CONSTRUCTION AND EVALUATION OF MATHEMATICAL MODELS FOR REAL-TIME PROCESS FAULT MONITORING

By

ROMAN ALEXANDROVICH SHAPOVALOV

Diplom
Chemical Cybernetics
Mendeleyev University
Moscow, Russia
1996

Master of Science
Chemical Engineering
Oklahoma State University
Stillwater, Oklahoma
1999

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 2004

CONSTRUCTION AND EVALUATION OF

MATHEMATICAL MODELS FOR

REAL-TIME PROCESS FAULT

MONITORING

Thesis approved:

[illegible]

Thesis Advisor

[illegible]

[illegible]

[illegible]

[illegible]

Dean of Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to my major advisor, Dr. James Robert Whiteley, for his guidance and inspiration of this research. I am also grateful to my parents, brother who supported me in many ways during the course of this research. In addition, I need to acknowledge the valuable suggestions by Drs. Russell Rhinehart, Sharad Bhartia (Indian Institute of Technology Mumbay), James Davis (University of California Los Angeles), James Moore (University of Tennessee), and Christopher Bilder (University of Nebraska Lincoln). Finally, I want to thank ConocoPhillips Company and School of Chemical Engineering of Oklahoma State University for their financial support.

TABLE OF CONTENTS

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

Figure **Page**

Figure **Page**

NOMENCLATURE

α	level of significance
$\boldsymbol{b}_l$	vector of biases for the multi-layer perceptron layer l
$B(X)$	potential net benefit of automating the monitoring of fault X (\$/yr)
$B_d(X)$	benefit of early detection of fault X by a fault model (\$/yr)
B_{dX}	benefit of early detection of a single instance of fault X (\$)
$C_{dm}(X)$	prorated cost of developing model M and running it in real time to monitor fault X (\$/yr)
$C_{fa}(X)$	cost per unit time associated with false warnings of fault X (\$/yr)
$C_m(X)$	cost associated with the possibility of misclassifying other faults as fault X (\$/yr)
C_{nj}	expected cost of the final products that would have been manufactured minus the cost of the inputs consumed during the time span of the fault instance j if the TE process had operated normally during that period (\$)
C_{Rj}	cost of the "*Reagent & product loss plus an incomplete capacity utilization*" for instance j of the "*Reaction Kinetics Drift*" (\$)
CART	classification and regression tree
d	dimensionality of the network inputs
f	frequency number
I	number of iterations
$\boldsymbol{I}$	identity matrix
K	number of classes distinguished by the classifier
K_Z	cost of chemical **Z** (\$/Kgmole)
$l(Y_i)$	time from fault inception to fault detection (detection lag) of instance Y_i of fault Y

L	total effective length of the normal operation intervals selected for model evaluation
L_{bij}	cost of consequence C_i for fault instance X_j at the imaginary time of the automated detection of X_j (\$)
$L_{dm}(X_j)$	total cost of the consequences of fault instance X_j if X_j were detected by the fault model (\$)
$L_{dma}(X_j)$	post-detection consequence cost for fault instance X_j (\$)
$L_{dmb}(X_j)$	pre-detection consequence cost for fault instance X_j (\$)
$L_{du}(X_j)$	total cost of the consequences of fault instance X_j if X_j is detected manually by the operator (\$)
$\overline{L}_{fa}(X)$	average cost of a single false warning of fault X produced by the fault model (\$)
$\overline{L}_{fa}(Y,X)$	average cost of a single misclassification fault Y as fault X (\$)
L_i	length of the i^{th} interval of the normal operation historical data used for model evaluation
L_{tij}	total cost of consequence C_i for fault instance X_j (\$)
M_i	number of incorrectly classified training set prototypes belonging to class i
n_f	number of monitored faults
n_M	number of models identified for the evaluated model design
n_X	total number of historicized and labeled instances of fault X
n_{XN}	number of the instances X_j of fault X for which $L_{du}(X_j) - L_{dm}(X_j)$ is non-negative
N_b	number of basis functions
N_C	total number of possible model component combinations
N_{ds}	number of data smoothing techniques

N_f	number of frequencies for the detection lags or pre-detection consequence costs or the number of feature generators in the model component library
N_i	number of the data vectors (prototypes) in the training set
N_N	number of test intervals selected for model evaluation
N_o	number of optimized criteria
N_S	number of process variables smoothed by the model
N_X	total number of instances of fault *X* observed during the plant lifetime
N_{vsc}	number of variable selection criteria available in the component library
N_{vsm}	number of variable selection methods
N_W	number of monitoring windows that can fit into a one-year interval
N_{XN}	number of the fault instances X_j for which $L_{du}(X_j) - L_{dm}(X_j)$ is non-negative
N_Z	number of times that fault of type *Z* has been observed during the plant history
$O(T)$	order of the amount of time required to train a fault model
p	probability
$P(n)$	probability of event *n*
$P(a\|b)$	probability of event *a* given an occurrence of event *b*
poslin	a function performing the mapping to zero for negative arguments and the identity mapping for the non-negative arguments
PCA	Principal Component Analysis
PNN	Probabilistic Neural Network
r_{base}	base process data sampling rate (4 points/minute) for the problem in Section 4.1.3

$r_{fa}(X)$	frequency of false warnings of fault X produced by a fault model during normal operation of the monitored process
$r_{fa}(Y_i,X)$	frequency of false warnings of fault X produced by the model during the existence of fault instance Y_i
$r_f(X)$	frequency of the occurrences of fault X
R	classifier error rate
$\boldsymbol{R}$	input data correlation matrix
S	Standard deviation
S^2	Variance
SFFS	Sequential Forward Floating Search
t_0	time of fault inception
t_B	time intervals between consecutive model runs or the time required to identify the basis function parameters
t_C	time it takes a fault model to perform a single classification
t_D	time stamp of the latest process measurement used by the model when the fault is detected
t_{DL}	detection lag duration
t_F	time to identify the feature generator parameters
t_P	time to pass data from process sensors to the computer running the fault model
t_{PO}	total length of time the plant has been in operation
t_{RX}	time required for the actions to prevent the consequences of fault X
t_S	time required to identify data smoothing method parameters
t_{SC}	time it takes to pass data from sensors and analytical devices to the computer

t_u	computing time taken by fault model unit u during the model identification
TE	Tennessee-Eastman
W	monitoring window width
$\boldsymbol{W}_l$	matrix of vector multi-layer perceptron weights in layer l
w_{kj}	weight of jth neuron belonging to class k
$\boldsymbol{x}$	input data vector
X	fault X
$\boldsymbol{X}$	input data matrix
$\boldsymbol{y}$	vector of actual values of the modeled variable
Y	fault Y
z	vector of model outputs
Z_α	α^{th} percentile of the standard normal cumulative distribution function
$\boldsymbol{\Delta}$	decision space
λ_i	i^{th} eigenvalue of the input data correlation matrix
μ_j^k	is the center of jth neuron belonging to class k
$\boldsymbol{O}$	space of process data points
$\boldsymbol{O}'$	intermediate measurement space
$\boldsymbol{\Xi}$	feature or residual space
σ	neuron spread parameter for a probabilistic neural network
$\boldsymbol{\Omega}$	space of faults an their root causes

Chapter I

The Need for a Framework for Selecting Optimal Methods for Fault Monitoring

1.1 Importance of Fault Monitoring

Despite the advent of the postindustrial era, the importance of the manufacturing sector in the economy remains high. Besides that, the global competition tightens, the environmental standards become more stringent, and the natural resources get exhausted. As a result, the economic benefits of manufacturing high-quality goods and operating industrial processes correctly grow. At the same time, industrial processes become increasingly more complex, especially in the developed world. It is hard to operate complex systems correctly because of the very large number of interactions and unaccounted factors.

