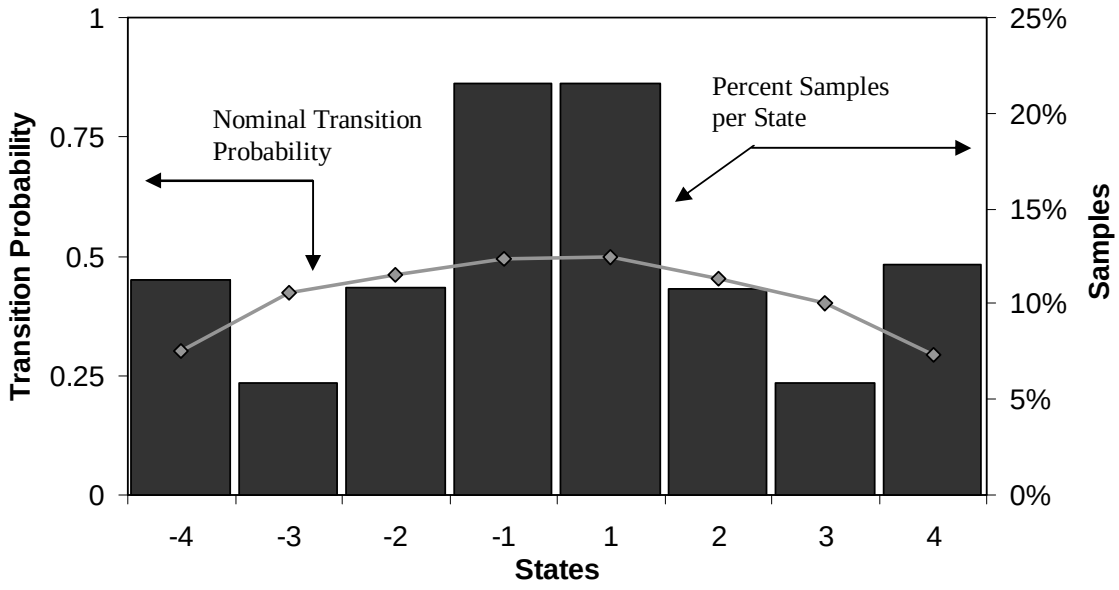

Figure 2.9: Markov Chain Model of Example Data (SR = 1, States = 8)

In Figure 2.9, with a sampling ratio of unity and number of states at 8, all transition probabilities fall within the 0.25 and 0.75 limits, fulfilling the first condition. However, the extreme states contain 23.3% of the samples, which violates the second condition that no more than 10% of all samples be found in the extreme states. Therefore, according to the algorithm outlined in Figure 2.6, the sampling ratio remains constant, but the number of states increases by 2 (1 for each "+" and "-" side). The results are found in Figure 2.10.

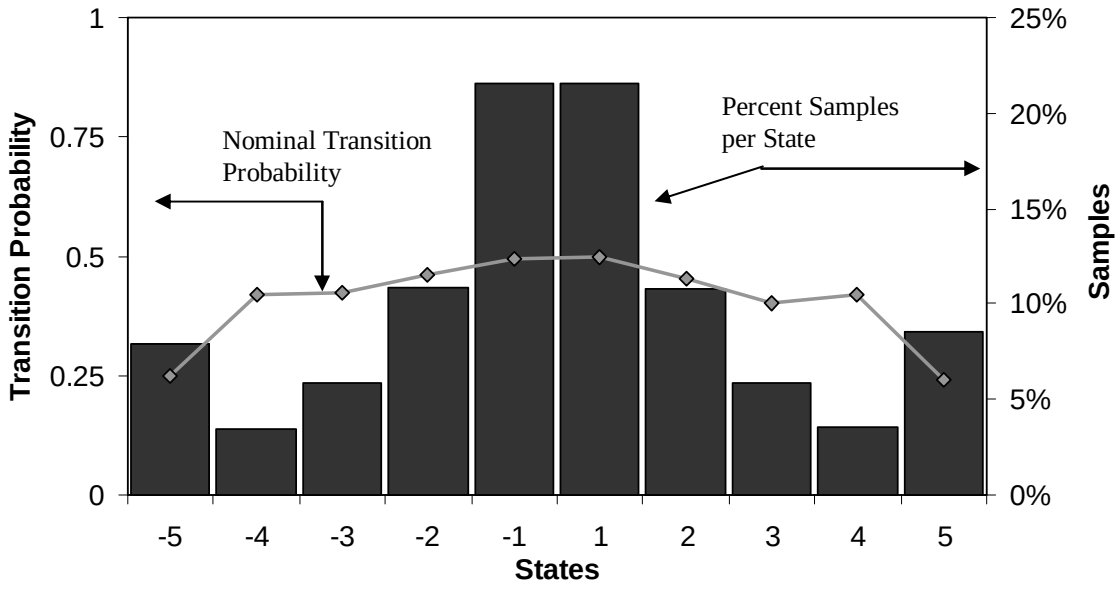

Figure 2.10: Markov Chain Model of Example Data (SR = 1, States = 10)

Figure 2.10 reports the result of increasing the number of states from 8 to 10. There are still 16.4% of all samples visiting the extreme states. Therefore, the sampling ratio is kept at unity and the number of states is increased from 10 to 12. Figure 2.11 reports these results.

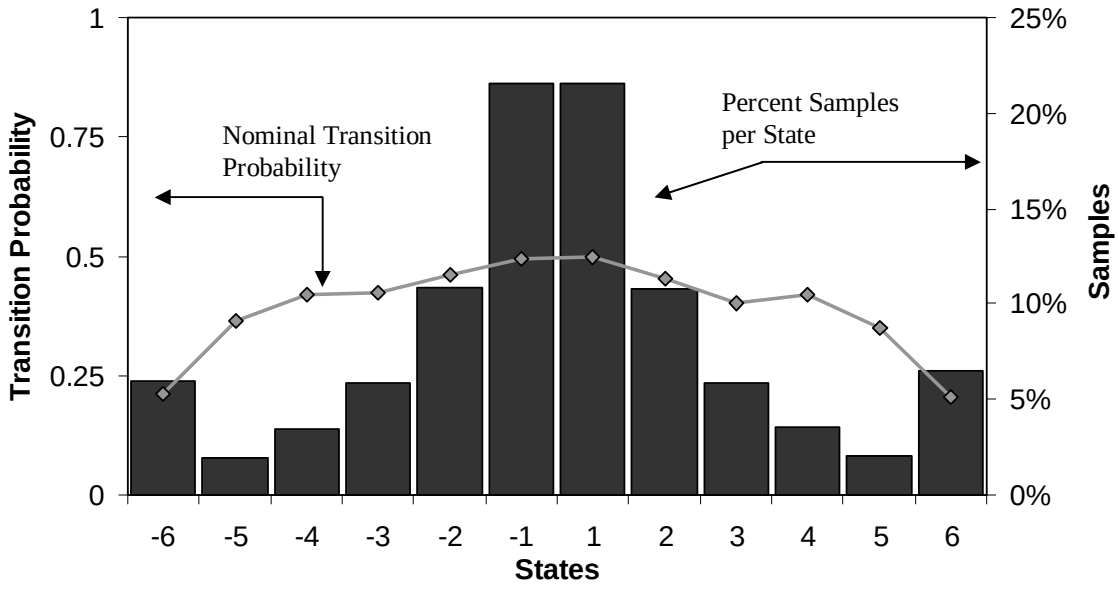

Figure 2.11: Markov Chain Model of Example Data (SR = 1, States = 12)

Figure 2.11 reports the results of increasing the number of states from 10 to 12. The percentage of samples now found in the extreme states has dropped to 12.4% which still violates the limit of 10%. The algorithm outlined in Figure 2.6 is continued until both conditions are met. Figure 2.12 is a matrix of percent of samples in the extreme states, lowest and highest transition probability for all states. These results are defined by the selected sampling ratio (Rows) and the number of states (Columns). The shaded regions are those results which violate one of the two conditions whether for percent of samples in the extreme states or transition probabilities.

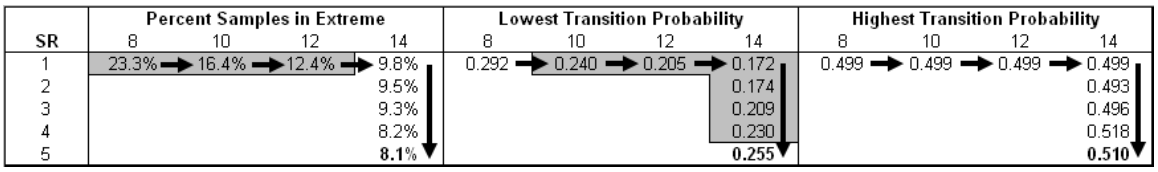

Figure 2.12: Matrix of Results from Changing Sampling Ratio (SR)

and Number of States

The arrows pointing to the right in Figure 2.12 lead the reader in following the first 3 steps taken from Figures 2.9 through 2.11 using this matrix. From Figure 2.9, the sampling ratio is unity and 8 states are used; 23.3% of the samples visited the extreme states and the lowest and highest probabilities reported, 0.292 and 0.499, are within the limits 0.25 and 0.75. From Figure 2.9, the sampling ratio is unity and 10 states are used; 16.4% of the samples visited the extreme states and the lowest and highest probabilities reported are 0.240 and 0.499 respectively. From Figure 2.10, the sampling ratio is 1 and 12 states are used; 12.4% of the samples visited the extreme states and the lowest and highest probabilities reported are 0.205 and 0.499 respectively.

The algorithm actually visits and rejects each combination listed in the matrix found in Figure 2.12 until the correct combination is obtained. The final Markov chain which fulfills both conditions is found where the sampling ratio is 5 and 14 total states are used. Figure 2.13 reports this final model with number of samples found in each state.

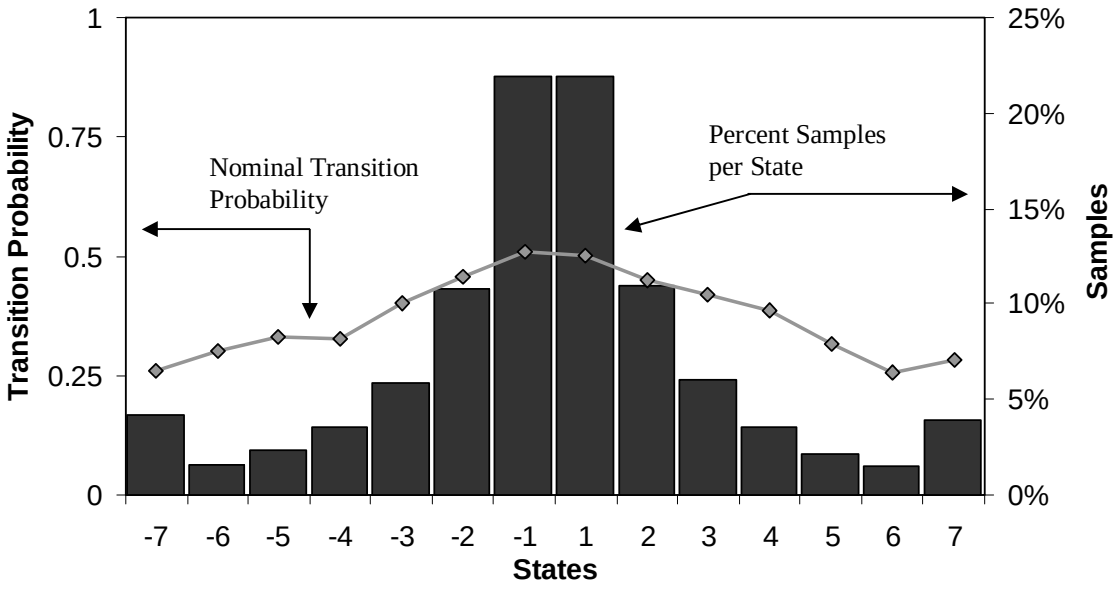

Figure 2.13: Markov Chain Model of Example Data (SR = 5, States = 14)

Notice that there are one fifth the total number of samples available to the final model as the initialized case where the sampling ratio is unity.

Now that the Markov chain model is determined, the algorithm found in Figure 2.7 is followed to find the transition probability limits and the statistical window length to be used for real-time monitoring. The base cases for this chain are the +/-6 states since those states contain the fewest visits for each chain half. The -6 state whose transition probability is 0.301 will be explored for the benefit of this example. As a reminder, the user has specified $\alpha = 0.10$, $\beta = 0.10$, $\lambda = 0.90$, therefore the low alternate transition probability is 0.0301 and the high alternate transition probability is 0.9301. Figure 2.14 reports the initial binomial distributions for all three transition probabilities starting with 5 samples.

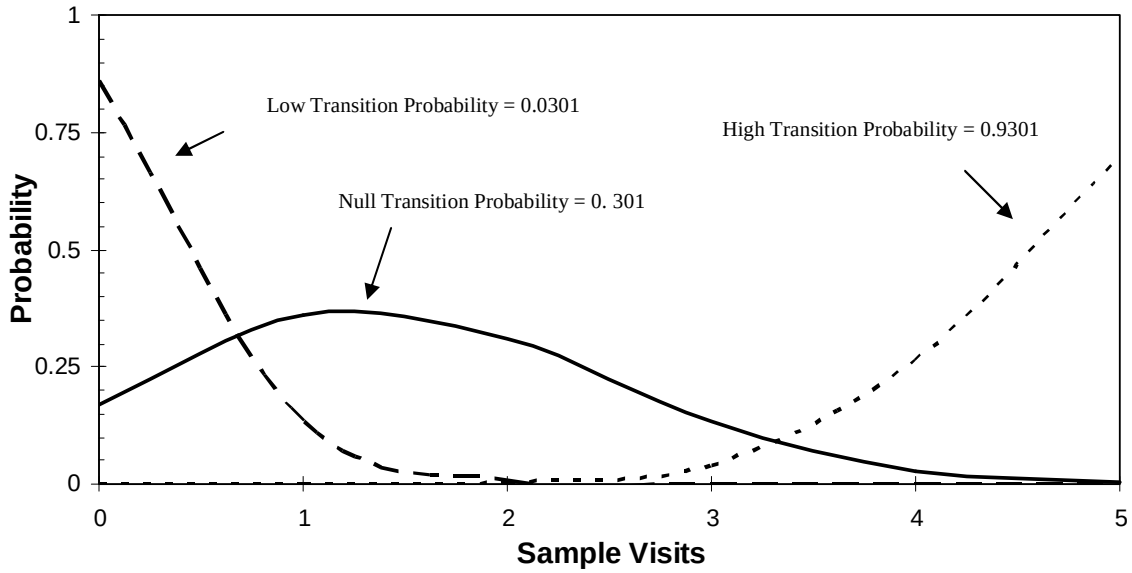

Figure 2.14: Binomial Distributions for Low, Null and High Transition Probabilities (n = 5, $\alpha = 0.10$, $\beta = 0.10$, $\lambda = 0.90$)

When the total number of visits n = 5, there is exists no success x which will satisfy the Type I test. Furthermore, since no limits may be obtained from the Type I test, the Type II test fails by virtue that all successes x belonging to the lower and higher alternate transition probability fall within the Type I test limits.

If the total number of visits n is increased to 20, the binomial distributions begin to separate as seen in Figure 2.15.

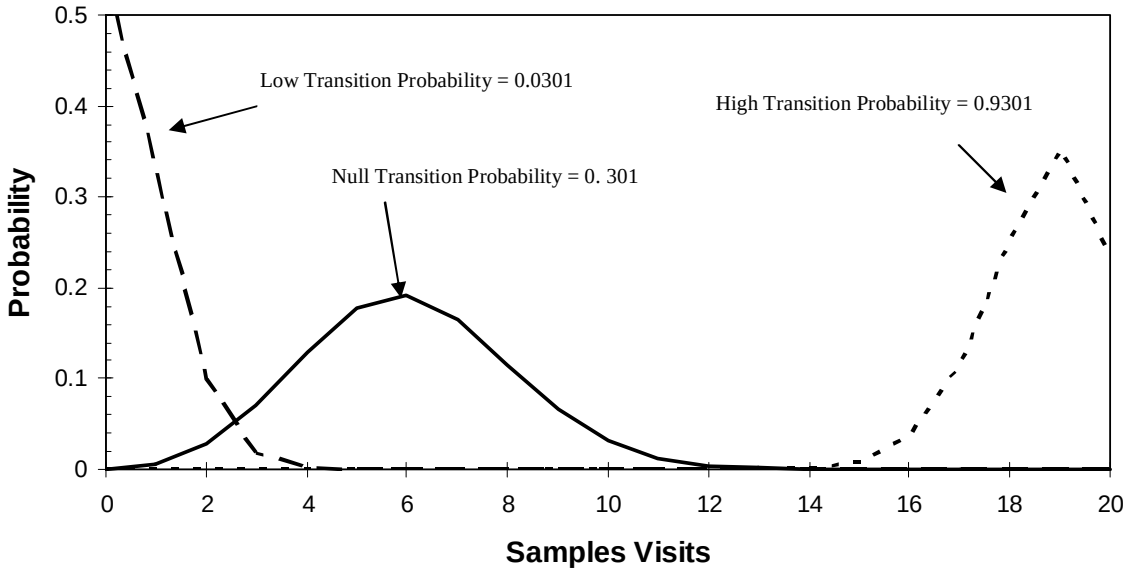

Figure 2.15: Binomial Distributions for Low, Null and High Transition Probabilities ($n = 20$, $\alpha = 0.10$, $\beta = 0.10$, $\lambda = 0.90$)

Figure 2.15 shows that there may be enough separation between the high and null mean transition probability that an upper limit may be obtained by a Type I test; however, a lower limit is still unobtainable, therefore no limits are show in Figure 2.15.

The number of sample visits n is increased until the α and β statistics for both the Type I and Type II tests are met. Figure 2.16 reports the final binomial distributions for both curves. Also shown, are the upper and lower transition probability limits.

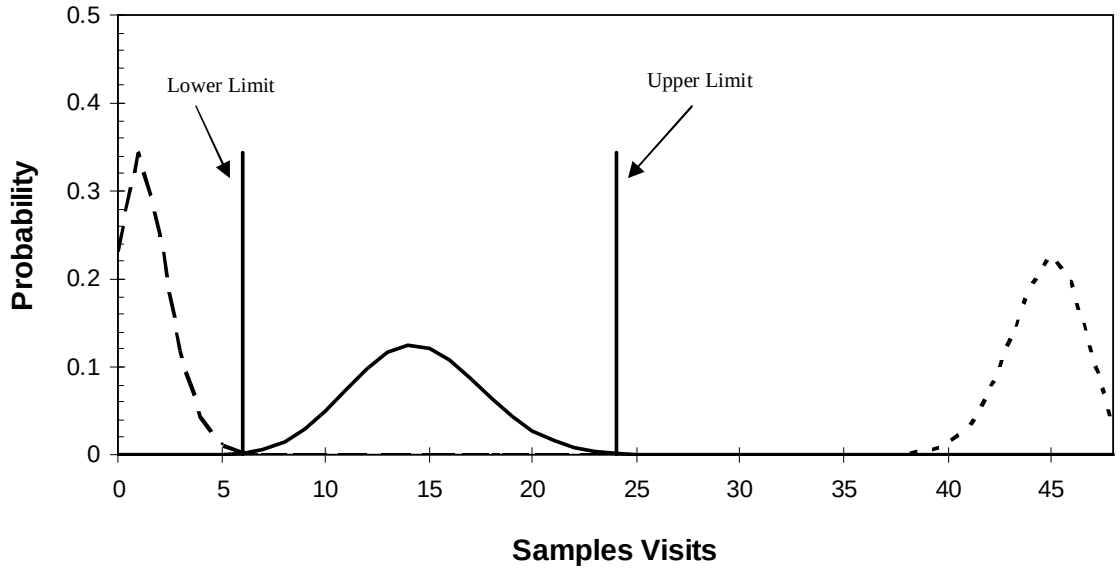

Figure 2.16: Binomial Distributions for Low, Null and High Transition Probabilities (n = 48, $\alpha = 0.10$, $\beta = 0.10$, $\lambda = 0.90$)

According to the binomial statistics, 48 samples must visit state -6 to meet both α and β conditions. The low transition probability limit is 6 samples ($6/48 = 0.125$) and the high transition limit is 24 ($24/48 = 0.500$). Limits are also calculated for +6 then used to find the samples expected in each of the other states.

Using the Equations (2.30) through (2.37), any state may be used to define the rest of the states. The original Owusu method calls for estimating the number of samples that must visit each positive and negative half of the chain, based on the base case on either side. The results shown in Figure 2.16 are done with the positive base case of +6 to find that a minimum of 58 samples must visit this state. The number of samples expected to visit the entire chain is now reported in Table 2.1.

Table 2.1: Expected Samples per State (base states -6, +6)

State	-7	-6	-5	-4	-3	-2	-1	+1	+2	+3	+4	+5	+6	+7
Samples	130	48	72	107	178	329	670	863	432	239	139	85	58	153

Notice that the +/- 6 states contain the number of samples previously calculated from the Type II test. Finally, upper and lower limits are calculated using a Type I binomial distribution test. With the number of samples to visit each state known, the binomial distribution is calculated and the limits found by Equations (2.38) and (2.39). Figure 2.17 reports the transition probability limits calculated for each state.

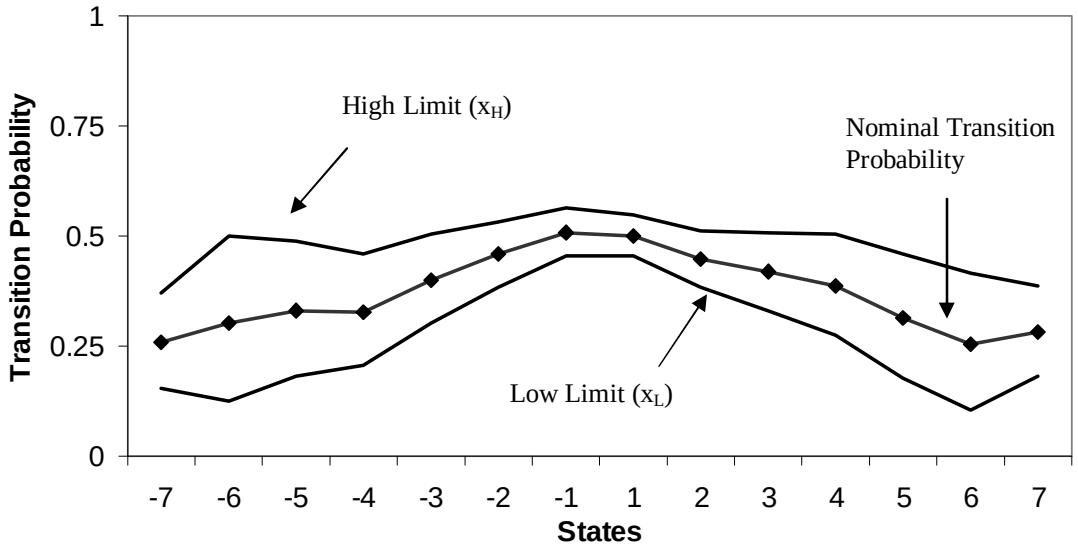

Figure 2.17: Transition Limits Surrounding Nominal Transition Probability

(Original Algorithm, $\alpha = 0.10$, $\beta = 0.10$, $\lambda = 0.90$)

The statistical window is then the sum of the samples over the entire Markov chain; in this case 3503 samples are required for the Type I and Type II statistics to be valid. The

settling time in number of samples (SR = 5) is now added to the statistical window to create a counter used in the real-time monitoring portion. For a settling time requiring 20 samples at a sampling ratio of 1 (equal to the DACS sampling frequency), it would require just 4 samples at a sampling ratio of 5. Therefore, the complete window (statistical window + settling time) is 3507 samples. This requires 17535 DACS samples to obtain.

In conclusion, the health monitor analyzes good control data and populates a Markov chain used to characterize these data. Transition probabilities are calculated based on the propensity that a run must end after a certain number of samples. Upper and lower limits to the transition probabilities are calculated using binomial statistics.

Finally, a model with transition limits and a required number of samples (statistical window) is used in real-time monitoring. The monitor begins collecting data as soon as the good control data has been characterized. The complete window length (statistical window + settling time) serves as a counter threshold value. The samples in the statistical window only are used to build a Markov chain of equal chain size as the good control data model. If any of the transition probability limits are violated, the counter begins. When the counter reaches the length of the complete window length, then the process is flagged, retroactively to the point when counting began. This designates the sample which started to move the transition probabilities outside of the limits. As soon as the transition probability for each state is once again inside the transition limits, the counter resets to 0. It is common for a well-controlled process to periodically start the monitor

counting since Type I errors are always possible, but the user specifies how probable. If the user wants fewer errors, the statistical window length must increase to collect more data. This allows the monitor to be more "sure" before declaring a limit violated.

2.2.3 Window Estimation Algorithm – Wooters

Two short comings are found in the above method:

1) Two conditions of the original work are difficult to support theoretically, called Markov Chain conditions in the next section.
    a. Nominal transition probabilities must lie between 0.25 and 0.75
    b. No more than 10% of the sample visits may be to the combined extreme states
2) The state on either the positive or negative side of the chain with the fewest samples may not be the true base state; it is a combination of number of samples and state transition probability. This is discussed under the Window Length Basis Determination later.

Finally, an improved algorithm is proposed after the limitations of the original methods are illustrated.

2.2.3.1 Markov Chain Conditions:

The final result of the good data analysis is a set of transition probability limits as shown in Figure 2.17 and the minimum number of samples needed to fulfill the user specified statistical tests. Unfortunately, as of yet, there is no way to ensure that the shortest window possible is obtained. However, if the first shortcoming is relieved, the health

monitor now has a degree of freedom with which it can minimize the statistical window (with respect to DACS samples) and still fulfill the user specified statistical requirements.

The first requirement of maintaining all transition probabilities between 0.25 and 0.75 is superfluous. The goal in this condition was to keep the final calculated limits from resting on 1.0 and 0.0, because there would be no way of violating these limits they would be useless. This end is more easily achieved by simply not allowing a model where limits would rest on these outermost bounds. Indeed, the Type II test will ensure this by increasing the number of samples required to separate the null and alternate hypothesis until the beta statistical requirement is met. For the final limits to rest on 1.0 or 0.0 would mean that the user specified a beta so high, that the entire alternate hypothesis fell within the null hypothesis. This situation is neither likely nor logical, as it would mean that 100% of the time the null hypothesis would be rejected when it should not be.

Therefore, it is proposed that the first assumption be dropped and in its place, allow the health monitor to change the sampling ratio to find the window requiring the fewest DACS samples. With all statistical requirements met, this would ensure the fastest notification of poor control.

In addition, the need for no more than 10% of all samples in the extreme states seems is called into question. Nearly all controlled systems will have autocorrelation where each state is dependent on the other. One cannot assume that perfect control for a particular system result in a transition probability of 0.5 for every state. This is the case used to

justify the 10% value. The value was already somewhat arbitrary since the previous derivation actually led to 12.5% of the samples in the combined extreme states and 10% was substituted as an approximation. What is not clear, is what the best value should be. Is 10% better than 15% or 20%? This work does not seek to answer these questions theoretically; however, it is proposed that a value is necessary from the standpoint of state shaping.

To understand this point, consider the extreme case where no limits are provided for the number of samples in the extreme states. Initializing the health monitor with 2 states (1 on the "+" side and one on the "-" side) then all data is found in either the positive or negative state. This chain would have very little sensitivity since all positive runs fall in the +1 state and all negative runs fall in the -1 state. Similarly, if the limit is set very low, say 1% or 2%, many states, perhaps 20 or more might be required. To necessitate so many states, run lengths of 10 or more would need to be found in the good control period. However, with very few samples possible in the extreme states, two problems arise:

1) A larger statistical window becomes required to collect sufficient long runs
2) These extreme states are more sensitive than the states closer to +/-1 since these closer states will have comparatively more samples over which short anomalies might be averaged. Too few long runs and too many long runs would have amplified affects on the extreme states when compared to the lower states.

This work seeks to use the limit of samples in the extreme state to shape the states somewhere between these two extremes. Since 10% seems already smaller than even the

insufficient theoretical value calculated to be 12.5% this work will use 20%. This value seems to allow for multiple states while rarely necessitating large number of states which prove too sensitive. In the final section of Chapter 2, the main advantage to specifying 20% is discussed.

With this distinction made, however, so that the two methods might be compared more directly, 10% is used in the examples that follow.

2.2.3.2 Window Length Basis Determination:

The number of samples expected in each state for the positive and negative half of the chain is assumed to be the state where the fewest samples are found. The base case is determined not only by the number of samples counted in a particular state, but is also a function of the transition probability of said state. Figure 2.18 shows just such an example.

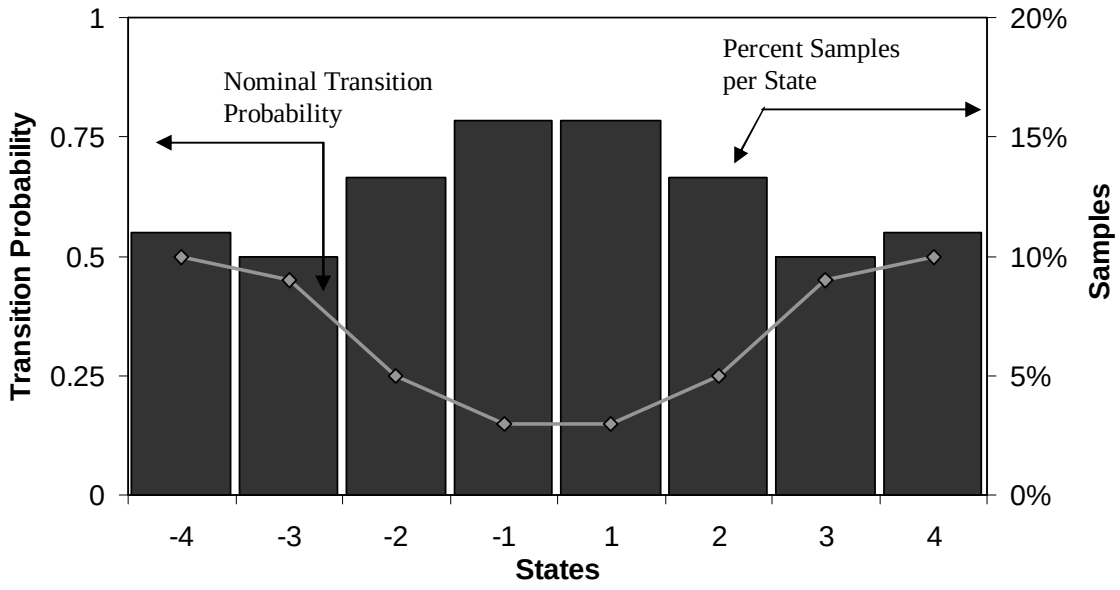

Figure 2.18: Markov Chain Model to Show Basis Problem (SR = 1, States = 4)

For the following analysis assume that $\alpha = 0.10$, $\beta = 0.10$, $\lambda = 0.90$ as the examples above used. Considering the positive side of the Markov chain, the original method would have us choose state +3 to base state since it contains the fewest samples of all the positive states. Consider each step in the algorithm found in Figure 2.7. The transition probability for State +3 is 0.45, meaning that 45% of all run lengths which reach +3 samples make a zero crossing. Binomial statistics would be performed using this state, providing limits that met the user specified α and β. The +3 state requires 25 samples to fulfill the Type II. The expected number of samples to visit each of the other positive states is calculated assuming 25 samples visit State +3 using Equations (2.30) through (2.37). Table 2.2 provides the expected number of samples per positive state.

Table 2.2: Expected Samples per

Positive State (base states +3)

State	+1	+2	+3	+4
Samples	39	33	25	28

Unfortunately, since the Type II test was performed with only the +3 State, it is not immediately evident whether enough expected samples visit remaining states to fulfill the statistical requirements for each state. As a check, calculate the minimum number of samples needed to visit each state to fulfill these requirements reported in Table 2.3. The +3 State has already been calculated.

Table 2.3: Minimum Required Samples per State to Fulfill

Statistical Requirements ($\alpha = 0.10$, $\beta = 0.10$, $\lambda = 0.90$)

State	+1	+2	+3	+4
Samples	95	53	25	23

The +3 and +4 (extreme) States would both have sufficient sample visits to be statistically valid. The +3 State receives the exact minimum required, and the +4 State would have more than sufficient visits. However, the +1 and +2 States would not have enough samples to fulfill the statistical requirements. Therefore, the +3 State could not possibly be the minimum or base state for the positive side of the chain. It is proposed that an alternate algorithm be used to ensure that all states be visited with at least the minimum number required by the Type II test.

2.2.3.3 Improved Algorithm

The previous two claims require an improved algorithm, one which eliminates the arbitrary transition bounds and consistently selects the correct base state when defining the statistical window. Most of the techniques outlined by Owusu are still used, except where they are explicitly discussed in this section. The selection of the sampling ratio and number of states outlined in Figure 2.6 and the binomial statistics used to find the limits and window length in Figure 2.7 must now be combined into a single algorithm. Also, the previous search technique of increasing sample ratio and states based on whether they were with their assumed bounds is modified.

As before, the health monitor user selects a period of good data by which future real-time data will be tested. The user specifies α , β , λ and settling time, and the monitor initializes the sampling ratio at unity and the number of states at 8. If the number of samples in the extreme states is more than 10%, then 2 states are added to the Markov chain (1 on each side).

When the number of states is found for a sampling ratio of unity, Type I and Type II tests are performed on each state given the transition probability found in the previous step. Again, the result of the Type II test is the minimum number of samples which must visit a particular state to fulfill the user specified α and β.

Now, the base state is determined. Each side of the Markov chain will have its own base case from which the rest of each half will be estimated using Equations (2.30) through (2.37). Starting at state +/-1, the minimum number of samples found during the Type II test is propagated down the chain. As each half is being estimated, if a state requires more samples than is being estimated by the current base case, then the proposed base case is discarded. The algorithm then moves down the chain towards the extreme state until the true base state is found. There can only exist one base case per chain half, therefore, as soon as a state is tested and, through propagating along the rest of the chain, it is found to provide enough samples to all other states, the proposed base case is accepted. Figure 2.19 provides a flowchart of the base state selection algorithm.

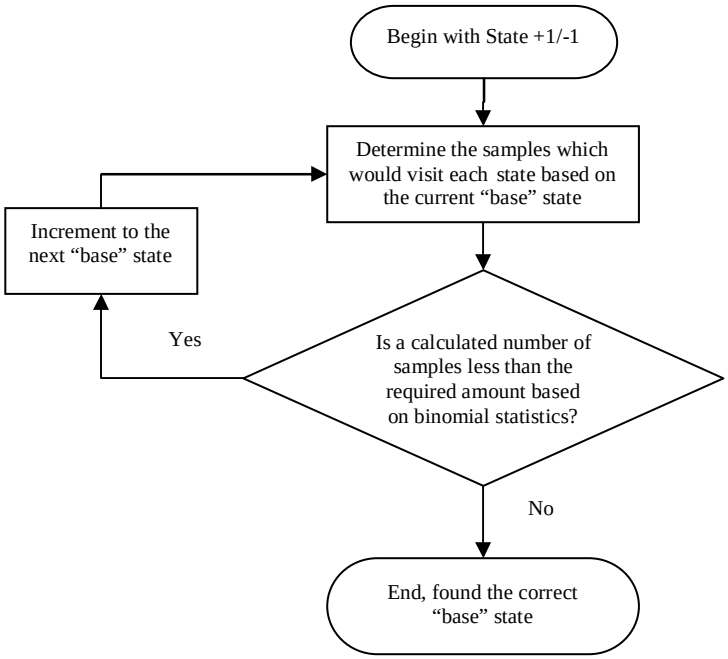

Figure 2.19: Improved Base Case Selection Algorithm

After the base case is found for each the positive and negative half of the Markov chain, then the entire window is estimated. The final statistical window length is found in terms of the number of DACS samples required to obtain enough health monitor samples to fill the window.

The sampling ratio is increased to 2, and the number of states reset to 8. States are added to the chain until once again there are less than 20% of samples found in the extreme states. The Type I and Type II tests are performed over all states and the base case selection algorithm repeated for this new Markov chain. The window length in the arbitrary time unit is stored with this new state and sampling ratio combination.

These steps are repeated by increasing the sampling ratio and storing window lengths until a model is found with 8 states and 10% or less of the total samples in the extreme states. Next, the state and sampling ratio combination which created the shortest time window is selected as the best Markov chain. The short time window allows for quicker decisions regarding real-time control monitoring. Figure 2.20 outlines all the steps found in this improved algorithm.

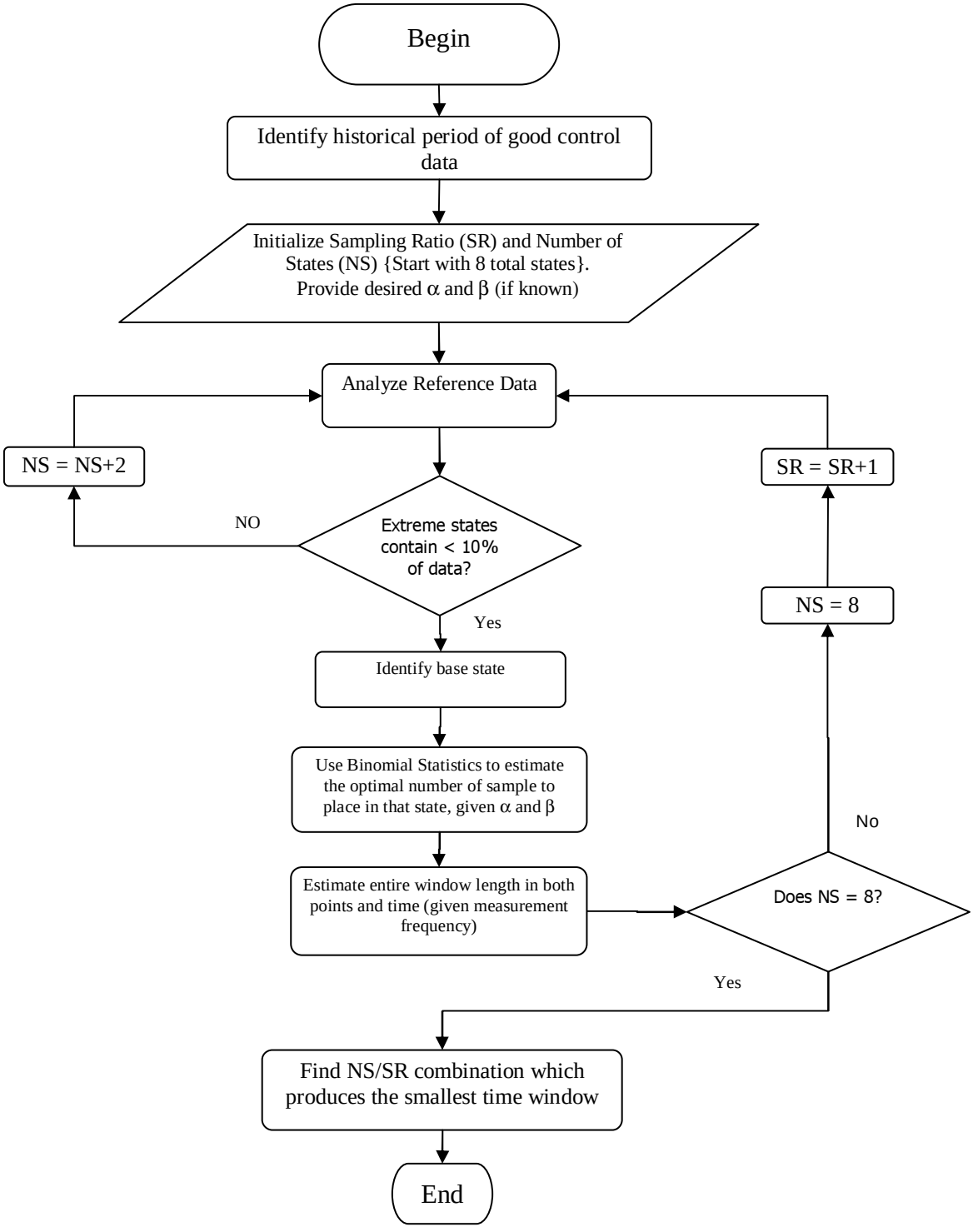

Figure 2.20: Improved Window Estimation Algorithm

As with the original algorithm it is instructive to provide an example. To illustrate the superiority of this method in obtaining the shortest window length, the same example as above is used. To remind the reader $\alpha = 0.10$, $\beta = 0.10$, $\lambda = 0.90$ and settling time is 20 samples of the DACS. As stated before, to compare both methods, no more than 10% of all sample visits may be to the extreme states. Beginning where the sampling ratio is unity and a chain of 8 states, the model and transition probabilities are equal to those found in Figure 2.9. Continuing the algorithm outlined in Figure 2.20 finds that the first number of states where only 10% of the samples fall in the extreme states is a Markov Chain with 14 states. The original algorithm also came to the sampling rate of unity and 14 states. Figure 2.21 shows the samples and transition probabilities.

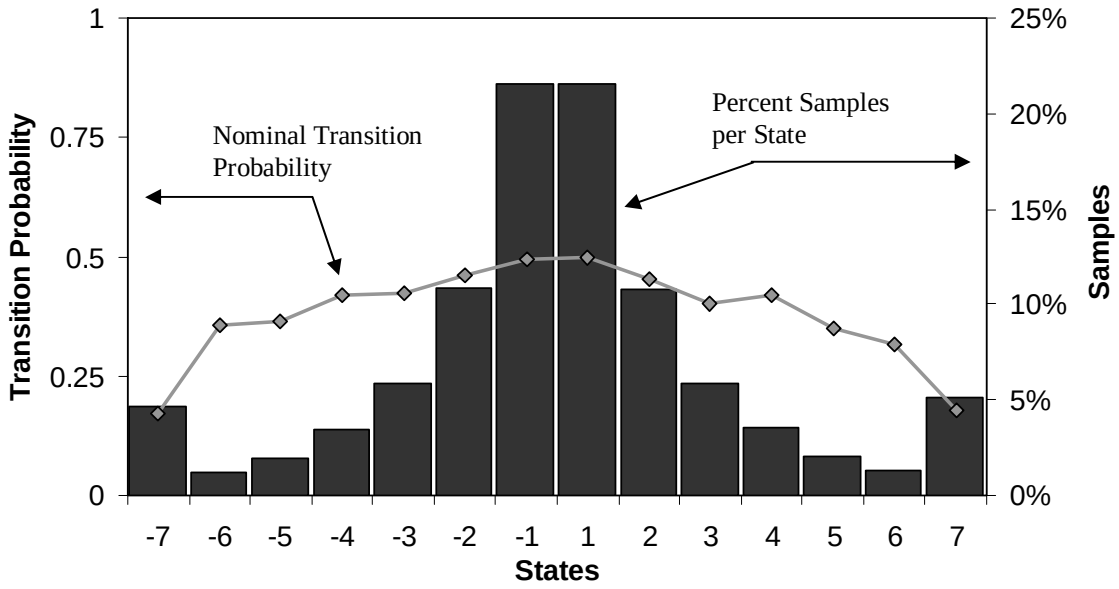

Figure 2.21: Markov Chain Model of Example Data (SR = 1, States = 14)

As determined by the modified algorithm detailed in Figure 2.19 the base states for this model are the +/- 6 States. The number of samples expected to visit each state is

calculated then combined for a statistical window length of 3,335 health monitor samples, equivalent to 3,335 DACS samples. Now add the 20 DACS sample settling time for a complete window length of 3,355 DACS samples.

This is the place where the original algorithm chose to increase the sampling ratio while maintaining 14 states because the extreme states obviously have transition probabilities below 0.25. The improved algorithm in this work increases the sampling ratio and resets the number of states to 8. The next number of states which fulfill the requirement to have no more than 10% of the data in the extreme states is again 14. Figure 2.22 reports this sample distribution and transition probability profile.

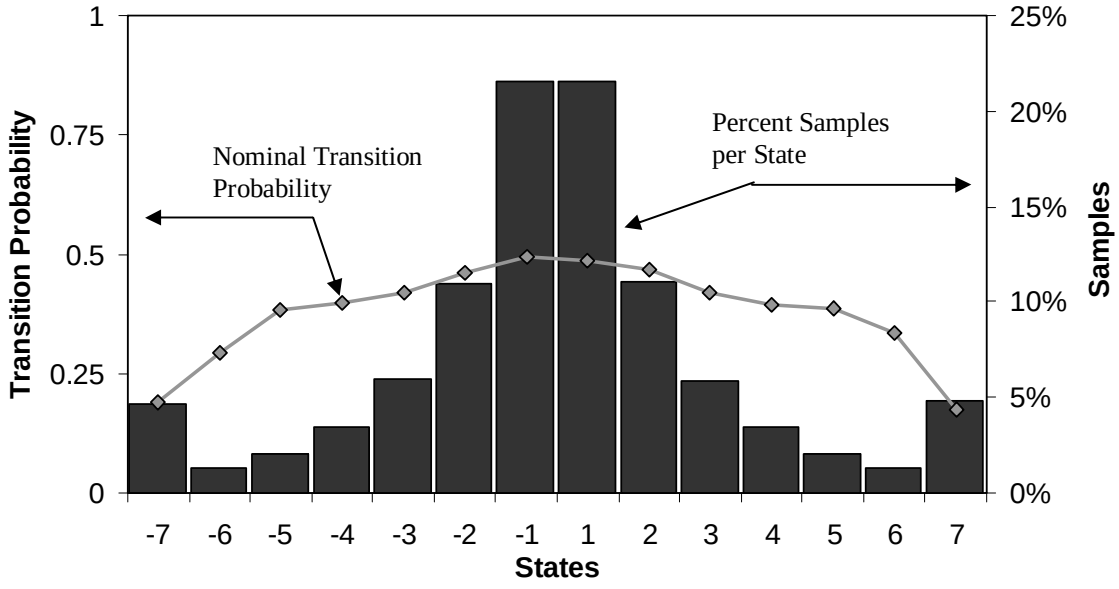

Figure 2.22: Markov Chain Model of Example Data (SR = 2, States = 14)

The base states for this model are again the +/- 6 States. The combined expected number of samples for each state finds a statistical window length of 3,660 health monitor

samples, equivalent to 7,320 DACS samples. Adding the settling time of 20 samples leads to a complete window length of 7,340 DACS samples.

The process continues until every complete window length is determined for the combination of sample ratio by number of states until a model of 8 states is found to fill this requirement. Figure 2.23 shows the progression of changing sampling ratio and states until model candidates are found, shown by the time calculated for the complete window. This figure is similar to Figure 2.12, however there is no check for minimum and maximum transition probability as this assumption is no longer needed. Instead is found a set of columns reporting the number of DACS samples required to fill the health monitor window.

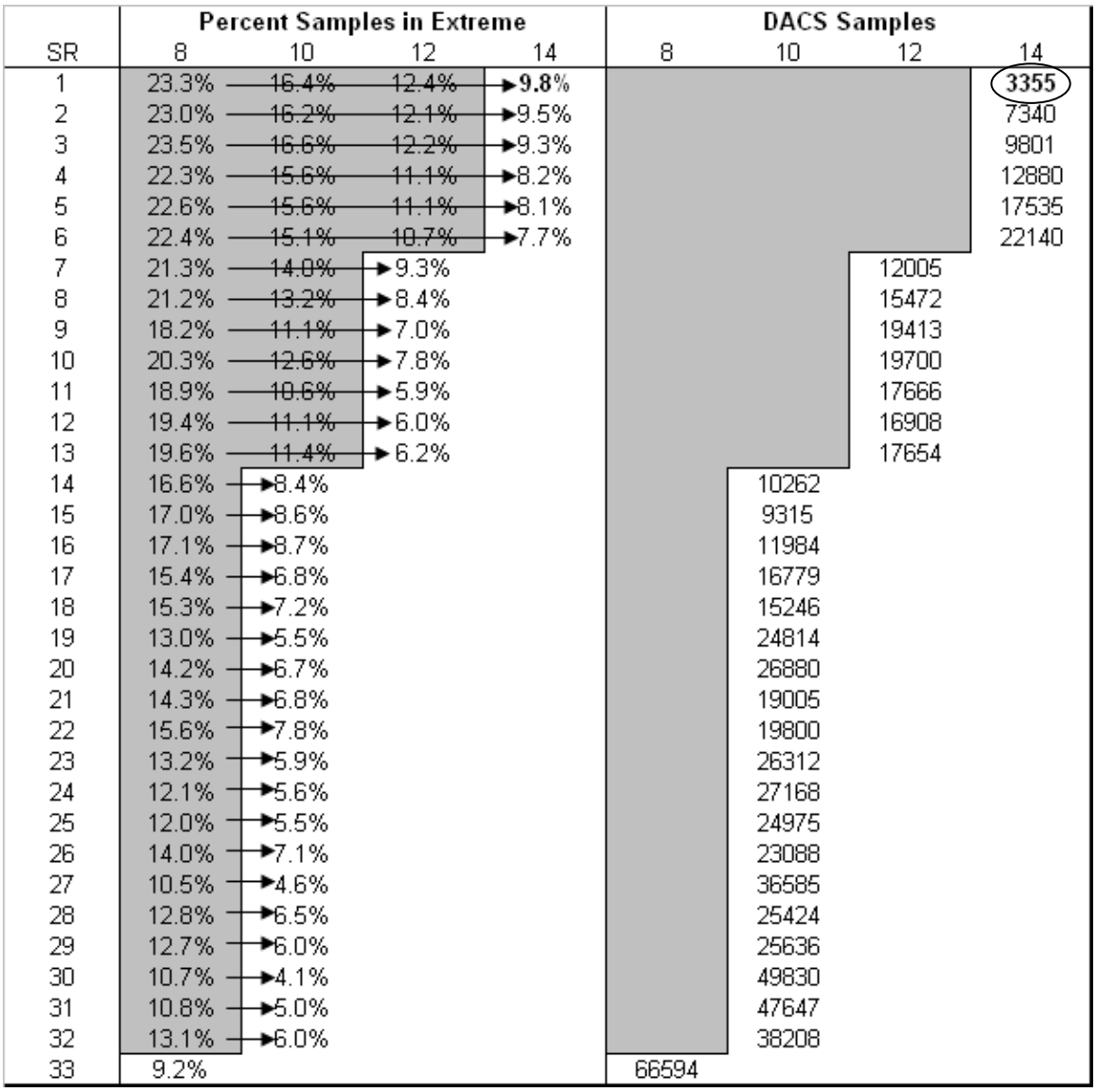

Figure 2.23: Improved Algorithm - Matrix of Results from Changing Sampling Ratio (SR) and Number of States

Notice that the algorithm terminated when a candidate 8-state model is found at a sampling ratio of 33. The regions marked in grey are those state and sampling ratio combinations which resulted in more than 10% of sample visits to the extreme states. The arrows represent the actual combinations probed by the modified algorithm. Although

this particular example shows that increasing the sampling ratio also increases the required number of DACS samples, this is not necessarily always the case. The window length in time is dependent on many factors, including the transition probabilities of the individual states which may be strongly affected by changing the sampling ratio. Comparing the number of DACS samples required to fill a statistical window for any particular number of states and sampling ratio, it is found that a Markov Chain with 14 states and a sampling ratio of 1 requires the fewest DACS samples (circled in Figure 2.23).

2.2.3.3 Number of Samples in "Good" Data Set

An example provided in Chapter 1 contained only 14 samples, raising the question of how many samples should be included in the "good" data set. One solution is to consider the number of DACS samples required by the health monitor determined statistical window. If the original data set does not contain at least as many DACS samples as are required by the health monitor statistical window, then a larger "good" data set should be collected. The example "good" data set provided in this Chapter contains 36,000 DACS samples; meaning that if the health monitor suggests a statistical window length requiring more than 36,000 DACS samples, more "good" data should be provided to the health monitor. The longest statistical window suggested for this "good" data set requires 17,535 DACS samples when found by Owusu's method. Furthermore, the statistical window suggested by the improved algorithms in this work is much shorter requiring 3,355 DACS samples, and therefore demonstrates that sufficient DACS samples are found in the "good" data set the user provided.

2.2.4 Final Model Comparison

Figure 2.16 reports the transition probabilities found using the original algorithm. During real-time monitoring, the health monitor requires 3,503 samples plus 4 samples for settling time. This takes 17,535 DACS samples.

The purpose of the improved algorithm is to find the model requiring the fewest number of DACS samples. Figure 2.24 shows the transition probability limits found when using the improved algorithm in this work.

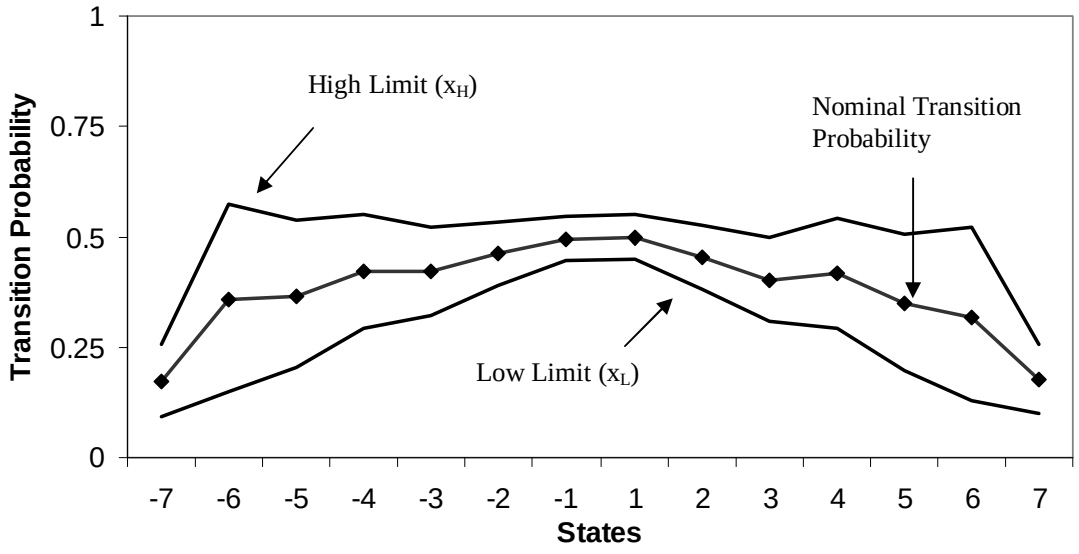

Figure 2.24: Transition Limits Surrounding Nominal Transition Probability

(Improved Algorithm, $\alpha = 0.10$, $\beta = 0.10$, $\lambda = 0.90$)

The real-time monitor requires 3,355 health monitor samples when including the settling time requiring 3,355 DACS samples. Since statistical requirements are met for both

models, there is no theoretical benefit to either model, they should both work fine. However, the improved model has the advantage of requiring fewer DACS samples. In this case the improved statistical window is 19% as long as the original Owusu statistical window, meaning that "poor" control is more quickly indicated.

For interest sake, it would be interesting to see what difference it makes to accept 20% of the samples in the extreme state, instead of the 10% used for comparison of the two models. Figure 2.25 reports the samples and nominal transition limits found during the analysis portion of the improved algorithm.

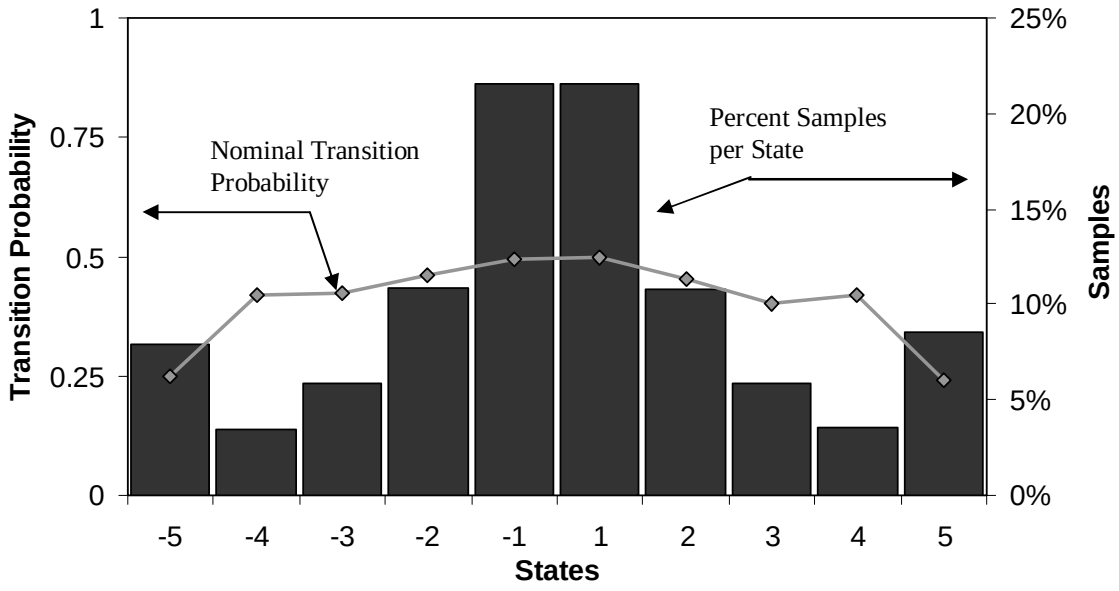

Figure 2.25: Markov Chain Model of Example Data

(Extreme = 20%, SR = 1, States = 10)

The transition probability limits for the above sampled data are based on states +/-4 found by the base state algorithm. Figure 2.26 reports the transition limits found by the health monitor.

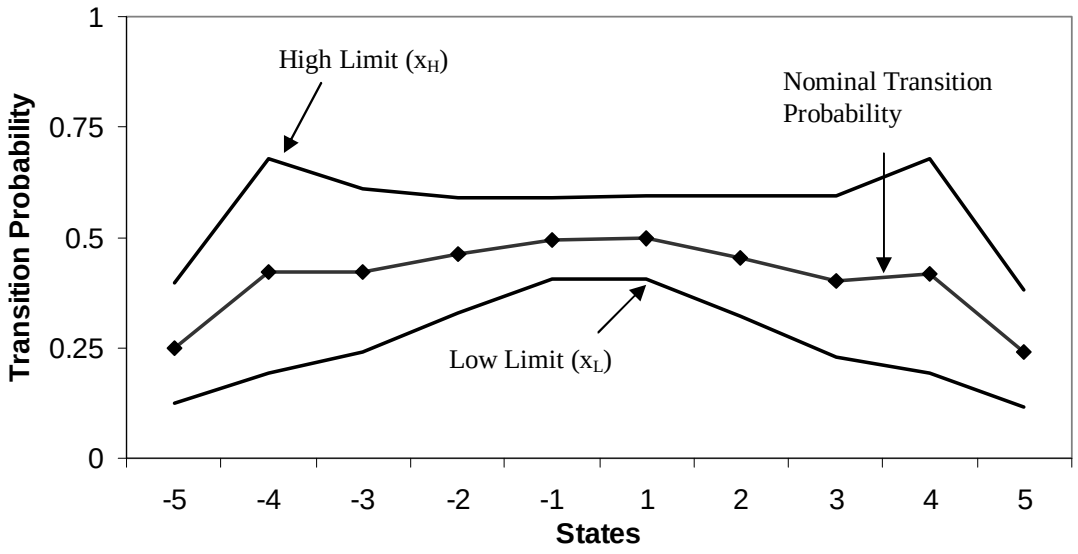

Figure 2.26: Transition Limits Surrounding Nominal Transition Probability

(Improved Algorithm, $\alpha = 0.10$, $\beta = 0.10$, $\lambda = 0.90$, Extreme = 20%)

Also, the statistical window contains 921 health monitor samples requiring 921 DACS samples. Using 20% as the condition instead of 10% further lowers the number of DACS samples required. The window would now require only 27% as many samples as even the improved method and a 10% condition. Furthermore, the window requires 5.3% as many DACS samples as the original Owusu statistical window. Since it is unknown whether there is really any difference in model quality it seems that tolerating more samples in the extreme states produces a model requiring a shorter time window. As will be shown in Chapter 3, this assumption still leads to a monitor which will correctly identify "good" and "poor" control.

CHAPTER III

EXPERIMENTAL SETUP

Two experimental apparatuses are used to validate the health monitor for effectiveness. Effectiveness is essentially defined as:

1) Flagging when control is no longer “good”

and

2) Not flagging when control is “good”

Quotes are used since real-time control monitoring is compared against the period of “good” control selected by someone familiar with the process. The first experiment is performed using a first-order-plus time-delay (FOPTD) simulator coded using VBA in conjunction with Microsoft Excel. The second experiment is physical apparatus involving two-phase through a vertical pipe.

3.1 Simulator – FOPTD

In Laplace domain, the first-order-plus time-delay model is expressed by

$$g_p(s) = \frac{K_p e^{-\theta s}}{\tau_\rho s + 1} \tag{3.1}$$

where K_p is the process gain, τ_ρ is the process time and θ is the time delay. All time units expressed in the process and controller functions are in minutes. The controller is PI given by

$$g_c(s) = K_c\left(1 + \frac{1}{\tau_I s}\right) \tag{3.2}$$

where Kc is the controller gain and τ_I is the integral time. The feedback block diagram is shown in Figure 3.1.

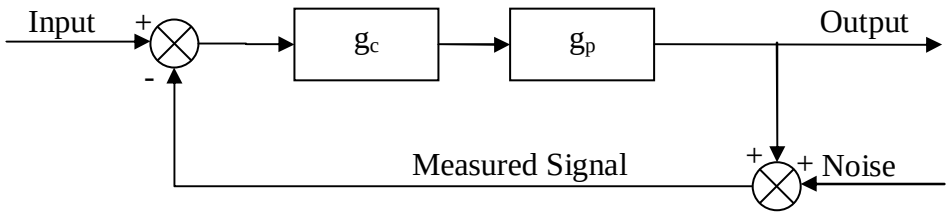

Figure 3.1: FOPTD Block Diagram

According Owusu's work, Gaussian noise is included in output measurements using the Box-Muller transform where any two uniformly distributed random numbers where $0 < U \leq 1$ may be used to calculate a normally distributed random value using

$$R = \sqrt{-2\ln(U_1)}, \tag{3.3}$$

$$\theta = 2\pi U_2 \tag{3.4}$$

and finally

$$Z = R\cos(\theta). \tag{3.5}$$

Both U_1 and U_2 are uniformly distributed random numbers in the range $0 < U \leq 1$ and R and θ are intermediate values found in polar coordinates. The final value, Z, is actually one of two independent random values that may be obtained from the Box-Muller technique, the other being

$$Z = R\sin(\theta), \tag{3.6}$$

however, only one is used per iteration.

Although the transfer functions are provided above, the system of equations is solved numerically. The FOPTD function given by

$$\tau_p \frac{dy}{dt} + y(t) = K_p u(t) \tag{3.7}$$

where y is the output and u is the input found in Figure 3.1. This does not include the delay, which is incorporated with a simple wait period in the program. According to High (2006) a first order ordinary differential equation may be solved by the 4th-order Runge-Kuta technique. The following outlines this technique. The first step is to calculate the slope from the current step using Euler's method.

$$\frac{dy}{dt} = y'_1(u_j, y_j) = \frac{1}{\tau_p}\left[K_p u_j - y_j\right] \tag{3.8}$$

Next, a half-step is taken using the Euler's derived slope from Equation (3.8).

$$y_2(u_j, y_j) = y_j + \frac{\Delta t}{2}\left[y'_1(u_j, y_j)\right] \tag{3.9}$$

The slope is now determined at this half-step using

$$y'_2\left(u_j, y_2\right) = \frac{1}{\tau_p}\left[K_p u_j - y_2\right]. \tag{3.10}$$

And, now another half step is taken using the updated slope at the previous half-step. If this next step were taken using the full time-step, it would be equivalent to a 2nd-order Runge-Kuta; however, here we take only a half-step.

$$y_3\left[u_j, y'_2\right] = y_j + \frac{\Delta t}{2}\left[y'_2\left(u_j, y_2\right)\right] \tag{3.11}$$

Next, a third slope is taken from half-step calculated in Equation (3.11)

$$y'_3\left(u_j, y_3\right) = \frac{1}{\tau_p}\left[K_p u_j - y_3\right]. \tag{3.12}$$

Then, a full-step is taken with the slope calculated in Equation (3.12)

$$y_4\left[u_j, y'_3\right] = y_j + \Delta t\left[y'_3\left(u_j, y_3\right)\right] \tag{3.13}$$

The slope is then calculated at the full-step length.

$$y'_4\left(u_j, y_4\right) = \frac{1}{\tau_p}\left[K_p u_j - y_4\right]. \tag{3.14}$$

Now there are four slopes from which a-full step may be taken with greater accuracy than either the single step Euler's method or a 2nd-order Runge-Kuta. The new y-value at the full step in u is reported in Equation (3.15)

$$y_{j+1} = y_j + \Delta t\left[\begin{array}{l} \frac{1}{6}y'_1\left(u_j, y_j\right) + \frac{1}{3}y'_2\left(u_j, y_2\right) + \cdots \\ \cdots + \frac{1}{3}y'_3\left(u_j, y_3\right) + \frac{1}{6}y'_4\left(u_j, y_4\right) \end{array}\right] \tag{3.15}$$

The input (u) is held at its initial value throughout because the input signal from the controller is sampled and held.

The PI controller uses the trapezoid rule to calculate the integral or cumulative error in the following manner

$$\int_{j-1}^{j} e \quad dt \approx \frac{\Delta t}{2}\left(e_j + e_{j-1}\right) \tag{3.16}$$

where “e” is the error signal and “j” is the current step.

The controller makes 10 samples per second with a zero order hold in between. Each sample incorporates white noise calculated from the Box-Muller transform. The process simulation is performed at 10 times the resolution of control. In other words, the process simulation time step is $1/10^{th}$ the length of time between controller samples and calculations. This is to simulate the underlying process which continues to act on the controller input in between samples. The 4^{th} order Runge-Kuta is chosen to model the process as correctly as possible, while the trapezoid rule is selected in the controller due to its common use in computer control to calculate integral error. In both cases, the intent is to more closely model reality.

The simulator also has a sticky valve replication mechanism. Valve stiction occurs when the valve stem does not move until enough force causes it to jerk to a new position. It often results in a sawtooth pattern in the process output. From Rhinehart (2007), expression (3.17) is used to replicate stiction

$$u_{input} = \begin{cases} \left| u_{input} - u_{output} \right| < \delta & u_{input} = u_{input} \\ else & u_{input} = u_{output} \end{cases} \tag{3.17}$$

where u_{input} is the input to the FOPTD process and u_{output} is the controller output. The value δ is some predefined difference between u_{input} and u_{output} such that if less than δ, the u_{input} does not update for each sample period.

Four simulations are performed on the FOPTD process:

1) Well tuned, normal process data with no setpoint changes (good data)
2) Process control response to setpoint changes
3) Process control response to varying controller gains tested with setpoint changes
4) Process control response to simulated valve stiction
5) Process control response to gradual process change

The good data is used to create the Markov chain, transition probability limits, and length of window required to satisfy binomial statistics. The process response to setpoint changes is to test whether the health monitor flags normal response to a setpoint change. The health monitor should flag the process when controller gain changes are made. Likewise, a controller facing valve stiction is in poor control, and the health monitor will flag these periods as well. Chapter 4 reports the results of the tests.

3.2 Cascaded Control in Two-Phase Flow

Testing the effectiveness of a controller health monitor using simulation alone is unsatisfactory. To establish that this method is useful, health monitor tests are performed

on both the primary and secondary controllers of a pilot-scale two-phase flow unit controlling pressure drop by cascade.

The two-phase flow apparatus is equipped with a single vertical pipe fed by air and water. One flow valve controls air flow and the other controls water flow. The combination of air and water forced to flow upwards through the pipe produces two-phased flow of various regimes from bubble to annular mist flow. The experiments in this study run primarily in the churn flow regime. Figures 3.2 through 3.5 are photographs of the experimental setup including the pipe, flow controllers and data acquisition and control system CamilleTG 2000.

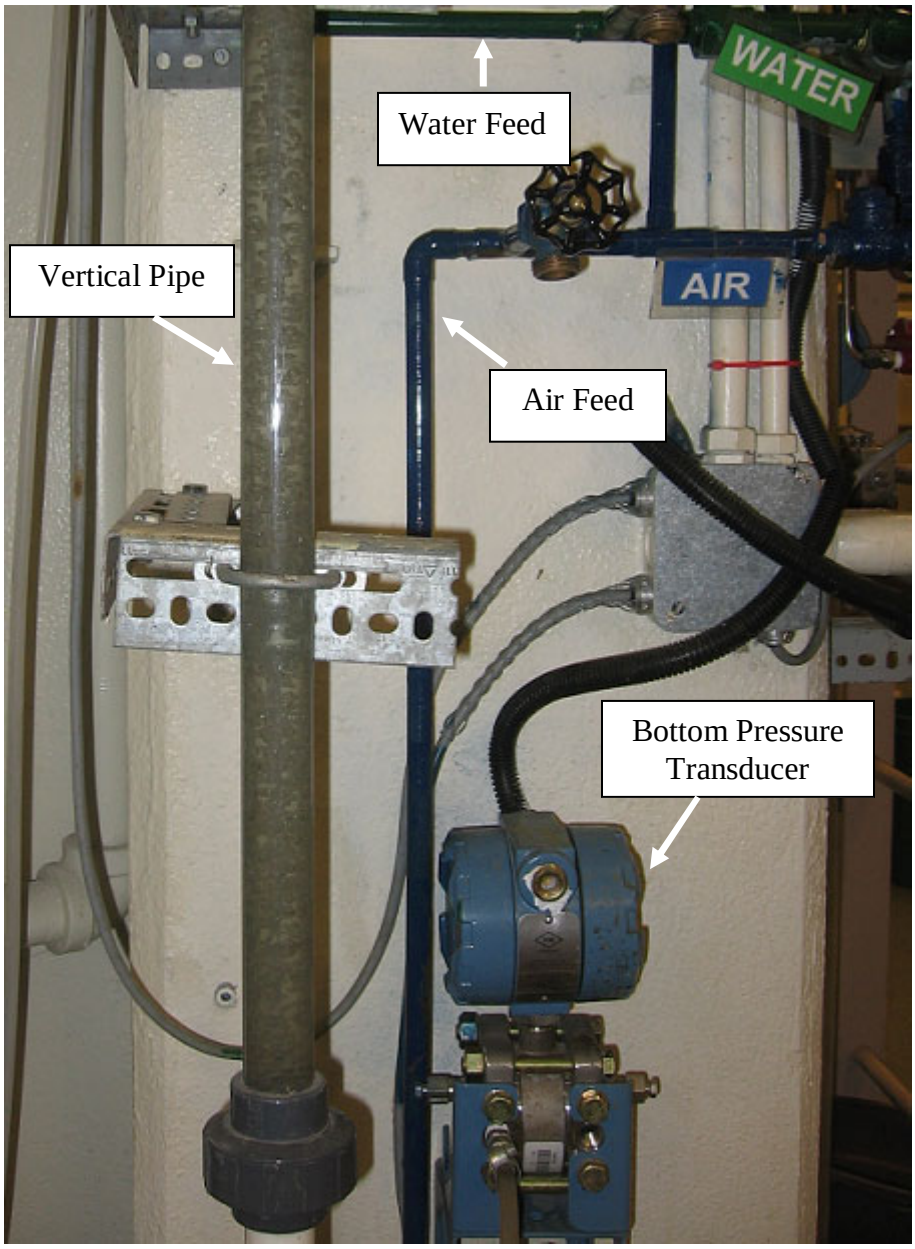

Figure 3.2: Two-Phase Flow Photograph – Bottom of Pipe

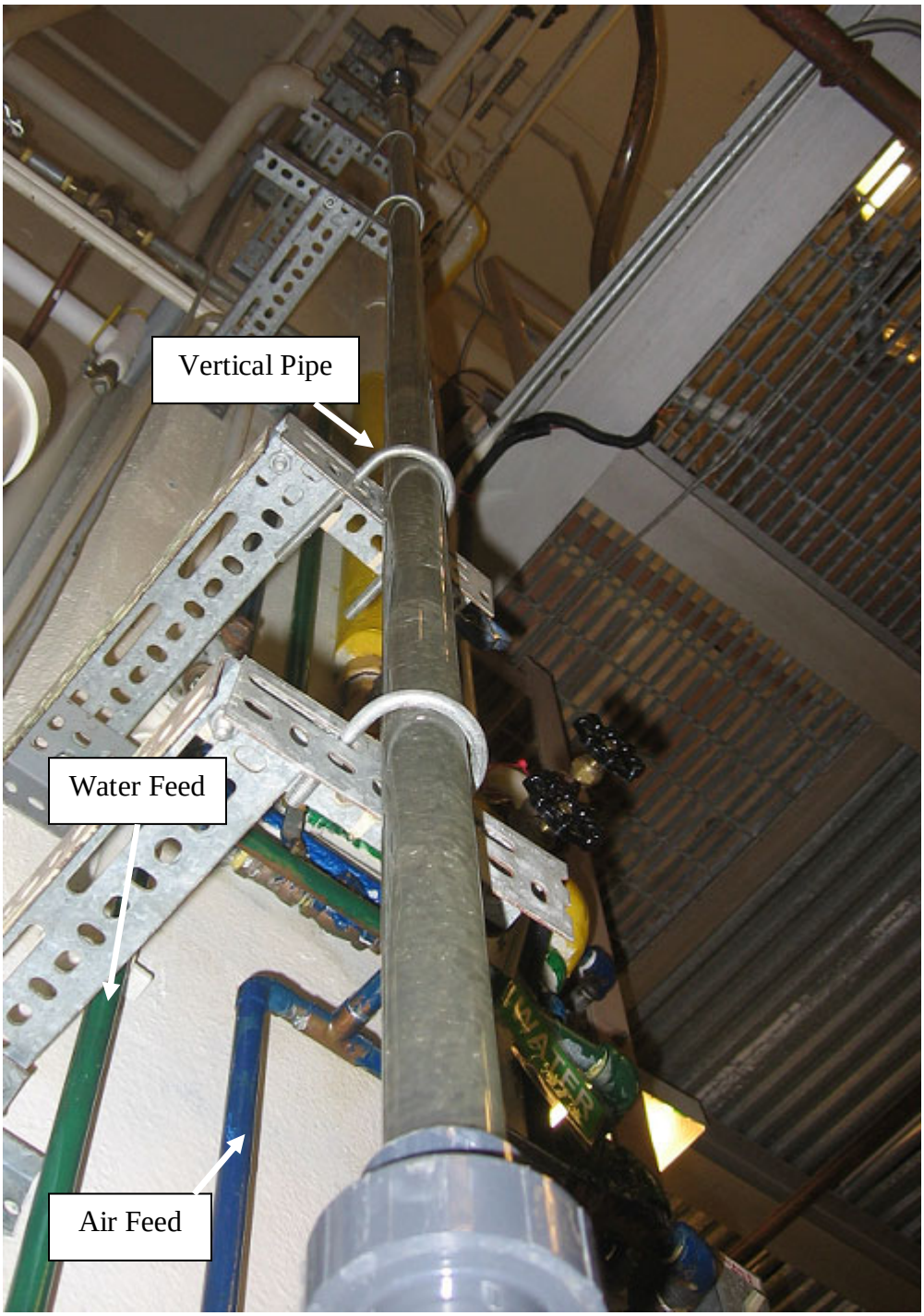

Figure 3.3: Two-Phase Flow Photograph – Entire Pipe

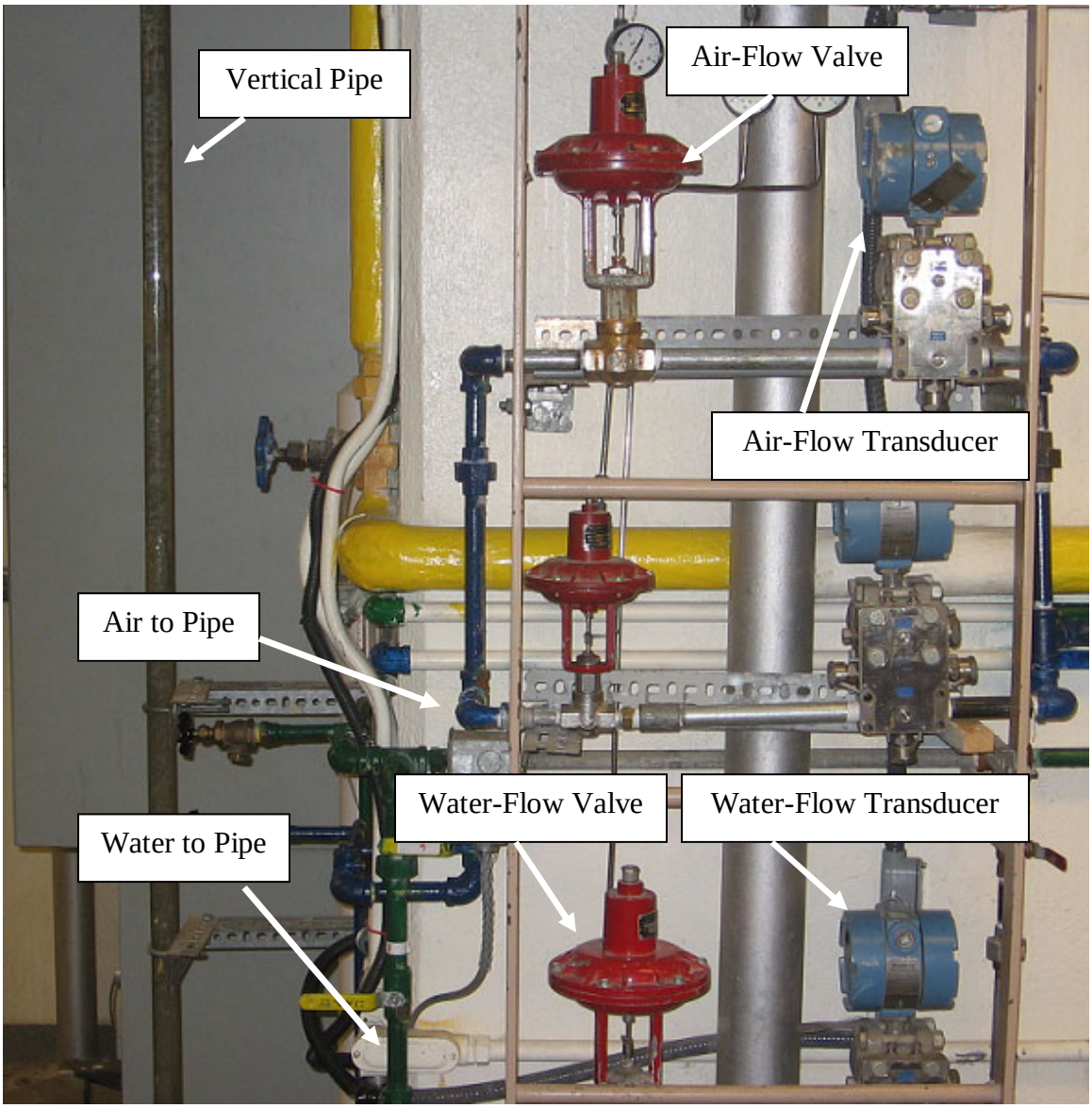

Figure 3.4: Two-Phase Flow Photograph – Control Valves

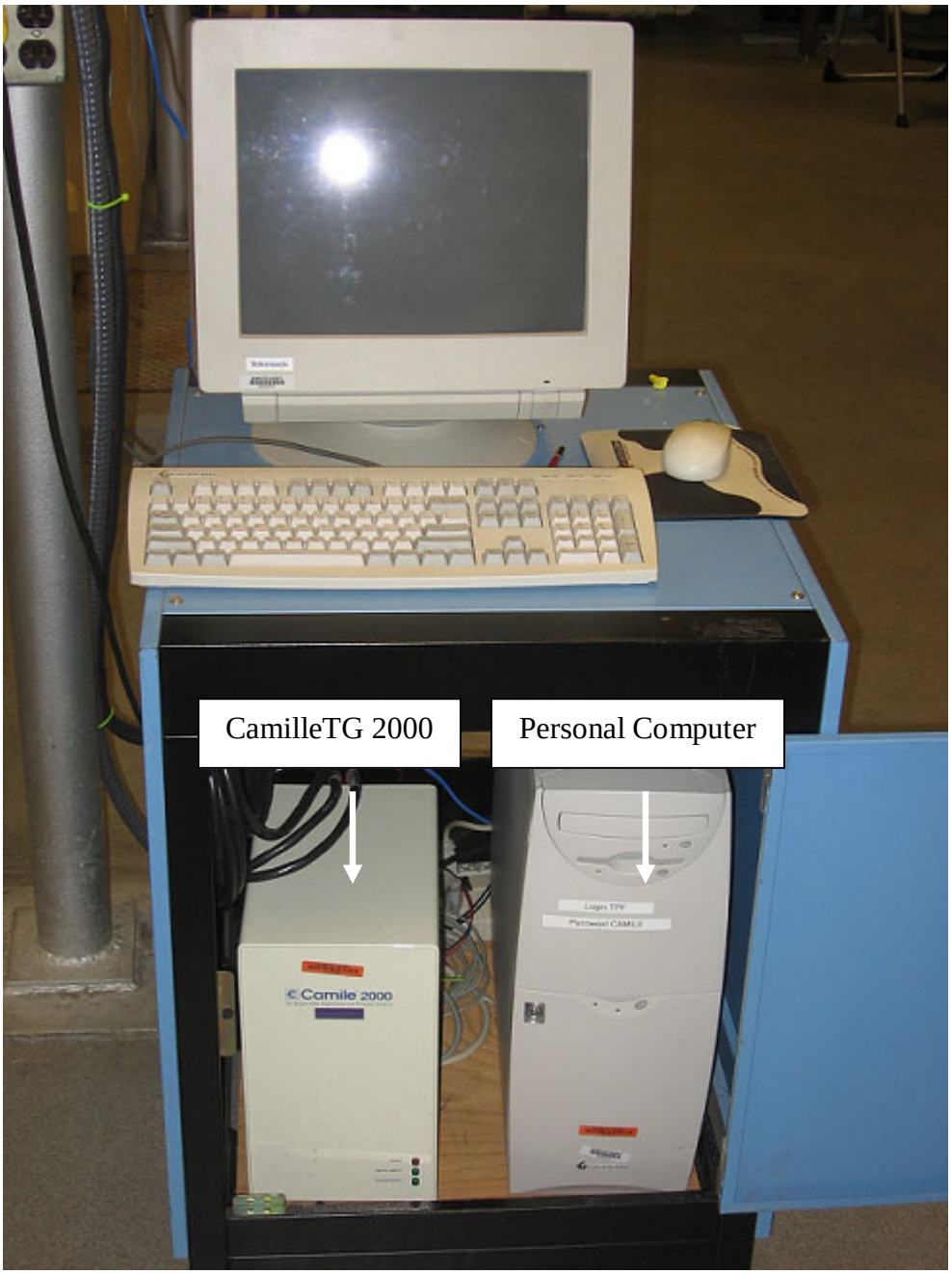

Figure 3.5: Two-Phase Flow Photograph – CamilleTG 2000 DACS

Figure 3.5 shows the CamileTG 2000 used for data acquisition and control. Various control schemes may be realized using the Windows based CamileTG 2000 software. In

this work, pressure drop is controlled by cascade where the secondary controller controls the water flow rate. Figure 3.6 is a diagram of the cascaded control strategy employed.