Aside from the well-studied issues linked with the control of complex systems, there is a problem of a timely detection of faults. In this work, **fault** is defined as an undesirable change in the process, process operation outside of the acceptable region. Most faults can be corrected with a fair amount of effort. Faults must be discovered as they appear because they often lead to serious consequences associated with a significant loss in the profitability of the process and the company running the process. Fault consequences can be non-dangerous, but undesirable, such as a substandard process operation reducing the product quality and increasing the use of components per unit product. Alternatively, faults can lead to accidents that pose danger to the health and lives of the plant personnel and the residents of the area where the plants are located.

1.2 Fault Monitoring in the Process Industries

Discovery of faults in real time is the task of fault monitoring in the process industries. Process industries deal primarily with continuous streams of raw materials and products and produce industrial commodities, such as chemicals, petroleum products, metals, foodstuffs, or electricity. To monitor faults in the process industries, the values of the process measurements and product characteristics (process variables) are constantly collected and analyzed for the presence of faults. Fault monitoring in the process industries is a big issue for two reasons.

First, faults in the process industries may potentially cause enormous damage. The two worst technological accidents in history: the Chernobyl′ and Bhopal disasters were caused by faults at a nuclear power plant and at a chemical plant, respectively. It was estimated that in 1994, abnormal situations cost the world's petrochemical industries alone about US$20 billion a year (Nimmo 1995). If the other industrial commodity manufacturers were accounted for, this cost would be at least doubled and would be equal to at least 0.2% of the world's gross domestic product for that year. This figure does not account for the human lives lost and the environment polluted as a result of these abnormal situations. Due to the fear of fault-related accidents, the US has not built a single nuclear plant for 25 years.

Second, many faults in the process industries are hard to detect. This difficulty is due to the highly nonlinear, non-stationary, multivariate nature of chemical, petrochemical, food processing, metallurgical, and power generation processes. Moreover, there is a great variability among how different faults display themselves in the observed process variables. Hence, fault detection in the process industries requires dealing simultaneously with a large number of values of different process variables using different custom approaches. The operators responsible for fault detection continuously analyze current process data and use their expertise to determine if the supervised processes have faults.

1.3 Difficulties in Automating Process Fault Monitoring

The recent advances in the applied mathematics and in computer hardware and software combined with the high-tech frenzy of the late nineties brought about a large number of relatively complex multivariate techniques, such as the advanced multivariate statistical methods, neural nets, nonlinear multiple regressions, classification and regression trees, advanced linear models, etc. These techniques can be potentially used for performing fault monitoring in the process industries. Several thousand research papers, describing potential and actual uses of different advanced multivariate methods for fault monitoring in the process industries, have been published in the last 15 years. However, these proposed methods have found only a very limited application in the area of process fault monitoring for several reasons.

First, the ability of numerical models to monitor each particular user-specified fault in real time is unknown. There is no single universal method for monitoring all possible faults because different processes have different qualitative and quantitative characteristics plus there are different requirements to how the monitoring of each individual fault should be performed. Hence, not only will different methods show different performance for different faults, but there may also be incompatibilities between some faults and mathematical methods for fault monitoring. The superabundance of these methods complicates things further.

Second, there is no well-established procedure for predicting and/or estimating the performance of different methods when monitoring each particular user-specified fault. While some performance measures, such as the detection lags, error rate and others have been proposed, they primarily aim at satisfying common quality and safety standards and do not convey the benefits of automating the process fault monitoring in monetary terms. Without the ability to estimate this measure for different faults and different monitoring methods, the plant management and operation personnel may not be convinced that it makes sense to replace the "good old" manual fault monitoring with a computerized one.

1.4 An Overview of the Solution Proposed for Finding Best Fault Detectors

The widely accepted procedure for selecting models for describing any phenomena consists of four steps:

1) Finding the model types suitable for the
2) Identifying the parameters of the appropriate model designs
3) Evaluating the performance of the resulting models
4) Selecting the best-performing model.

This work proposes an algorithm that performs the four above steps to construct well-performing methods for user-defined fault monitoring problems.

First, the proposed algorithm views all the possible methods for fault monitoring as a combination of structural parts. The proposed algorithm combines these parts into operational methods for fault monitoring using the proposed rules that define the compatibility of the structural parts with the user requirements, monitored process and fault specifics, the properties of the processed data, and other structural parts of the model for fault monitoring. Automation of this algorithm allows searching vast arrays of mathematical methods, including those that have never been described in the technical literature and those that will be developed in the future. This organized search will help find optimal methods for fault monitoring in the cases very common in the process industry where no a priori information is available on which methods are suitable for fault monitoring and which methods will perform well.

Chapter II describes the proposed approach for decomposing any mathematical methods for fault monitoring into structural components, formalizing the properties of these components and user-specified fault monitoring problems, and combining the components into methods applicable to solving the user-specified monitoring problems. Essentially, Chapter II explains how to perform the first step of the model selection sequence for the case of fault monitoring.

Second, a procedure for estimating the performance of different fault monitoring methods in monetary terms is proposed. The proposed statistical estimator indicates the monetary return on the investment in the automation of monitoring a particular fault using different fault models. This monetary return is evaluated as an expectation of the reduction in the risk of fault consequences that will be achieved by the automation of fault monitoring. This reduction is corrected by the risk of false alarms. Chapter III describes the proposed metric in detail.

Chapter IV summarizes the proposed algorithm and illustrates its application in practice. Every step of the algorithm is described in great detail and the illustration uses a commonly known simulation of a chemical process available in the public domain. Although the proposed algorithm has not been computer-coded yet, the illustration suggests how to implement a piece of software for selecting methods for fault monitoring using an operator-friendly interface.

This algorithm overcomes the problems outlined in the previous section, which have been limiting the automation of fault monitoring in the process industry. To the knowledge of the author, the proposed algorithm is the first one that evaluates the return on the investments in automating fault monitoring in the process (continuous-stream) industry. In addition, this proposed approach might help commercialize many previously developed methods for fault monitoring that never made it to the industry.