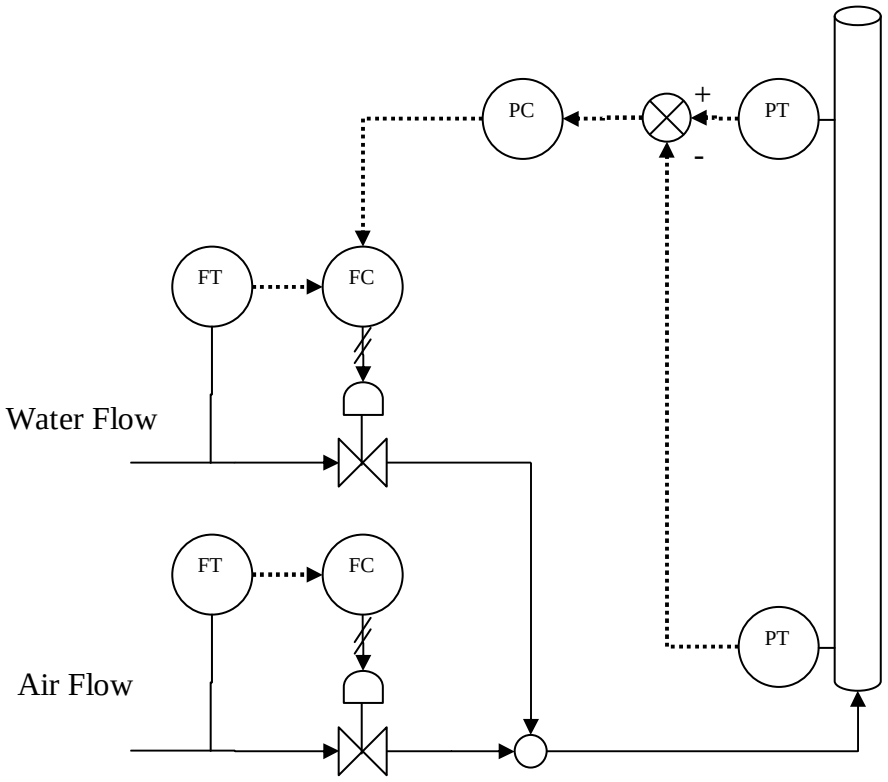

Figure 3.6: Pilot-Scale Two-Phased Flow Unit

The large air flow valve is also used to introduce disturbances and is setup as a SISO PI controller. Two pressure transducers, one at the bottom, the other at the top of the column measure pressure drop. Nominal pressure drops for this study are from 25 to 75 inches H_2O, water flow rates range from 0 to 25 lbs/min which are the physical limits of this valve and nominally the air flow valve allows 10-15 cfps depending on the case under study.

The specific series of steps and methods for each of two separate cases are described in Chapter 4 as introduction to each set of results.

CHAPTER IV

EXPERIMENTAL RESULTS

4.1 Simulated Testing – FOPTD model

The simulated first order plus time delay system (FOPTD) controlled by PI tuned under ITAE servo rules is used to generate several test cases. The simulation is of no particular process so arbitrary time and process units are used. The process gain (K_p) is 1.2 process output units per controller input units, the time constant (τ_p) is 0.5 time units and the time delay (θ) is 0.1 time units. The servo PI controller tuning rules for ITAE reported by Ogunnaike (1994) for the proportional gain is

$$K_c = \frac{0.586}{K_p}\left(\frac{\theta}{\tau_p}\right)^{-0.916} \tag{4.1}$$

and for the integral time is

$$\tau_I = \frac{\tau_p}{1.03 - 0.165\left(\frac{\theta}{\tau_p}\right)}. \tag{4.2}$$

Therefore, the controller gain (K_c) is found to be 2.133 controller output per error and the integral time (τ_I) is 0.502 time units.

The simulator runs to collect 120 time units of data measured at a data acquisition and control system (DACS) frequency of 600 per time unit during which no step changes are made to the process. This means that during the run of "good" data 72,000 DACS samples are collected. The relationship between the arbitrary time unit used in the simulation and the DACS frequency is the same as if minutes represent the time unit and 10 DACS samples are taken every second. The measurement noise is Gaussian and independent with a standard deviation of 0.05 process units NID(0, 0.0025). Figure 4.1 reports a sampling of the good data actuating errors collected during this run.

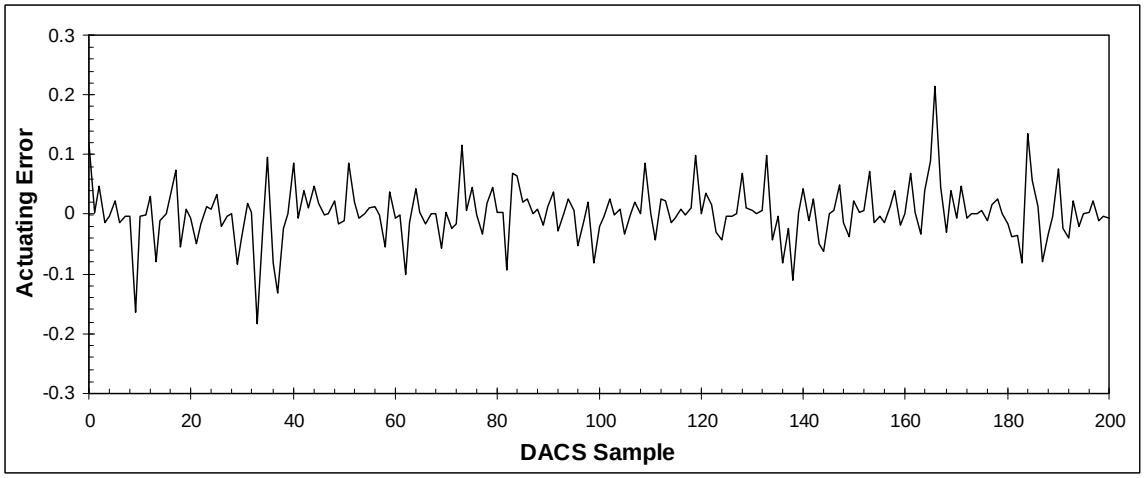

Figure 4.1: Good Data Actuating Errors (FOPTD Simulated Process)

The health monitor analyzes the good control data where Type I error (α) is specified as 0.003 and Type II error (β) is specified as 0.003 corresponding to +/- 3σ limits. In addition, the change of transition probability for the alternate hypothesis (λ) is 0.900. Finally, during step testing of the simulated FOPTD process, it is found that the settling time should be 1,200 samples at the DACS frequency.

The health monitor determines that a sampling ratio of 1 and a Markov chain of 8 states sufficiently describe the run-lengths found in the “good” data set. The transition ratios are all close to 0.5 and there is approximately 15.7% of the data found in the extreme (+/-4) states. Figure 4.2 shows the number of samples per state with bars and the transition probability for each state as a line.

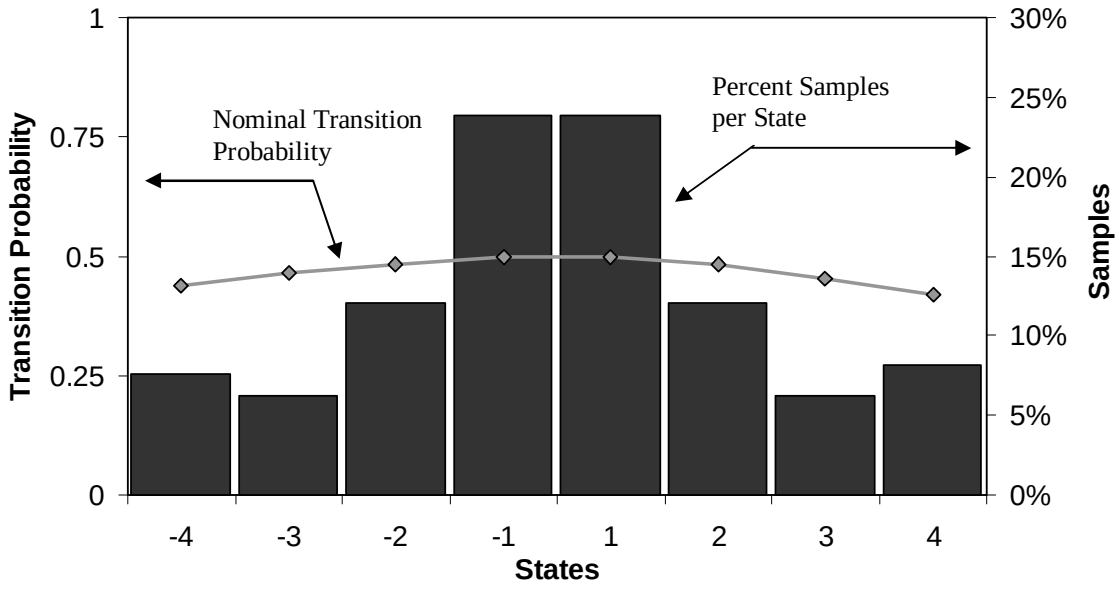

Figure 4.2: Markov Chain Model of Simulation Data (SR = 1 and 8 states)

The Type I test determines the transition probability limits found in Figure 4.3. Since the nominal transition probabilities are all close to 0.5, fewer samples are required to distinguish the null hypothesis from the alternate hypothesis. This is evident by the window length found during Type II tests which is short in comparison to the example provided in Chapter 2. The statistical window length to be used during real-time monitoring is 774 DACS samples. Including the settling time for a complete window requires 1,974 DACS samples (3.3 arbitrary time units).

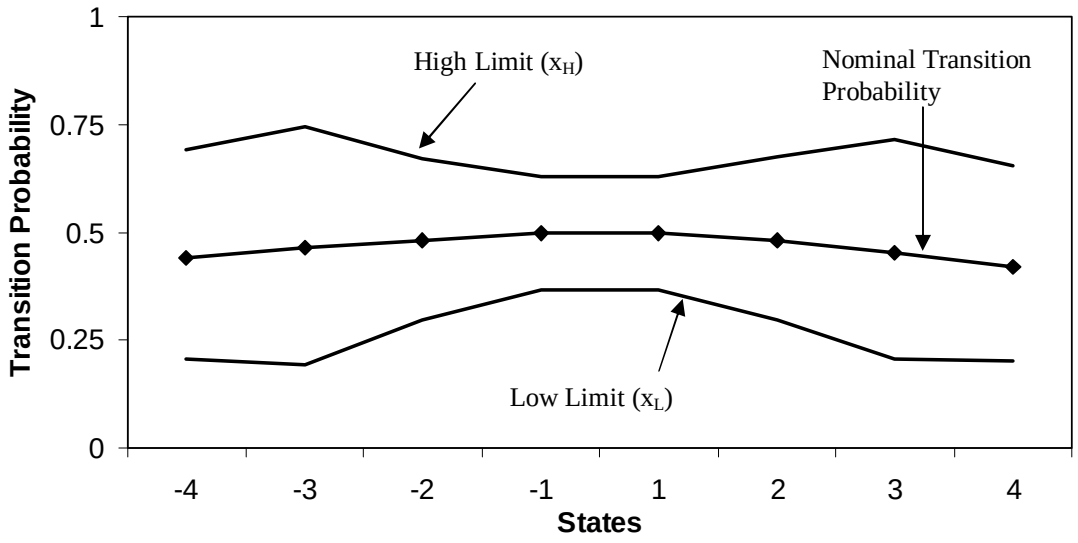

Figure 4.3: Transition Limits Surrounding Nominal Transition Probability for Simulated Process ($\alpha=\beta=0.003$, $\lambda=0.900$, Extreme = 20%)

During real-time monitoring, whenever a state transition probability violates a limit, a counter begins. If the counter continues (violation continues) for the entire length of the complete window (statistical + settling time) the process is flagged retroactively. Three cases are presented to test the real-time monitoring capabilities of the health monitor.

4.1.1 Case 1 – Setpoint changes

During normal process operation, setpoint changes should not cause the monitor to flag. However, setpoint changes cause long negative or positive run lengths until the process settles at the new setpoint. Depending on system dynamics and the magnitude of the setpoint change, enough samples may visit the extreme states to alter the transition probability and cause a limit violation. Since a violation must last through a complete window length, which includes the settling time, the monitor should never flag a setpoint

change only. During this simulation, a setpoint change of 2 process units is initiated every 6,000 DCS samples. Figure 4 demonstrates the ability of the health monitor to avoid flagging when there has been no change to the process control.

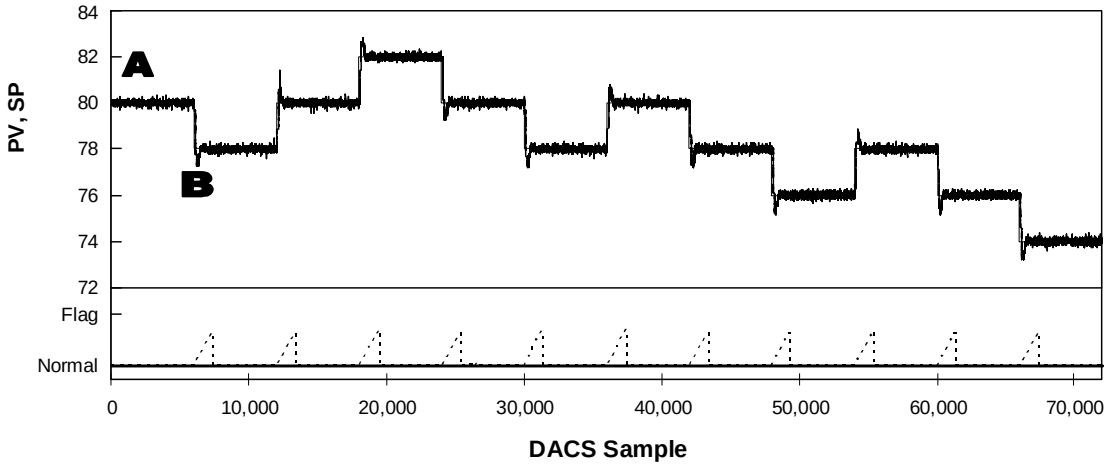

Figure 4.4: Simulated Real-time Analysis During Setpoint Changes

Figure 4.4 shows the setpoint and process variable tracking in the upper part and a real-time view of counting and flagging in the lower portion. The sawtooth pattern in the lower part shows the monitor counting during a setpoint change. This example shows no flagging occurred, meaning that transition probabilities are not violated for a complete window length, which is what this demonstration seeks to test.

Notice that the period of time marked by an “A” is in a section where there is no sawtooth pattern in the lower portion of the figure. This means that control in this region is still considered “good”, or that the null hypothesis has not been rejected for any of the 8 states. Figure 4.5 reports the transition probabilities of a sample window found during the period marked “A”.

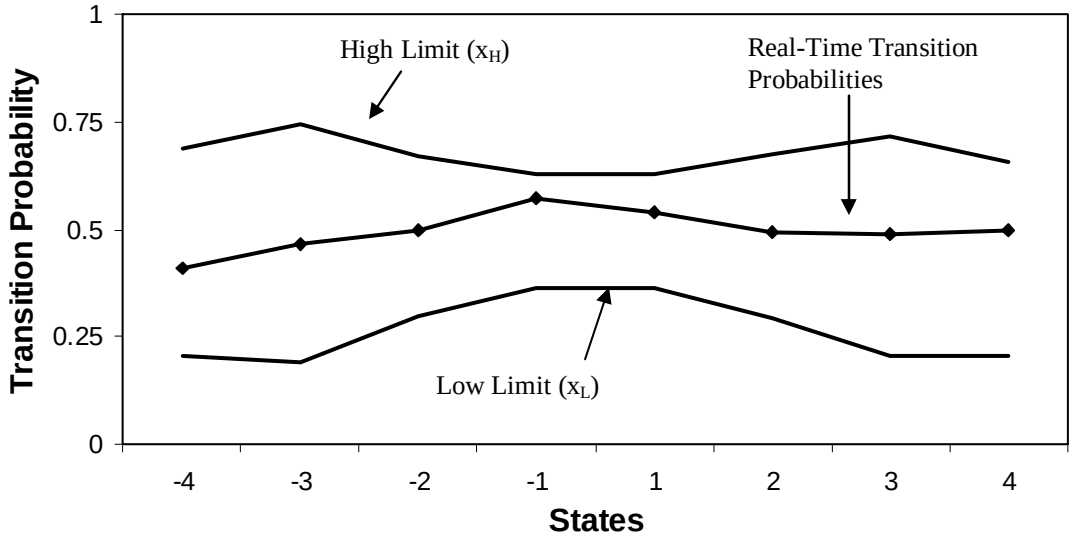

Figure 4.5: Simulated Real-Time Transition Probabilities with Limits

(Time Period 'A' During Setpoint Changes)

Indeed Figure 4.5 shows that all transition probabilities are still within predefined limits. However, when a setpoint change occurs, as in the time period marked "B", the lower portion of Figure 4.4 shows some counting occurring that if it continued for the length of a complete window would produce a flagged event.

Figure 4.6 shows the transition probabilities calculated for a window found within the time period marked "B" which is a setpoint change which occurred at DACS sample 6,000.

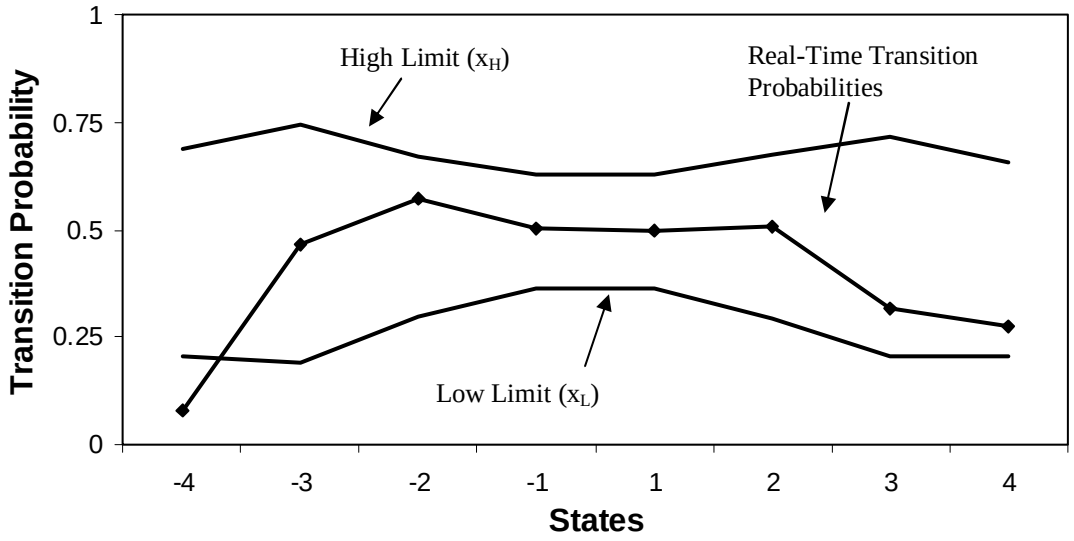

Figure 4.6: Simulated Real-Time Transition Probabilities with Limits

(Time Period 'B' During Setpoint Changes)

During the time period marked "B" a setpoint change from 80 to 78 produces a period of long negative runs until good control is again realized. Figure 4.6 shows the transition probability for state -4 below the lower transition limit, meaning that fewer of the negative runs reaching 4 or more in length are making zero crossings. Figure 4.6 does not generalize all setpoint changes as only violating one state transition probability limit.

4.1.2 Case 2 – Controller Retuning

One of the purposes of the health monitor is to notify the user when a loop has become detuned. One way to demonstrate this is by changing gains during the simulation. Since the controller is the standard or parallel algorithm, K_c is multiplied across the proportional and integral portions. First, the gain is maintained at design specifications ($K_c = 2.133$), then for a period of time gain is doubled ($K_c = 4.266$), then returned to

design specs. Finally, the gain is decreased by 4 times ($K_c = 0.533$) then returned back to specs. These steps are reported below in Table 4.1 where $K_I = K_c/\tau_I$.

Table 4.1: Controller Gain Change Steps Simulated Process

DACS Sample	Kc	K_I
0	2.133	4.249
9000	4.266	8.498
27000	2.133	4.249
45000	0.533	1.062
63000	2.133	4.249

The first column gives the sample which begins the specified controller parameter; therefore, the gain change increase occurs from DACS sample 9,000 to DACS sample 26,999. Figure 4.7 shows the real time monitoring of this simulation.

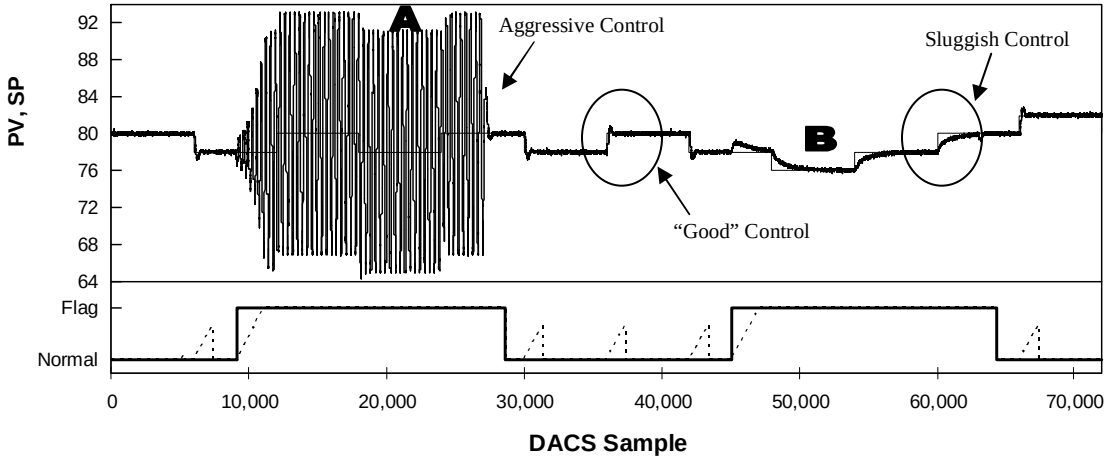

Figure 4.7: Simulated Real-time Analysis During Gain Changes

When the gain is increased at DACS sample 9,000, oscillations begin and grow quite large compared to the good control signal. Counting begins immediately, and as soon as

violations last for complete window length the flag is marked retroactively. Figure 4.8 shows the transition limits for the time period marked “A”.

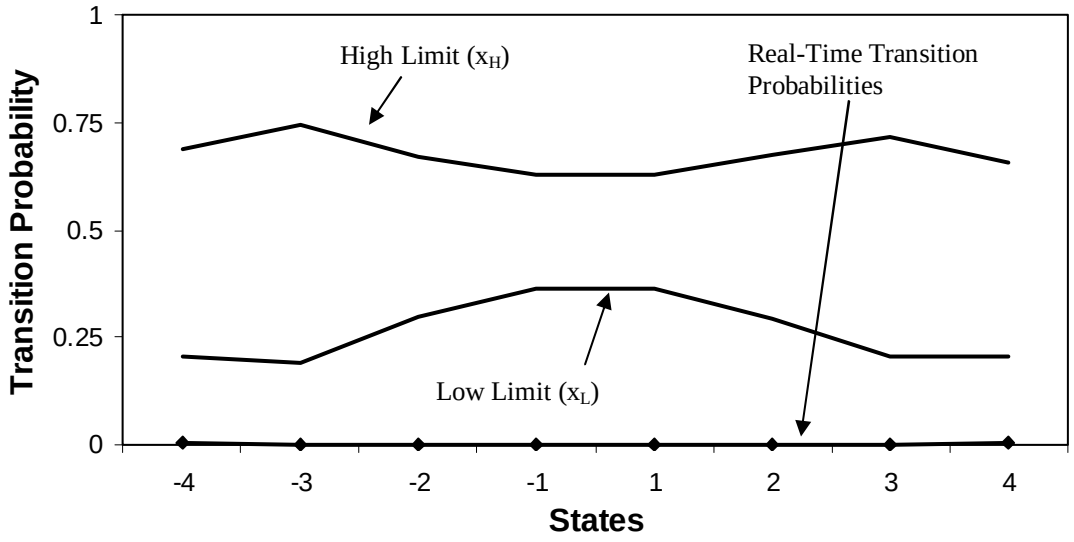

Figure 4.8: Simulated Real-Time Transition Probabilities with Limits (Time Period ‘A’ During Gain Changes)

The real-time transition probabilities are nearly zero because the window contains exclusively long runs, which is indicative of a process in oscillation. Notice that in this case every state has violated low transition probability limits. Oscillations do not always cause all transition probabilities to be zero, but they often result in very low transition probabilities, especially for the extreme states.

Finally, when the gain is returned to specs at DACS sample 27,000 the system takes some time to return to good control noted by transition limits returning to within their limits. The last period marked by “B” denotes a region of sluggish control because the controller gain has been decreased by four times. The controller gain is decreased at

DACS sample 45,000. Figure 4.9 reports the real-time transition probabilities during this period.

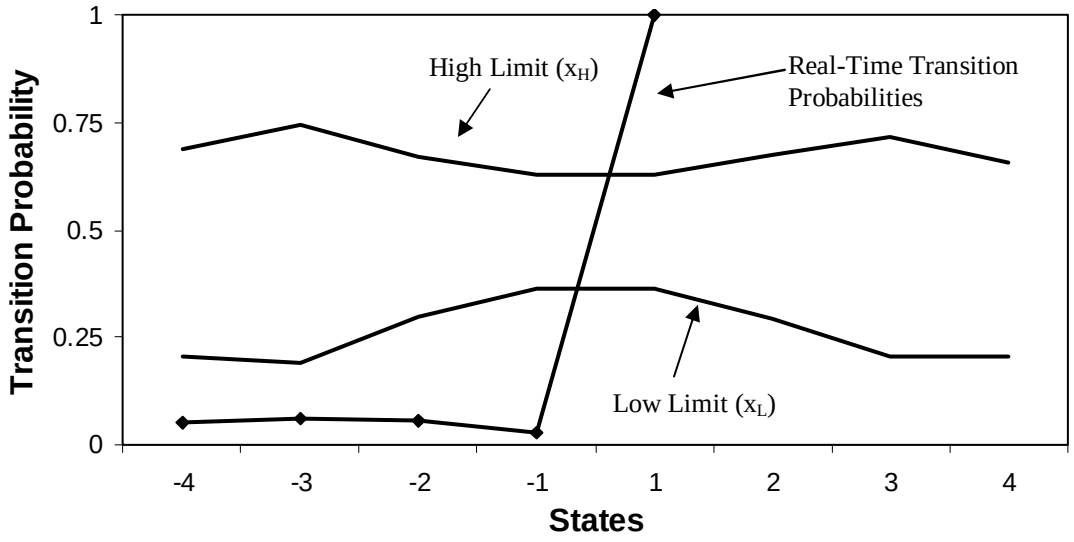

Figure 4.9: Simulated Real-Time Transition Probabilities with Limits

(Time Period 'B' During Gain Changes)

All of negative state real-time probabilities are very low, meaning long runs are occurring on that side of the setpoint. The positive states show something very different. The +1 state reports that all samples reaching a run length of 1 on the positive side will have a zero crossing, therefore, there will be no run lengths longer than 1 in the positive direction. Figure 4.9 shows no transition probability for States +2, +3 and +4 since no run-lengths of greater than +1 exist in the sample window. This means the process is experiencing on offset and only periodically crossing the setpoint, a result that would be common for a process sluggishly tracking a setpoint change.

Finally, when the gain is returned to specs, the controller can now track the setpoint and control is returned to normal.

4.1.3 Case 3 – Valve Stiction

An additional control problem which faces the process industry is valve stiction. The health monitor can identify poor control caused by stiction. This is simulated by not allowing the process input signal to change unless a difference between the controller output and current process input is greater than a specified value. As the process fails to track the setpoint, the error signal increases causing the controller output signal to increase. When the disparity reaches a certain value the process input is set to the controller output. In reality, this jerk reflects a similar response from a sticky valve. The valve moves only when enough force is applied, then it jerks to a new position. In the simulation, the difference is initially set to 0 (no stiction), then at DACS sample 9,000 the difference is set to 2, meaning when the controller output and process input have a difference of 2, the process input will be updated to the controller output. At DACS sample 27,000 the difference is reset to spec (no stiction). Finally, at DACS sample 45,000 a different differential of 3 is tested, then returned to spec at DACS sample 63,000. Figure 4.10 reports the real-time results of monitoring a system experiencing valve stiction.

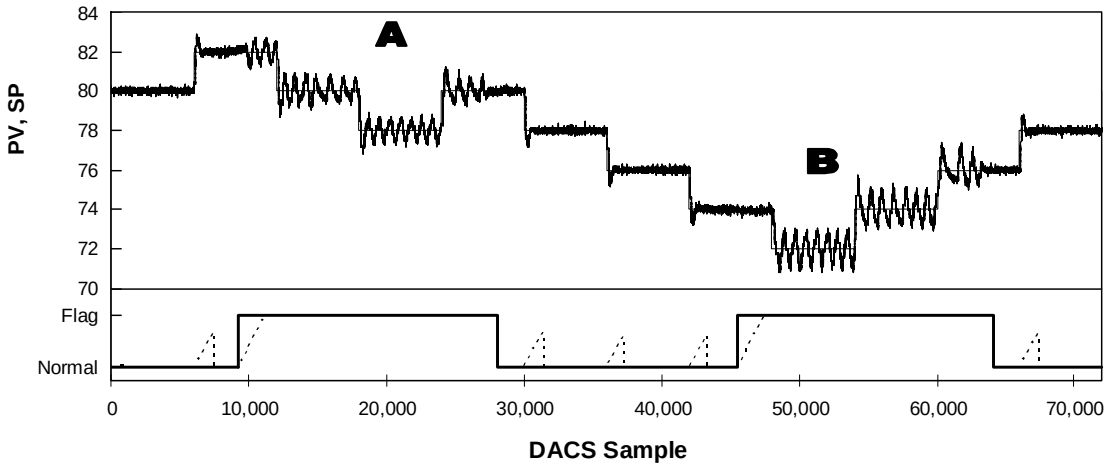

Figure 4.10: Simulated Real-Time Analysis During Periods of Stiction

The first section of valve stiction is denoted by the letter "A". Figure 4.11 reports the transition probabilities for this period of stiction which began at DACS sample 9,000.

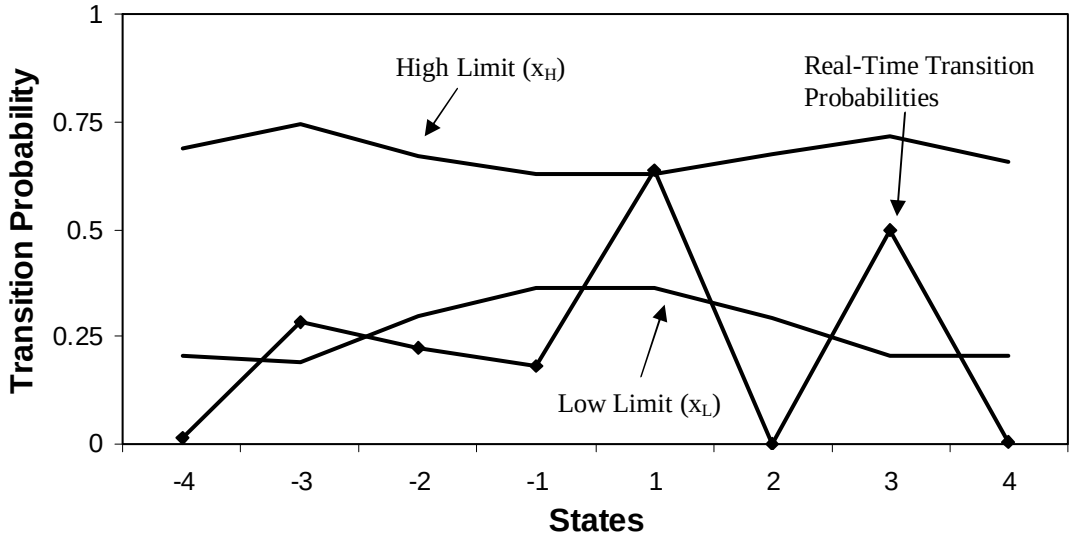

Figure 4.11: Simulated Real-Time Transition Probabilities with Limits

(Time Period 'A' During Stiction)

The low probabilities in the extreme states denote long runs, which is seen in Figure 4.10 as a sawtoothed response. The second section of stiction marked with a "B" is very similar to the first, and a plot of real-time transition probabilities is similar to Figure 4.11. It would be redundant to show an additional plot for the second period of stiction. The real-time health monitor flags both periods of stiction as having poor control. In addition, when stiction stops, the health monitor stops flagging the process.

4.1.4 Case 4 – Process Change

A final problem which plagues even good control is the gradual changes all processes experience. Even the best tuned controllers need periodic retuning. To simulate this effect and the ability of the health monitor to indicate poor control, process parameters are changed during data collection. For the first 60 time units or 36,000 DACS samples, the characteristic process parameters remain at their original spec values ($K_p = 1.2$, $\tau_p = 0.5$, $\theta = 0.1$). Then at DACS sample 36,000, the process gain (K_p) and process time-constant (τ_p) begin to gradually change until by DACS sample 54,000 $K_p = 1.81$, $\tau_p = 0.25$ and the time delay remains constant at $\theta = 0.1$. These process parameters are maintained through the end of the simulation. Figure 4.12 reports the real-time monitoring of this simulation.

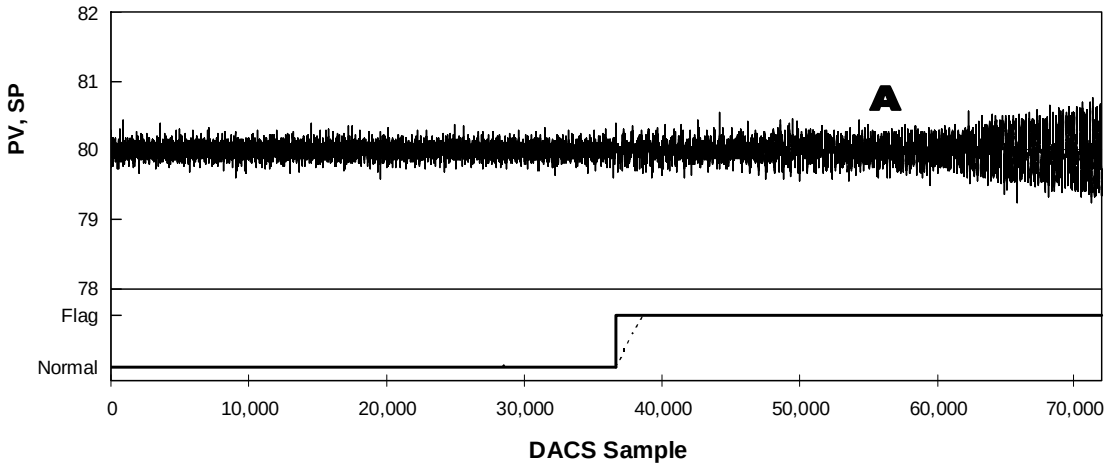

Figure 4.12: Simulated Real-Time Analysis During Process Change

As the system changes, the process output begins to oscillate slightly. Near the end of the simulation, oscillations have increased noticeably and, were the simulation to continue, their amplitude would continue to increase. This is because the final process gain and time constants caused the previously well tuned PI controller to become too aggressive for the changed system. Figure 4.13 reports the transition probabilities for a sample window found during the time period marked “A”.

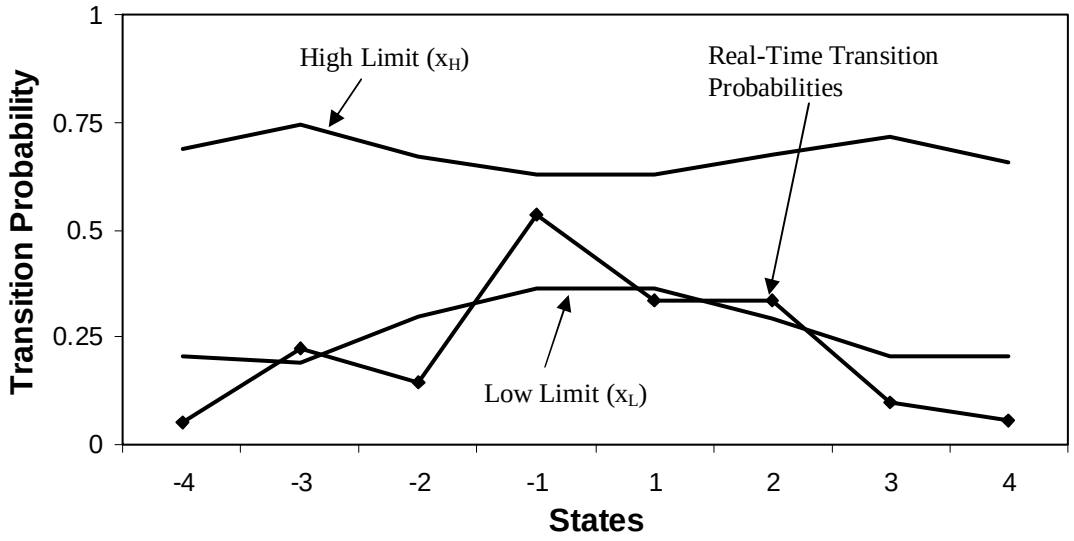

Figure 4.13: Simulated Real-Time Transition Probabilities with Limits

(Time Period 'A' During Process Change)

Lower transition limits for States -4, -2, +1, +3 and +4 are all violated. As has already been noted, oscillations may cause transition probabilities to be low since long run-lengths are more common.

4.2 Experimental Testing – Two-Phase Flow

The two-phase flow experiment described in the experimental method section of Chapter 3 is controlled by a cascade strategy. The large air flow valve setpoint is set to 10 scfm to act as a disturbance. The Primary controller monitors pressure drop between the tube base and top and sends setpoints to the water flow, or Secondary, controller. The secondary controller is tuned first by the process reaction technique yielding a K_c of 2 %/error and a K_I of 0.40 seconds^{-1} where the PI controller algorithm is parallel. Next, the primary

controller is tuned with the same process reaction technique where K_c = 4 %/error and a K_I = 2 seconds^{-1} also using a parallel controller algorithm.

After tuning, "good" control data for both the Primary and Secondary controllers are collected on the CamileTG 2000 process control and acquisition system for approximately 1587 seconds (26.5 minutes). Data is collected at a data acquisition and control (DAC) frequency of 10 samples per second collecting a total of 15,870 DACS samples. Figure 4.14 reports a small sampling of actuating errors from the "good" period of data for the Primary controller.

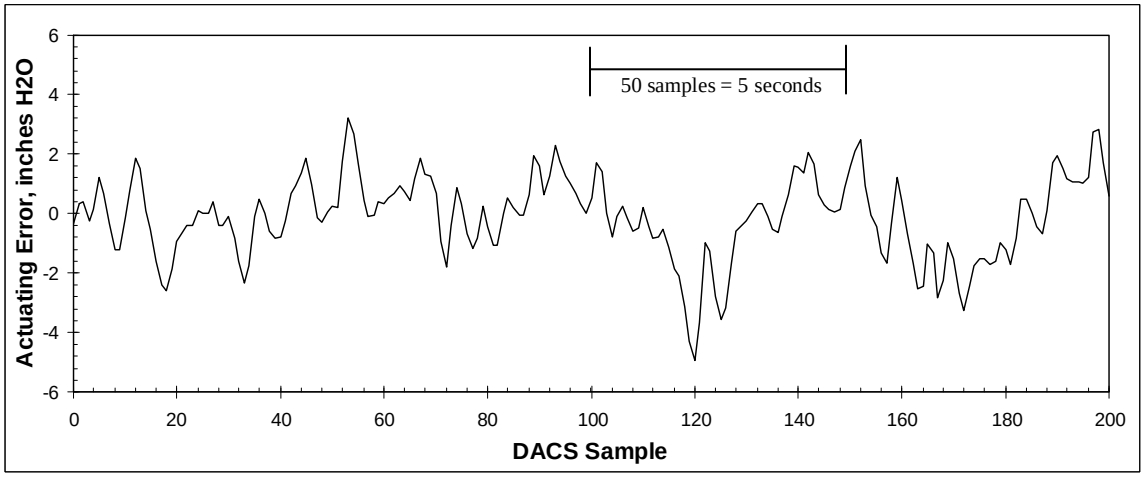

Figure 4.14: Good Data Actuating Errors (Primary Controler, Two-Phase Flow)

In the "good" data set collected for the primary controller there is a discrepancy between the actuating error mean -0.00033 inches H_2O and the mode 1.9291 inches H_2O. Therefore more samples actually reside above the zero axis. A Markov Chain where there are more samples in the positive runs is expected.

The health monitor analyzes the data using a specified Type I error (α) of 0.003 and a Type II error (β) of 0.003 corresponding to +/-3σ limits. In addition, the change of transition probability for the alternate hypothesis (λ) is 0.900. Finally, during step tests it is estimated that the settling time should be 20 seconds or 200 samples at the DACS frequency of 10 samples per second. Figure 4.15 reports the number of samples per state in vertical black bars and the null transition probabilities as a line.

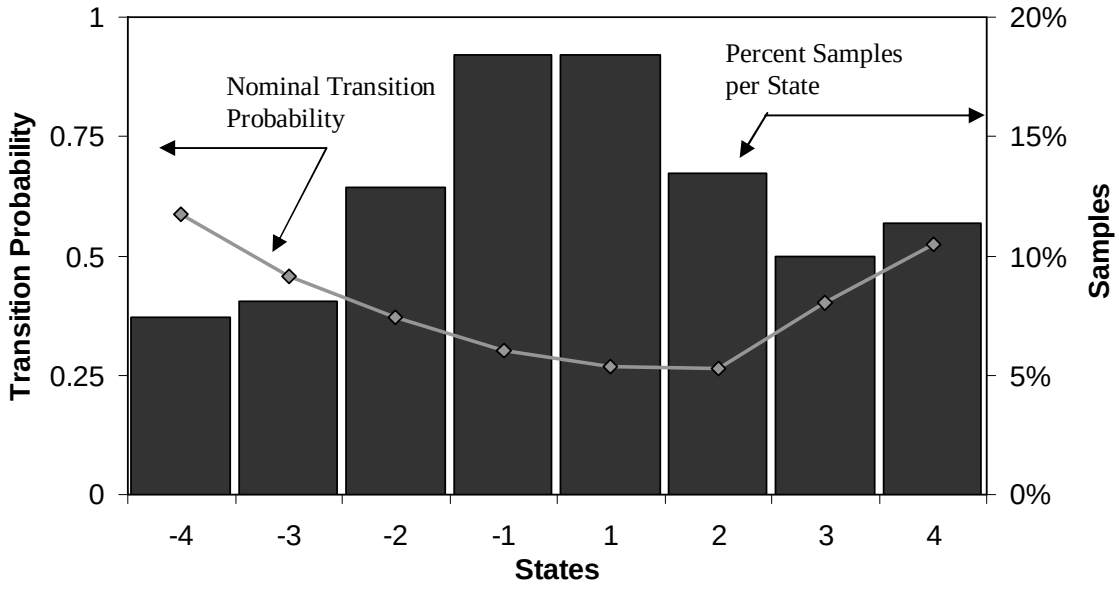

Figure 4.15: Markov Chain Model of Two-Phase Flow

(Primary Controller, SR = 8 and 8 states)

The health monitor determines that a sample ratio of 8 with an 8 state Markov Chain fulfills the requirement to have 20% of the data in the extreme states (+/-4) and finds the model requiring the shortest time window. The statistical window requires 719 samples which takes 575.2 seconds (9.6 minutes) to acquire due to the higher than unity sampling ratio. In addition, the settling time adds an additional 25 samples to the window for a complete window of 744 samples. In terms of the DACS, the complete window requires

5,952 samples due to the sampling ratio of 8. The transition limits obtained through Type I error testing are reported in Figure 4.16.

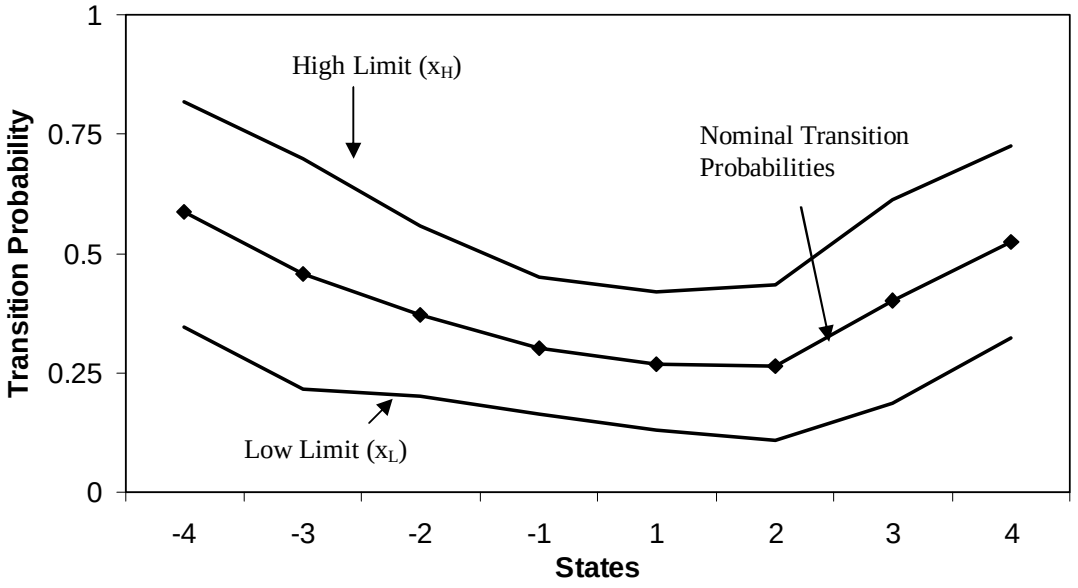

Figure 4.16: Transition Limits Surrounding Nominal Transition Probability for Two-Phase Flow (Primary Controller, $\alpha=\beta=0.003$, $\lambda=0.900$, Extreme = 20%)

The limits presented in Figure 4.16 confirm that there exists a higher propensity for negative runs to make zero crossings than for positive runs. Pressure drop is the controlled variable. Since water and air run co-current in the tube, the higher the desired pressure drop, the more water required to fill the tube. More water means more work pushing that water up the vertical pipe. Therefore, tracking the setpoint requires more work in increasing pressure drop than in decreasing pressure drop. This results in the propensity for the process to make fewer visits above the setpoint (negative runs) and spend more time below the setpoint. The null transition probabilities, along with their limits, demonstrate this fact.

In addition to collecting the primary controller response, the secondary controller actuating errors are collected and a small sampling of the good data set is reported in Figure 4.17.

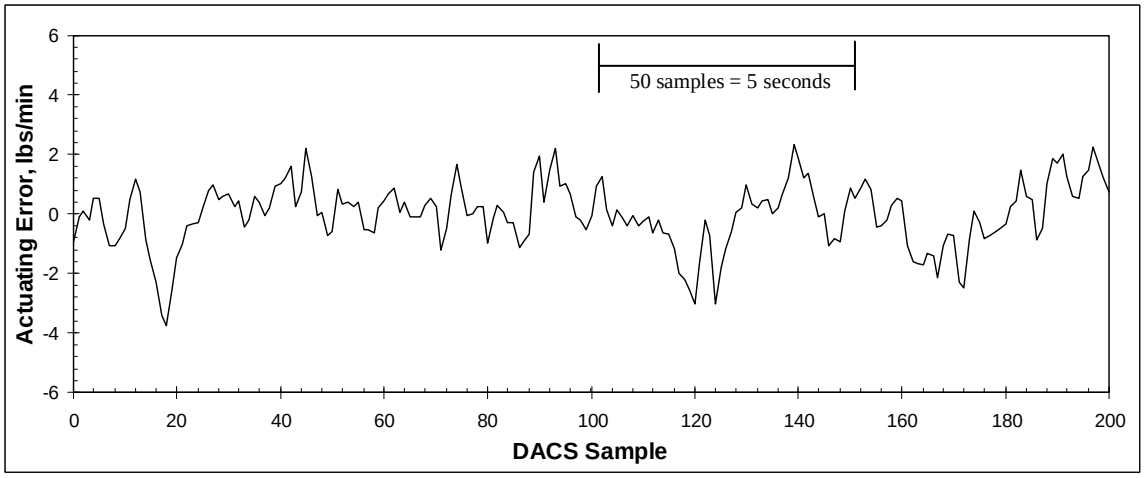

Figure 4.17: Good Data Actuating Errors (Secondary Controller, Two-Phase Flow)

The health monitor now analyzes the secondary controller error signal using a specified Type I error (α) of 0.003 and a Type II error (β) of 0.003 corresponding to +/-3σ limits. In addition, the change of transition probability for the alternate hypothesis (λ) is 0.900. Finally, during step tests it is estimated that the settling time should be 10 seconds or 100 samples at the DACS frequency of 10 samples per second. Figure 4.18 reports the number of samples per state in vertical black bars and the null transition probabilities as a line.

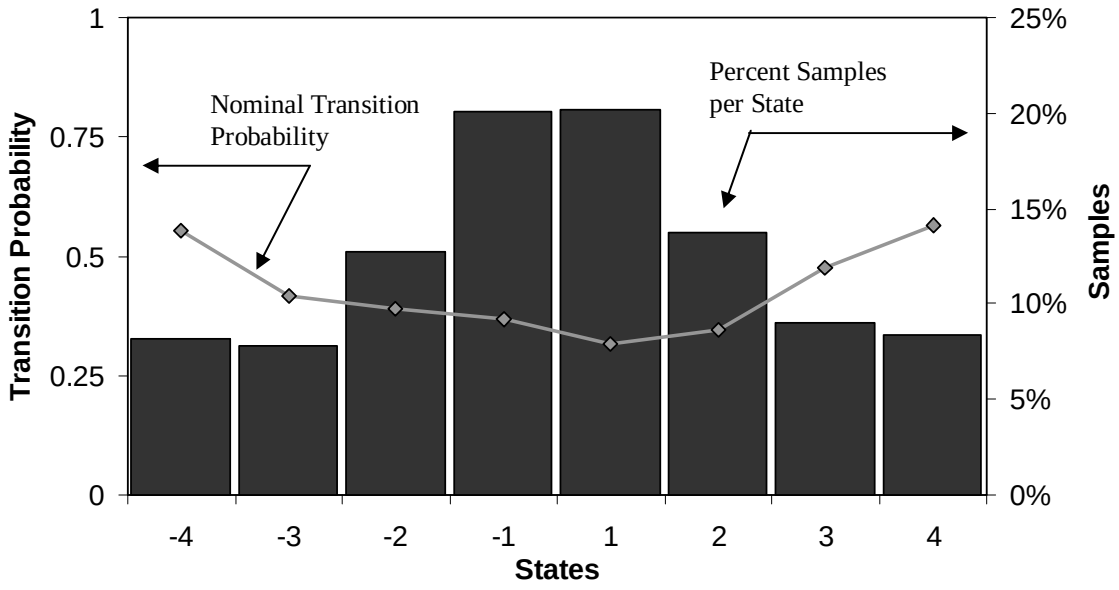

Figure 4.18: Markov Chain Model of Two-Phase Flow

(Secondary Controller, SR = 7 and 8 States)

The health monitor determines that a sampling ratio of 7 and a Markov Chain of 8 states meet the requirement of less than 20% of the data in the extreme states. This model also requires the fewest number of DACS samples. The statistical window found through the Type II error test is 669 samples requiring 468.3 seconds (7.8 minutes) to obtain. Including the settling time adds an additional 15 samples. In terms of the DACS, the complete window requires 4,788 samples due to the sampling ratio of 7. Figure 4.19 reports the limits found during Type I error testing.

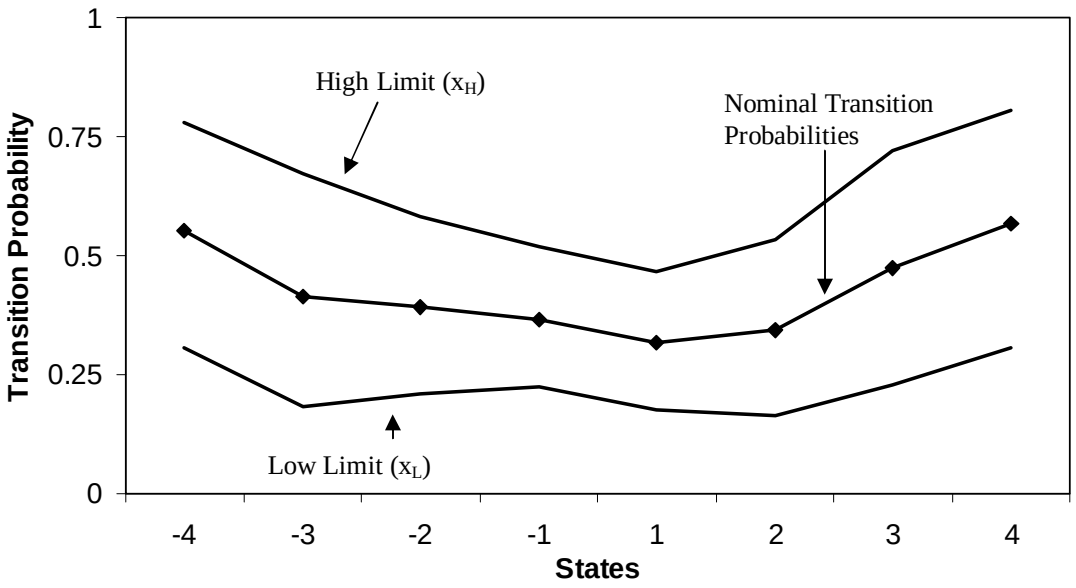

Figure 4.19: Transition Limits Surrounding Nominal Transition Probability for Two-Phase Flow (Secondary Controller, $\alpha=\beta=0.003$, $\lambda=0.900$, Extreme = 20%)

The transition limits in the secondary controller model are slightly further from the '0' probability bound. This results in fewer samples required in the Type II test to decide between the null and alternate hypothesis.

The health monitor is validated through real-time testing. If the monitor flags a process which is in "poor" control, and does not flag a process which is still in "good" control, the monitor is working as designed. Two cases show the effectiveness of the health monitor.

4.2.1 Case 1 – Setpoint Changes

Due to nonlinearities many processes are tuned only for a particular range of operation; the two-phase flow apparatus is such a process. Originally, the primary controller is tuned

to operate from 45 to 65 inches H_2O pressure drop. Outside of this pressure drop range the controller will not control in the same manner.

By step testing the primary controller to the physical operating limits of the secondary controller, these nonlinearities may be amplified and picked up by the health monitor. A nonlinear process operating outside of the region for which it is tuned is no longer under "good" control; therefore, these circumstances should cause the health monitor to flag.

All setpoint changes in this experiment are in increments of 10 inches H_2O and held for approximately 15 minutes (about 9,000 DACS samples). First the controller is tested within the region for which it is tuned. Second a region above the well tuned area is tested. This region is found to be the upper physical operating limit of the secondary controller. Third, the process is brought back into the well tuned region. Fourth, a region below the well tuned area is tested. This region is found to be the lower physical operating limit of the secondary controller. Fifth, the primary controller is once again brought into a region for which it is tuned. The experimental run ends at 136.2 minutes (total samples 81,725). Table 4.2 reports the setpoint changes made during this experiment.

Table 4.2: Setpoint Changes Two-Phase Flow Experiment

Region Letter	Time (minutes)	DACS Sample	Setpoint (inches H_2O)
A	0.0	0	50
-	15.7	9,425	60
B	30.8	18,506	70
-	45.9	27,573	60
-	61.1	36,659	50
-	76.2	45,714	40
-	91.3	54,769	30
C	106.4	63,861	40
-	121.6	72,962	50

Figure 4.20 reports the outcome of this test in relation to the primary controller. Note that the top section of the figure reports the setpoint and process variable, while the lower half shows the counting (dotted lines) and flagging (solid lines).

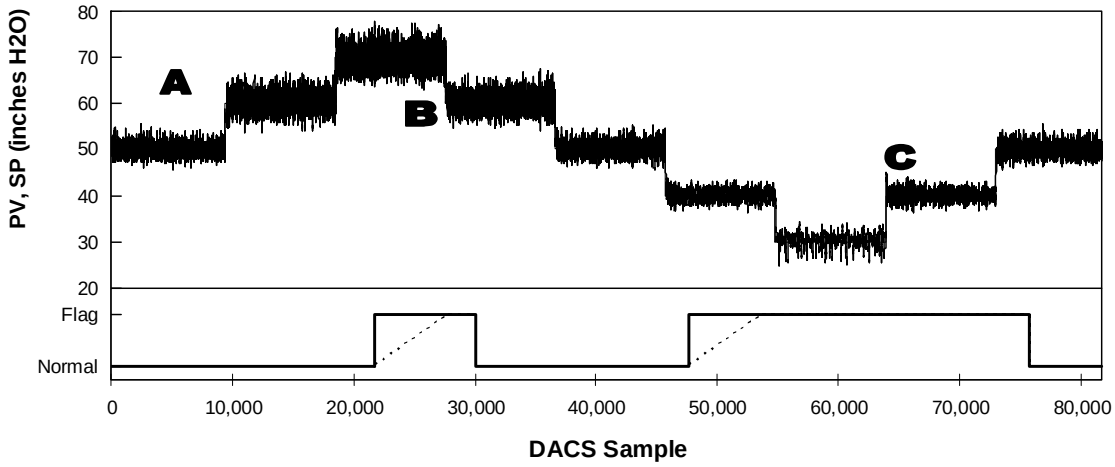

Figure 4.20: Two-Phase, Primary Controller Real-time Analysis During Setpoint Changes

The first section marked by an “A” is that region for which the primary controller is tuned. Figure 4.21 reports the transition probabilities of an example window included in period “A” and the transition limits found during the previous model acquisition.

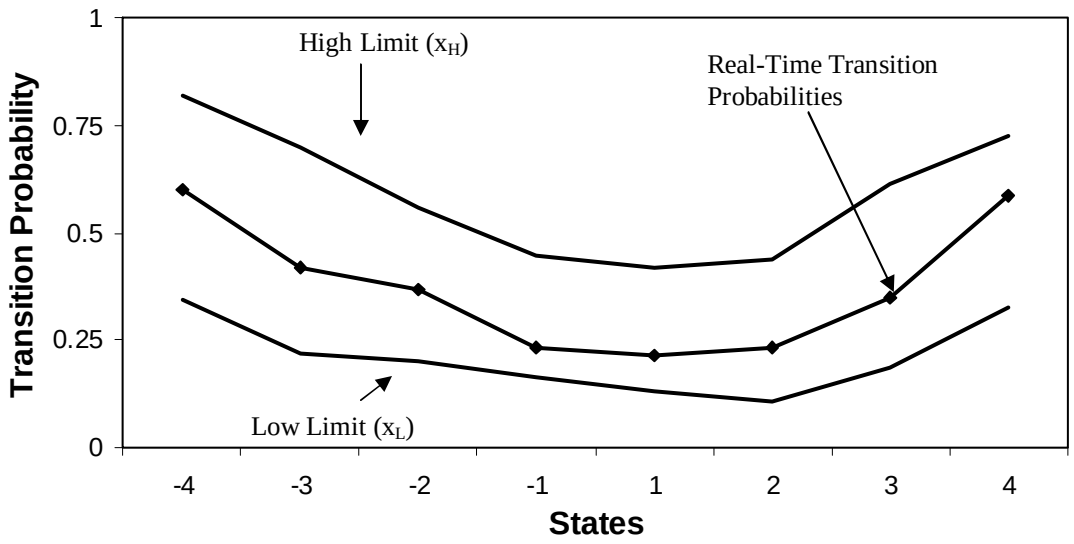

Figure 4.21: Two-Phase Flow, Primary Controller Real-Time Transition Probabilities with Limits (Time Period 'A' During Setpoint Changes)

All transition probabilities are within the limits for its corresponding state, which is expected for "good" control. However, when the setpoint is moved upward, the process moves into a region for which it is not tuned. Figure 4.22 reports the transition probabilities for an example window from the time period marked "B" in Figure 4.20.

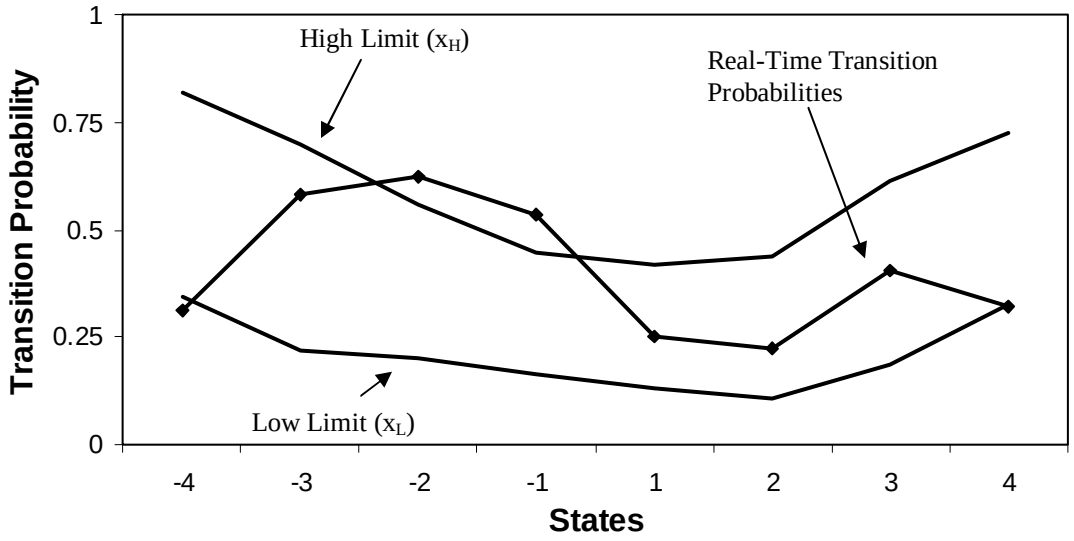

Figure 4.22: Two-Phase Flow, Primary Controller Real-Time Transition Probabilities with Limits (Time Period 'B' During Setpoint Changes)

Transition limits during the time period marked "B" are violated for states -4, -2, -1 and +4. The increased transition probability for negative states comes back to the original reasoning for why the nominal case contained more samples for the positive states than did the negative. As the setpoint increases, the secondary water flow valve is less able to increase flow rate to meet pressure drop setpoints. This causes for negative run lengths to have a high propensity to make zero crossings after only 1, 2 or 3 samples. The health monitor begins to notice the "poor" control somewhere around 2,160 seconds; however, the setpoint change is made at 1,850 seconds, meaning that 310 seconds pass before any transition limits are violated. This occurs because enough "poor" data points must populate the window such that remaining "good" data points do not dominate. In this case almost half a window length had to include the new "poor" data to draw the transition probabilities outside of their limits.

After the setpoint is set to the upper physical limits of the secondary controller flow valve, the primary controller setpoint is once again brought to a region of "good" control. The health monitor is slow to stop flagging because enough new "good" control data points must fill the window before transition probabilities return to within their limits

Finally, the setpoint is deceased to the lowest physical constraint of the secondary control flow valve. This period of time is marked by a "C". Figure 4.23 reports the transition probabilities for an example window found in this time period.

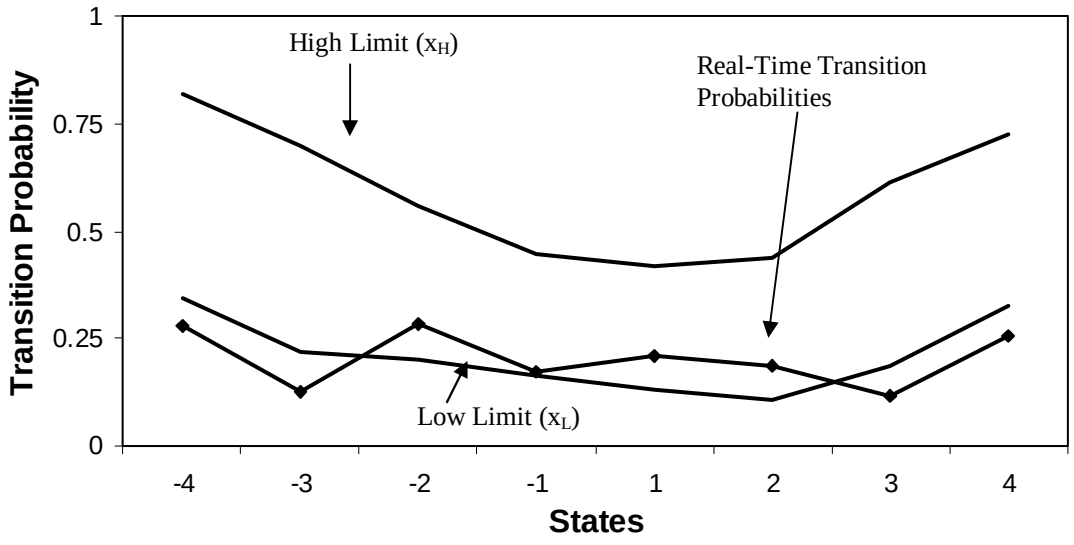

Figure 4.23: Two-Phase Flow, Primary Controller Real-Time Transition Probabilities with Limits (Time Period 'C' During Setpoint Changes)

The transition limits for states -4, -3, +3 and +4 all violate the lower control limit for their corresponding state. In fact, the low transition probabilities denote oscillations as was seen during the simulation section in the earlier portion of this chapter. Although difficult

to see in Figure 4.20 due to the time scale, if time period “C” is expanded to fill the plot, oscillations are visible.

Also of interest is the response of the secondary controller to each of these regions of control. Although setpoint changes are made to the primary controller, many of the nonlinear effects are caused by operating near the physical limits of the secondary controller flow valve. The health monitor is employed to monitor both controllers. Figure 4.24 reports the response of the secondary controller along with the health monitor evaluation of each control region.

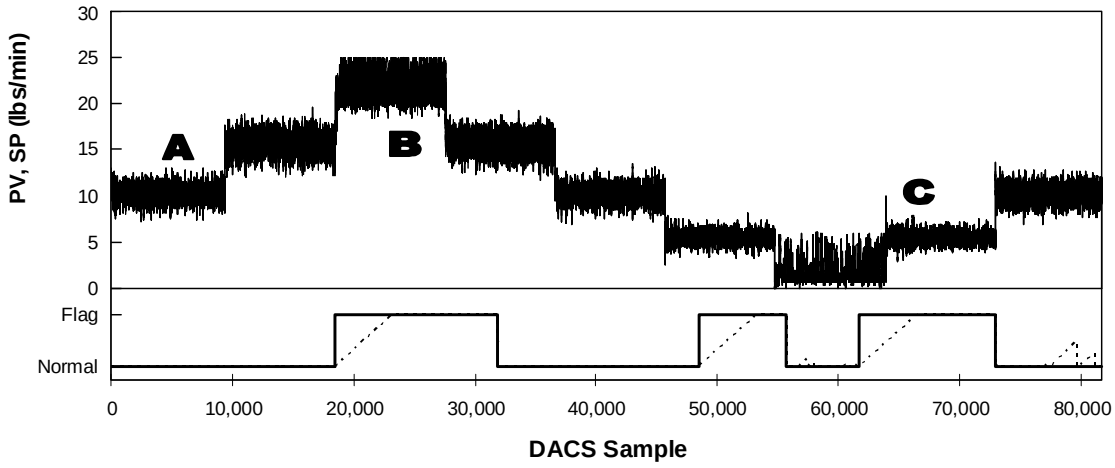

Figure 4.24: Two-Phase, Secondary Controller Real-time Analysis During Setpoint Changes

The letters marked on Figure 4.20 are marked again on Figure 4.24 as a reference. The health monitor correctly identifies each region; however seems to struggle identifying period “C” completely. From 92.9 minutes (DACS sample 55,720) to 102.9 minutes (DACS sample 61,740) during the time period “C” the flagging turned off. The counter is

frequently reset (denoted by dotted line) and never reaches the entire length of the complete window. This is indicative of a process right on the edge of "poor" control. So while the entire region may contain mostly "poor" control, it is possible for the monitor to stop flagging for brief periods of time when control is between "good" and "poor". Also, the setpoint changes are being made in the primary controller, which may mean that the secondary controller is still controlling the flow rate well enough to not cause significant changes to actuating error run-lengths. Whatever the reason may be, run-lengths were similar enough to the 'good' control data set, that the statistical parameters provided by the user could reject the null transition probabilities.

The health monitor effectively flags the "poor" control time periods and turns the flagging off during time periods of "good" control.

4.2.2 Case 2 – Controller Retuning

The second case used to demonstrate the effectiveness of the health monitor is to show that the monitor can flag a process which has been retuned. The pressure drop setpoint remains a constant 50 inches H_2O. The disturbance (air flow) is modulated between the initial value of 10 scfm and 15 scfm every 2 minutes. During the test the primary controller proportional and integral gains are retuned then the secondary controller proportional and integral gains are retuned. Each setting in Table 4.3 is held for approximately 15 minutes (9,000 DACS samples).

Table 4.3: Controller Retuning Two-Phase Flow Experiment

Region Letter	Time (minutes)	DACS Sample	Primary K_c	Primary K_I	Secondary K_c	Secondary K_I
A	0.0	0	2.0	0.4	4.0	2.0
B	15.1	9,070	8.0	1.6	4.0	2.0
-	30.3	18,195	2.0	0.4	4.0	2.0
C	45.5	27,289	0.5	0.1	4.0	2.0
D	60.6	36,379	2.0	0.4	4.0	2.0
E	75.7	45,455	2.0	0.4	16.0	8.0
-	90.9	54,535	2.0	0.4	4.0	2.0
F	106.0	63,621	2.0	0.4	1.0	0.5
-	121.1	82,378	2.0	0.4	4.0	2.0

Table 4.3 also provides the reader with a list of the letters designating regions of control used in Figures 4.25 through 4.34. The health monitor analyzes both the real-time primary controller error signal and the real-time secondary error signal. Figure 4.25 reports the outcome from the primary controller view and Figure 4.26 graphically shows the controller retuning pattern.

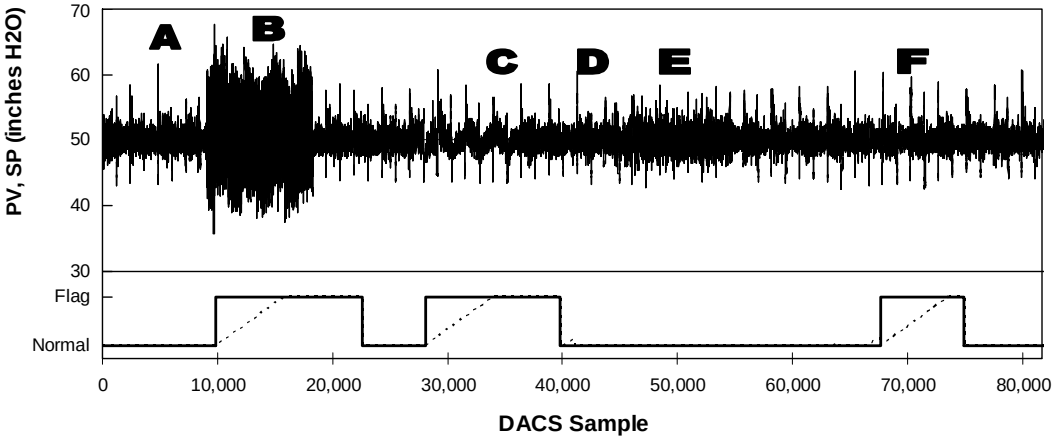

Figure 4.25: Two-Phase, Primary Controller Real-time Analysis

During Controller Retuning

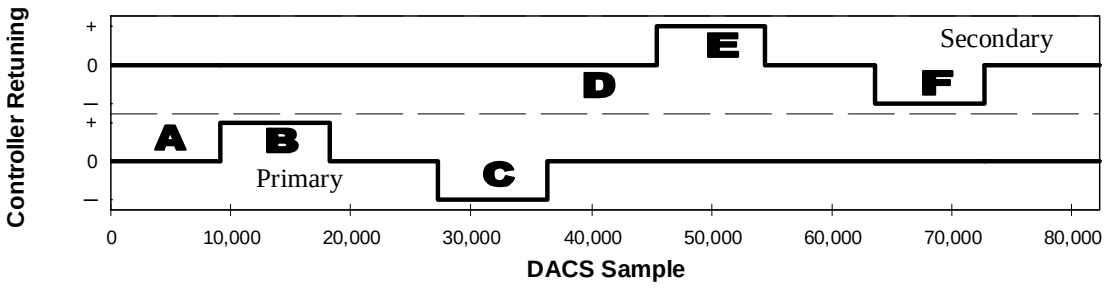

Figure 4.26: Two-Phase, Real-Time Controller Retuning

(+ = Increased Gain, - = Decreased Gain)

The first time period marked by "A" represents a period of "good" control where primary and secondary controller gains are properly tuned. Figure 4.27 reports the transition probabilities compared to model limits for an example window found in time period "A".

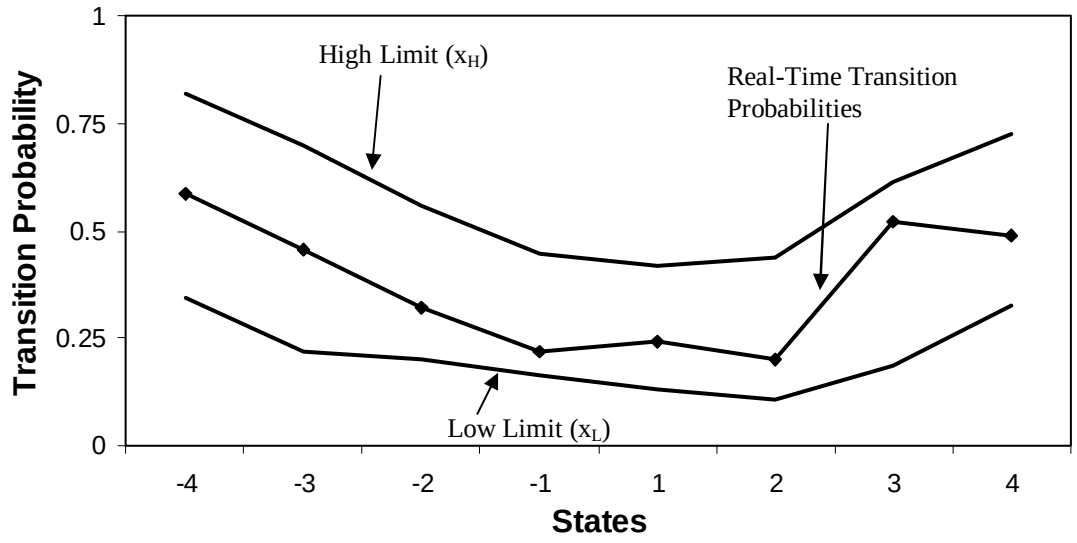

Figure 4.27: Two-Phase Flow, Primary Controller Real-Time Transition Probabilities with Limits (Time Period 'A' During Controller Retuning)

All transition probabilities are found within their limits, verifying that control is still as it was intended during tuning. At 15.1 minutes (DACS sample 9,070) the controller is made

more aggressive by increasing the proportional and integral gains by 400%. In Figure 19 this is seen as a change from "0" to "+" on the lower half marked "Primary". Also note that the secondary controller gains remain tuned correctly. The time period marked "B" in Figure 4.25 represents the effects of the increased gains. The transition probabilities compared to limits are reported in Figure 4.28 for a representative window found during time period "B".

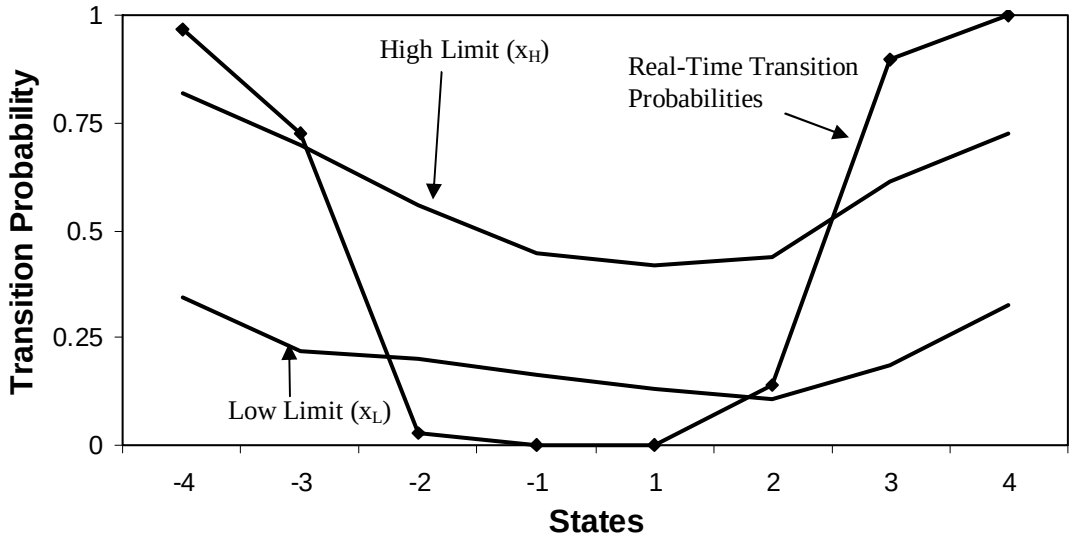

Figure 4.28: Two-Phase Flow, Primary Controller Real-Time Transition Probabilities with Limits (Time Period 'B' During Controller Retuning)

Only the transition probability for state +2 remains within the limits for the primary controller transition probabilities. The aggressive controller has pushed control into oscillations; however, the oscillations have short run lengths and behave more like quickly correcting overshooting. Transition probabilities for the states closest to "0" are very low denoting that run lengths of +/-1, +/-2 seldom end, but continue to longer run lengths to at least 3 and sometimes 4 samples. Recall that the sampling ratio (SR) for the

real-time monitor is 8, meaning 1 out of every 8 DACS samples is collected in the real-time window. Were the sampling ratio unity, the run lengths would be longer and may look more similar to the simulated process response to increased gain found in Figure 4.8.

Next, at 30.3 minutes (DACS sample 18,195) gains are returned to proper tuning, which results in "good" control as indicated by a cease in flagging. Then at 45.5 minutes (DACS sample 27,289), the primary controller is retuned to sluggish control by decreasing both gains to 25% that of proper tuning. Figure 4.29 reports the resulting primary controller transition probabilities for a representative window during the time period marked "C".