Chapter II

Constructing Fault Models from Structural Components

Section 2.1 of this chapter discusses faults in the process industries, methods to prevent faults, and the role and functions of automated (computerized) model-based monitoring of process faults. Section 2.2 describes a functional hierarchy of various methods for fault monitoring presented in the technical literature. Section 2.3 proposes a template that can be used to build mathematical models for fault monitoring out of structural components. Section 2.4 proposes the rules that determine how fault models can be created from the proposed template for monitoring different user-specified faults.

2.1 Faults and the Role of Automated Fault Monitoring in the Process Industries

2.1.1 Types of Faults in the Process Industries

In the process-engineering vernacular, process faults (or abnormal situations) mean an incorrect operation of the process. All the processes are designed to operate without faults. However, from time to time, some components of a process may **fail**, i.e., lose the ability to perform their functions correctly. For example, a control valve may be damaged and thus lose the ability to entirely block the flow of the fluid through it. As a result, the process controlled by this valve may not be operated correctly. Thus, each process fault has a **root cause**, i.e., a **failure** that causes the fault. Once a fault occurs, it is important to find and correct the failures that are the root causes of the fault as soon as possible.

Process faults are divided into classes according to the type of root causes, fault onset, and the risk. A recent survey on fault monitoring (Venkatasubramanian, Rengaswamy et al. 2003c) enumerates several types of process faults (abnormal situations) according to their root causes. The first type of root cause is comprised by the external causes, such as changes in the feeds or ambient conditions. Failures in the control hardware (sensors, actuators, control valves, communication lines, and control computers) make up the root cause of the second type. The third type of root cause is made up of the rest of the process hardware, such as pipes, reactors, separators, etc. Faults with the second type of root cause are the hardest to detect as they affect the equipment delivering the process data that can be used for fault detection.

Faults can be gradual or abrupt. Gradual faults may arise as a result of the tear and wear of the process equipment or as a result of a gradual change in the ambient conditions or process feeds. An example of a gradual fault can be a slowly accumulating fouling of a pipe resulting in a gradual increase of the head loss across the pipe. Abrupt faults are the results of catastrophic changes in the process equipment or process feeds. Control valve sticking is an example of an abrupt fault.

It is important to note that the impact (or effect) of a fault on the process generally increases with time. For abrupt faults, this impact may first drop and then grow again as shown in Figure 2.1. Hence, there is frequently a substantial time lag between process fault occurrence and fault discovery.

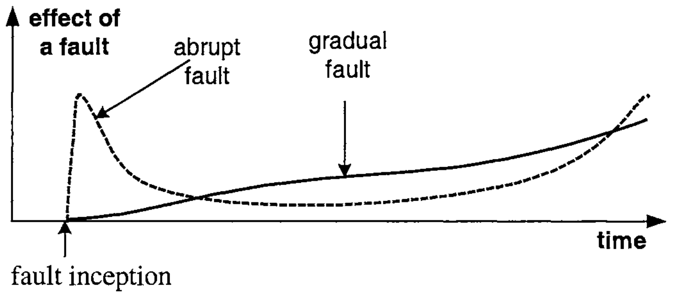


Figure 2.1. The effect of faults on the process variables

2.1.2 Process Safety Management and Automation of Fault Monitoring

To reduce the risk associated with process faults and their consequences, each processing plant should have a safety management program. Several approaches to safety management, such as what-if, interaction analysis, zonal analysis, checklist, Failure Mode Effect Analysis, and

HAZOP (Steinbach 1999) have been proposed to date. There are various software tools that automate safety management in the process industries. These tools are expert systems HAZOPExpert that model the circumstances of faults (Srinivasan, Dimitriadis et al. 1998), relational databases filled with reactivity and hazard information (Al-Qurashi, Sharma et al. 2001), and "data farms" containing taxonomically organized fault histories recorded at various plants (Thomas and Moosemiller 2001).

There are four basic steps that must be taken to reduce the risk associated with faults and their consequences at a process facility:

1) Improvements in plant design, such as installation of additional sensors and fault-proof process equipment
2) Safety training of the personnel
3) Scheduled preventive maintenance
4) Real-time management of faults as they occur.

There are government-issued regulations and recommendations on how to implement the safety measures that reduce the risk of faults and their consequences (CCPS 2000c). Despite that fact, there are some degrees of freedom in the implementation of the above safety program. In particular, human operators can perform the real-time fault management or it can be automated. Automation of fault management can lower the risk associated with process faults and their consequences. As was outlined by Isermann (Isermann 1997), an automated management of faults in a process should consist of at least five stages, as shown in Figure 2.2:

1) **Fault modeling:** determining the relationships between the process variables and faults;
2) **Fault detection:** using a fault model to determine in real time if a process operates inside or outside of the applicable appropriate region;

3) **Fault diagnosis and evaluation:** using a fault model to determine the origin of the fault that causes the process to operate outside the appropriate region and the extent of the fault;
4) **Deciding** on what to do to rectify the fault;
5) **Fault Rectification:** acting to reduce the negative effect of the diagnosed fault on the process and returning the process to the normal operation.

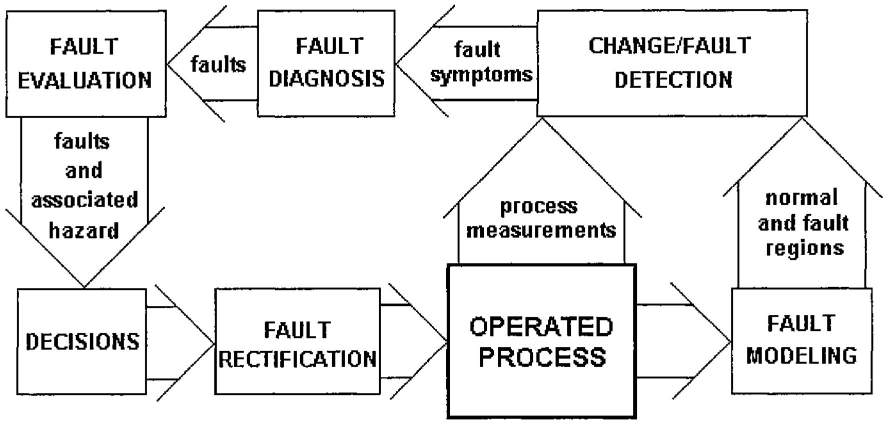


Figure 2.2. Stages in the Automated Process Safety Management (Isermann 1997)

2.1.3 The Concept and Functions of a Fault Model

Fault models are mathematical and logical relationships between the process variables and faults possible in the process. These models serve for fault detection (Step 2 of the automated fault management process). Some fault models also perform fault diagnosis by finding the root causes of process faults (Step 3 of the automated fault management process). Fault models effectively solve pattern recognition event-monitoring problems. Event monitoring is the type of pattern recognition that consists in identifying the occurrence of certain events in real time.

To perform fault detection, the monitored process is divided into process blocks that may be entire process plants, process units, or particular streams and pieces of control equipment. Each block has one or, sometimes, several modes of abnormal operation known as "fault states." For each process block, the latest process data collected over a fixed-length time period called **monitoring window (grab size)** are input into a fault model. All the process data falling into this window at a given time instance make up a **process data vector**. This vector consists of the sequentially historicized **process data points** caught inside the monitoring window. Each process data point contains the values of the process variables recorded by the process data historian at a given point in time. The constant time interval between acquisitions of adjacent process data points is called **sampling period** or **sampling interval.**

A fault model finds out from the process data vector that includes the latest observed process data point if the monitored process is in the region corresponding to the normal operation of the process or in the region corresponding to a certain fault. Fault models perform the mappings from the space $\boldsymbol{O}$ of process data points to the space $\boldsymbol{\Omega}$ of faults.

$$\zeta : O \rightarrow \Omega \tag{2.1}$$

These mappings imitate the actual relationships between $\boldsymbol{O}$ and $\boldsymbol{\Omega}$ for a specific fault. To create these imitations, fault model parameters are identified by using either the **training sets**: the sets of historical data recorded (historicized) during the occurrences of the monitored faults as well as normal operation of the monitored process or by using the first principles. This proposed definition of a fault model is broader than the conventional one. This definition takes account all the stages of the transformation from the sensor readings to the prediction of faults and not just the mapping of process data vectors onto meaningful descriptions of the monitored process states.

The concept of automated fault monitoring is schematically shown in Figure 2.3. If the process data vector is in the faulty region, the process variables whose current values are least likely for the normal operation of the process are singled out. Then a logical fault-diagnosing model, such as a fault tree (CCPS 2000b), is run to name the possible root causes of the detected fault. The use of the shared equipment reliability databases together with statistical classifiers (CCPS 1998) helps in narrowing down the list of possible root causes. After that, a manual check and replacement of faulty equipment and/or correction of external root causes are performed.

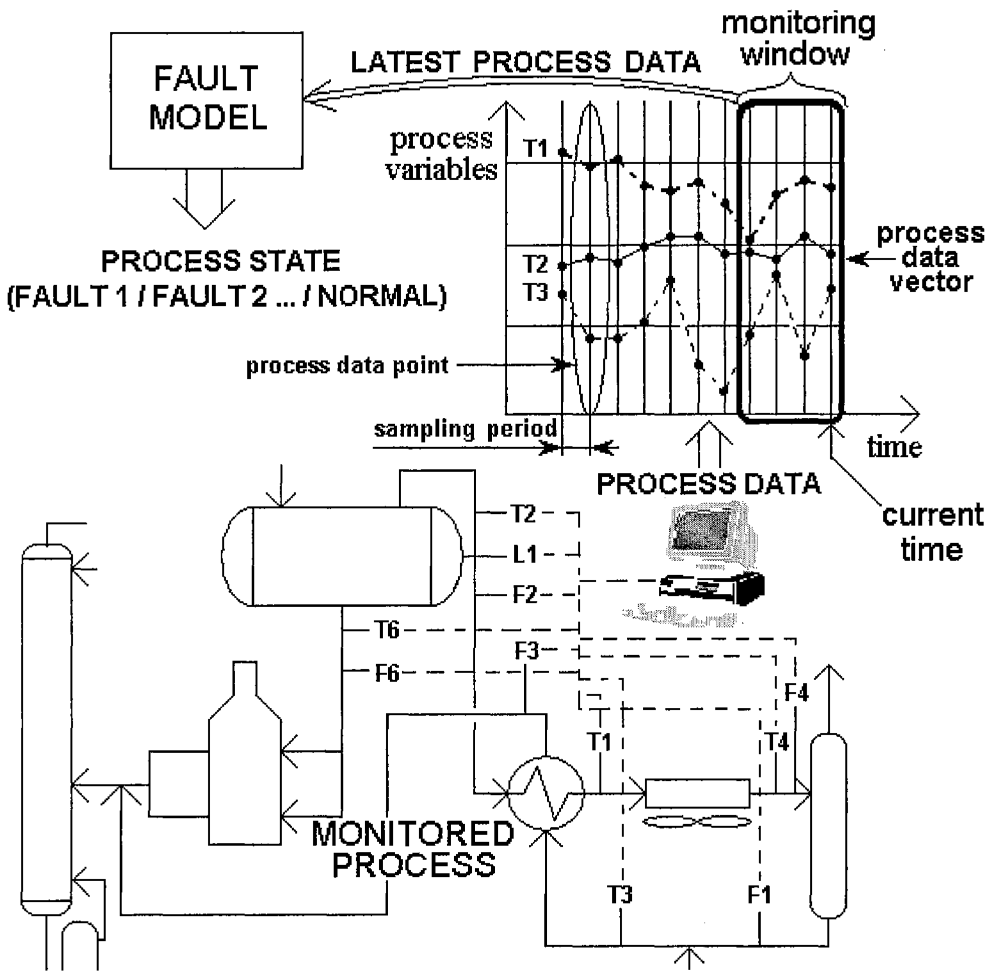


Figure 2.3. Generic flowchart of process fault monitoring

There exists a vast array of human knowledge on how to get to the root causes of faults. Many activities included in fault diagnosis have to be performed manually. Hence, this work primarily focuses on automated real-time fault detection that **may** also involve partial automation of fault diagnosis. It is important not to confuse fault monitoring with fault detection and fault diagnosis.

2.2 Implementations of Fault Models as Combinations of Different Methods

2.2.1 Fault Models as Hierarchies of Methods for Fault Monitoring

As was mentioned in Chapter I, there can be no method universally good for monitoring all the possible process faults. A plethora of different numerical and logical methods to model process faults have been proposed to date. This work on process fault modeling has been summarized in the multiple surveys on methods for process fault detection and diagnosis (Chiang, Russell et al. 2001b; Davis, Piovoso et al. 1999; Frank, Ding et al. 2000; Frank and Ding 1997; Isermann 1997; Isermann and Balle 1997; Kramer and Mah 1993; Leonardt and Ayoubi 1997; MacGregor and Kourti 1995; Phatak and Sparks 1995; Venkatasubramanian, Rengaswamy et al. 2003a; Venkatasubramanian, Rengaswamy et al. 2003b; Venkatasubramanian, Rengaswamy et al. 2003c; and Wise and Gallagher 1996).

Most of these surveys primarily focus on a particular class or a set of classes of methods for fault detection and diagnosis where the authors are the experts. In addition, these surveys do not show a distinction between fault models that help imitate faulty operation of industrial processes and methods for automated fault detection and diagnosis performing specific tasks in this imitation. It is important to show what these tasks are and present a generic fault model as a combination of different methods for fault detection and diagnosis.