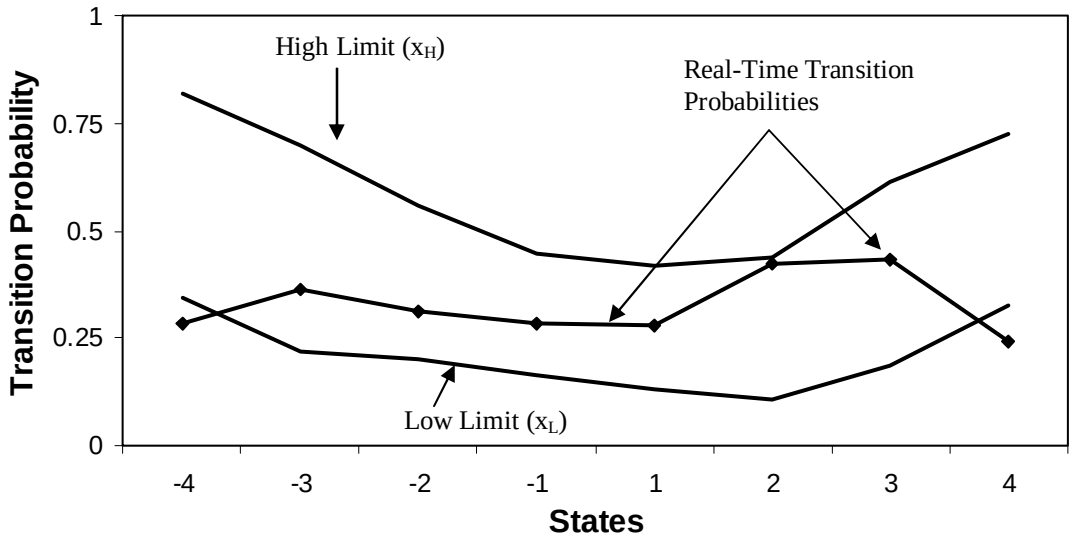

Figure 4.29: Two-Phase Flow, Primary Controller Real-Time Transition Probabilities with Limits (Time Period 'C' During Controller Retuning)

Only states +/- 4 violate transition limits. Since the experiment is performed with periodic disturbance steps (large air flow valve), when the controller is sluggish, the

controlled variable struggles to track the setpoint 50 inH_2O through periodic air disturbances. Therefore, longer runs on either side of the chain is expected and with longer runs comes a lower propensity to cross the zero axis.

After the gain is once again reset to proper tuning values at 60.6 minutes (DACS sample 36,379), the secondary controller proportional and integral gains are retuned. During time period ‘E’ the secondary controller gain is increased by 400% while the primary controller gains remain constant. The real-time health monitor does not flag the primary controller process during this period. This simply indicates that in retuning the secondary controller, gains may not have been moved enough to cause poor control in the primary controller. In addition, it is possible that oscillations induced in the secondary controller may not translate to the primary controller if the primary controller reacts more slowly than the secondary controller.

Figure 4.30 reports the outcome of the gain changes reported in Table 4.3 from the secondary controller viewpoint; also Figure 4.26 is presented again as Figure 4.31.

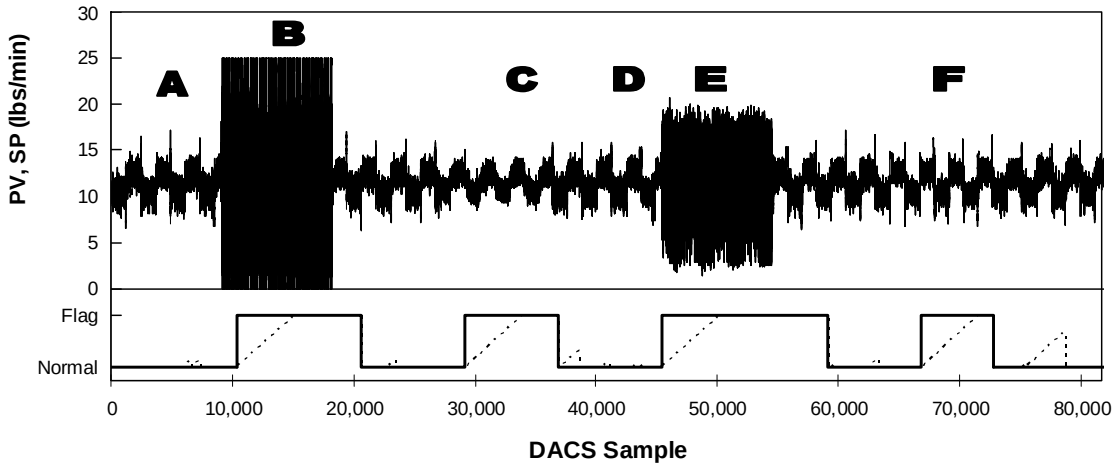

Figure 4.30: Two-Phase, Secondary Controller Real-time Analysis

During Controller Retuning

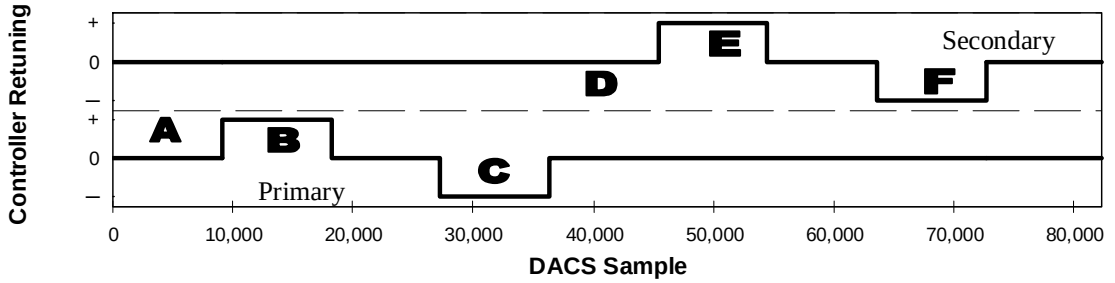

Figure 4.31: Two-Phase, Real-Time Controller Retuning

(+ = Increased Gain, - = Decreased Gain)

The health monitor recognized all four periods of “poor” control by raising a flag and, turned flagging off during periods of “good” control. Only transition probabilities resulting from secondary controller changes are shown here. First, Figure 4.32 reports the brief period of “good” control preceding secondary controller retuning which begins at 60.6 minutes (DACS sample 36,379). The transition probabilities for the period marked ‘D’ is reported in Figure 4.30.

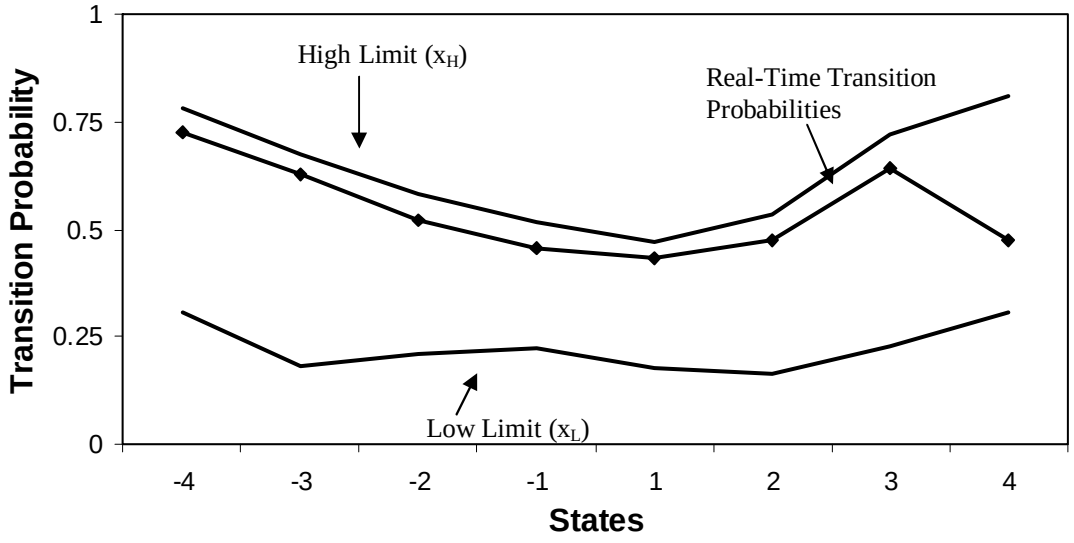

Figure 4.32: Two-Phase Flow, Secondary Controller Real-Time Transition Probabilities with Limits (Time Period 'D' During Controller Retuning)

All transition limits are within their limits before the secondary controller gains are increased. At 75.7 minutes (DACS sample 45,455) the secondary controller is made more aggressive by increasing the proportional and integral gains by 400% introducing oscillations into the system. Figure 4.33 reports the secondary controller transition probabilities for a representative window in time period 'E'.

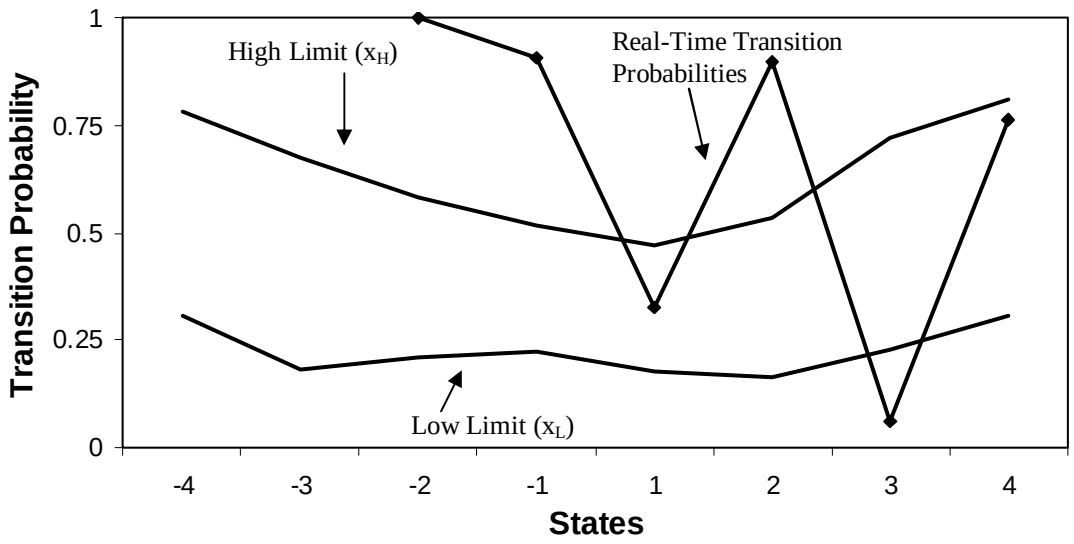

Figure 4.33: Two-Phase Flow, Secondary Controller Real-Time Transition Probabilities with Limits (Time Period 'E' During Controller Retuning)

Only two states, State +1 and +4, did not violate transition probabilities during this test. No transition probability is reported for States -4 and -3 because all run-lengths end after 2 negative samples. The response is very similar to the oscillations found when the primary controller gains are increased during the time period marked 'B' found in Figure 4.28. This period featured short run-lengths. Also, since the sampling ratio is 7, run-lengths may actually be between 7 and 14 DACS samples in length. In addition, Figure 4.33 reports very different transition probabilities between negative and positive states, likely due to process nonlinearity.

Beginning at 106 minutes (DACS sample 63,621) the controller gains are decreased by 400% to produce sluggish control. Figure 4.34 reports the transition limits for a representative window during time period "F".

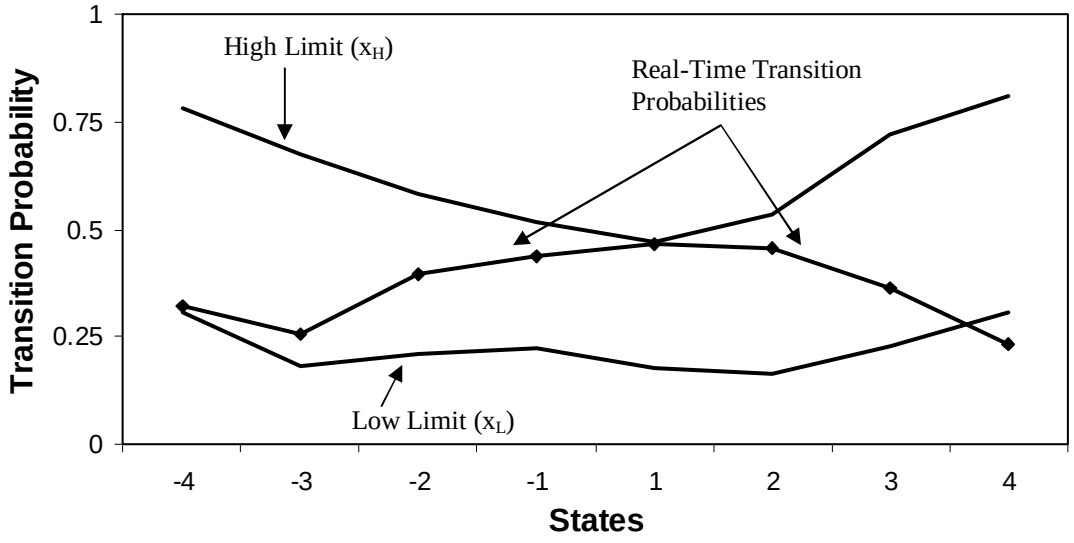

Figure 4.34: Two-Phase Flow, Secondary Controller Real-Time Transition Probabilities with Limits (Time Period 'F' During Controller Retuning)

Only state +4 violates a limit and two other states -4 and +1 are close to limit violations; however, only the violations start the health monitor counter. In the upper half of Figure 4.30, where actual process response is reported, it is difficult to distinguish sluggish control from "good" control, however this is a result of time scale. Figure 4.35 reports from 101.7 minutes (DACS sample 61,000) through 110 minutes (DACS sample 66,000).

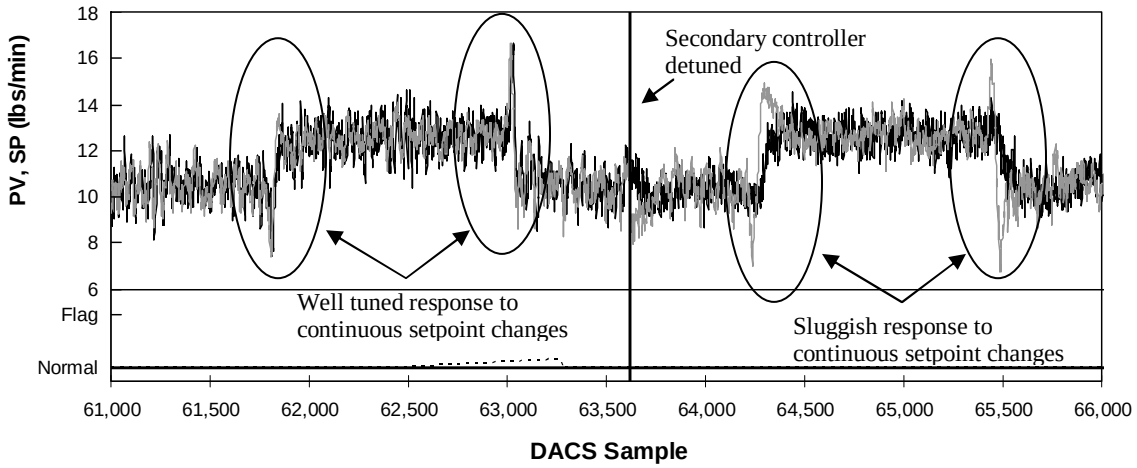

Figure 4.35: Two-Phase, Secondary Controller Real-time Analysis During Controller Retuning (Showing Sluggish Control)

The setpoint line is grey and overlays the process response line in black. Prior to minute 106 (DACS sample 63,621), demarcated by the black vertical line, there is little discernable difference between the setpoint and process response. This is because the controller is well tuned in this region. After minute 106 (DACS sample 63,621) the process tracks the setpoint more sluggishly. This is most obvious immediately following disturbances. Also note that there is no flagging during the period reported in Figure 4.35 because not enough sluggish control data has filled the health monitor window yet.

After 121.1 minute (DACS sample 72,680) the secondary controller is retuned to proper values and the performance improves until the health monitor ceases to flag the secondary controller, mirroring the health monitor's response to a return to "good" control in the primary controller.

4.3 Effects of Type I and Type II Error Rates

To determine the Markov Chain model used in real-time monitoring, the user specified a Type I error (α) of 0.003 and a Type II error (β) of 0.003 corresponding to +/-3σ limits. Also, λ was specified as 0.900, meaning that a 90% change in transition probability should be identified. These statistical parameters were set for both the primary and secondary controller. The primary controller was found to require a statistical window of 719 health monitor samples, which takes 9.6 minutes to collect at a sampling ratio of 8. The secondary controller was found to require a statistical window of 669 health monitor samples, which takes 7.8 minutes to collect at a sampling ratio of 7. These statistical windows seem long given that the primary and secondary systems respond to setpoint changes in 20 and 10 seconds respectively. This is likely due to the stringent statistical parameters provided by the user. Were these parameters to be relaxed, the windows would be shortened. This idea is explored with the secondary controller. Then, when a suitable Markov Chain model has been found using the more relaxed statistical parameters, real-time identification is demonstrated on the controller retuning experiment.

Let $\alpha = 0.1$, $\beta = 0.1$ and $\lambda = 0.990$ when determining the correct Markov Chain model. The health monitor analyzes the 15,870 DACS samples during the period of "good" control. Figure 4.36 reports the number of samples per state in vertical black bars and the null transition probabilities as a line.

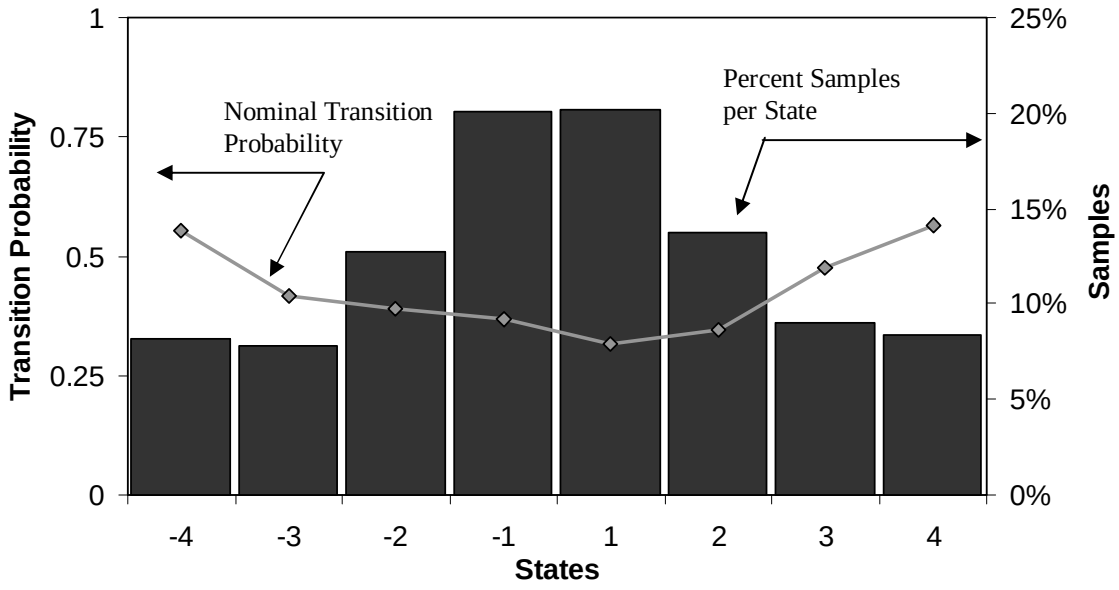

Figure 4.36: Markov Chain Model of Two-Phase Flow

(Secondary Controller, SR = 7 and 8 States)

The health monitor determines that a sampling ratio of 7 and a Markov Chain of 8 states exactly the same combination selected for the original statistical parameters. The statistical window found through the Type II error test is 220 health monitor samples requiring 154 seconds (2.6 minutes) to obtain. Including the settling time adds an additional 15 health monitor samples. In terms of the DACS, the complete window requires 1,645 DACS samples due to the sampling ratio of 7. The window obtained through relaxing the statistical parameters requires 34% as many DACS samples as the window obtained through more stringent statistical parameters. Figure 4.37 reports the limits found during Type I error testing.

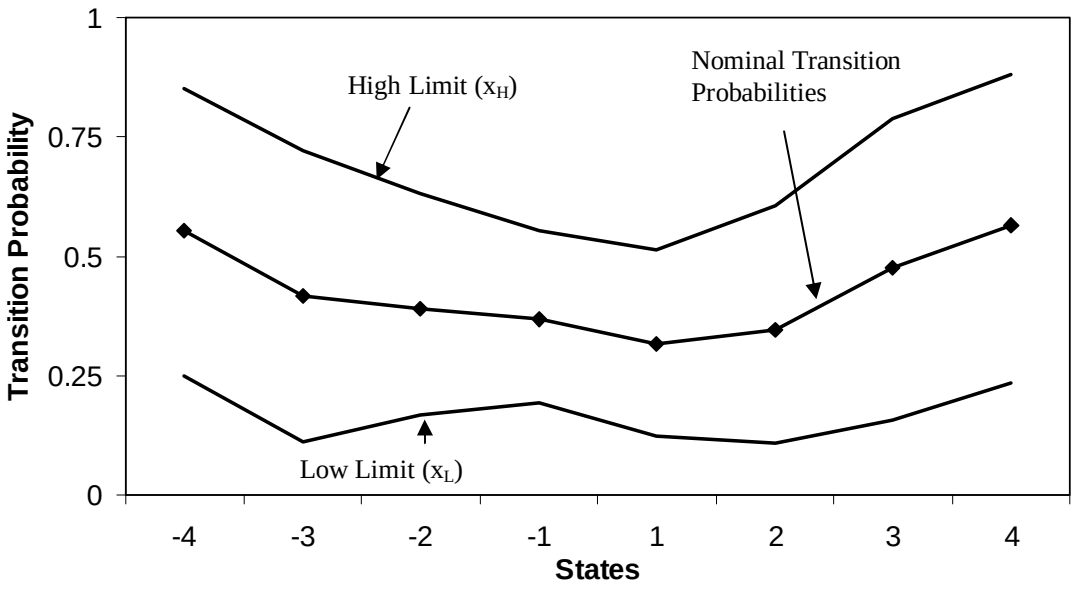

Figure 4.37: Transition Limits Surrounding Nominal Transition Probability for Two-Phase Flow (Secondary Controller, $\alpha=\beta=0.1$, $\lambda=0.990$, Extreme = 20%)

Now, the more relaxed statistical window with the limits reported in Figure 4.37 are tested against the real-time controller gain chance experiment. Figure 4.30 reported the outcome of the gain changes found in Table 4.3 from the secondary controller viewpoint when more stringent statistical parameters were used. Figure 4.38 reports the real-time monitoring results using the relaxed statistical parameters.

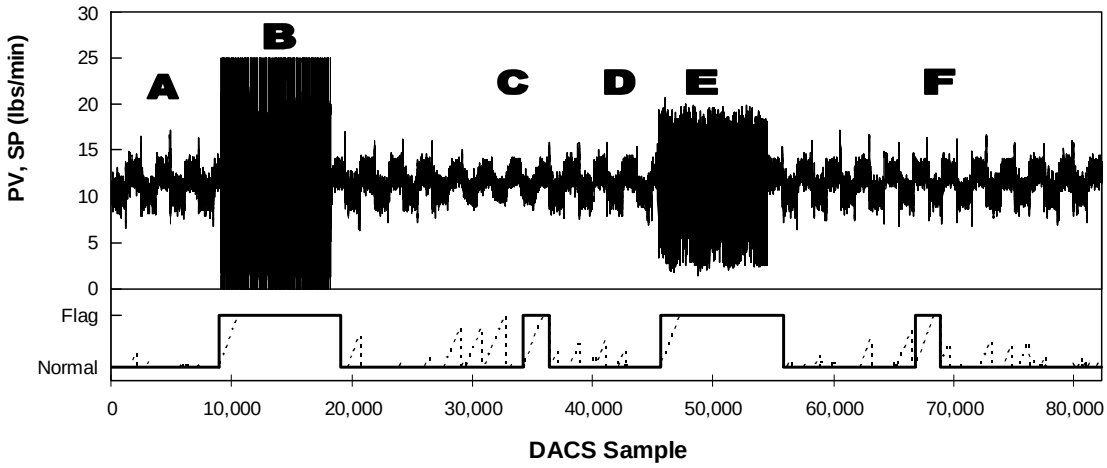

Figure 4.38: Two-Phase, Secondary Controller Real-time Analysis During Controller Retuning ($\alpha=\beta=0.1$, $\lambda=0.990$, Extreme = 20%)

The bottom portion of Figure 4.38 reports where the health monitor indentifies “poor” control by flagging. Because the window is shorter, the health monitor is much quicker in flagging; whereas in Figure 4.30 there appeared some lag between where obvious “poor” control began (such as visible oscillations during time periods ‘B’ and ‘E’) and when flagging begins. With a larger statistical window, more “poor” control samples are required to replace the “good” control samples, resulting in some lag. The smaller window simply means less “poor” samples need to populate the statistical window before they dominate and push real-time transition limits outside their limits. In addition, all 4 periods of poor control are indentified. However, during each sluggish period, ‘C’ and ‘F’, there are periods where the health monitor begins to count violated limits, but the counter is reset by very brief periods where the controller finally settles. This is seen as sawtoothed dotted lines in the lower portion of Figure 4.38.

4.4 Effects of Percent of Visits in Extreme States

The Markov Chains describing the run-lengths in each of the experiments found in Chapter 2 have been determined by ensuring that no more than 20% of all run-lengths visit the Extreme States. Owusu noted that a 8 state Markov Chain where all states are independent will have 12.5% of all run-lengths visit the +/-4 State. For simplicity, he proposed that 10% be selected as the threshold.

Return the Type I and Type II error rates to their more stringent values where $\alpha = 0.003$, $\beta = 0.003$ and $\lambda = 0.900$ when determining the correct Markov Chain model. The health monitor analyzes the 15,870 DACS samples during the period of "good" control, but now uses 10% as the threshold for run-length visits to the Extreme States. Figure 4.39 reports the number of samples per state in vertical black bars and the null transition probabilities as a line.

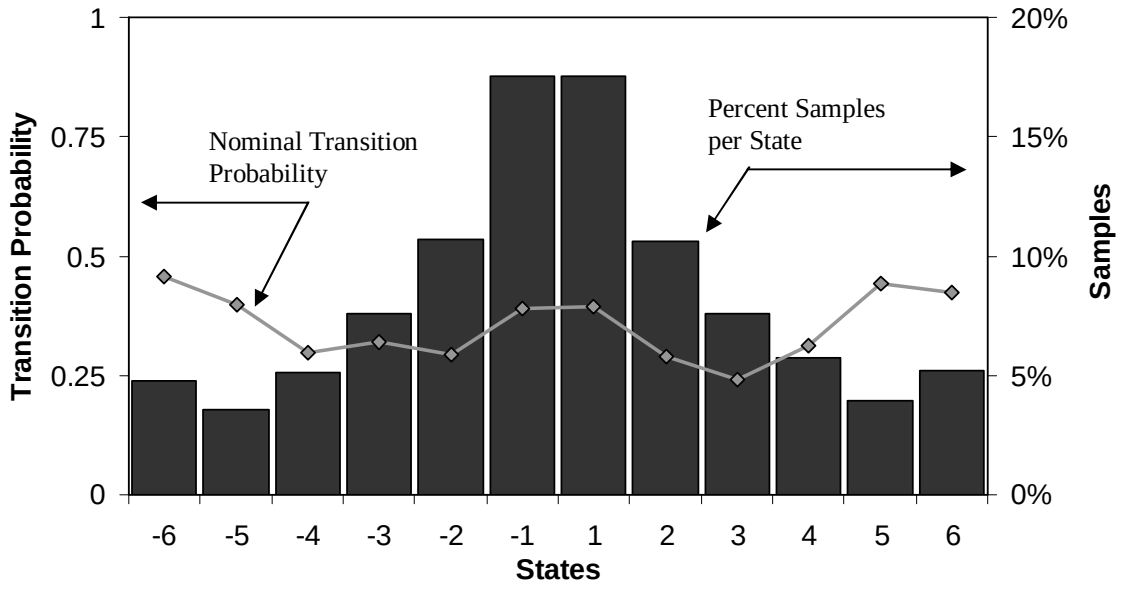

Figure 4.39: Markov Chain Model of Two-Phase Flow

(Secondary Controller, SR = 4 and 12 States)

The health monitor determines that a sampling ratio of 4 and a Markov Chain of 14 states contain fewer than 10% of run-length visits in the Extreme States (+/-6). The statistical window found through the Type II error test is 1568 health monitor samples requiring 627.2 seconds (10.5 minutes) to obtain. Including the settling time adds an additional 25 health monitor samples. In terms of the DACS, the complete window requires 6372 DACS samples due to the sampling ratio of 4. This window requires 25% more DACS samples than when the upper threshold for run-length visits to the Extreme States is 20%. Figure 4.40 reports the limits found during Type I error testing.

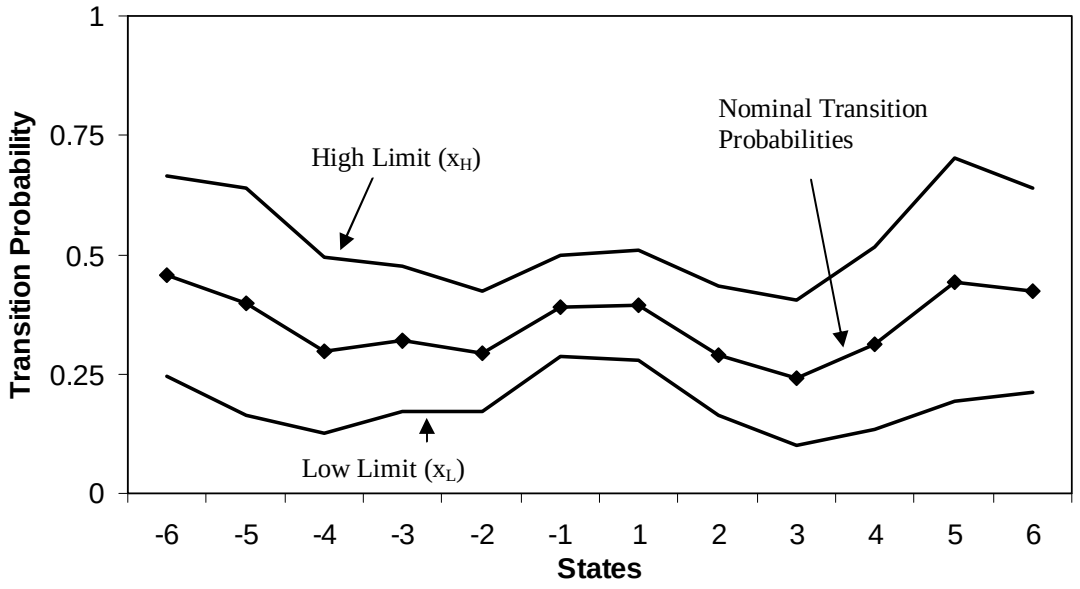

Figure 4.40: Transition Limits Surrounding Nominal Transition Probability for Two-Phase Flow (Secondary Controller, $\alpha=\beta=0.003$, $\lambda=0.900$, Extreme = 10%)

The window and limits are tested on the real-time controller retuning experiment to assess health monitor performance when 10% is the Extreme State threshold. Figure 4.41 reports the health monitor control identification.

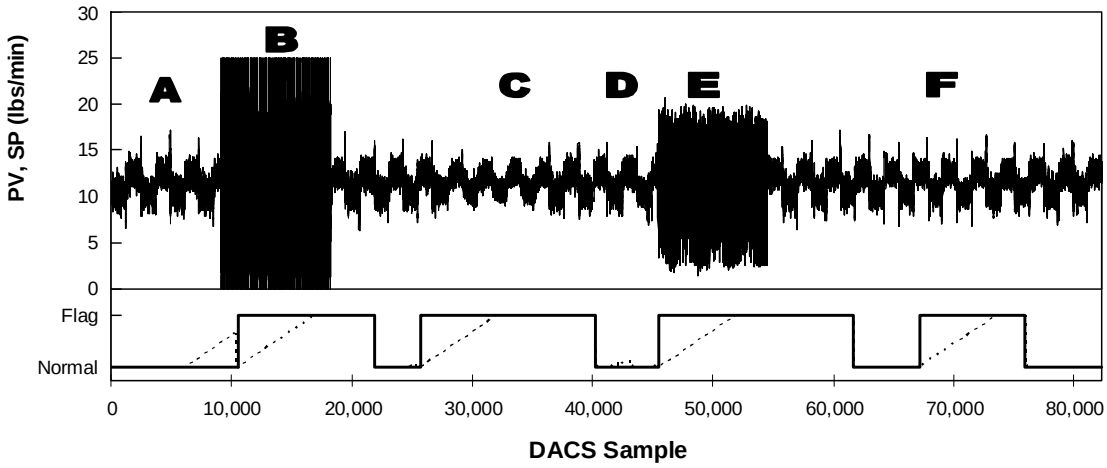

Figure 4.41: Two-Phase, Secondary Controller Real-time Analysis During Controller Retuning (Extreme = 10%)

The health monitor successfully identifies all periods of "poor" control. However, the increased window length does mean more 'poor' data must populate the window for the health monitor to flag 'poor' control. On the other hand, more 'good' data must also populate the window for the flagging to cease, which is evident immediately following time period 'E'. Oscillations stop at DACS sample 54,535, but 'poor' control is still flagged until DACS sample 61,608 (11.8 minutes later). This compared to Figure 4.30 when flagging persisted after time period 'E' for an additional 7.6 minutes.

Through this demonstration is appears that the most important reason for choosing a higher threshold for run-length visits to the Extreme States leads to a shorter statistical window, which entails more quick reaction to controller upsets. However, more experimental evidence may lead to other conclusions as well.

CHAPTER V

DISCUSSION AND RECOMMENDATIONS

Four future improvements may further improve upon the ideas presented in this work. The first improvement deals with the requirement to have the same number of positive and negative states. Nonlinear processes may naturally contain more samples on one side of the Markov chain than on the other. For instance Figure 4.15 reported the number of samples and nominal transition probabilities from the Case 2 “good” period of control for the primary controller. Here, the figure is reproduced as Figure 5.1 and only shows the number of samples which visited each state.

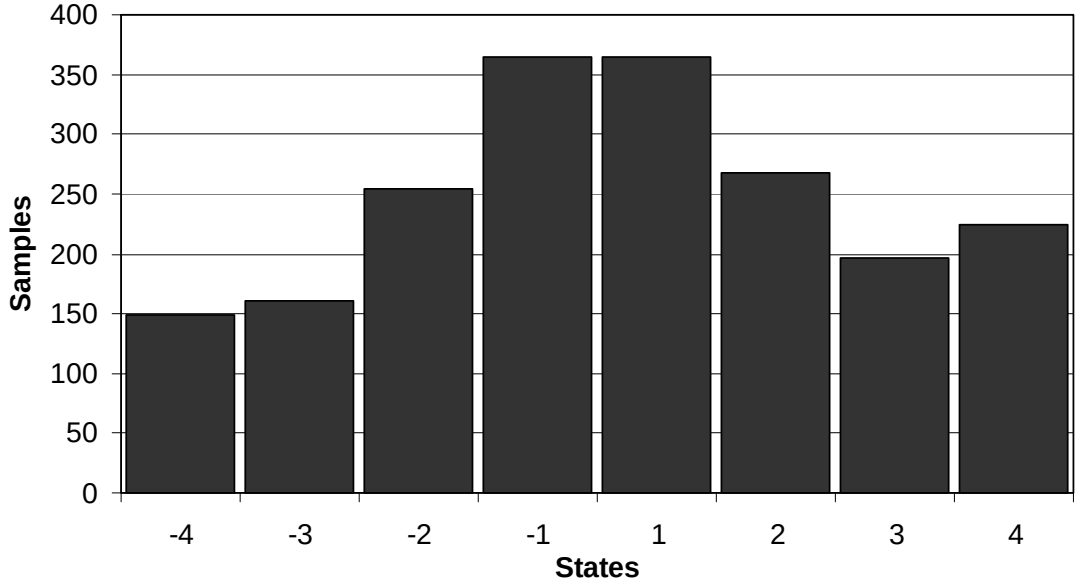

Figure 5.1: Markov Chain Model of Two-Phase Flow Without Transition Probabilities (Primary Controller, SR = 8 and 8 states)

The obvious comparison are States -4 and +4 where 77 more samples visit State +4 than State -4 during the period of "good" control. This means that 60.3% of all samples visiting an extreme state visit the +4 State. In addition, 1054 samples visited the positive states while 928 samples visited the negative states, meaning that 53.2% of all samples are found in the positive states (below the setpoint).

The algorithm used to determine sampling ratio (SR) and number of samples explained in Chapter 2 could be altered to only add one state to the side whose extreme state contained the most sample visits. If each extreme state is viewed on its own, instead of requiring less than 20% of all samples be found in the total extreme states, less than 10% should be found in each individual extreme state. For nonlinear systems where more samples visit

one side of the Markov chain over the other, this could result in fewer total states. For example, if the negative extreme state already contained just 10% of the total sample visits, a state is added to the positive side of the Markov chain. If the addition of this one state to the positive side decreases the number of sample visits in the positive extreme state below 10% of the total samples, the correct model is found.

The second improvement involves the way in which the statistical window used in real-time monitoring is determined. Currently, as explained in Chapter 2, each half of the Markov chain is independently evaluated. A state on each side of the chain is selected as the base state from which the expected number of sample visits per state is determined. With each half constructed based on separate states (one on each side) the statistical window is then found to be the sum of the expected number of samples visiting each chain half. For nonlinear systems, this produces a reasonable result since each half will appear similar. However, when the nominal transition probabilities for each half are substantially different, as is the case for nonlinear systems, the number of samples expected on each half can no longer be treated as independent.

Consider the primary controller nominal transition probabilities reported in Figure 4.16. These probabilities are reported in Figure 5.2 along with the expected number of samples to visit each state, based on the base state in each half.

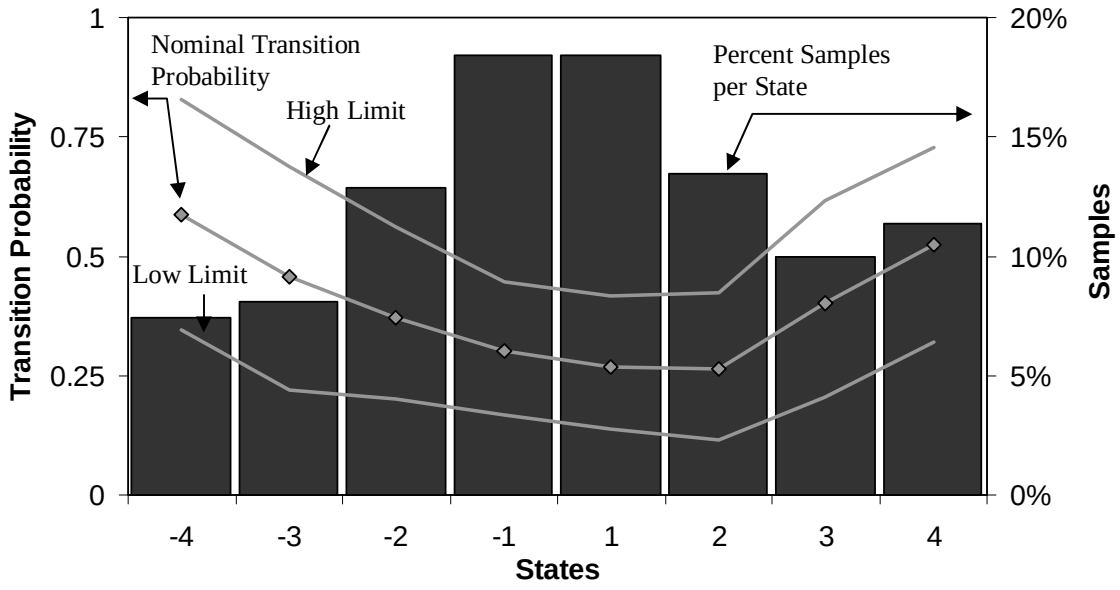

Figure 5.2: Markov Chain Model of Two-Phase Flow With Calculated Samples Per State (Primary Controller, SR = 8 and 8 states)

To remind the reader, the statistical window is found to require 719 total health monitor samples, found by summing the sample visits for each state in Figure 5.2. The sample visits to States +/-1 should differ by no more than 1 sample; however, since each half is calculated independently, the number of samples in State +1 will not necessarily equal the number of samples in State -1. A method will need to be developed to take the interdependence between each Markov Chain half into account when building the statistical window.