Process data typically consist of a very large number of continuous readings of many different process variables distorted by noise and random fluctuations due to sensor and process noise. In addition, the process control system is set to smooth out all the abnormal deviations in the process variables, including the deviations caused by faults. Therefore, the relationships between process variables and fault states are often very complex and a direct real-time mapping of the process variables into all the possible faults with the root causes of these faults is a task too challenging for a single method for fault modeling (Venkatasubramanian, Rengaswamy et al. 2003b). This is why a fault model is usually a combination of different numerical and logical methods, even though the presentations of fault models in the technical literature usually focus only on the key and most complex component of the proposed fault model and give little or no attention to other components of the model. The general strategy of processing data by fault models consists in trimming and de-noising a potentially boundless pool of process data, creating a small number of indicators sensitive to faults, and mapping those few indicators onto a set of process states.

There are a number of methods, with a long application history in various fields of human knowledge, which can perform each of these tasks separately. Each method essentially performs a part or a step of the mapping in Equation 2.1. Some methods map the space $\boldsymbol{O}$ of process measurements onto an intermediate measurement space $\boldsymbol{O}'$ that either has fewer measurements than $\boldsymbol{O}$ or contains measurements at least partially cleared of noise and outliers, or both. Other methods map $\boldsymbol{O}$ or $\boldsymbol{O}'$ onto a feature or residual space $\boldsymbol{\Xi}$ consisting of a few derived variables that are not trends of the monitored variables. The variables spanning $\boldsymbol{\Xi}$ are created to be very sensitive to the monitored faults. Some other methods map the feature space $\boldsymbol{\Xi}$ or the measurement spaces ($\boldsymbol{O}$ or $\boldsymbol{O}'$) onto the symbolic decision space $\boldsymbol{\Delta}$. The decision space spans the monitored faults or qualitatively described fault components. Finally, methods for fault diagnosis map the symbolic decision space $\boldsymbol{\Delta}$ onto the space $\boldsymbol{\Omega}$ of the root causes of faults. Figure 2.4 presents possible mapping types performed by the methods for fault detection and diagnosis. These mappings are shown as numbered arrows between the rectangles denoting different spaces. The remainder of Section 2.2 discusses different methods that perform each type of these mappings and proposes a generic template for designing fault models out of the methods for fault detection and diagnosis described in the technical literature.

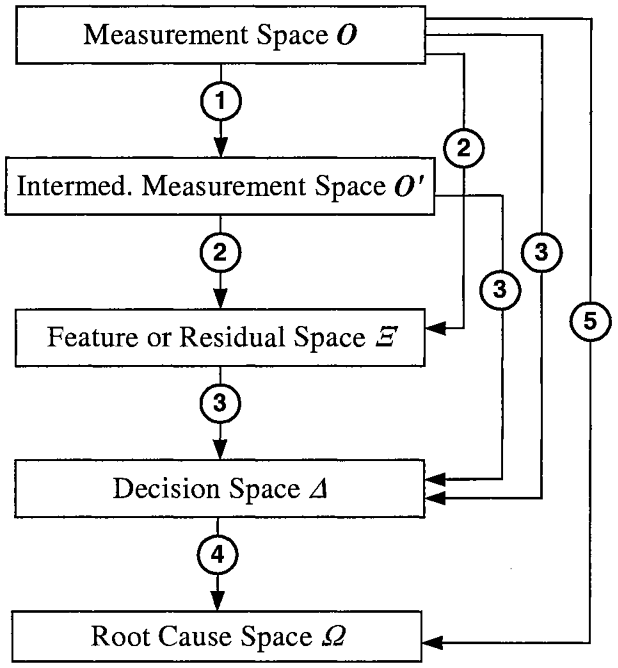

Measurement Space $\boldsymbol{O}$
Intermed. Measurement Space $\boldsymbol{O'}$
Feature or Residual Space Ξ
Decision Space Δ
Root Cause Space Ω
1
2
2
3
3
3
4
5

Figure 2.4. Mappings that can be performed by the methods for fault detection and diagnosis

2.2.2 Methods for the Mapping between the Measurement Spaces ($O \rightarrow O'$Mapping)

This type of mapping is marked '①' in Figure 2.4. Methods performing mappings between the measurements spaces are further divided into:

1) Methods for the selection of the relevant process variables
2) Methods for noise reduction and removal of outliers.

These methods are very important, but, for some reason, they were given very little attention in the surveys on fault detection and diagnosis.

2.2.2.1 Methods for the Selection of Relevant Process Variables. Supervised selection of the process variables most relevant to the monitored faults should be performed when there are too many process variables in the monitored block and when a first-principle process model is unknown (see Section 3.2.4). Variable selection can be performed in two general ways. One way to do variable selection is to identify a crude relationship between all the process variables and faults and see what variables have the greatest weights in this relationship. The reported schemes of the weight-based selection of process variables for process fault detection and diagnosis used a classification and regression tree (Mastrangelo and Porter 1998) and a neural network (Lezanski 2001) to create this crude relationship between the process variables and the fault.

Another way to select input variables is to use a discrete search technique, such as forward selection backward elimination, or a genetic algorithm, that finds a set of input variables that optimizes a certain criterion indicating the amount of information about the monitored faults contained in the selected process variables. Reported process fault detection applications optimize the mean distance between historical data points belonging to different process modes (Oyeleye and Lehtihet 1998) and the Kullback-Leibler discrimination information (Sadegh, Madsen et al. 1995).

2.2.2.2 Methods for Noise Reduction and Outlier Removal. There are three approaches to reduce the noise and remove the outliers and biases from the process data: traditional time-series analysis, direct removal of outliers, and data reconciliation.

The traditional time-series analyses are performed using linear or nonlinear filters that de-noise trends of a single process variable. Filters map the process measurements to maximize the retention of the relevant filter-dependent process information while minimizing the noise content. Process monitoring applications of linear filters, e.g. of the conventional finite impulse response filter (Davis, Piovoso et al. 1999) or the Kalman Filter (Qiu, Wen et al. 1995) use certain linear transformations of process measurements over a certain time horizon.

Linear filtering is not demanding to the computational resources of the computer. However, the nonlinear filters are more common in the reported process monitoring applications, especially the wavelet transform, because they are better suited to model the nonlinear nature of the process systems. Since classical continuous wavelet transform is significantly more computationally expensive than the linear filters, several applications of the faster discrete wavelet transform have been reported, e.g., (Aretakis and Mathioudakis 1996; Koh, Shi et al. 1999; and Shao, Jia et al. 1999). The coefficients of the discrete wavelet transform can be further refined using different techniques, e.g. the commonly known Principal Component Analysis as was suggested by (Bakshi 1998), before being put in use to de-noise the trends of the monitored variables.

Data reconciliation consists in correcting the biased values of the process variables. Data reconciliation maps the current measurements of the process variables into the measurements that simultaneously minimize the deviation of the new variables from the measured readings and the errors in the conservation laws that apply to the monitored process system (Crowe 1996).

The removal of outliers is performed by statistical tests for a trend in the measurements of the process variables. A sharp increase in the value of a monitored variable without an upward trend for this variable indicates an outlier that should be removed (Tham and Parr 1994).

2.2.3 Methods for the Mapping between the Measurement and Feature (Residual) Spaces ($O \rightarrow \Xi$ and $O' \rightarrow \Xi$ Mappings)

This type of mapping is marked '②' in Figure 2.4. Methods performing the mapping of the "$O \rightarrow \Xi$' type can be divided into those based and those not based on a model of the monitored process.

2.2.3.1 Residual-Generating Methods Based on the Monitored Process Model. Methods based on a model of the monitored process use process models that express relations between the variables of the monitored process. The argument variables are "inputs" and the dependent variables are outputs. These methods calculate model residuals: the differences between the outputs of these models and the actual process variables predicted by the model. The applicable models can be first-principle-based, linear, or nonlinear ones. A simplistic structure of the residual generating model-based methods for fault detection attributed to Frank and his colleagues (Frank, Ding et al. 2000) is shown schematically in Figure 2.5 below.

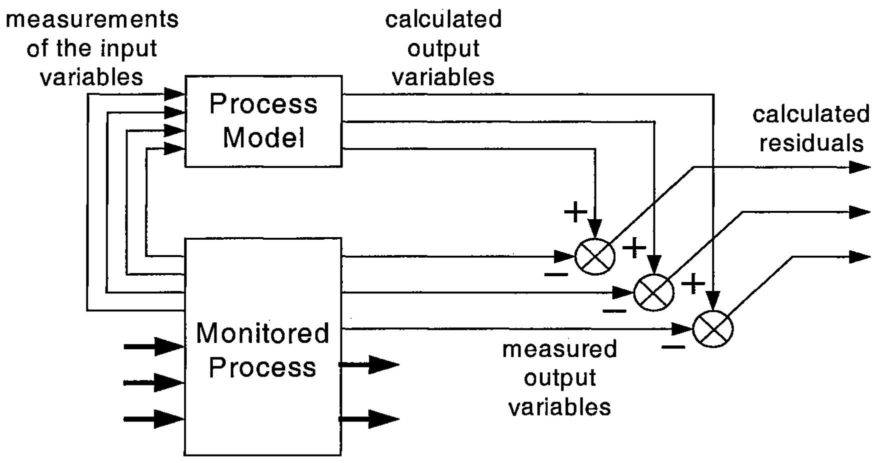

measurements of the input variables
Process Model
calculated output variables
calculated residuals
Monitored Process
measured output variables
+
−
+
−
+
−

Figure 2.5. A simplistic structure of the residual generating model-based methods for fault detection

The use of first principle-based models for residual generation is justified only for easy-to-model process systems. For these systems, detailed first-principle models of each process device are created and combined to simulate the monitored process system. The model runs in real time, along with the monitored process, and outputs the residuals: differences between the actual and simulated values of the measured process variables. Modeling of process systems for fault detection and diagnosis has been implemented in various pieces of software a long time ago, e.g., (Kelly and Lees 1986).

Generation of residuals for fault detection using a data-driven linear dynamic model of the monitored process is also a well-researched mature area. The reason is that linear models are used for fault detection in the dynamic systems with a small number of process variables, such as machinery and electronics. The conventional methods for generating residuals from a linear process model are observers (Clark 1979) and parity relations (Willsky 1976). Newer methods for residual generation using linear models include the generation of directional residuals (Gertler and Monajemy 1995) and the generation of structured residuals (Gertler, Fang et al. 1990). Kavuri and Venkatasubramanian reported an application of an observer to monitoring a chemical process (Kavuri and Venkatasubramanian 1992).

The downside of these methods is that the number of residuals they generate is equal to the minimal number of the variables necessary to identify the state of the monitored process uniquely. In the process industries, the number of such variables can be very large, so the number of residuals will also be large, which is inappropriate for further mapping of these residuals to the space of the monitored process modes. As a result, it makes sense to model relationships not between the "input" and "output" process variables, but between the linear combinations of these variables. The method called Projection to the Latent Structures (PLS) creates a linear relationship between a few best-correlated linear combinations of the "input" variables and a few linear combinations of the "output" variables (Chiang, Russell et al. 2001c). Similar to the PLS is the Canonical Variate Analysis (CVA) that creates the linear combinations of the input and output variables to maximize the covariances between these combinations (Chiang, Russell et al. 2001a). If the number of "output" variables is small, they are used in the PLS regression as they are and this method is called Principal Component Regression. Advanced multi-block versions of the projection to the latent structures are especially well suited for the monitoring of batch processes (Kourti, Nomikos et al. 1995). The Multi-block PLS adds another dimension to the process data, indicating the batch number, before decreasing the number of the "input" and "output" variables.

Monitored processes can also be modeled using nonlinear relationships, such as neural networks. Several neural network-based approaches to modeling the monitored processes have been proposed, e.g., (Tsai and Chang 1995).

2.2.3.2 Residual-Generating Methods not Based on the Monitored Process Model. Both linear and nonlinear non-model based methods for residual generation can be used in fault detection and diagnosis. The linear methods are based on the linear transformations of the original process variables into their uncorrelated principal components that explain most of the variance in the original process variables (Jackson 1991). Numerous variations of this method have been reported applicable to fault detection and diagnosis, e.g., the Principle Component Analysis or PCA, (Dunia, Qin et al. 1996), Multi-Way PCA (Nomikos and MacGregor 1994), Dynamic PCA or DPCA (Luo, Misra et al. 1999), and Independent Component Analysis or ICA (Hyvarinen and Oja 2000). Essentially, all these methods are similar to PCA modified to process input variables in steps or/and in blocks. The Multi-Way PCA, developed specifically for batch processes, adds another dimension, batch number, to the process data. The DPCA augments the value of each input variable with the time-delayed values of this variable. The ICA is a more general form of PCA. The ICA is a linear transformation of input variables into a set of components that are not only uncorrelated, but are statistically independent in other ways specified by the user. Essentially, ICA is a large family of methods.

There are a number of nonlinear methods for generating residuals for fault detection and diagnosis. These residuals can the coefficients of the wavelet transform methods discussed in Subsection 2.2.2.2. However, this is not a very good idea because the residuals generated this way do not take advantage of the interaction between the process variables. Other nonlinear methods group together the process data points located close to each other. Reported applications of these methods to fault detection and diagnosis include the *k*-nearest-neighbor and *k*-nearest prototype clustering algorithms (Guglielmi, Parisini et al. 1995) and the Self-Organizing Feature Map

(Fantoni and Mazzola 1996). Similar to these two methods is an application of the nonlinear PCA to generate residuals out of the process variables (Kramer 1991). In the nonlinear PCA, the inputs are nonlinearly mapped into a small number of variables used as the features. To train the nonlinear mapping, the features are mapped back onto the process variables. The nonlinear mapping is trained to minimize the difference between the values of the process variables used as the inputs and the values of the process variables mapped from the features.