Third, the required number of sample visits to the extreme states is a condition not well grounded. The original work called for no more than 10% of all sample visits to be made to the extreme states. This was a “first-take” at a decision value with not substantive theoretical basis. This work proposes that 10% of the samples may lead to more states

than is sufficient, which leads to excessive window length. Some decision boundary is necessary in the current version of the health monitor. The value of 20% is chosen for this work to show that a larger value may provide sufficient control performance identification. It provides fully adequate health monitor performance with a smaller statistical window. Further work is needed to develop a theoretical framework to support the inclusion of this condition.

Fourth, practical Statistical Process Control (SPC) employs a Type-I test from the user specified α to create SPC limits. The Type-II test performed in this work rigorously decides how many samples are required in a window to avoid accepting the null hypothesis when it should be rejected. Often, the Type-II test is omitted in favor of simply setting a practical window size of perhaps 100 or 200 samples. While this method is not fundamentally correct, it has found much success in industry. For this work to be practically implemented in an industrial setting, the Type-II test may need to be removed in favor of setting a fixed window length.

CHAPTER VI

CONCLUSIONS

6.1 Conclusions

A control loop health monitor is hereby proposed. This work makes the following five conclusions regarding updates to Owusu's original work:

1) The revised algorithm to compute the binomial distribution is both more robust and more efficient. Overflow run-time errors are now avoided by performing the order of calculations based on remaining close to the nominal value of 0.5. In addition, only binomial probabilities which would be larger than 1E-16 are calculated to further guard against overflow run-time errors. This also leads to more efficient calculations since only those calculations significant enough for the study are performed.
2) Modifications in this work shortened the window length while meeting Type I and Type II conditions demonstrated by results reported in Chapter 2. According to the example provided in this chapter, the improved algorithm estimated that the real-time window only be 19% of the DACS samples that the previous Owusu algorithm required. This results in quicker diagnoses of controller performance.

3) The health monitor flags when it should, and does not flag when it should not. This is demonstrated by four cases for a simulated process under PI control and two cases for a pilot-plant scale two-phase flow experiment under cascade control where both the primary and secondary controller are PI controllers.
4) Relaxing Type-I and Type-II error rate parameters leads to shorter window lengths. This increases the probability that a region of control performance may be misdiagnosed as is seen during sluggish control in Figure 4.38 where only a portion of 'poor' control is diagnosed. However, the health monitor did eventually flag the sluggish control. Faster controller performance diagnoses may be required for some processes. That the health monitor did raise a flag during each region of 'poor' control while sacrificing some statistical confidence is encouraging.
5) An advantage to allowing more than 10% of run-lengths to visit the extreme states in the Markov Chain model during 'good' control data analysis is to decrease the number of DACS samples required to fill the statistical window.

6.2 Future Work

It is proposed that five changes might further improve the health monitor:

1) States may be added one at a time to either side of the Markov chain, as opposed to two at a time, one to each side
2) The statistical window should not be determined as the sum of both halves of the Markov chain, but instead, the window length must be determined including interactions between the positive and negative states

3) One condition remains from the original algorithm which cannot be resolved. The second condition requiring no more than a specified percentage (10% in Owusu's work) of all sample visits be made to the extreme states has no theoretical underpinnings. This work chose to use 20% to minimize the window size. Future work must focus on either eliminating this condition or supporting it.
4) After limits are violated for an entire window health monitor window, the monitor retroactively flags to the sample when transition probabilities limits were first violated. The monitor stops flagging at the end of the window when transition probabilities return to within their limits. It is proposed that retroactively turning flagging off to the beginning of the window may better represent regions of poor control.
5) In practice, Statistical Process Control (SPC) sets a statistical window and performs a Type-I test with the user specified α. This method may be more practical if β and λ are dropped in favor of this standard practice.

REFERENCES

Bartels, N, “Outsourcing to the Rescue?”, Control. July 2007. http://www.controlglobal.com/articles/2007/261.html

Bezergianni, S and Georgakis, C, Controller performance assessment based on minimum and open-loop output variance, Control Engineering Practice **8** (2000), pp. 791–797.

Harris T. J, “Assessment of control loop performance”. Canadian Journal of Chemical Engineering, 67 (1989), pp. 856–861.

High, K., “Optimization Course Notes”, 2007

Hugo, J, A., “Process Controller Performance Monitoring and Assessment”, Control Arts Inc., 2000

Ingalls R., “Private Communication to Samuel Owusu”, 2002

Kadali, R. and B. Huang, “Controller Performance Analysis with LQG Benchmark Obtained under Closed Loop Conditions,” ISA Trans. 41, 521–537

Ko B., Edgar T. F., “Performance Assessment of Multivariable Feedback Control Systems”, Automatica, 37, pp 899-905, 2001

Li, Q., Whiteley, J. R., and Rhinehart, R. R., “An Automated Performance Monitor for Process Controllers”, Control Engineering Practice, 2004

Mosca, E, Agnoloni, T. “Closed-loop monitoring for early detection of performance losses in feedback-control systems”, Automatica, Vol 39, Issue 12, Dec 2003 pg 2071-2084

Ogunnaike, B A, Ray, W H, Process Dynamics, Modeling, and Control, Oxford University Press, New York, 1994

Owusu, Samuel, “A Control Loop Performance Monitor”, Oklahoma State University, Dissertation, 2006.

Rhinehart, R. R., “A Watch Dog for Controller Performance Monitoring”, Proceedings of the 1995 American Control Conference. Seattle, WA. 1995

Rhinehart, R. R., “Private Communation”, 2007

APPENDIX

Appendix A – Simulator Code

The code included in Appendix A and B is written in Visual Basic for Applications (VBA) used in conjunction with Excel.

```
Option Explicit
Const Pi = 3.14159265358979
Dim sDummy As String
Public Sub sub_PID()
' Author:           T. Judson Wooters
' Created Date:     18-DEC-2006
' Description:      Models a FOPDT process with FOPDT parrallel disturbance

' ____________________________________________________________
' ____________________TURN OFF SCREEN UPDATING_________________

Application.ScreenUpdating = False

' ____________________________________________________________
' ___________________________DECLARE VARIABLES_________________

Dim arr_Time() As Double        ' modeled time
Dim arr_DistU() As Double       ' modeled disturbance input
Dim arr_Dist() As Double        ' modeled disturbance output
Dim arr_ProcU() As Double       ' modeled process input
Dim a_dValveP() As Double
Dim a_dValvePM() As Double
Dim arr_Proc() As Double        ' modeled process output
Dim arr_Meas() As Double        ' measured time
Dim arr_DistUM() As Double      ' measured disturbance input
Dim arr_DistM() As Double       ' measured disturbance output
Dim arr_ProcUM() As Double      ' measured process input
Dim arr_ProcM() As Double       ' measured process output
Dim arr_TotalM() As Double      ' measured total output, includeds gaussian noise
Dim arr_ErrorM() As Double      ' measured error = setpoint - total_output
Dim arr_DerivM() As Double      ' measured derivative of error
Dim arr_IntegM() As Double      ' measured integral of error
Dim arr_SetPnt() As Double
Dim a_dStic() As Double
Dim a_dTime(1 To 6) As Double
Dim a_sTime(1 To 6) As String
Dim aSP() As Double
Dim aGain() As Double
Dim dbl_EndTime As Double       ' ending time for model
Dim dbl_Step As Double          ' model step size
Dim dbl_Measure As Double       ' measured step size (time between measurement)
Dim dbl_Setpoint As Double      ' setpoint
Dim dbl_InitProc As Double
Dim dbl_DistMean As Double      ' modeled average disturbance output
Dim dbl_DistStdev As Double     ' modeled disturbance standard deviation per
                                ' modeled time step (stdev * stepsize)
Dim dbl_ProcStdev As Double     ' measured standard deviation (noise)
Dim dbl_KProc As Double         ' process gain
```

```
Dim dbl_TProc As Double         ' process time
Dim dbl_ThetaProc As Double     ' process dead time
Dim dbl_KDist As Double         ' disturbance gain
Dim dbl_TDist As Double         ' disturbance time
Dim dbl_ThetaDist As Double     ' disturbance dead time
Dim dbl_KC As Double            ' controller gain
Dim dbl_TauI As Double          ' integral time
Dim dbl_TauD As Double          ' derivative time
Dim dbl_InitTime As Double      ' initial system clock time
Dim dbl_FinalTime As Double     ' end system clock time
Dim dDelStic As Double
Dim dDate_Day As Double
Dim dDiff As Double
Dim dStamp As Double
Dim lng_ArrSize As Long         ' modeled array size (# of modeled data points)
Dim lng_MeaSize As Long         ' measured array size (# of measured data points)
Dim iSPIndex As Integer
Dim iSPMaxIndex As Integer
Dim iGainIndex As Integer
Dim iSticIndex As Integer
Dim iSticMaxIndex As Integer
Dim iGainMaxIndex As Integer
Dim s_DistType As String        ' type of modeled disturbance
Dim var_SelectedItem As Variant
Dim s_FileData As String
Dim s_FileName As String
Dim sStamp As String
Dim sStampYMD As String
Dim fd_FileData As FileDialog
Dim K As Integer
Dim N As Long                   ' counter variable
Dim P As Long                   ' counter variable
Dim R As Long                   ' counter variable
Dim S As Long                   ' counter variable

' ____________________________________________________________
' ________________________INITIALIZE VARIABLES_________________________

dbl_InitTime = Timer
s_DistType = "Random"
dbl_Step = sht_FOPDT.Cells(1, 6)
dbl_EndTime = sht_FOPDT.Cells(2, 6)
dbl_Measure = sht_FOPDT.Cells(3, 6)
dbl_InitProc = sht_FOPDT.Cells(2, 9)
dbl_DistMean = sht_FOPDT.Cells(2, 10)
dbl_DistStdev = sht_FOPDT.Cells(3, 10)
dbl_ProcStdev = sht_FOPDT.Cells(3, 9)
dbl_KProc = sht_FOPDT.Cells(2, 2)
dbl_TProc = sht_FOPDT.Cells(3, 2)
dbl_ThetaProc = sht_FOPDT.Cells(4, 2)
dbl_KDist = sht_FOPDT.Cells(2, 3)
dbl_TDist = sht_FOPDT.Cells(3, 3)
dbl_ThetaDist = sht_FOPDT.Cells(4, 3)
dbl_KC = sht_FOPDT.Cells(2, 13)
dbl_TauI = sht_FOPDT.Cells(3, 13)
iSPIndex = 1
iGainIndex = 1
iSticIndex = 1

For K = 1 To 20000
'    If sSequence.Cells(2, 1) = "" Then
'        ReDim Preserve aSP(1 To 2, 1 To 1)
'        aSP(1, 1) = 0
'        aSP(2, 1) = dbl_Setpoint
'        Exit For
'    End If
    If sSequence.Cells(K + 1, 1) = "" Then Exit For
    ReDim Preserve aSP(1 To 2, 1 To K)
    aSP(1, K) = sSequence.Cells(1 + K, 1)
    aSP(2, K) = sSequence.Cells(1 + K, 2)
    iSPMaxIndex = K
```

```
Next K
For K = 1 To 20000
    If sSequence.Cells(2, 4) = "" Then
        ReDim Preserve aGain(1 To 2, 1 To 1)
        aSP(1, 1) = 0
        aSP(2, 1) = dbl_KC
        Exit For
    End If
    If sSequence.Cells(K + 1, 4) = "" Then Exit For
    ReDim Preserve aGain(1 To 2, 1 To K)
    aGain(1, K) = sSequence.Cells(1 + K, 4)
    aGain(2, K) = sSequence.Cells(1 + K, 5)
    iGainMaxIndex = K
Next K
For K = 1 To 20000
    If sSequence.Cells(2, 8) = "" Then
        ReDim Preserve a_dStic(1 To 2, 1 To 1)
        aSP(1, 1) = 0
        aSP(2, 1) = 0
        Exit For
    End If
    If sSequence.Cells(K + 1, 8) = "" Then Exit For
    ReDim Preserve a_dStic(1 To 2, 1 To K)
    a_dStic(1, K) = sSequence.Cells(1 + K, 8)
    a_dStic(2, K) = sSequence.Cells(1 + K, 9)
    iSticMaxIndex = K
Next K

' ____________________________________________________________
' ________________________CALCULATE CONSTANTS_________________________

lng_ArrSize = fun_Round(dbl_EndTime / dbl_Step, 1)

If dbl_Measure < dbl_Step Then
    dbl_Measure = dbl_Step
    sht_FOPDT.Cells(3, 6) = dbl_Measure
    MsgBox ("Measurement frequency set to stepsize")
End If

lng_MeaSize = fun_Round(dbl_EndTime / dbl_Measure, 1)

'____________________________________________________________
'________________________REQUEST FILENAME_____________________________
Set fd_FileData = Application.FileDialog(msoFileDialogSaveAs)

With fd_FileData
    .AllowMultiSelect = False
    If .Show = -1 Then
        For Each var_SelectedItem In .SelectedItems
            s_FileData = var_SelectedItem
        Next
    Else
        Exit Sub
    End If
End With

' ____________________________________________________________
' ____________________________RESIZE ARRAYS_____________________________

ReDim arr_Time(0 To lng_ArrSize)
ReDim arr_DistU(0 To lng_ArrSize)
ReDim arr_Dist(0 To lng_ArrSize)
ReDim arr_ProcU(0 To lng_ArrSize)
ReDim arr_Proc(0 To lng_ArrSize)
ReDim arr_Meas(0 To lng_MeaSize)
ReDim arr_DistUM(0 To lng_MeaSize)
ReDim arr_DistM(0 To lng_MeaSize)
ReDim arr_ProcUM(0 To lng_MeaSize)
ReDim a_dValveP(0 To lng_ArrSize)
ReDim arr_ProcM(0 To lng_MeaSize)
ReDim arr_TotalM(0 To lng_MeaSize)
```

```
ReDim arr_ErrorM(0 To lng_MeaSize)
ReDim arr_DerivM(0 To lng_MeaSize)
ReDim arr_IntegM(0 To lng_MeaSize)
ReDim arr_SetPnt(0 To lng_MeaSize)
ReDim a_dValvePM(0 To lng_MeaSize)

' ____________________________________________________________
' ____________________CREATE TIME ARRAYS_____________________

For N = 0 To lng_ArrSize
    arr_Time(N) = N * dbl_Step
Next

For N = 0 To lng_MeaSize
    arr_Meas(N) = N * dbl_Measure
Next

' ____________________________________________________________
' ____________________MODEL / MEASUREMENT ARRAYS_____________
' Steps:
'   1) Determine disturbance input/output for current step
'   2) Determine process input/output for current step
'   3) If it is time to measure, (a) then the total output is measured
'      assuming guassian noise, (b) the error from setpoint,
'      the error integral and error derivative are determined,
'      (c) finally the process input is found for the next modeled step
'      based on the controller output
'   4) If it is not time to measure, the process input for the next
'      modeled step is considered constant and the loop returns to
'      step 1

For N = 0 To lng_ArrSize

' --- STEP 1 ---
    Call subDist(arr_Time(), arr_DistU(), arr_Dist(), _
                        dbl_Step, dbl_ThetaDist, dbl_DistMean, _
                        dbl_DistStdev, dbl_KDist, dbl_TDist, _
                        s_DistType, N, R)
' --- STEP 2 ---
    Call subProc(arr_Time(), a_dValveP(), arr_ProcU(), arr_Proc(), _
                        dbl_Step, dbl_ThetaProc, dbl_DistMean, _
                        dbl_ProcStdev, dbl_KProc, dbl_TProc, _
                        dbl_InitProc, N, S)
' --- STEP 3 ---
    If arr_Time(N) >= arr_Meas(P) Then

        If iSPIndex <= iSPMaxIndex Then
            If arr_Time(N) >= aSP(1, iSPIndex) Then
                dbl_Setpoint = aSP(2, iSPIndex)
                iSPIndex = iSPIndex + 1
            End If
        End If
        If iGainIndex <= iGainMaxIndex Then
            If arr_Time(N) >= aGain(1, iGainIndex) Then
                dbl_KC = aGain(2, iGainIndex)
                iGainIndex = iGainIndex + 1
            End If
        End If
        If iSticIndex <= iSticMaxIndex Then
            If arr_Time(N) >= a_dStic(1, iSticIndex) Then
                dDelStic = a_dStic(2, iSticIndex)
                iSticIndex = iSticIndex + 1
            End If
        End If

        arr_SetPnt(P) = dbl_Setpoint
' ---- (a) -----
        arr_DistUM(P) = arr_DistU(N)
        arr_DistM(P) = arr_Dist(N)
        arr_ProcUM(P) = arr_ProcU(N)
        arr_ProcM(P) = arr_Proc(N)
```

```
        a_dValvePM(P) = a_dValveP(N)
        arr_TotalM(P) = (arr_ProcM(P) + arr_DistM(P)) + _
                        fun_Gauss(0, dbl_ProcStdev)
' ---- (b) -----
        arr_ErrorM(P) = dbl_Setpoint - arr_TotalM(P)
        Call subInteg(arr_IntegM(), dbl_Measure, arr_ErrorM(), P)
' ---- (c) -----
        If arr_Time(N) > 2 Then
            sDummy = "HERE"
        End If
        Call subProcU(dbl_TauI, arr_ProcU(), dbl_InitProc, _
                        dbl_DistMean, dbl_KProc, dbl_KC, _
                        arr_ErrorM(), arr_IntegM(), lng_MeaSize, N, P)
        If N < lng_ArrSize Then
            If Abs(arr_ProcU(N) - a_dValveP(N)) < dDelStic Then
                a_dValveP(N + 1) = a_dValveP(N)
            Else
                a_dValveP(N + 1) = arr_ProcU(N + 1)
            End If
        End If
        P = P + 1
' --- STEP 4 ---
    Else
        If N < lng_ArrSize Then
            arr_ProcU(N + 1) = arr_ProcU(N)
            a_dValveP(N + 1) = a_dValveP(N)
        End If
    End If
Next

' ____________________________________________________________
' ____________________FIX FINAL STEP__________________________
' Description:
'   The final time step does not seem to always map to the measurement
'   array so it is specifically done so in this if section

If arr_TotalM(lng_MeaSize) = 0 And dbl_Setpoint <> 0 And _
                                arr_ErrorM(lng_MeaSize) = 0 Then
    arr_DistUM(lng_MeaSize) = arr_DistU(lng_ArrSize)
    arr_DistM(lng_MeaSize) = arr_Dist(lng_ArrSize)
    arr_ProcUM(lng_MeaSize) = arr_ProcU(lng_ArrSize)
    arr_ProcM(lng_MeaSize) = arr_Proc(lng_ArrSize)
    arr_TotalM(lng_MeaSize) = fun_Gauss((arr_ProcM(lng_MeaSize) + _
                                arr_DistM(lng_MeaSize)), dbl_ProcStdev)
    arr_ErrorM(lng_MeaSize) = dbl_Setpoint - arr_TotalM(lng_MeaSize)
    Call subInteg(arr_IntegM(), dbl_Measure, arr_ErrorM(), lng_MeaSize)
    'Call subDeriv(arr_DerivM(), dbl_Measure, arr_ErrorM(), lng_MeaSize)
End If

' ____________________________________________________________
' ___________________ DETERMINE STARTING TIMESTAMP _____________

dDate_Day = Now()
'a_dTime(1) = Year(dDate_Day)
a_dTime(2) = Month(dDate_Day)
a_dTime(3) = Day(dDate_Day)
dDiff = fRoundDown(dDate_Day)
a_dTime(4) = (dDate_Day - dDiff) * 24
dDiff = fRoundDown(a_dTime(4))
a_dTime(5) = (a_dTime(4) - dDiff) * 60
dDiff = fRoundDown(a_dTime(5))
a_dTime(6) = (a_dTime(5) - dDiff) * 60
a_dTime(4) = fRoundDown(a_dTime(4))
a_dTime(5) = fRoundDown(a_dTime(5))
a_dTime(6) = fRoundDown(a_dTime(6))

'a_sTime(1) = a_dTime(1)
'sStampYMD = a_sTime(1)
sStampYMD = ""
For K = 2 To 3
    If a_dTime(K) < 10 Then
```

```
        a_sTime(K) = "0" & a_dTime(K)
    Else
        a_sTime(K) = a_dTime(K)
    End If
    sStampYMD = sStampYMD & a_sTime(K)
Next K

' ____________________________________________________________
' ______________________________OUTPUT________________________________

Open s_FileData For Output Access Write Lock Write As #1
For N = 0 To lng_MeaSize

    If sht_FOPDT.Cells(7, 3).Value = "Actuate Err" Then
        Print #1, arr_Meas(N) * 60, arr_ErrorM(N)
    Else
        Print #1, arr_Meas(N) * 60, arr_TotalM(N), arr_SetPnt(N)
    End If
    If dbl_EndTime <= 30 Then
        sht_FOPDT.Cells(N + 11, 1) = arr_Meas(N)
        sht_FOPDT.Cells(N + 11, 2) = arr_ErrorM(N)
        sht_FOPDT.Cells(N + 11, 3) = arr_ProcUM(N)
        sht_FOPDT.Cells(N + 11, 4) = a_dValvePM(N)
        sht_FOPDT.Cells(N + 11, 5) = arr_TotalM(N)
        sht_FOPDT.Cells(N + 11, 6) = arr_ProcM(N)
        sht_FOPDT.Cells(N + 11, 7) = arr_SetPnt(N)
        'sht_FOPDT.Cells(N + 11, 8) = arr_IntegM(N)
        sht_FOPDT.Cells(N + 11, 8) = arr_DistM(N)
    End If
Next
Close #1

sht_FOPDT.Cells(4, 6) = lng_MeaSize + 1            ' record # measured data points

dbl_FinalTime = Timer
sht_FOPDT.Cells(7, 1) = dbl_FinalTime - dbl_InitTime ' record total execution time (s)

' ____________________________________________________________
' ____________________TURN ON SCREEN UPDATING____________________

Application.ScreenUpdating = True

End Sub
Private Function fun_Round(dbl_Value As Double, dbl_Sig As Double) As Double
' Description:  Takes two inputs (any number and the desired output
' significant digits) and rounds up

fun_Round = Application.WorksheetFunction.Ceiling(dbl_Value, dbl_Sig)

End Function
Private Function fun_Gauss(dbl_Mean As Double, dbl_StDev As Double) As Double
' Description:  Takes two inputs (mean and standard deviation) and returns a
'               random normally distributed value

Dim dbl_U1 As Double        ' first random number
Dim dbl_U2 As Double        ' second random number
Dim dbl_R As Double         ' intermediate cosine side
Dim dbl_Theta As Double     ' intermediate natural log side
Dim dbl_Z As Double         ' random gaussian coefficient

dbl_U1 = Rnd()
dbl_U2 = Rnd()
If dbl_U1 = 0 Then dbl_U1 = Rnd()
If dbl_U2 = 0 Then dbl_U2 = Rnd()
dbl_R = Math.Cos(2# * Pi * dbl_U1)
dbl_Theta = (-2# * Math.Log(dbl_U2)) ^ 1 / 2
dbl_Z = dbl_R * dbl_Theta

fun_Gauss = dbl_Mean + dbl_Z * dbl_StDev

End Function
```

```
Private Function fun_Integ(dbl_Step As Double, dbl_X1 As Double, _
                           dbl_X2 As Double) As Double
' Description:  Takes three inputs (stepsize, value 1 and value 2)
' and returns the integral using the trapezoid rule

fun_Integ = (dbl_Step * (dbl_X2 + dbl_X1)) / 2

End Function
Private Function fun_Back1(dbl_Step As Double, dbl_X0 As Double, _
                     dbl_X1 As Double) As Double
' Description:  Takes three inputs (stepsize, value 1 and value 2)
' and returns the derivative using 1st order error backwards finite difference

fun_Back1 = (dbl_X1 - dbl_X0) / (2 * dbl_Step)

End Function
Private Function fun_Back2(dbl_Step As Double, dbl_X0 As Double, _
                           dbl_X1 As Double, dbl_X2 As Double) As Double
' Description:  Takes four inputs (stepsize, value 1, value 2 and value 3)
' and returns the derivative using 2nd order error backwards finite difference

fun_Back2 = (3 * dbl_X2 - 4 * dbl_X1 + dbl_X0) / (2 * dbl_Step)

End Function
Private Function fun_RK4(dbl_Input As Double, dbl_OutPrev As Double, _
                   dbl_Gain As Double, dbl_Step As Double, dbl_Tau As Double) _
                   As Double
' Description:  Takes five inputs (input, previous output, gain, stepsize,
' time constant) and returns the next step based on runga kuta 4th order

Dim dbl_EulSlope As Double          'euler slope from initial point
Dim dbl_HalfEul As Double           'half of the next step based on initial
                                    'step euler slope
Dim dbl_HalfEulSlp As Double        'euler slope from half step
Dim dbl_HalfRK As Double            'half of the next step from first point
                                    'based on half step euler slope
Dim dbl_HalfRKSlp As Double         'slope at the next step based on halfrunga step
Dim dbl_FullRK As Double            'full step based on the halfrunga slope
Dim dbl_FullRKSlp As Double         'slope at the full step
Dim dbl_RK4Slp As Double            '4th order runga kuta full step by weighing
                                    'each slope

dbl_EulSlope = (1 / dbl_Tau) * (dbl_Gain * dbl_Input - dbl_OutPrev)
dbl_HalfEul = dbl_OutPrev + ((dbl_EulSlope * dbl_Step) / 2)
dbl_HalfEulSlp = (1 / dbl_Tau) * (dbl_Gain * dbl_Input - dbl_HalfEul)
dbl_HalfRK = dbl_OutPrev + ((dbl_HalfEulSlp * dbl_Step) / 2)
dbl_HalfRKSlp = (1 / dbl_Tau) * (dbl_Gain * dbl_Input - dbl_HalfRK)
dbl_FullRK = dbl_OutPrev + (dbl_HalfRKSlp * dbl_Step)
dbl_FullRKSlp = (1 / dbl_Tau) * (dbl_Gain * dbl_Input - dbl_FullRK)

dbl_RK4Slp = (((1 / 6) * dbl_EulSlope) + ((1 / 3) * dbl_HalfEulSlp) + _
               ((1 / 3) * dbl_HalfRKSlp) + ((1 / 6) * dbl_FullRKSlp))
fun_RK4 = dbl_OutPrev + dbl_RK4Slp * dbl_Step

End Function
Private Sub subDist(arr_Time() As Double, arr_DistU() As Double, _
                      arr_Dist() As Double, dbl_Step As Double, _
                      dbl_ThetaDist As Double, dbl_DistMean As Double, _
                      dbl_DistStdev As Double, dbl_KDist As Double, _
                      dbl_TDist As Double, s_DistType As String, N As Long, _
                      R As Long)
' Description:  Takes twelve inputs and changes both the disturbance
' input and output

Select Case s_DistType
' ____________________________________________________________
' ____________________RANDOM DISTURBANCE INPUT_____________________
' Description:
'   Generates a gaussian random walk input that is then transfered to the output
Case "Random"
    If arr_Time(N) <= dbl_ThetaDist Or N = 0 Then
```

```
        arr_DistU(N) = (dbl_DistMean / dbl_KDist) + fun_Gauss(0, dbl_DistStdev) _
        * dbl_Step
        arr_Dist(N) = dbl_DistMean
        If arr_Dist(N) < 0 Then arr_Dist(N) = 0         ' output is confined
                                                        ' between 0 and 100
        If arr_Dist(N) > 100 Then arr_Dist(N) = 100
    Else
        arr_DistU(N) = arr_DistU(N - 1) + fun_Gauss(0, dbl_DistStdev) * dbl_Step
        arr_Dist(N) = fun_RK4(arr_DistU(R), arr_Dist(N - 1), dbl_KDist, _
                      dbl_Step, dbl_TDist)
        If arr_Dist(N) < 0 Then arr_Dist(N) = 0           'output is confined
                                                          'between 0 and 100
        If arr_Dist(N) > 100 Then arr_Dist(N) = 100
        R = R + 1
    End If
' ____________________________________________________________
' ____________________PULSE DISTURBANCE INPUT___________________
' Description:
'   Generates a pulse 3* as large as the steady state disturbance input after
'   1 second
Case "Pulse"
    If arr_Time(N) <= dbl_ThetaDist Or N = 0 Then
        arr_DistU(N) = (dbl_DistMean / dbl_KDist)
        If arr_DistU(N) < 0 Then arr_DistU(N) = 0           ' input / output is
                                                            ' confined between 0 and 100
        If arr_DistU(N) > 100 Then arr_DistU(N) = 100
        arr_Dist(N) = dbl_DistMean
    ElseIf arr_Time(N) > (dbl_ThetaDist + 1) And arr_Time(N) < _
    (dbl_ThetaDist + 1 + 0.01000001) Then
        arr_DistU(N) = 3 * (dbl_DistMean / dbl_KDist)
        If arr_DistU(N) < 0 Then arr_DistU(N) = 0           ' input / output is confined
                                                            ' between 0 and 100
        If arr_DistU(N) > 100 Then arr_DistU(N) = 100
        arr_Dist(N) = fun_RK4(arr_DistU(R), arr_Dist(N - 1), dbl_KDist, _
                      dbl_Step, dbl_TDist)
        R = R + 1
    Else
        arr_DistU(N) = (dbl_DistMean / dbl_KDist)
        If arr_DistU(N) < 0 Then arr_DistU(N) = 0           ' input / output is confined
                                                            ' between 0 and 100
        If arr_DistU(N) > 100 Then arr_DistU(N) = 100
        arr_Dist(N) = fun_RK4(arr_DistU(R), arr_Dist(N - 1), dbl_KDist, _
                      dbl_Step, dbl_TDist)
        R = R + 1
    End If
' ____________________________________________________________
' ____________________PULSE DISTURBANCE INPUT___________________
' Description:
'   Generates a step input of 1.5* the initial disturbance input
Case "Step"
    If arr_Time(N) <= dbl_ThetaDist Or N = 0 Then
        arr_DistU(N) = (dbl_DistMean / dbl_KDist)
        If arr_DistU(N) < 0 Then arr_DistU(N) = 0           ' input / output is confined
                                                            ' between 0 and 100
        If arr_DistU(N) > 100 Then arr_DistU(N) = 100
        arr_Dist(N) = dbl_DistMean
    ElseIf arr_Time(N) > dbl_ThetaDist And arr_Time(N) <= dbl_ThetaDist + 1 Then
        arr_DistU(N) = (dbl_DistMean / dbl_KDist)
        If arr_DistU(N) < 0 Then arr_DistU(N) = 0           ' input / output is confined
                                                            ' between 0 and 100
        If arr_DistU(N) > 100 Then arr_DistU(N) = 100
        arr_Dist(N) = fun_RK4(arr_DistU(R), arr_Dist(N - 1), dbl_KDist, _
                      dbl_Step, dbl_TDist)
        R = R + 1
    ElseIf arr_Time(N) > dbl_ThetaDist + 1 Then
        arr_DistU(N) = 1.5 * (dbl_DistMean / dbl_KDist)
        If arr_DistU(N) < 0 Then arr_DistU(N) = 0           ' input / output is confined
                                                            ' between 0 and 100
        If arr_DistU(N) > 100 Then arr_DistU(N) = 100
        arr_Dist(N) = fun_RK4(arr_DistU(R), arr_Dist(N - 1), dbl_KDist, _
                      dbl_Step, dbl_TDist)
```

```
        R = R + 1
    End If
Case Else
End Select

End Sub
Private Sub subProc(arr_Time() As Double, a_dValveP() As Double, arr_ProcU() As Double, _
                    arr_Proc() As Double, dbl_Step As Double, dbl_ThetaProc As Double, _
                    dbl_DistMean As Double, dbl_ProcStdev As Double, dbl_KProc As Double,
_
                    dbl_TProc As Double, dbl_InitProc As Double, N As Long, S As Long)
' Description:  Takes twelve inputs and changes both the process input and output

If arr_Time(N) <= dbl_ThetaProc Or N = 0 Then
    If N = 0 Then
        arr_ProcU(N) = dbl_InitProc / dbl_KProc
        a_dValveP(N) = arr_ProcU(N)
    End If
    If a_dValveP(N) < 0 Then a_dValveP(N) = 0           ' input / output is confined
                                                        ' between 0 and 100
    If a_dValveP(N) > 100 Then a_dValveP(N) = 100
    arr_Proc(N) = dbl_InitProc
Else
    If a_dValveP(N) < 0 Then a_dValveP(N) = 0           ' input / output is confined
                                                        ' between 0 and 100
    If a_dValveP(N) > 100 Then a_dValveP(N) = 100
    arr_Proc(N) = fun_RK4(a_dValveP(S), arr_Proc(N - 1), dbl_KProc, _
                          dbl_Step, dbl_TProc)
    S = S + 1
End If

End Sub
Private Sub subInteg(arr_Integ() As Double, dbl_Step As Double, _
                    arr_Error() As Double, N As Long)
' Description:  Takes four inputs (integral array, stepsize, error array
' and N counter) and returns the current integrated error

If N = 0 Then
    arr_Integ(N) = 0
Else
    arr_Integ(N) = arr_Integ(N - 1) + fun_Integ(dbl_Step, _
                                arr_Error(N - 1), arr_Error(N))
End If

End Sub
Private Sub subProcU(dbl_TauI As Double, arr_ProcU() As Double, dbl_InitProc As Double, _
                        dbl_DistMean As Double, dbl_KProc As Double, dbl_KC As Double, _
                        arr_Error() As Double, arr_Integ() As Double, _
                        lng_ArrSize As Long, N As Long, P As Long)
' Description:  Takes eleven inputs and returns the next controller output (process
input)

If P < lng_ArrSize Then
    If dbl_TauI = 0 Then
        arr_ProcU(N + 1) = (dbl_InitProc / dbl_KProc) + dbl_KC * arr_Error(P)
    Else
        arr_ProcU(N + 1) = (dbl_InitProc / dbl_KProc) + dbl_KC _
                                * (arr_Error(P) + (dbl_KC / dbl_TauI) * (arr_Integ(P)))
    End If
End If

End Sub
Function fRoundDown(dValue As Double) As Double
' Description: Takes a double and returns the value rounded down

Dim myDec As Long

myDec = InStr(1, CStr(dValue), ".", vbTextCompare)
If myDec > 0 Then
    fRoundDown = CDbl(Left(CStr(dValue), myDec))
Else
```

```
    fRoundDown = dValue
End If

End Function
```

Appendix B – Health Monitor Code

The code included in Appendix A and B is written in Visual Basic for Applications (VBA) used in conjunction with Excel.