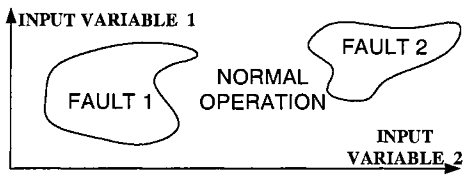
2.2.4 Methods for the Mappings from the Space of Process Measurements onto the Decision Space ($\Xi \rightarrow \Delta$, $O \rightarrow \Delta$, and $O' \rightarrow \Delta$)

This type of mapping is marked '③' in Figure 2.4. Two major classes of methods performing this type of mapping: trend analyzers and neural networks are used in fault detection and diagnosis.

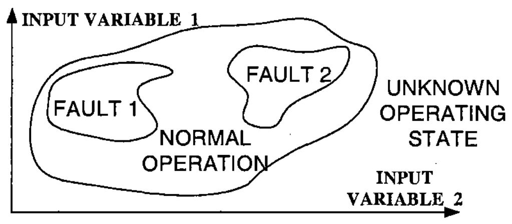
Different methods performing this mapping can partition the input space corresponding to the monitored faults and the normal operating state into bounded (local) or unbounded (global) sectors. As shown in Figure 2.6, there may be four possible situations of how the input space sectors corresponding to different operating states of the monitored process can be bounded or unbounded.

2.2.4.1 Trend Analyzers. Trend analyzers perform a mapping from the space of trended variables, such as process variables or residuals. The most common trend analyzers are control charts.

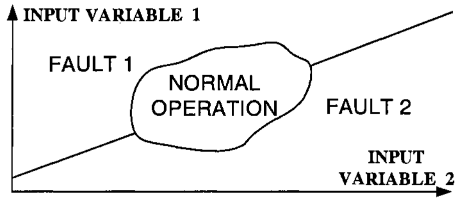
Control charts are supervised data-driven methods. Control charts (Xie, Goh et al. 2002) process sequences of the monitored process variables (or features/residuals extracted from these variables) sampled over time. The control charts used in the process fault monitoring are, in essence, two-sided statistical hypotheses tests for the consistency in the statistical characteristics of the process variables.



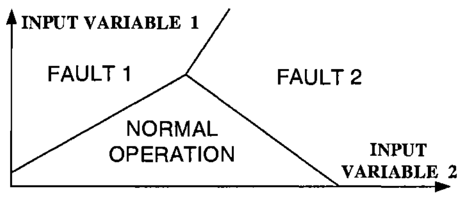
TYPE 1. BOUNDED FOR FAULTS, UNBOUDED FOR NORMAL OPERATION



TYPE 2. BOUNDED FOR BOTH FAULTS AND NORMAL OPERATION



TYPE 3. BOUNDED FOR NORMAL OPERATION, UNBOUNDED FOR FAULTS



TYPE 4. UNBOUNDED FOR BOTH FAULTS AND NORMAL OPERATION

Figure 2.6. Types of partitioning the input space into classes

For each input variable, univariate control charts test the null hypotheses that the variable is not significantly shifted from its median value (X and $\overline{X}$ charts) and/or whether its variance does not significantly exceed the variance observed during the normal process operation (R_m, R, and s charts). Thus, control charts map input variable values to a decision space consisting of the categorical parameters indicating if each of the input variables is consistent with the normal operation of the monitored process or if there is a certain type of inconsistency, e.g., the variable value is too high or the variable's variance is too large.

To diminish the errors caused by the outliers and noise, control charts may pre-filter the trends of the input variables. Filtering the trends by using a moving average filter makes a Cumulative Sum (CUSUM) control chart (Woodward and Goldsmith 1964) and an Exponentially Weighted Moving Average filtering makes an EWMA control chart (Roberts 1959). Multivariate control charts have the capability to test for inconsistencies in a combination of input variables. The use of multivariate analogs of the X-chart: the Hotelling Chart (Phatak and Sparks 1995), CUSUM chart (Healy 1987), and EWMA chart (Lowry, Woodall et al. 1992) is reported in the technical literature. The above-mentioned control charts perform the Type 3 partitioning of the space of the input variables.

More sophisticated control charts test several statistical hypotheses. Each of these hypotheses corresponds to a specific monitored mode of the process (Al-Ghanim and Jordan 1996Raich and Cinar 1996). A similar idea is behind the proposed testing of sensor faults based on the so-called "sensor validity index" that uses test the hypotheses that the current measurements have been sampled from a hypothetical distribution describing a sensor failure (Dunia, Qin et al. 1996). More sophisticated reported approaches to statistical fault identification involve using angular discriminants based on Machalanobis angles between trend points (Raich and Cinar 1997).

The qualitative trend analysis (QTA) is an alternative to the control charts based on "common" first principles. This approach serves for generating qualitative information describing faults. QTA is generally not suitable for the $\Xi \rightarrow \Delta$ mappings as it takes advantage of the information encoded in the process variable trends. This technique analyzes process variables and residuals and finds certain patterns, called primitives, specific for the monitored faults. Each process variable trend can be characterized by a combination of these primitives (Janusz and Venkatasubramanian 1991). Thus, the variables spanning the decision space would be sets or meaningful combinations of such primitives for each process variable. Recent papers on the application of the QTA to the process fault detection and diagnosis report extraction of primitives using neural networks (Rengaswamy and Venkatasubramanian 1995), wavelets (Vedam and Venkatasubramanian 1997), dynamic time warping techniques (Kassidas, Taylor et al. 1998), and the dyadic B-Splines (Vedam, Venkatasubramanian et al. 1998).

2.2.4.2 Neural Networks. Neural networks are combinations of multiple transforms of the same type or of few similar types performed in parallel and sequentially. A great number of neural-network-based applications for fault detection and diagnosis have been reported. The simplest neural networks for modeling process faults are feed-forward multiplayer perceptrons (Hoskins and Himmelblau 1991), a Hamming network (Marcu and Mirea 1997), a Group Methods of Data Handling (Korbicz and Kus 1998), and a Learning Vector Quantization network (Wang and Xu 1996).

Most other neural networks applicable for modeling faults in the process industries are either kernel-based or Adaptive Resonance Theory-based. The kernel-based networks create several kernels that span sectors of adjustable shape and size that cover the area of the input space corresponding to a fault. Applications of a number of different types of a kernel-based neural network have been reported for fault detection. These applications include the networks that create round Gaussian kernels (Tzafestas and Dalianis 1996), elliptical Gaussian kernels (Kavuri and Venkatasubramanian 1993), and non-orthogonal wavelet-sigmoid kernels (Zhao, Chen et al. 1998). The Adaptive Resonance Theory-based networks create densely packed hyper-spherical shapes that cover the input space corresponding to a fault. Reported applications of these networks in process fault detection used the ART2 (Whiteley and Davis 1994), a cascade of

ART2 networks (Yamashita, Komori et al. 1999), the Fuzzy ART (Roehl 1995), and the Fuzzy ARTMAP (Aldrich, Moolman et al. 1995).