---- Markov Chain and Window Length Code ----

```
Const Pi = 3.14159265358979     ' Fixes constant for PI
Const dEpsilon = 1E-16          ' how small is zero?
Dim sDummy As String            ' Dummy variable used to stop a loop
Sub History()
' Author:           T. Judson Wooters
' Created Date:     3-JAN-2008
' Description:      Main program

Dim a_dTotTime() As Double      ' time corresponding to each actuating error data point
Dim a_dTotErr() As Double       ' each actuating error data point
Dim a_dTot() As Double          ' used in Charact sub, holds transition probabilities
Dim a_dStats() As Double        ' holds transition probability limits in samples
Dim a_lngWindow() As Long       ' holds number of samples expected to visit states
Dim a_dTranLimits() As Double   ' holds transition probability limits as a fraction
Dim a_dSP() As Double           ' each setpoint data point
Dim a_dCV() As Double           ' each controlled variable data point
Dim dInitTime As Double         ' initial system clock time
Dim dFinalTime As Double        ' end system clock time
Dim dExtremeMax As Double       ' allowed maximum sample fraction in extreme states
Dim dExtreme As Double          ' sample fraction in extreme states
Dim dAlphaT As Double           ' allowed maximum alpha test
Dim dBetaT As Double            ' allowed maximum beta test
Dim dLamda As Double            ' allowed maximum lambda change
Dim dMeasFreq As Double         ' optional measurement frequency in seconds
Dim lngSettle As Long           ' process settling time
Dim dSampleFreq As Double       ' sampling frequency in seconds
Dim dStatWinTime As Double      ' statistical window in seconds
Dim dTotWinTime As Double       ' total window = statistical window + settling time
                                ' (seconds)
Dim dAlphaK As Double           ' alpha for each state
Dim lngDataPts As Long          ' number of samples based on sampling ratio
Dim lngArrSize As Long          ' total number of samples
Dim lngXL As Long               ' sample number lower limit
Dim lngXH As Long               ' sample number high limit
Dim lngMinPoints As Long        ' minimum samples required for a given state
Dim lngStatWin As Long          ' statistical window in samples
Dim lngSampleSettle As Long     ' samples in settling time
Dim lngTotWin As Long           ' total window = statistical window + settling time
                                ' (samples)
Dim lngN As Long                ' temporary storage for min number of samples
Dim lngModelSamples As Long     ' number of samples in final model
Dim iMaxStates As Integer       ' maximum number of states
Dim iSR As Integer              ' sampling ratio
Dim iInitState As Integer       ' initial number of states
Dim iMaxSR As Integer           ' maximum sampling ratio
Dim dOptTime As Double          ' time required by optimum states / sampling ratio
Dim lngOptSample As Long        ' samples requred by optimum states / sampling ratio
Dim iOptStates As Integer       ' optimum number of states
Dim iOptSR As Long              ' optimum sampling ratio
Dim iExitErr As Integer         ' exit error number (0 = ok, -1 = error)
Dim vSelectedItem As Variant    ' holds value of selection for file name
Dim sFileData As String         ' complete file name with path
Dim sFileTypeName As String     ' file type found in header
Dim sFileType As String         ' file type
Dim sErrFileData As String      ' temporary storage for file name with path
Dim sErrFileDir As String       ' error file path
Dim sReason As String           ' explains model output
Dim fdFileData As FileDialog    ' file dialog object for user input
Dim bModCand As Boolean         ' model candidate
Dim bNotZero As Boolean         ' if a state has 0 visits
```

```
Dim bDebug As Boolean           ' if model output should be viewed
Dim dCompleteP As Double        ' percent complete in finding model
Dim N As Long                   ' counting variable
Dim P As Long                   ' counting variable
Dim R As Integer                ' counting variable

' ==== INITIALIZE VARIABLES ====
dExtreme = 1
dAlphaT = shtModel.Cells(1, 3).Value
dBetaT = shtModel.Cells(2, 3).Value
dLamda = shtModel.Cells(3, 3).Value
dExtremeMax = shtModel.Cells(7, 3).Value
iInitState = shtModel.Cells(8, 3).Value
iSR = 1
lngSettle = shtModel.Cells(4, 3).Value
iMaxStates = shtModel.Cells(9, 3).Value
bDebug = shtModel.Cells(10, 3).Value

' ==== REQUEST ERROR FILE ====
sFileType = ".aer"
sFileTypeName = "Actuating Error"
If RequestFile(sErrFileData, sErrFileDir, sFileType, sFileTypeName) = -1 Then Exit Sub
sFileData = sErrFileData


dInitTime = Timer           ' Initialize the timer

' ==== READ ERROR FILE ====
Open sFileData For Input As #1
Do While Not EOF(1)
    lngArrSize = lngArrSize + 1
    ReDim Preserve a_dTotErr(1 To lngArrSize)
    ReDim Preserve a_dTotTime(1 To lngArrSize)
    ReDim Preserve a_dSP(1 To lngArrSize)
    ReDim Preserve a_dCV(1 To lngArrSize)
    Input #1, a_dTotTime(lngArrSize), a_dCV(lngArrSize), a_dSP(lngArrSize)
    a_dTotErr(lngArrSize) = a_dSP(lngArrSize) - a_dCV(lngArrSize)
Loop
Close #1

' ==== REQUEST NAME FOR MODEL OUTPUT ====
Set fdFileData = Application.FileDialog(msoFileDialogSaveAs)

With fdFileData
    .AllowMultiSelect = False
    If .Show = -1 Then
        For Each vSelectedItem In .SelectedItems
            sFileData = vSelectedItem
        Next
    Else
        Exit Sub
    End If
End With

' ==== DETERMINE MAXIMUM SAMPLING RATIO ====
iMaxSR = fOptMax(lngArrSize, False, iSR, iInitState, lngArrSize, dExtreme, _
            a_dTotErr(), dExtremeMax)

' ==== DETERMINE MAXIMUM STATES ====
iMaxStates = fOptMax(CLng(iMaxStates), True, iSR, iInitState, lngArrSize, dExtreme, _
            a_dTotErr(), dExtremeMax)

dCompleteP = 100 / ((iMaxSR * (iMaxStates - iInitState)) + 2)     ' Compute % complete

' ==== SETUP EXCEL WORKSHEETS FOR OUTPUT ====
shtModel.Activate
shtModel.Cells(12, 3).ClearContents
shtModel.Cells(13, 3).ClearContents
shtModel.Cells(15, 3).ClearContents
shtModel.Cells(16, 3).ClearContents
shtModel.Cells(18, 3).ClearContents
```

```
shtModel.Cells(19, 3).ClearContents
shtModel.Cells(21, 3).ClearContents
shtModel.Cells(22, 3).ClearContents
If bDebug Then
    shtData.Activate
    shtData.Range(Cells(1, 2), Cells(5, 200)).ClearContents
    shtData.Range(Cells(8, 2), Cells(1000, 200)).Clear
    shtData.Range(Cells(8, 2), Cells(9 + iMaxSR, (3 * (iMaxStates - iInitState) _
                                / 2# + 1) + 5)).Interior.ColorIndex = 2
    For N = 1 To 40
        shtData.Columns(N).ColumnWidth = 8.43
    Next N
    shtModel.Activate
End If

' ==== MAIN MODEL ACQUISITION ====
For P = iSR To iMaxSR                           ' Loops through sampling ratio
    For R = iInitState To iMaxStates Step 2     ' Loops through states
        sReason = ""                            ' Initialize reason
        lngDataPts = lngArrSize                 ' reset number of data points
        ReDim a_dTot(1 To 4, 1 To R)            ' reset tot array
        Call Charact(R, dExtreme, P, lngDataPts, a_dTotErr(), a_dTot())   ' determine
                                                    ' transition probabilities
        ReDim a_dStats(1 To 4, 1 To R)          ' reset stats array
        ReDim a_lngWindow(1 To 3, 1 To R)       ' reset window array

        ' ==== CHECK FOR STATE WITH ZERO VISITS ====
        For N = 1 To R
            If a_dTot(4, N) <> 1 And a_dTot(4, N) <> 0 Then
                bNotZero = True
            Else
                bNotZero = False
                Exit For            ' exit if there is a state with zero visits
            End If
        Next N

        ' ==== CHECK IF MAX EXTREME FRACTION VIOLATED ====
        If dExtreme <= dExtremeMax Then
            If bNotZero Then
                bModCand = True         ' Possible model candidate
            Else
                bModCand = False
            End If
        Else
            bModCand = False
        End If

        ' ==== FIND LIMITS USING MIDPOINT OPTIMIZATION ====
        For N = 1 To R
            If bModCand Then
                RunLimitsAlg dAlphaT, dBetaT, dLamda, R, a_dTot(4, N), lngXL, _
                             lngXH, lngMinPoints, False
                a_dStats(2, N) = lngXL
                a_dStats(3, N) = lngXH
                a_dStats(4, N) = lngMinPoints
                a_dStats(1, N) = a_dTot(1, N)
            Else
                a_dStats(2, N) = 0
                a_dStats(3, N) = 0
                a_dStats(4, N) = 0
                a_dStats(1, N) = a_dTot(1, N)
            End If
        Next
        If P = 5 Then
            sDummy = "HERE"
        End If

        ' ==== FIND WINDOW SIZE / DETERMINE SHORTEST WINDOW IN TIME ====
        If bModCand Then
            FindWindow a_dStats(), a_dTot(), a_lngWindow(), lngStatWin
            lngSampleSettle = fRoundUp((lngSettle / P), 0)
```

```
            lngTotWin = lngStatWin + lngSampleSettle
            dStatWinTime = lngStatWin * P
            dTotWinTime = lngTotWin * P
            If iOptStates = 0 Then      ' initialize optimum parameters
                iOptStates = R
                iOptSR = P
                dOptTime = dStatWinTime
                lngOptSample = lngStatWin
            ElseIf dStatWinTime < dOptTime Then
                iOptStates = R
                iOptSR = P
                dOptTime = dStatWinTime
                lngOptSample = lngStatWin
            End If
        Else
            lngStatWin = 0
        End If

        ' ==== EXCEL OUTPUT ====
        shtModel.Cells(12, 3).Value = Round(((P - 1) * (iMaxStates - iInitState) + _
                                        (R - iInitState) + 2) * dCompleteP, 2)
        If bDebug Then
            If OutputModelExcel(iMaxStates, iInitState, P, lngDataPts, R, dExtreme, _
                            dExtremeMax, dTotWinTime, bNotZero, sReason, bModCand, iSR) _
                            = -1 Then Exit For
        End If
        DoEvents
        If Not bDebug And dExtreme <= dExtremeMax Then
            Exit For
        End If
        If Not bNotZero Then Exit For       ' Choose new sampling ratio if zero
                                            'visits occurs
    Next R
    If R = iInitState And bNotZero Then
        Exit For
    End If
Next P
shtModel.Cells(12, 3).Value = Round((iMaxSR * (iMaxStates - iInitState) + 2) _
                                * dCompleteP, 2)

' ==== RERUN FOR OPTIMUM PARAMETERS ====
lngDataPts = lngArrSize                         ' reset total number of data points
ReDim a_dTot(1 To 4, 1 To iOptStates)
Call Charact(iOptStates, dExtreme, iOptSR, lngDataPts, a_dTotErr(), a_dTot())
ReDim a_dStats(1 To 4, 1 To iOptStates)
ReDim a_lngWindow(1 To 3, 1 To iOptStates)
ReDim a_lngWindow(1 To 3, 1 To iOptStates)
ReDim a_dTranLimits(1 To 2, 1 To iOptStates)
For N = 1 To iOptStates
    RunLimitsAlg dAlphaT, dBetaT, dLamda, iOptStates, a_dTot(4, N), lngXL, _
                lngXH, lngMinPoints, False
    a_dStats(2, N) = lngXL
    a_dStats(3, N) = lngXH
    a_dStats(4, N) = lngMinPoints
    a_dStats(1, N) = a_dTot(1, N)
Next N
FindWindow a_dStats(), a_dTot(), a_lngWindow(), lngStatWin
lngSampleSettle = fRoundUp((lngSettle / iOptSR), 0)
lngTotWin = lngStatWin + lngSampleSettle
dStatWinTime = lngStatWin * iOptSR
dTotWinTime = lngTotWin * iOptSR
dAlphaK = 1# - (1# - dAlphaT) ^ (1# / iOptStates)
For N = 1 To iOptStates
    lngN = a_lngWindow(1, N)
    Limits dAlphaK, a_dTot(4, N), 1, 0, lngXL, lngXH, False, lngN, 0, True
    a_lngWindow(2, N) = lngXH
    a_lngWindow(3, N) = lngXL
    a_dTranLimits(1, N) = (lngXH / a_lngWindow(1, N))
    a_dTranLimits(2, N) = (lngXL / a_lngWindow(1, N))
Next
```

```
For N = 1 To iOptStates
    lngModelSamples = lngModelSamples + a_dTot(2, N)
Next N

' ==== WRITE MODEL PARAMETERS TO EXCEL ====
shtModel.Cells(16, 3).Value = iOptStates
shtModel.Cells(15, 3).Value = iOptSR
shtModel.Cells(18, 3).Value = lngStatWin
shtModel.Cells(19, 3).Value = lngTotWin
shtModel.Cells(21, 3).Value = Round(dStatWinTime, 2)
shtModel.Cells(22, 3).Value = Round(dTotWinTime, 2)

For N = 1 To iOptStates
    shtData.Cells(1, N + 2) = a_dTot(1, N)
    shtData.Cells(2, N + 2) = a_dTot(2, N) / lngModelSamples
    shtData.Cells(3, N + 2) = a_dTranLimits(1, N)
    shtData.Cells(4, N + 2) = a_dTot(4, N)
    shtData.Cells(5, N + 2) = a_dTranLimits(2, N)
Next N

' ==== WRITE MODEL FILE ====
Open sFileData For Output Access Write Lock Write As #1
Print #1, iOptStates
Print #1, iOptSR
Print #1, lngOptSample
Print #1, lngTotWin
For N = 1 To iOptStates
    Print #1, a_dTot(1, N), a_dTranLimits(1, N), a_dTranLimits(2, N)
Next N
Close #1

' ==== CLEANUP EXCEL OUTPUT ====
If bDebug Then
    CleanUpExcel iMaxStates, iInitState, iMaxSR
End If

' ==== FINISH WITH SOME STATISTICS ====
dFinalTime = Timer
shtModel.Cells(13, 3).Value = (dFinalTime - dInitTime)

' ==== SET-UP CHART ====
ModifyChart iOptStates, "ModelChart"

End Sub
Sub Charact(iNumState As Integer, dExtreme As Double, iSR As Long, _
               lngArrSize As Long, a_dTotErr() As Double, a_dTot() As Double)
' Author:           T. Judson Wooters
' Created Date:     18-DEC-2006
' Description:      Accepts an array of errors, assigns states and returns statistics

Dim a_iStates() As Integer         ' Array holding the state value of each error
Dim a_dErr() As Double             ' Array holding all error
Dim N As Long                      ' Internal counter
Dim P As Integer                   ' Internal counter (used to count through states)

If ((lngArrSize / iSR) Mod 1) > 0 Then
    lngArrSize = Int(lngArrSize \ iSR) + 1
Else
    lngArrSize = Int(lngArrSize \ iSR)
End If

' ==== RESIZE ARRAYS ====
ReDim a_iStates(1 To lngArrSize)
ReDim a_dErr(1 To lngArrSize)
ReDim a_dTot(1 To 4, 1 To iNumState)

' ==== FILL 'TOT' ARRAY WITH STATES (1) ====
For N = 1 To iNumState
    If (N - (iNumState / 2#)) > 0 Then
        a_dTot(1, N) = N - (iNumState / 2#)
    Else
```

```
        a_dTot(1, N) = N - (iNumState / 2#) - 1
    End If
Next

' ==== FILL 'ERR', 'STATES' AND 'TOT' ARRAYS ====
For N = 1 To lngArrSize
    a_dErr(N) = a_dTotErr(((iSR * (N - 1)) + 1))
    If N = 1 Then                                          ' determine current state
        a_iStates(N) = fState(a_dErr(N), 0, (iNumState / 2#))
    Else
        a_iStates(N) = fState(a_dErr(N), a_iStates(N - 1), (iNumState / 2#))
    End If
    If N > 1 Then
        For P = 1 To iNumState
            If a_iStates(N) = a_dTot(1, P) Then
                a_dTot(2, P) = a_dTot(2, P) + 1
                Exit For
            End If
        Next
        If a_iStates(N - 1) > 0 And a_iStates(N) = -1 Then
            For P = 1 To iNumState
                If a_iStates(N - 1) = a_dTot(1, P) Then
                    a_dTot(3, P) = a_dTot(3, P) + 1
                    Exit For
                End If
            Next
        ElseIf a_iStates(N - 1) < 0 And a_iStates(N) = 1 Then
            For P = 1 To iNumState
                If a_iStates(N - 1) = a_dTot(1, P) Then
                    a_dTot(3, P) = a_dTot(3, P) + 1
                    Exit For
                End If
            Next
        End If
    End If
Next

' ==== TRANSITION PROB = PROBABILIY TO CROSS ZERO ====
For N = 1 To iNumState
    If a_dTot(2, N) = 0 Then
        a_dTot(4, N) = 0
    Else
        a_dTot(4, N) = a_dTot(3, N) / a_dTot(2, N)
    End If
Next

' ==== DETERMINE STATISTICS OF ENTIRE STATE CHARACTERIZATION ====
dExtreme = (a_dTot(2, 1) + a_dTot(2, iNumState)) / (lngArrSize)

End Sub

Function fState(dError As Double, iStateP As Integer, iNumState) As Integer
' Author:           T. Judson Wooters
' Created Date:     28-DEC-2006
' Description:      Accepts the current error, the previous state and the max
'                   number of states

' ==== DECLARE VARIABLES ====
Dim iMaxState As Integer                ' Maximum state
Dim iMinState As Integer                ' Minimum state

iMaxState = Math.Abs(iNumState)
iMinState = -1 * Math.Abs(iNumState)

If iStateP < iMaxState And iStateP > iMinState Then     ' If the previous state
                                                        ' was not an extreme
    If iStateP = 0 Then             ' If the previous state = 0
        If dError = 0 Then          ' If current error = 0
            fState = 0              ' assign current state = 0
        ElseIf dError < 0 Then      ' If current error < 0
            fState = -1             ' assign current state = -1
```

```
        Else                            ' If current error > 0
            fState = 1                  ' assign current state > 1
        End If
    ElseIf iStateP < 0 Then             ' If the previous state < 0
        If dError <= 0 Then             ' If the current error <= 0
            fState = iStateP - 1        ' assign current state to one less than previous
        Else                            ' If the current error > 0
            fState = 1                  ' assign current state = 1
        End If
    Else                                ' If the previous state > 0
        If dError >= 0 Then             ' If the current error >= 0
            fState = iStateP + 1        ' assign current state to one more than previous
        Else                            ' If the current error < 0
            fState = -1                 ' assign current state = -1
        End If
    End If
Else                                    ' If at an extreme state
    If iStateP = 0 Then                 ' If the previous state = 0
        If dError = 0 Then              ' If current error = 0
            fState = 0                  ' assign current state = 0
        ElseIf dError < 0 Then          ' If current error < 0
            fState = -1                 ' assign current state = -1
        Else                            ' If current error > 0
            fState = 1                  ' assign current state > 1
        End If
    ElseIf iStateP < 0 Then             ' If the previous state < 0
        If dError <= 0 Then             ' If the current error <= 0
            fState = iStateP            ' assign current state to be the same as previous
        Else                            ' If the current error > 0
            fState = 1                  ' assign current state = 1
        End If
    Else                                ' If the previous state > 0
        If dError >= 0 Then             ' If the current error >= 0
            fState = iStateP            ' assign current state to be the same as previous
        Else                            ' If the current error < 0
            fState = -1                 ' assign current state = -1
        End If
    End If
End If

End Function
Sub RunLimitsAlg(dAlphaT As Double, dBetaT As Double, dLamda As Double, _
                 iStates As Integer, dProbRef As Double, lngXL As Long, _
                 lngXH As Long, lngMinPoints As Long, bPrint As Boolean)
' Author:           T. Judson Wooters
' Created Date:     3-JAN-2007
' Description:      Inverval halving method to find the number of samples required
'                   for beta test.
'                   Step 1) Bounding using the exact binomial distribution
'                   Step 2) After bounding we use interval halving with the exact
'                           binomial distribution

Dim a_lngSample(1 To 3) As Long     ' number of samples (1 - Low, 2 - Mid, 3 - High)
Dim a_dBeta(1 To 3) As Double       ' beta for each number of samples
Dim a_iSign(1 To 3) As Integer      ' sign, above or below beta limit for each
                                    ' number of samples
Dim a_lngXL(1 To 3) As Long         ' lower limit for each set of samples
Dim a_lngXH(1 To 3) As Long         ' upper limit for eacch set of samples
Dim iDirection As Integer           ' +/- 1 used during step 3 to bound correctly
Dim dMaxBeta As Double              ' maximum beta obtained through limits sub
Dim lngSamLow As Long               ' number of samples obtained from step 2
Dim lngFLow As Long                 ' beta found with optimum number of samples
                                    ' (step 2 and 4)
Dim lngIncrement As Long            ' step size used in bounding
Dim dAlphaK As Double               ' alpha for each state
Dim dBetaK As Double                ' beta for each state
Dim dProbH As Double
Dim dProbL As Double
Dim N As Long                       ' counting variable
Dim P As Long                       ' counting variable
Dim M As Long                       ' counting variable
```

```
Dim K As Long                       ' counting variable

dAlphaK = 1# - (1# - dAlphaT) ^ (1# / iStates)
dBetaK = 1# - (1# - dBetaT) ^ (1# / iStates)
dProbH = dProbRef + dLamda * (1# - dProbRef)
dProbL = dProbRef - dLamda * dProbRef

N = 5
lngIncrement = 1

' ==== STEP 1 ====
For P = 1 To 10000
    Limits dAlphaK, dProbRef, dProbH, dProbL, lngXL, lngXH, bPrint, N, dMaxBeta, False
    If dMaxBeta < dBetaK / 2# Then      ' Check if crossed allowable beta on high side
    'If dMaxBeta < dBetaK Then      ' Check if crossed allowable beta on high side
        a_lngSample(3) = N
        a_dBeta(3) = Abs((dMaxBeta - (dBetaK / 2#)) / (dBetaK / 2#))
        'a_dBeta(3) = Abs((dMaxBeta - (dBetaK)) / (dBetaK))
        If ((dMaxBeta - (dBetaK / 2#)) / (dBetaK / 2#)) > 0 Then
        'If ((dMaxBeta - (dBetaK)) / (dBetaK)) > 0 Then
            a_iSign(3) = 1
        Else
            a_iSign(3) = -1
        End If
        a_lngXL(3) = lngXL
        a_lngXH(3) = lngXH
        Exit For                        ' Once the high side is found, exit for
    Else                                ' Check if low side of allowable beta
        lngIncrement = lngIncrement * 2#    ' Increase stepsize
        a_lngSample(1) = N
        a_dBeta(1) = Abs((dMaxBeta - (dBetaK / 2#)) / (dBetaK / 2#))
        'a_dBeta(1) = Abs((dMaxBeta - (dBetaK)) / (dBetaK))
        If ((dMaxBeta - (dBetaK / 2#)) / (dBetaK / 2#)) > 0 Then
        'If ((dMaxBeta - (dBetaK)) / (dBetaK)) > 0 Then
            a_iSign(1) = 1
        Else
            a_iSign(1) = -1
        End If
        a_lngXL(1) = lngXL
        a_lngXH(1) = lngXH
    End If
    N = N + lngIncrement      ' Increase number of samples if no high side found yet
Next P

' ==== Get Middle Point ====
a_lngSample(2) = Round((a_lngSample(1) + a_lngSample(3)) / 2#, 0)       ' Middle point
Limits dAlphaK, dProbRef, dProbH, dProbL, lngXL, lngXH, bPrint, a_lngSample(2), _
               dMaxBeta, False
a_dBeta(2) = Abs((dMaxBeta - (dBetaK / 2#)) / (dBetaK / 2#))
'a_dBeta(2) = Abs((dMaxBeta - (dBetaK)) / (dBetaK))
If ((dMaxBeta - (dBetaK / 2#)) / (dBetaK / 2#)) > 0 Then
'If ((dMaxBeta - (dBetaK)) / (dBetaK)) > 0 Then
    a_iSign(2) = 1
Else
    a_iSign(2) = -1
End If
a_lngXL(2) = lngXL
a_lngXH(2) = lngXH

lngFLow = 0

' ==== STEP 2 ====
For P = 1 To 10000
    If (a_lngSample(3) - a_lngSample(1)) <= 2 Then      ' Did we find the optimum?
        For M = 1 To 3
            If a_iSign(M) < 0 And lngFLow = 0 Then
                lngFLow = a_dBeta(M)
                lngMinPoints = a_lngSample(M)
                lngXL = a_lngXL(M)
                lngXH = a_lngXH(M)
                Exit For                                ' Exit when optimum found
```

```
            End If
        Next M
        Exit For
    End If
    ' ==== If the sign of the bottom two are the same, then you have to move up ====
    If a_iSign(1) = a_iSign(2) Then
        a_lngSample(1) = a_lngSample(2)
        a_dBeta(1) = a_dBeta(2)
        a_iSign(1) = a_iSign(2)
        a_lngXL(1) = a_lngXL(2)
        a_lngXH(1) = a_lngXH(2)
    ' ==== If the sign of the top two are the same, then you have to move down ====
    Else
        a_lngSample(3) = a_lngSample(2)
        a_dBeta(3) = a_dBeta(2)
        a_iSign(3) = a_iSign(2)
        a_lngXL(3) = a_lngXL(2)
        a_lngXH(3) = a_lngXH(2)
    End If
    ' ==== Get Middle Point ====
    a_lngSample(2) = Round((a_lngSample(1) + a_lngSample(3)) / 2#, 0)
    Limits dAlphaK, dProbRef, dProbH, dProbL, lngXL, lngXH, bPrint, a_lngSample(2), _
                    dMaxBeta, False
    a_dBeta(2) = Abs((dMaxBeta - (dBetaK / 2#)) / (dBetaK / 2#))
    'a_dBeta(2) = Abs((dMaxBeta - (dBetaK)) / (dBetaK))
    If ((dMaxBeta - (dBetaK / 2#)) / (dBetaK / 2#)) > 0 Then
    'If ((dMaxBeta - (dBetaK)) / (dBetaK)) > 0 Then
        a_iSign(2) = 1
    Else
        a_iSign(2) = -1
    End If
    a_lngXL(2) = lngXL
    a_lngXH(2) = lngXH
Next P

End Sub
Sub Limits(dAlphaK As Double, dProbRef As Double, dProbH As Double, dProbL As Double, _
                lngXL As Long, lngXH As Long, bPrint As Boolean, N As Long, _
                dMaxBeta As Double, bFixLimits As Boolean)
' Author:           T. Judson Wooters
' Created Date:     10-JAN-2007
' Description:      This program finds the alpha and beta given a mean and number of
samples

Dim arrRefCumProb() As Double       ' cummulative probability for the reference tran prob
Dim arrRefProb() As Double          ' discrete probability for the reference tran prob
Dim arrHighCumProb() As Double      ' cummulative probability for the high side tran prob
Dim arrHighProb() As Double         ' discrete probability for the high side tran prob
Dim arrLowCumProb() As Double       ' cummulative probability for the low side tran prob
Dim arrLowProb() As Double          ' discrete probability for the low side tran prob
Dim arrRevRefCumProb() As Double    ' reverse cummulative probability for the reference
                                    ' tran prob
Dim arrRevHighCumProb() As Double   ' reverse cummulative probability for the high side
                                    ' tran Prob
Dim arrRevLowCumProb() As Double    ' reverse cummulative probability for the low side
                                    ' tran prob
Dim dCumProbH As Double             ' beta on the high side
Dim dCumProbL As Double             ' beta on the low side
Dim dX1Prob As Double               ' 1st probability used to find slope
Dim dX2Prob As Double               ' 2nd probability used to find slope
Dim dSlope As Double                ' slope between # of samples used to find # of
samples
Dim dYIntercept As Double           ' y intersept of samples used to find # of samples
Dim dXLowActual As Double           ' fraction value for # of samples (low side)
Dim dXHighActual As Double          ' fraction value for # of samples (low side)
Dim lngX1 As Long                   ' 1st number samples used to find slope
Dim lngX2 As Long                   ' 2nd number samples used to find slope
Dim lngMean As Long
Dim lngMeanLow As Long
Dim lngMeanHigh As Long
Dim X As Long
```

```
' ==== INITIALIZE VARIABLES ====
ReDim arrRefCumProb(0 To N)
ReDim arrRevRefCumProb(0 To N)
ReDim arrRefProb(0 To N)
ReDim arrHighCumProb(0 To N)
ReDim arrRevHighCumProb(0 To N)
ReDim arrHighProb(0 To N)
ReDim arrLowCumProb(0 To N)
ReDim arrRevLowCumProb(0 To N)
ReDim arrLowProb(0 To N)
dCumProbL = 0
dCumProbH = 0
lngMean = Math.Round(N * dProbRef, 0)
lngMeanLow = Math.Round(N * dProbL, 0)
lngMeanHigh = Math.Round(N * dProbH, 0)
lngXL = 0
lngXH = N

' ==== CREATE THE 3 BINOMIAL DISTRIBUTIONS ====
UpperHalf N, arrRefProb(), dProbRef, lngMean

LowerHalf N, arrRefProb(), dProbRef, lngMean

If Not bFixLimits Then
    UpperHalf N, arrHighProb(), dProbH, lngMeanHigh

    UpperHalf N, arrLowProb(), dProbL, lngMeanLow

    LowerHalf N, arrHighProb(), dProbH, lngMeanHigh

    LowerHalf N, arrLowProb(), dProbL, lngMeanLow
End If

' ==== FORWARD CUMMULATIVE PROBABILITIES ====
For X = 0 To N
    If X = 0 Then
        arrRefCumProb(X) = arrRefProb(X)
        If Not bFixLimits Then
            arrHighCumProb(X) = arrHighProb(X)
            arrLowCumProb(X) = arrLowProb(X)
        End If
    Else
        arrRefCumProb(X) = arrRefProb(X) + arrRefCumProb(X - 1)
        If Not bFixLimits Then
            arrHighCumProb(X) = arrHighProb(X) + arrHighCumProb(X - 1)
            arrLowCumProb(X) = arrLowProb(X) + arrLowCumProb(X - 1)
        End If
    End If
Next

' ==== REVERSE CUMMULATIVE PROBABILITIES ====
For X = N To 0 Step -1
    If X = N Then
        arrRevRefCumProb(X) = arrRefProb(X)
        If Not bFixLimits Then
            arrRevHighCumProb(X) = arrHighProb(X)
            arrRevLowCumProb(X) = arrLowProb(X)
        End If
    Else
        arrRevRefCumProb(X) = arrRefProb(X) + arrRevRefCumProb(X + 1)
        If Not bFixLimits Then
            arrRevHighCumProb(X) = arrHighProb(X) + arrRevHighCumProb(X + 1)
            arrRevLowCumProb(X) = arrLowProb(X) + arrRevLowCumProb(X + 1)
        End If
    End If
Next

' ==== FIND LOW SAMPLE ====
For X = 0 To N
    If arrRefCumProb(X) < dAlphaK / 2# Then
```

```
        lngX1 = X
        dX1Prob = arrRefCumProb(X)
    Else
        If X = 0 Then
            lngXL = 0
            Exit For
        End If
        lngX2 = X
        dX2Prob = arrRefCumProb(X)
        dSlope = (dX2Prob - dX1Prob) / (lngX2 - lngX1)
        dYIntercept = dX1Prob - (dSlope * lngX1)
        dXLowActual = ((dAlphaK / 2#) - dYIntercept) / dSlope   ' This is the
                                                                ' interpolated low limit
        lngXL = Round(dXLowActual, 0)
        Exit For
    End If
Next X

' ==== FIND HIGH SAMPLE ====
For X = N To 0 Step -1
    If arrRevRefCumProb(X) < dAlphaK / 2# Then
        lngX1 = X
        dX1Prob = arrRevRefCumProb(X)
    Else
        If X = N Then
            lngXH = N
            Exit For
        End If
        lngX2 = X
        dX2Prob = arrRevRefCumProb(X)
        dSlope = (dX2Prob - dX1Prob) / (lngX2 - lngX1)
        dYIntercept = dX1Prob - (dSlope * lngX1)
        dXHighActual = ((dAlphaK / 2#) - dYIntercept) / dSlope  ' This is the
                                                                ' interpolated high limit
        lngXH = Round(dXHighActual, 0)
        Exit For
    End If
Next X

' ==== FIND LOW PROBABILITY ====
If Not bFixLimits Then
    If lngXL = 0 Then
        dCumProbL = arrRevLowCumProb(lngXL)
    Else
        If dXLowActual < lngXL Then
            lngX1 = lngXL - 1
            lngX2 = lngXL
        Else
            lngX1 = lngXL
            lngX2 = lngXL + 1
        End If
        dX1Prob = arrRevLowCumProb(lngX1)
        dX2Prob = arrRevLowCumProb(lngX2)
        dSlope = (dX2Prob - dX1Prob) / (lngX2 - lngX1)
        dYIntercept = dX1Prob - (dSlope * lngX1)
        dCumProbL = dSlope * dXLowActual + dYIntercept  ' This is the interpolated beta
    End If

    ' ==== FIND HIGH PROBABILITY ====
    If lngXH = N Then
        dCumProbH = arrHighCumProb(lngXH)
    Else
        If dXHighActual < lngXH Then
            lngX1 = lngXH - 1
            lngX2 = lngXH
        Else
            lngX1 = lngXH
            lngX2 = lngXH + 1
        End If
        dX1Prob = arrHighCumProb(lngX1)
        dX2Prob = arrHighCumProb(lngX2)
```

```
        dSlope = (dX2Prob - dX1Prob) / (lngX2 - lngX1)
        dYIntercept = dX1Prob - (dSlope * lngX1)
        dCumProbH = dSlope * dXHighActual + dYIntercept  ' This is the interpolated beta
    End If

    dMaxBeta = fFindMax(dCumProbL, dCumProbH)         ' All I care about is the highest
                                                      ' beta (low/high)
End If

' Print information to spreadsheet only if debugging
If bPrint Then
    shtTestLimits.Range(Cells(13, 1), Cells(11000, 10)).Clear

    shtTestLimits.Cells(6, 2) = N
    shtTestLimits.Cells(8, 3) = dCumProbL
    shtTestLimits.Cells(10, 3) = dCumProbH
    shtTestLimits.Cells(8, 2) = lngXL
    shtTestLimits.Cells(10, 2) = lngXH
    shtTestLimits.Cells(8, 4) = dProbL
    shtTestLimits.Cells(9, 4) = dProbRef
    shtTestLimits.Cells(10, 4) = dProbH


    For X = 0 To N
        shtTestLimits.Cells(X + 13, 1) = X
        shtTestLimits.Cells(X + 13, 2) = arrLowProb(X)
        shtTestLimits.Cells(X + 13, 3) = arrRefProb(X)
        shtTestLimits.Cells(X + 13, 4) = arrHighProb(X)
        'shtTestLimits.Cells(X + 13, 5) = arrLowCumProb(X)
        'shtTestLimits.Cells(X + 13, 6) = arrRevLowCumProb(X)
        'shtTestLimits.Cells(X + 13, 7) = arrRefCumProb(X)
        'shtTestLimits.Cells(X + 13, 8) = arrRevRefCumProb(X)
        'shtTestLimits.Cells(X + 13, 9) = arrHighCumProb(X)
        'shtTestLimits.Cells(X + 13, 10) = arrRevHighCumProb(X)
    Next

End If


End Sub
Function fBinomial(lngN As Long, lngX As Long, dProb As Double) As Double
' T. Judson Wooters
' The Binomial function is also refered to as the direct method in this program.
' || Due to overflow concerns, each individual element in the binomial equation is
' || assigned to an array of lenght 'N'. If there are not enough unique elements to
' || fill an array then '1' fills the rest. For example, if X = 2 but lngN = 10
' || then the N! array will contain 10, 9, 8, ... and the X! array will
' || contain 2, 1, 1, 1. Further efforts are made to reduce overflow risk in this order
' || 1) Each of the 5 element arrays are multiplied to together and assigned to a
'       new array
' || 2) The cases where overflow are of the most concern have arrays that are already
'       mostly sorted. Very low numbers on one end increasing to some max value not found
'       in the middle then decreasing to small values. The peak will never be in the
'       middle and an algorithm makes sure that the number stays as close to "0.5" as
'       possible.

Dim arrX() As Double               ' Array of lngX!
Dim arrN() As Double               ' Array of lngN!
Dim arrNminusX() As Double         ' Array of (lngN-lngX)!
Dim arrPX() As Double              ' Array of P^lngX
Dim arrPNminusX() As Double        ' Array of P^(lngN-lngX)
Dim arrEachIter() As Double        ' Array created by step 1 above
Dim lngNminusX As Long             ' Calculated lngN-lngX value
Dim lngSmallIndex As Long
Dim lngLargeIndex As Long
Dim lngArrUpper() As Long
Dim lngArrLower() As Long
Dim lngMaxRun As Long
Dim dArrCombined() As Double
Dim PWA As Double
Dim PWB As Double
```

```
Dim lngthR As Long
Dim S As Long                          ' Counter variable
Dim J As Long                          ' Counter variable

' ------ INITIAL CALCULATIONS ------
fBinomial = 1
lngNminusX = lngN - lngX
lngMaxRun = fFindMaxLong(lngX, lngNminusX)

' ------ RESIZE ARRAYS ------
ReDim arrX(1 To lngN)
ReDim arrN(1 To lngN)
ReDim arrNminusX(1 To lngN)
ReDim arrPX(1 To lngN)
ReDim arrPNminusX(1 To lngN)
ReDim arrEachIter(1 To lngN)

' ------ STEP 1 ------
For S = 1 To lngMaxRun
    If S <= lngX Then
        arrN(S) = lngN - S + 1
        arrX(S) = lngX - S + 1
    Else
        arrN(S) = 1
        arrX(S) = 1
    End If
    If S <= lngX Then
        arrPX(S) = dProb
    Else
        arrPX(S) = 1
    End If
    If S <= lngNminusX Then
        arrPNminusX(S) = (1 - dProb)
    Else
        arrPNminusX(S) = 1
    End If
    arrEachIter(S) = (arrN(S) / arrX(S)) * arrPX(S) * arrPNminusX(S)
Next

lngSmallIndex = 0
lngLargeIndex = lngMaxRun + 1

' ------ STEP 2 ------
Do While lngSmallIndex + 1 < lngLargeIndex
    If fBinomial > 0.5 Then
        If arrEachIter(lngSmallIndex + 1) < arrEachIter(lngLargeIndex - 1) Then
            lngSmallIndex = lngSmallIndex + 1
            fBinomial = fBinomial * arrEachIter(lngSmallIndex)
        Else
            lngLargeIndex = lngLargeIndex - 1
            fBinomial = fBinomial * arrEachIter(lngLargeIndex)
        End If
    Else
        If arrEachIter(lngSmallIndex + 1) > arrEachIter(lngLargeIndex - 1) Then
            lngSmallIndex = lngSmallIndex + 1
            fBinomial = fBinomial * arrEachIter(lngSmallIndex)
        Else
            lngLargeIndex = lngLargeIndex - 1
            fBinomial = fBinomial * arrEachIter(lngLargeIndex)
        End If
    End If
Loop

End Function
Function fFindMax(dbl_A As Double, dbl_B As Double) As Double
' T. Judson Wooters
' FindMax finds the maximum number between two numbers

If dbl_A > dbl_B Then
    fFindMax = dbl_A
Else
```

```
    fFindMax = dbl_B
End If

End Function
Function fFindMaxLong(dbl_A As Long, dbl_B As Long) As Long
' T. Judson Wooters
' FindMaxLong finds the maximum number between two integers

If dbl_A > dbl_B Then
    fFindMaxLong = dbl_A
Else
    fFindMaxLong = dbl_B
End If

End Function
Function fFindMinLong(dbl_A As Long, dbl_B As Long) As Long
' T. Judson Wooters
' FindMaxLong finds the maximum number between two integers

If dbl_A < dbl_B Then
    fFindMinLong = dbl_A
Else
    fFindMinLong = dbl_B
End If

End Function
Function FindWindow(a_dStats() As Double, a_dTot() As Double, _
                    a_lngWindow() As Long, lngWin As Long)
' Author:           T. Judson Wooters
' Created Date:     10-JAN-2007
' Description:      This program finds the base state on either side
'                   of the Markov Chain

Dim arrWin() As Double
Dim iExtState As Integer
Dim iArrSize As Integer
Dim bRightFound As Boolean
Dim bLeftFound As Boolean
Dim N As Integer
Dim P As Integer
Dim R As Integer

iExtState = (a_dStats(1, 1) ^ 2) ^ (1 / 2#)
iArrSize = 2 * iExtState
bRightFound = False
bLeftFound = False
lngWin = 0

ReDim arrWin(1 To iArrSize)

' ---- FIND THE POSITIVE BASE STATE ----
For N = iExtState + 1 To iArrSize
    arrWin(N) = a_dStats(4, N)
    For P = N + 1 To iArrSize - 1
        arrWin(P) = arrWin(P - 1) * (1 - a_dTot(4, P - 1))
    Next
    If N < iArrSize Then
        arrWin(iArrSize) = (arrWin(iArrSize - 1) * (1 - a_dTot(4, iArrSize - 1))) _
                           / a_dTot(4, iArrSize)
    End If
    If N > iExtState + 1 Then
        For P = N - 1 To iExtState + 1 Step -1
            If P = iArrSize - 1 Then
                arrWin(P) = (arrWin(P + 1) * a_dTot(4, P + 1)) / (1 - a_dTot(4, P))
            Else
                arrWin(P) = arrWin(P + 1) / (1 - a_dTot(4, P))
            End If
        Next
    End If
    For P = iExtState + 1 To iArrSize
        If arrWin(P) < a_dStats(4, P) Then
```

```
            For R = iExtState + 1 To iArrSize
                arrWin(R) = 0
            Next
            Exit For
        ElseIf P = iArrSize Then bRightFound = True
        End If
    Next
    If bRightFound = True Then Exit For
Next

' ---- FIND THE NEGATIVE BASE STATE ----
For N = iExtState To 1 Step -1
    arrWin(N) = a_dStats(4, N)
    For P = N - 1 To 2 Step -1
        arrWin(P) = arrWin(P + 1) * (1 - a_dTot(4, P + 1))
    Next
    If N > 1 Then
        arrWin(1) = (arrWin(2) * (1 - a_dTot(4, 2))) / a_dTot(4, 1)
    End If
    If N < iExtState Then
        For P = N + 1 To iExtState
            If P = 2 Then
                arrWin(P) = (arrWin(P - 1) * a_dTot(4, P - 1)) / (1 - a_dTot(4, P))
            Else
                arrWin(P) = arrWin(P - 1) / (1 - a_dTot(4, P))
            End If
        Next
    End If
    For P = iExtState To 1 Step -1
        If arrWin(P) < a_dStats(4, P) Then
            For R = iExtState To 1 Step -1
                arrWin(R) = 0
            Next
            Exit For
        ElseIf P = 1 Then bLeftFound = True
        End If
    Next
    If bLeftFound = True Then Exit For
Next

For N = 1 To iArrSize
    arrWin(N) = fRoundUp(arrWin(N), 0)
    lngWin = lngWin + arrWin(N)
    a_lngWindow(1, N) = arrWin(N)
Next

End Function
Function fRoundUp(dValue As Double, iDecimal As Integer) As Double
' Author:           T. Judson Wooters
' Created Date:     10-JAN-2007
' Description:      This program rounds up to the next decimal

Dim myDec As Long

myDec = InStr(1, CStr(dValue), ".", vbTextCompare) + iDecimal
If myDec > 0 Then
    fRoundUp = CDbl(Left(CStr(dValue), myDec)) + (1 / (10 ^ iDecimal))
Else
    fRoundUp = dValue
End If

End Function
Function fRoundDown(dValue As Double, iDecimal As Integer) As Double
' Author:           T. Judson Wooters
' Created Date:     10-JAN-2007
' Description:      This program rounds down to the next decimal

Dim myDec As Long

myDec = InStr(1, CStr(dValue), ".", vbTextCompare) + iDecimal
If myDec > 0 Then
```

```
    fRoundDown = CDbl(Left(CStr(dValue), myDec))
Else
    fRoundDown = dValue
End If

End Function
Sub UpperHalf(N As Long, arrProb() As Double, dProb As Double, lngMean As Long)
' Author:           T. Judson Wooters
' Created Date:     10-JAN-2007
' Description:      This program calculates the binomial for the upper
'                   half of the distribution

Dim X As Long

For X = lngMean To N
    arrProb(X) = fBinomial(N, X, dProb)
    If arrProb(X) < dEpsilon Then
        Exit For
    End If
Next X

End Sub
Sub LowerHalf(N As Long, arrProb() As Double, dProb As Double, lngMean As Long)
' Author:           T. Judson Wooters
' Created Date:     10-JAN-2007
' Description:      This program calculates the binomial for the lower
'                   half of the distribution

Dim X As Long

For X = lngMean - 1 To 0 Step -1
    arrProb(X) = fBinomial(N, X, dProb)
    If arrProb(X) < dEpsilon Then
        Exit For
    End If
Next X

End Sub
Function RequestFile(sFileData As String, sFileDir As String, sFileType As String, _
                sFileTypeName As String) As Integer
' Sub program to request file name from user

Dim vSelectedItem As Variant    ' temporarily contains file name
Dim fdFileData As FileDialog    ' object of file dialog

Set fdFileData = Application.FileDialog(msoFileDialogFilePicker)

With fdFileData
    .Filters.Clear
    .Filters.Add sFileTypeName, "*" & sFileType, 1
    .AllowMultiSelect = False
    .Title = "Select " & sFileTypeName & " File"
    If .Show = -1 Then
        For Each vSelectedItem In .SelectedItems
            sFileData = vSelectedItem
        Next
        sFileDir = .InitialFileName
        RequestFile = 0
    Else
        RequestFile = -1
        Exit Function
    End If
End With

End Function
Function fOptMax(iMax As Long, bStates As Boolean, iSR As Integer, _
            iInitState As Integer, lngArrSize As Long, dExtreme As Double, _
            a_dTotErr() As Double, dExtremeMax As Double) As Long
' Author:           T. Judson Wooters
' Created Date:     10-JAN-2007
' Description:      This program finds the upper range of SR and States ratios
```

```
Dim iMaxSR As Long
Dim iMaxStates As Integer
Dim lngDataPts As Long
Dim a_dTot() As Double
Dim bModCand As Boolean
Dim P As Long
Dim R As Integer
Dim N As Integer

If bStates Then
    iMaxStates = iMax
    iMaxSR = iSR
Else
    iMaxSR = iMax
    iMaxStates = iInitState
End If

For P = iSR To iMaxSR
    For R = iInitState To iMaxStates Step 2
        lngDataPts = lngArrSize
        ReDim a_dTot(1 To 4, 1 To R)
        Call Charact(R, dExtreme, P, lngDataPts, a_dTotErr(), a_dTot())
        For N = 1 To R
            If dExtreme <= dExtremeMax Then
                bModCand = True
            Else
                bModCand = False
                Exit For
            End If
        Next N
        If bModCand Then
            Exit For
        End If
    Next R
    If bModCand Then        ' Found max number of states or SR
        If bStates Then
            fOptMax = R
        Else
            fOptMax = P
        End If
        Exit For
    Else
        If bStates Then
            fOptMax = R
        Else
            fOptMax = P
        End If
    End If
Next P

End Function
```

---- Run Individual Subs ----

```
Option Explicit
Sub HistoryTest()
' Author:           T. Judson Wooters
' Created Date:     3-JAN-2008
' Description:      Main program

Dim a_dTotTime() As Double      ' time corresponding to each actuating error data point
Dim a_dTotErr() As Double       ' each actuating error data point
Dim a_dTot() As Double          ' used in Charact sub, holds transition probabilities
Dim a_dStats() As Double        ' holds transition probability limits in samples
Dim a_lngWindow() As Long       ' holds number of samples expected to visit states
Dim a_dTranLimits() As Double   ' holds transition probability limits as a fraction
Dim a_dSP() As Double           ' each setpoint data point
Dim a_dCV() As Double           ' each controlled variable data point
Dim dInitTime As Double         ' initial system clock time
Dim dFinalTime As Double        ' end system clock time
Dim dExtremeMax As Double       ' allowed maximum sample fraction in extreme states
Dim dExtreme As Double          ' sample fraction in extreme states
Dim dAlphaT As Double           ' allowed maximum alpha test
Dim dBetaT As Double            ' allowed maximum beta test
Dim dLamda As Double            ' allowed maximum lambda change
Dim dMeasFreq As Double         ' optional measurement frequency in seconds
Dim dSettle As Double           ' process settling time
Dim dSampleFreq As Double       ' sampling frequency in seconds
Dim dStatWinTime As Double      ' statistical window in seconds
Dim dTotWinTime As Double       ' total window = statistical window + settling time
(seconds)
Dim dAlphaK As Double           ' alpha for each state
Dim lngDataPts As Long          ' number of samples based on sampling ratio
Dim lngArrSize As Long          ' total number of samples
Dim lngXL As Long               ' sample number lower limit
Dim lngXH As Long               ' sample number high limit
Dim lngMinPoints As Long        ' minimum samples required for a given state
Dim lngStatWin As Long          ' statistical window in samples
Dim lngSampleSettle As Long     ' samples in settling time
Dim lngTotWin As Long           ' total window = statistical window + settling time
(samples)
Dim lngN As Long                ' temporary storage for min number of samples
Dim iMaxStates As Integer       ' maximum number of states
Dim iSR As Integer              ' sampling ratio
Dim iInitState As Integer       ' initial number of states
Dim iMaxSR As Integer           ' maximum sampling ratio
Dim dOptTime As Double          ' time required by optimum states / sampling ratio
Dim lngOptSample As Long        ' samples requred by optimum states / sampling ratio
Dim iOptStates As Integer       ' optimum number of states
Dim iOptSR As Long           ' optimum sampling ratio
Dim iExitErr As Integer         ' exit error number (0 = ok, -1 = error)
Dim vSelectedItem As Variant    ' holds value of selection for file name
Dim sFileData As String         ' complete file name with path
Dim sFileTypeName As String     ' file type found in header
Dim sFileType As String         ' file type
Dim sErrFileData As String      ' temporary storage for file name with path
Dim sErrFileDir As String       ' error file path
Dim sReason As String           ' explains model output
Dim fdFileData As FileDialog    ' file dialog object for user input
Dim bModCand As Boolean         ' model candidate
Dim bNotZero As Boolean         ' if a state has 0 visits
Dim bDebug As Boolean           ' if model output should be viewed
Dim dCompleteP As Double        ' percent complete in finding model
Dim N As Long                   ' counting variable
Dim P As Long                ' counting variable
Dim R As Integer                ' counting variable

' ==== INITIALIZE VARIABLES ====
dExtreme = 1
dAlphaT = shtModel.Cells(1, 3).Value
dBetaT = shtModel.Cells(2, 3).Value
dLamda = shtModel.Cells(3, 3).Value
dExtremeMax = shtModel.Cells(6, 3).Value
iInitState = shtModel.Cells(7, 3).Value
```

```
iSR = 1
dMeasFreq = shtModel.Cells(8, 3).Value
dSettle = shtModel.Cells(4, 3).Value
iMaxStates = shtModel.Cells(9, 3).Value
bDebug = shtModel.Cells(10, 3).Value

' ==== REQUEST ERROR FILE ====
sFileType = ".aer"
sFileTypeName = "Actuating Error"
If RequestFile(sErrFileData, sErrFileDir, sFileType, sFileTypeName) = -1 Then Exit Sub
sFileData = sErrFileData


dInitTime = Timer           ' Initialize the timer

' ==== READ ERROR FILE ====
Open sFileData For Input As #1
Do While Not EOF(1)
    lngArrSize = lngArrSize + 1
    ReDim Preserve a_dTotErr(1 To lngArrSize)
    ReDim Preserve a_dTotTime(1 To lngArrSize)
    ReDim Preserve a_dSP(1 To lngArrSize)
    ReDim Preserve a_dCV(1 To lngArrSize)
    Input #1, a_dTotTime(lngArrSize), a_dCV(lngArrSize), a_dSP(lngArrSize)
    a_dTotErr(lngArrSize) = a_dSP(lngArrSize) - a_dCV(lngArrSize)
Loop
Close #1

' ==== RERUN FOR OPTIMUM PARAMETERS ====
iOptSR = shtTestTran.Cells(1, 2).Value
iOptStates = shtTestTran.Cells(2, 2).Value
lngDataPts = lngArrSize                         ' reset total number of data points
ReDim a_dTot(1 To 4, 1 To iOptStates)
Call Charact(iOptStates, dExtreme, iOptSR, lngDataPts, a_dTotErr(), a_dTot())
ReDim a_dStats(1 To 4, 1 To iOptStates)
ReDim a_lngWindow(1 To 3, 1 To iOptStates)
ReDim a_lngWindow(1 To 3, 1 To iOptStates)
ReDim a_dTranLimits(1 To 2, 1 To iOptStates)
For N = 1 To iOptStates
    RunLimitsAlg dAlphaT, dBetaT, dLamda, iOptStates, a_dTot(4, N), lngXL, _
                 lngXH, lngMinPoints, False
    a_dStats(2, N) = lngXL
    a_dStats(3, N) = lngXH
    a_dStats(4, N) = lngMinPoints
    a_dStats(1, N) = a_dTot(1, N)
Next N
FindWindow a_dStats(), a_dTot(), a_lngWindow(), lngStatWin
dSampleFreq = dMeasFreq * iOptSR
lngSampleSettle = fRoundUp((dSettle / dSampleFreq), 0)
lngTotWin = lngStatWin + lngSampleSettle
dStatWinTime = lngStatWin * dSampleFreq
dTotWinTime = lngTotWin * dSampleFreq
dAlphaK = 1# - (1# - dAlphaT) ^ (1# / iOptStates)
For N = 1 To iOptStates
    lngN = a_lngWindow(1, N)
    Limits dAlphaK, a_dTot(4, N), 1, 0, lngXL, lngXH, False, lngN, 0, True
    a_lngWindow(2, N) = lngXH
    a_lngWindow(3, N) = lngXL
    a_dTranLimits(1, N) = (lngXH / a_lngWindow(1, N)) 'fRoundUp(lngXH / a_lngWindow(1,
N), 6)
    a_dTranLimits(2, N) = (lngXL / a_lngWindow(1, N)) 'fRoundDown(lngXL / a_lngWindow(1,
N), 6)
Next

' ==== WRITE MODEL PARAMETERS TO EXCEL ====
shtTestTran.Cells(15, 3).Value = iOptStates
shtTestTran.Cells(14, 3).Value = iOptSR
shtTestTran.Cells(16, 3).Value = lngStatWin
shtTestTran.Cells(17, 3).Value = lngTotWin
shtTestTran.Cells(18, 3).Value = Round(dStatWinTime, 2)
shtTestTran.Cells(19, 3).Value = Round(dTotWinTime, 2)
```

```
For N = 1 To iOptStates
    shtTestTran.Cells(1, N + 4) = a_dTot(1, N)
    shtTestTran.Cells(2, N + 4) = a_dTot(2, N)
    shtTestTran.Cells(3, N + 4) = a_dTranLimits(1, N)
    shtTestTran.Cells(4, N + 4) = a_dTot(4, N)
    shtTestTran.Cells(5, N + 4) = a_dTranLimits(2, N)
Next N

End Sub
Sub sub_RunLimitsMH()

Dim dbl_AlphaT As Double
Dim dbl_BetaT As Double
Dim dbl_Lamda As Double
Dim int_States As Integer
Dim dbl_Prob0 As Double
Dim lng_XL As Long
Dim lng_XH As Long
Dim lng_MinPoints As Long
Dim dTimeStart As Double

dTimeStart = Timer

Application.ScreenUpdating = False

dbl_AlphaT = shtTestLimits.Cells(1, 2)
dbl_BetaT = shtTestLimits.Cells(2, 2)
dbl_Lamda = shtTestLimits.Cells(3, 2)
int_States = shtTestLimits.Cells(1, 5)
dbl_Prob0 = shtTestLimits.Cells(2, 5)

RunLimitsAlg dbl_AlphaT, dbl_BetaT, dbl_Lamda, int_States, dbl_Prob0, lng_XL, _
            lng_XH, lng_MinPoints, True

Application.ScreenUpdating = True

shtTestLimits.Cells(1, 7) = Timer - dTimeStart

End Sub
Sub sub_FindAlphaLimits()

Dim dAlphaT As Double
Dim dAlphaK As Double
Dim iStates As Integer
Dim dProbRef As Double
Dim lngXL As Long
Dim lngXH As Long
Dim lngN As Long
Dim iNumSamples As Long

dAlphaT = shtTestLimits.Cells(1, 2).Value
iStates = shtTestLimits.Cells(1, 5).Value
dProbRef = shtTestLimits.Cells(2, 5).Value
lngN = shtTestLimits.Cells(6, 2).Value

dAlphaK = 1# - (1# - dAlphaT) ^ (1# / iStates)

Limits dAlphaK, dProbRef, 1, 0, lngXL, lngXH, True, lngN, 0, True

End Sub
Sub ModifyChartTestTran(iStates As Integer, sChartName As String)
' T. Judson Wooters, 20-DEC-2007
' Inputs:  iStates - number of states to be ploted
' Outputs: None
' Purpose: Makes a chart visible and changes its source

    Dim K As Integer        ' Counting variable

    'iStates = 8
    'sChartName = "ModelChart"
```

```
    With shtTestTran.ChartObjects(sChartName)
        .Visible = True     ' Set the chart visible
    End With

    With shtTestTran.ChartObjects(sChartName).Chart
        For K = 1 To 3
            .SeriesCollection(K).XValues = "='Test States'!R1C6:R1C" & (5 + iStates) '
Change the X axis source
            .SeriesCollection(K).Values = "='Test States'!R" & (1 + K) & "C6:R" & (1 + K)
& "C" & (5 + iStates) ' Change the Y axis source
        Next K
    End With

End Sub
```

---- Plot Update Code ----

```
Option Explicit
Sub ModifyChart(iStates As Integer, sChartName As String)
' T. Judson Wooters, 20-DEC-2007
' Inputs:  iStates - number of states to be ploted
' Outputs: None
' Purpose: Makes a chart visible and changes its source

    Dim K As Integer        ' Counting variable

    'iStates = 8
    'sChartName = "ModelChart"

    With shtModel.ChartObjects(sChartName)
        .Visible = True     ' Set the chart visible
    End With

    With shtModel.ChartObjects(sChartName).Chart
        For K = 1 To 4
            .SeriesCollection(K).XValues = "=Data!R1C3:R1C" & _
                        (2 + iStates) ' Change the X axis source
            .SeriesCollection(K).Values = "=Data!R" & (1 + K) & _
                        "C3:R" & (1 + K) & "C" & (2 + iStates)
                        ' Change the Y axis source
        Next K
    End With

End Sub
Function OutputModelExcel(iMaxStates As Integer, iInitState As Integer, _
                        P As Long, lngDataPts As Long, R As Integer, _
                        dExtreme As Double, dExtremeMax As Double, _
                        dTotWinTime As Double, bNotZero As Boolean, _
                        sReason As String, bModCand As Boolean, _
                        iSR As Integer) As Integer

    If iInitState Then
        shtData.Cells(9, 2) = "SR"
        shtData.Cells(8, 4) = "Extreme"
        shtData.Cells(8, ((iMaxStates - iInitState) / 2) + 5) = _
                        "Statistical Window (sec)"
        shtData.Cells(8, (2 * (iMaxStates - iInitState) / 2 + 1) + 5) = _
                        "Reason"
        shtData.Cells(8, 3) = "Total"
        shtData.Cells(9 + P, 2) = P
        shtData.Cells(9 + P, 3) = lngDataPts
    End If
    If P = iSR Then
        shtData.Cells(9, (R / 2)).Value = R
        shtData.Cells(9, (iMaxStates / 2) + (R / 2) - 3).Value = R
        shtData.Cells(9, (2 * (iMaxStates - iInitState) / 2 + 1) + 5 + _
                        (R - iInitState) / 2).Value = R
    End If
    shtData.Cells(9 + P, (R / 2)) = dExtreme
    If bModCand Then
        shtData.Cells(9 + P, ((iMaxStates / 2) + (R / 2) - 3)) = dTotWinTime
    End If
    If dExtreme > dExtremeMax Then
        sReason = "Ext > " & dExtremeMax
        If Not bNotZero Then
            sReason = sReason & ", TranProb=0/1"
        End If
    ElseIf Not bNotZero Then
        sReason = "TranProb=0/1"
    End If
    If bModCand Then
        shtData.Cells(9 + P, (R / 2)).Interior.ColorIndex = 45
        shtData.Cells(9 + P, ((iMaxStates / 2) + (R / 2) - 3)) _
                    .Interior.ColorIndex = 45
        shtData.Cells(9 + P, (2 * (iMaxStates - iInitState) / 2 + 1) _
                    + 5 + (R - iInitState) / 2).Interior.ColorIndex = 45
```

```
            shtData.Cells(9 + P, (2 * (iMaxStates - iInitState) / 2 + 1) _
                        + 5 + (R - iInitState) / 2).Value = "MODEL"
            OutputModelExcel = -1
        Else
            shtData.Cells(9 + P, (2 * (iMaxStates - iInitState) / 2 + 1) _
                        + 5 + (R - iInitState) / 2).Value = sReason
            shtData.Cells(9 + P, (2 * (iMaxStates - iInitState) / 2 + 1) _
                        + 5 + (R - iInitState) / 2).Interior.ColorIndex = 15
            shtData.Cells(9 + P, (R / 2)).Interior.ColorIndex = 15
            shtData.Cells(9 + P, ((iMaxStates / 2) + (R / 2) - 3)) _
                        .Interior.ColorIndex = 15
            shtData.Columns((2 * (iMaxStates - iInitState) / 2 + 1) _
                        + 5 + (R - iInitState) / 2).EntireColumn.AutoFit
            OutputModelExcel = 0
        End If

    End Function
    Sub CleanUpExcel(iMaxStates As Integer, iInitState As Integer, iMaxSR As Integer)

        shtData.Activate

        ' ==== FIXES HEADERS ====
        If iMaxStates > 8 Then
            With shtData.Range(Cells(8, 4), Cells(8, (iMaxStates / 2)))
                .Merge
                .Font.Bold = True
                .HorizontalAlignment = xlCenter
            End With
            With shtData.Range(Cells(8, (iMaxStates / 2) + 1), _
                    Cells(8, iMaxStates - 3))
                .Merge
                .Font.Bold = True
                .HorizontalAlignment = xlCenter
            End With
            With shtData.Range(Cells(8, (2 * (iMaxStates - iInitState) / _
                            2 + 1) + 5), Cells(8, (3 * (iMaxStates - iInitState) _
                            / 2 + 1) + 5))
                .Merge
                .Font.Bold = True
                .HorizontalAlignment = xlCenter
            End With
        ElseIf iMaxStates = 8 Then
            With shtData.Cells(8, 4)
                .Font.Bold = True
                .HorizontalAlignment = xlCenter
            End With
            With shtData.Cells(8, (iMaxStates / 2) + 1)
                .Font.Bold = True
                .HorizontalAlignment = xlCenter
            End With
        End If

        With shtData.Cells(8, 3)
            .Font.Bold = True
            .HorizontalAlignment = xlCenter
        End With

        ' ==== BORDERS ====
        With shtData.Range(Cells(8, 2), Cells(9 + iMaxSR, (3 * (iMaxStates _
                    - iInitState) / 2 + 1) + 5))
            .Borders(xlEdgeRight).LineStyle = xlContinuous
            .Borders(xlEdgeRight).Weight = xlThick
            .Borders(xlEdgeLeft).LineStyle = xlContinuous
            .Borders(xlEdgeLeft).Weight = xlThick
            .Borders(xlEdgeTop).LineStyle = xlContinuous
            .Borders(xlEdgeTop).Weight = xlThick
            .Borders(xlEdgeBottom).LineStyle = xlContinuous
            .Borders(xlEdgeBottom).Weight = xlThick
        End With

        With shtData.Range(Cells(9, 2), Cells(9, (3 * (iMaxStates - iInitState) _
```

```
                / 2 + 1) + 5))
        .Borders(xlEdgeBottom).LineStyle = xlContinuous
        .Borders(xlEdgeBottom).Weight = xlThin
        .HorizontalAlignment = xlCenter
    End With

    With shtData.Range(Cells(8, 2), Cells(9 + iMaxSR, 2))
        .Borders(xlEdgeRight).LineStyle = xlContinuous
        .Borders(xlEdgeRight).Weight = xlThin
        .HorizontalAlignment = xlCenter
    End With

    With shtData.Range(Cells(8, 4), Cells(9 + iMaxSR, (iMaxStates / 2)))
        .Borders(xlEdgeRight).LineStyle = xlContinuous
        .Borders(xlEdgeRight).Weight = xlThin
        .Borders(xlEdgeLeft).LineStyle = xlContinuous
        .Borders(xlEdgeLeft).Weight = xlThin
    End With

    With shtData.Range(Cells(8, (2 * (iMaxStates - iInitState) / 2 + 1) _
            + 5), Cells(9 + iMaxSR, (2 * (iMaxStates - iInitState) / 2 + 1) + 5))
        .Borders(xlEdgeLeft).LineStyle = xlContinuous
        .Borders(xlEdgeLeft).Weight = xlThin
    End With
    shtModel.Activate

End Sub
```

---- Real-Time Data Analyzer ----

```
Option Explicit
Sub RealTime()

Dim aLimit() As Double
Dim aTempErr() As Double
Dim aTotTime() As Double
Dim aSample() As Double
Dim aState() As Integer
Dim aTran() As Double
Dim aTot() As Double
Dim a_dSP() As Double
Dim a_dCV() As Double
Dim aF() As Double
Dim aT() As Double
Dim aFlagVal() As Double
Dim aFlagTime() As Double
Dim dCurrTime As Double
Dim dPrevTime As Double
Dim dCurrErr As Double
Dim dSampleFreq As Double
Dim dSeed As Double
Dim dFlag As Double
Dim dCurrFlag As Double
Dim dPrevFlag As Double
Dim dFlagTime As Double
Dim dCurrSP As Double
Dim iFlagStates As Integer
Dim lngFlagSamples As Long
Dim iStates As Integer
Dim iNextState As Integer
Dim iPrevState As Integer
Dim lngSR As Long
Dim lngSample As Long
Dim lngArrSize As Long
Dim lngIndexS As Long
Dim lngIndexFlag As Long
Dim lngPrevIndex As Long
Dim lngNextIndex As Long
Dim lngFlagIndex As Long
Dim lngFIndex As Long
Dim lngStartIndex As Long
Dim vSelectedItem As Variant
Dim sFileData As String
Dim fdFileData As FileDialog
Dim aTotTest() As Double
Dim aStateTest() As Integer
Dim aSampleTest() As Double
Dim IndexTest As Long
Dim iSeed As Integer
Dim dummy As String
Dim bContinousSP As Boolean
Dim K As Long
Dim J As Long
Dim N As Long

Application.ScreenUpdating = False
dSampleFreq = 0.1
bContinousSP = shtRT.Cells(1, 2).Value

Set fdFileData = Application.FileDialog(msoFileDialogFilePicker)

With fdFileData
    .AllowMultiSelect = False
    If .Show = -1 Then
        For Each vSelectedItem In .SelectedItems
            sFileData = vSelectedItem
        Next
```

```
    Else
        Exit Sub
    End If
End With

Open sFileData For Input As #1
Input #1, iStates
Input #1, lngSR
Input #1, lngSample
Input #1, lngFlagSamples
lngSample = lngSample + 1               ' Must get 1 more sample to find final state tran
lngFlagSamples = lngFlagSamples     ' Must add 1 more sample
ReDim aLimit(1 To 3, 1 To iStates)
ReDim aTran(1 To 3, 1 To iStates)
ReDim aSample(1 To lngSample)
ReDim aTot(1 To 4, 1 To iStates)
ReDim aTotTest(1 To 4, 1 To iStates)
ReDim aState(1 To lngSample)
ReDim aStateTest(1 To lngSample)
ReDim aSampleTest(1 To lngSample)
ReDim aFlagVal(1 To 1)
ReDim aFlagTime(1 To 1)

For K = 1 To iStates
    If (K - (iStates / 2)) > 0 Then
        aLimit(1, K) = K - (iStates / 2)
        aTran(1, K) = aLimit(1, K)
        aTot(1, K) = aLimit(1, K)
        aTotTest(1, K) = aLimit(1, K)
    Else
        aLimit(1, K) = K - (iStates / 2) - 1
        aTran(1, K) = aLimit(1, K)
        aTot(1, K) = aLimit(1, K)
        aTotTest(1, K) = aLimit(1, K)
    End If
Next
For K = 1 To iStates
    Input #1, aLimit(1, K), aLimit(2, K), aLimit(3, K)
Next K
Close #1

With fdFileData
    .AllowMultiSelect = False
    If .Show = -1 Then
        For Each vSelectedItem In .SelectedItems
            sFileData = vSelectedItem
        Next
    Else
        Exit Sub
    End If
End With

Open ActiveWorkbook.Path & "\CV.txt" For Output Access Write Lock Write As #4
Open ActiveWorkbook.Path & "\SP.txt" For Output Access Write Lock Write As #5
Open sFileData For Input As #1
Do While Not EOF(1)
    lngArrSize = lngArrSize + 1
    ReDim Preserve aTempErr(1 To lngArrSize)
    ReDim Preserve aTotTime(1 To lngArrSize)
    ReDim Preserve a_dSP(1 To lngArrSize)
    ReDim Preserve a_dCV(1 To lngArrSize)
    Input #1, aTotTime(lngArrSize), a_dCV(lngArrSize), a_dSP(lngArrSize)
    aTempErr(lngArrSize) = a_dSP(lngArrSize) - a_dCV(lngArrSize)
    Print #4, aTotTime(lngArrSize), a_dCV(lngArrSize)
    If bContinousSP Then
        Print #5, aTotTime(lngArrSize), a_dSP(lngArrSize)
    Else
        If dCurrSP <> a_dSP(lngArrSize) Then
            If lngArrSize = 1 Then
                Print #5, aTotTime(lngArrSize), a_dSP(lngArrSize)
            Else
```

```
                Print #5, aTotTime(lngArrSize), dCurrSP
            End If
            dCurrSP = a_dSP(lngArrSize)
            Print #5, aTotTime(lngArrSize), dCurrSP
        End If
    End If
Loop
Print #5, aTotTime(lngArrSize), dCurrSP
Close #1
Close #4
Close #5

dFlag = 1 / (lngFlagSamples + 1)
dCurrFlag = 0

dSeed = aTempErr(1)
iPrevState = fState(dSeed, 0, (iStates / 2))
For K = 1 To lngSample
    aSample(K) = aTempErr(lngSR * (K - 1) + 1)
Next K
dCurrTime = aTotTime(lngSR * (lngSample - 1) + 1)
lngStartIndex = lngSR * lngSample + 1

Open ActiveWorkbook.Path & "\RealTime.txt" For Output Access Write Lock Write As #1
For K = 1 To iStates
    Print #1, aLimit(1, K),
Next K
Print #1,
For K = 1 To iStates
    Print #1, aLimit(2, K),
Next K
Print #1,
For K = 1 To iStates
    Print #1, aLimit(3, K),
Next K
Print #1,

RealCharact iStates, 0, 1, lngSample, aSample(), aTot(), aState(), 0

iFlagStates = 0
For N = 1 To iStates
    If aTot(4, N) > aLimit(2, N) Or aTot(4, N) < aLimit(3, N) Then
        If dCurrFlag = 0 Then
            dFlagTime = dCurrTime
        End If
        dPrevFlag = dCurrFlag
        dCurrFlag = dCurrFlag + dFlag
        Exit For
    End If
    iFlagStates = iFlagStates + 1
Next N

Print #1, dCurrTime,
For J = 1 To iStates
    Print #1, aTot(4, J),
Next J
Print #1, dCurrFlag

If iFlagStates = iStates Then
    dCurrFlag = 0
    dFlagTime = 0
End If

lngFIndex = lngFIndex + 1
ReDim Preserve aT(1 To lngFIndex)
ReDim Preserve aF(1 To lngFIndex)
aT(lngFIndex) = dCurrTime
aF(lngFIndex) = dCurrFlag
aFlagVal(1) = dCurrFlag
aFlagTime(1) = dCurrTime
ReDim Preserve aFlagVal(1 To 2)
```

```
ReDim Preserve aFlagTime(1 To 2)
lngIndexFlag = 2

For K = lngStartIndex To lngArrSize Step lngSR
    dCurrErr = aTempErr(K)
    dPrevTime = dCurrTime
    dCurrTime = aTotTime(K)
    lngIndexS = lngIndexS + 1
    If lngIndexS > lngSample Then lngIndexS = 1
    If lngIndexS = 1 Then
        lngPrevIndex = lngSample
    Else
        lngPrevIndex = lngIndexS - 1
    End If
    If lngIndexS = lngSample Then
        lngNextIndex = 1
    Else
        lngNextIndex = lngIndexS + 1
    End If
    iNextState = modDevModel.fState(dCurrErr, aState(lngPrevIndex), iStates / 2)
    For J = 1 To iStates                                ' fill (2) element of 'tot' array
        If aState(lngIndexS) = aTot(1, J) Then
            aTot(2, J) = aTot(2, J) - 1
            Exit For
        End If
    Next J
    If aState(lngIndexS) > 0 And aState(lngNextIndex) = -1 Then
        For N = 1 To iStates
            If aState(lngIndexS) = aTot(1, N) Then
                aTot(3, N) = aTot(3, N) - 1
                Exit For
            End If
        Next
    ElseIf aState(lngIndexS) < 0 And aState(lngNextIndex) = 1 Then
        For N = 1 To iStates
            If aState(lngIndexS) = aTot(1, N) Then
                aTot(3, N) = aTot(3, N) - 1
                Exit For
            End If
        Next
    End If
    If aState(lngPrevIndex) > 0 And iNextState = -1 Then
        For N = 1 To iStates
            If aState(lngPrevIndex) = aTot(1, N) Then
                aTot(3, N) = aTot(3, N) + 1
                Exit For
            End If
        Next
    ElseIf aState(lngPrevIndex) < 0 And iNextState = 1 Then
        For N = 1 To iStates
            If aState(lngPrevIndex) = aTot(1, N) Then
                aTot(3, N) = aTot(3, N) + 1
                Exit For
            End If
        Next
    End If
    For J = 1 To iStates                              ' fill (2) element of 'tot' array
        If aState(lngPrevIndex) = aTot(1, J) Then
            aTot(2, J) = aTot(2, J) + 1
            Exit For
        End If
    Next J

    iSeed = aState(lngIndexS)
    aState(lngIndexS) = iNextState
    aSample(lngIndexS) = dCurrErr

    For N = 1 To iStates
        If aTot(2, N) = 0 Then
            aTot(4, N) = 0
        Else
```

```
        aTot(4, N) = aTot(3, N) / aTot(2, N)
    End If
Next

iFlagStates = 0
For N = 1 To iStates
    If aTot(4, N) > aLimit(2, N) Or aTot(4, N) < aLimit(3, N) Then
        If dCurrFlag = 0 Then
            dFlagTime = dCurrTime
        End If
        dPrevFlag = dCurrFlag
        dCurrFlag = dCurrFlag + dFlag
        If dCurrFlag >= 1 Then
            dCurrFlag = 1
            aFlagVal(lngIndexFlag) = 1
        End If
        If aFlagVal(lngIndexFlag) = 0 Then
            aFlagTime(lngIndexFlag) = dCurrTime
            lngIndexFlag = lngIndexFlag + 1
            ReDim Preserve aFlagVal(1 To lngIndexFlag)
            ReDim Preserve aFlagTime(1 To lngIndexFlag)
            aFlagVal(lngIndexFlag) = dCurrFlag
            aFlagTime(lngIndexFlag) = dCurrTime
        End If
        Exit For
    End If
    iFlagStates = iFlagStates + 1
Next N
' **** PRINTS ALL TRANSITION PROBABILITIES ****
Print #1, dCurrTime,
For J = 1 To iStates
    Print #1, aTot(4, J),
Next J
Print #1, dCurrFlag
' ******************************************
If iFlagStates = iStates Then
    dPrevFlag = dCurrFlag
    dCurrFlag = 0
    dFlagTime = 0
    If aFlagVal(lngIndexFlag) = 1 Then
        lngIndexFlag = lngIndexFlag + 1
        ReDim Preserve aFlagVal(1 To lngIndexFlag)
        ReDim Preserve aFlagTime(1 To lngIndexFlag)
        aFlagVal(lngIndexFlag) = 1
        aFlagTime(lngIndexFlag) = dCurrTime
        lngIndexFlag = lngIndexFlag + 1
        ReDim Preserve aFlagVal(1 To lngIndexFlag)
        ReDim Preserve aFlagTime(1 To lngIndexFlag)
        aFlagVal(lngIndexFlag) = 0
        aFlagTime(lngIndexFlag) = dCurrTime
        lngIndexFlag = lngIndexFlag + 1
        ReDim Preserve aFlagVal(1 To lngIndexFlag)
        ReDim Preserve aFlagTime(1 To lngIndexFlag)
    Else
        If aFlagVal(lngIndexFlag) > 0 Then
            lngIndexFlag = lngIndexFlag - 1
            ReDim Preserve aFlagVal(1 To lngIndexFlag)
            ReDim Preserve aFlagTime(1 To lngIndexFlag)
        End If
        aFlagVal(lngIndexFlag) = 0
        aFlagTime(lngIndexFlag) = dCurrTime
    End If
End If

If dCurrFlag < dPrevFlag Then
    lngFIndex = lngFIndex + 1
    ReDim Preserve aT(1 To lngFIndex)
    ReDim Preserve aF(1 To lngFIndex)
    aT(lngFIndex) = dPrevTime
    aF(lngFIndex) = dPrevFlag
    lngFIndex = lngFIndex + 1
```

```
        ReDim Preserve aT(1 To lngFIndex)
        ReDim Preserve aF(1 To lngFIndex)
        aT(lngFIndex) = dCurrTime
        aF(lngFIndex) = dCurrFlag
    ElseIf dCurrFlag > dPrevFlag And dPrevFlag = 0 Then
        lngFIndex = lngFIndex + 1
        ReDim Preserve aT(1 To lngFIndex)
        ReDim Preserve aF(1 To lngFIndex)
        aT(lngFIndex) = dPrevTime
        aF(lngFIndex) = dPrevFlag
        lngFIndex = lngFIndex + 1
        ReDim Preserve aT(1 To lngFIndex)
        ReDim Preserve aF(1 To lngFIndex)
        aT(lngFIndex) = dCurrTime
        aF(lngFIndex) = dCurrFlag
    ElseIf dCurrFlag > dPrevFlag And dCurrFlag = 1 Then
        lngFIndex = lngFIndex + 1
        ReDim Preserve aT(1 To lngFIndex)
        ReDim Preserve aF(1 To lngFIndex)
        aT(lngFIndex) = dPrevTime
        aF(lngFIndex) = dPrevFlag
        lngFIndex = lngFIndex + 1
        ReDim Preserve aT(1 To lngFIndex)
        ReDim Preserve aF(1 To lngFIndex)
        aT(lngFIndex) = dCurrTime
        aF(lngFIndex) = dCurrFlag
    End If


Next K

lngIndexFlag = lngIndexFlag + 1
ReDim Preserve aFlagVal(1 To lngIndexFlag)
ReDim Preserve aFlagTime(1 To lngIndexFlag)
aFlagVal(lngIndexFlag) = aFlagVal(lngIndexFlag - 1)
aFlagTime(lngIndexFlag) = dCurrTime
lngFIndex = lngFIndex + 1
ReDim Preserve aT(1 To lngFIndex)
ReDim Preserve aF(1 To lngFIndex)
aT(lngFIndex) = dCurrTime
aF(lngFIndex) = aF(lngFIndex - 1)

Close #1

Open ActiveWorkbook.Path & "\Flag.txt" For Output Access Write Lock Write As #2
For K = 1 To lngFIndex
    Print #2, aT(K), aF(K)
Next K
Close #2
Open ActiveWorkbook.Path & "\SquareFlag.txt" For Output Access Write Lock Write As #3
For K = 1 To lngIndexFlag
    Print #3, aFlagTime(K), aFlagVal(K)
Next K
Close #3

Application.ScreenUpdating = True

End Sub
Sub RealCharact(iNumState As Integer, dExtreme As Double, iSR As Integer, _
                lngArrSize As Long, a_dTotErr() As Double, a_dTot() As Double, _
                arr_States() As Integer, iSeed As Integer)
' Author:           T. Judson Wooters
' Created Date:     18-DEC-2006
' Description:      Accepts an array of errors, assigns states and returns statistics

Dim N As Long                       ' Internal counter
Dim P As Integer                    ' Internal counter (used to count through states)

' ---------- FILL 'TOT' ARRAY WITH STATES (1) -----------
For N = 1 To iNumState
```

```
    a_dTot(2, N) = 0
    a_dTot(3, N) = 0
    a_dTot(4, N) = 0
    If (N - (iNumState / 2)) > 0 Then
        a_dTot(1, N) = N - (iNumState / 2)
    Else
        a_dTot(1, N) = N - (iNumState / 2) - 1
    End If
Next

For N = 1 To lngArrSize
    arr_States(N) = 0
    If N = 1 Then                                                ' determine current state
        arr_States(N) = fState(a_dTotErr(N), iSeed, (iNumState / 2))
    Else
        arr_States(N) = fState(a_dTotErr(N), arr_States(N - 1), (iNumState / 2))
    End If
Next N

' ---------- FILL 'ERR', 'STATES' AND 'TOT' ARRAYS -----------
For N = 1 To lngArrSize - 1
    For P = 1 To iNumState                          ' fill (2) element of 'tot' array
        If arr_States(N) = a_dTot(1, P) Then
            a_dTot(2, P) = a_dTot(2, P) + 1
            Exit For
        End If
    Next
    If arr_States(N) > 0 And arr_States(N + 1) = -1 Then
        For P = 1 To iNumState
            If arr_States(N) = a_dTot(1, P) Then
                a_dTot(3, P) = a_dTot(3, P) + 1
                Exit For
            End If
        Next
    ElseIf arr_States(N) < 0 And arr_States(N + 1) = 1 Then
        For P = 1 To iNumState
            If arr_States(N) = a_dTot(1, P) Then
                a_dTot(3, P) = a_dTot(3, P) + 1
                Exit For
            End If
        Next
    End If
Next

' --------- TRANSITION PROB = PROBABILIY TO CROSS ZERO ----------
For N = 1 To iNumState
    If a_dTot(2, N) = 0 Then
        a_dTot(4, N) = 0
    Else
        a_dTot(4, N) = a_dTot(3, N) / a_dTot(2, N)
    End If
Next

' -------- DETERMINE STATISTICS OF ENTIRE STATE CHARACTERIZATION ---------
dExtreme = (a_dTot(2, 1) + a_dTot(2, iNumState)) / (lngArrSize)

End Sub

Sub LoadRTFile()

Dim aStates() As Integer
Dim aLimits() As Double
Dim aTran() As Double
Dim aTemp() As Double
Dim dTimeInit As Double
Dim vSelectedItem As Variant
Dim sFileData As String
Dim fdFileData As FileDialog
Dim iStates As Integer
Dim iIndex As Integer
Dim iSR As Integer
```

```
Dim iRecordCnt As Long

Dim K As Long
Dim J As Long
Dim M As Long

Application.ScreenUpdating = False

shtRT.Range(Cells(2, 4), Cells(30001, 20)).ClearContents
shtRT.Range(Cells(30002, 4), Cells(60002, 20)).ClearContents
ReDim aTemp(1 To 2)
iIndex = 0
iSR = 1

J = 0
Open ActiveWorkbook.Path & "\Flag.txt" For Input As #2
Do While Not EOF(2)
    J = J + 1
    For K = 1 To 2
        Input #2, aTemp(K)
        shtRT.Cells(J + 1, 15 + K).Value = aTemp(K)
    Next K
Loop
Close #2

ReDim aTemp(1 To 2)
J = 0
Open ActiveWorkbook.Path & "\SquareFlag.txt" For Input As #3
Do While Not EOF(3)
    J = J + 1
    For K = 1 To 2
        Input #3, aTemp(K)
        shtRT.Cells(J + 1, 18 + K).Value = aTemp(K)
    Next K
Loop
Close #3

ReDim aTemp(1 To 2)
J = 1
iRecordCnt = 1
iIndex = 0
Open ActiveWorkbook.Path & "\CV.txt" For Input As #4
For K = 1 To 2
    Input #4, aTemp(K)
    shtRT.Cells(J + 1, iIndex + K + 3).Value = aTemp(K)
Next K
Do While Not EOF(4)
    iRecordCnt = iRecordCnt + 1
    If (iRecordCnt Mod iSR) = 0 Then
        J = J + 1
        For K = 1 To 2
            Input #4, aTemp(K)
            shtRT.Cells(J + 1, iIndex + K + 3).Value = aTemp(K)
        Next K
    Else
        For K = 1 To 2
            Input #4, aTemp(K)
        Next K
    End If
    If J >= 60000 Then
        J = 0
        iIndex = iIndex + 3
    End If
Loop
Close #4

ReDim aTemp(1 To 2)
J = 0
iIndex = 0
Open ActiveWorkbook.Path & "\SP.txt" For Input As #5
Do While Not EOF(5)
```

```
    J = J + 1
    For K = 1 To 2
        Input #5, aTemp(K)
        shtRT.Cells(J + 1, iIndex + K + 9).Value = aTemp(K)
    Next K
    If J >= 60000 Then
        J = 0
        iIndex = iIndex + 3
    End If
Loop
Close #5

Application.ScreenUpdating = True

End Sub
```

VITA

Thomas Judson Wooters

Candidate for the Degree of

Master of Science

Thesis: AN IMPROVED CONTROL LOOP PERFORMANCE MONITOR

Major Field: Chemical Engineering

Biographical:

Education:
Bachelor of Science Chemical Engineering**,** Dec 2005, Brigham Young University, Provo, Utah. Completed the requirements for the Master of Science in Chemical Engineering at Oklahoma State University, Stillwater, Oklahoma in May, 2008.

Experience:
Employed by Oklahoma State University as a Teaching Assistant for ENSC 3233 – Fluid Mechanics, Fall 2006 and ENGR 1412 – VBA from Spring 2007 to present. Also employed by Oklahoma State University as a Research Assistant from Fall 2006 to present.

Interned at Chevron Phillips Chemical Summer of 2007 in the field of my specialty as a Process Control Engineer.

Professional Memberships:
Member of Omega Chi Epsilon-Oklahoma State University Chapter

Name: Thomas Judson Wooters Date of Degree: July, 2008

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: AN IMPROVED CONTROL LOOP PERFORMANCE MONITOR

Pages in Study: 183 Candidate for the Degree of Master of Science

Major Field: Chemical Engineering

Scope and Method of Study:
Improved Control Loop Monitoring Using Markov Chains and Binomial Statistics

Findings and Conclusions:
A simple technique to detect and flag poor and degrading control loop performance using Markov chains and binomial statistics was developed by Owusu, S (2006). The original work is improved upon in computational efficiency and calculation correctness through updated algorithms. The technique is capable of detecting various control loop problems as demonstrated by simulated and pilot-plant scale experiments.

ADVISER'S APPROVAL: Dr. R. Russell Rhinehart