2.2.5 Methods for the Mapping onto the Monitored Mode Space ($\Delta \rightarrow \Omega$, $O \rightarrow \Omega$, and $O' \rightarrow \Omega$ Mappings)

These mappings are marked '④' and '⑤' in Figure 2.4. To perform this type of mapping for fault diagnosis, two basic approaches have been devised: tables and semantic networks. A table is a one-level symbolic mapping. A fault-diagnostic table lists all possible diagnostic symptoms, including the detected faults, against possible root causes of these faults. Semantic networks consist of nodes representing process entities (units and concepts) and relationships between these nodes. Three basic types of semantic network are used for fault diagnosis: digraphs, fault trees, and event trees.

Diagraphs (Kramer and Palowitch 1987) are process-specific, first-principle-based graphical methods that generally perform the mapping from the space of measurements onto the space of root causes. To create a digraph, the user should create a mathematical and logical first principle model of the monitored process. Then he or she should create nodes out of the important variables of the model, faults and possible root causes of the faults. After that, the user should connect the nodes with arcs representing the known relationships between these variables. Use of digraphs in fault diagnosis in a whipped topping manufacturing process is illustrated in (Rich and Venkatasubramanian 1987).

Fault trees (CCPS 2000b) are similar to digraphs. The difference is that a digraph processes quantitative information and converts it to the qualitative one. A fault tree connects qualitative symptoms, faults, and their root causes with logical relationships (fault trees perform the $\Delta \rightarrow \Omega$ mapping). Figure 2.7 shows an example of a fault tree (Davis, Piovoso et al. 1999).

This fault tree connects a fault located at the top node "High Catalyst Loss in the Reactor," fault symptoms located at the bottom nodes and possible root causes of a fault: "Cyclone Damage," "Cyclone/Dipleg Damage," and "Hole in Reactor Plenum." Fault tree nodes are connected with "and" and "or" arcs. Thus, the "High Catalyst Loss in the Reactor" can be caused by any of the root causes, whereas the "Hole in Reactor Plenum" would require that the "Rates of Loss Increasing" and "Fines Content Decreased Slightly" occur simultaneously.

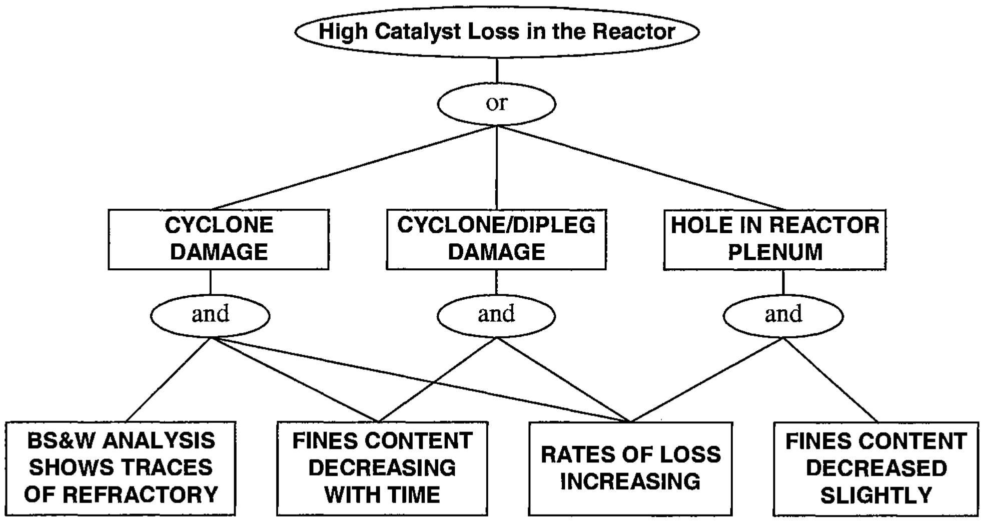

High Catalyst Loss in the Reactor
or
CYCLONE DAMAGE
CYCLONE/DIPLEG DAMAGE
HOLE IN REACTOR PLENUM
and
and
and
BS&W ANALYSIS SHOWS TRACES OF REFRACTORY
FINES CONTENT DECREASING WITH TIME
RATES OF LOSS INCREASING
FINES CONTENT DECREASED SLIGHTLY

Figure 2.7. A fault tree for high catalyst loss in a reactor (Davis, Piovoso et al. 1999)

Event trees (CCPS 2000a) are graphical logical models that identify possible outcomes following an initiating event along with the probabilities of those outcomes. Just like fault trees, event trees connect qualitative symptoms, faults, and their root causes. Figure 2.8 shows a scheme of a reactor that can have coolant failures during which the coolant is no longer supplied to the reactor. Figure 2.9 shows an event tree connecting a coolant failure and an accident, a runaway reaction that can result from this failure for this reactor. In the case of a coolant failure, a runaway reaction can be prevented if the reaction temperature alarm or the coolant flow alarm or both are operational at the time of a failure. Once the coolant is no longer supplied to the reactor, either the reaction temperature alarm or the coolant flow alarm goes on. As a result, the reactor dump valve is opened and the reagents drained from the reactor. The event tree in Figure 2.10 illustrates how different conditions of the alarms and the dump valve can lead to a safe shutdown of the reactor or to a runaway reaction in the absence of a fault model that would monitor the reactor coolant failures. The probabilities of these two outcomes are calculated using the probabilities of the conditions shown in the event tree.

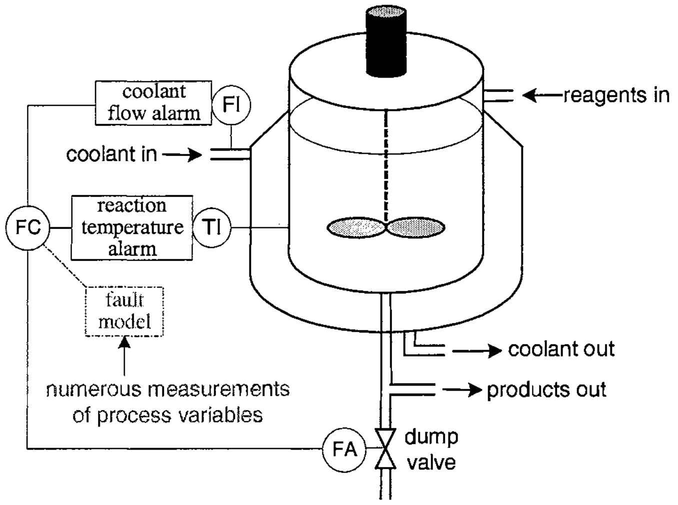


Figure 2.8. A simplified scheme of monitoring reactor coolant failure (CCPS 2000a)

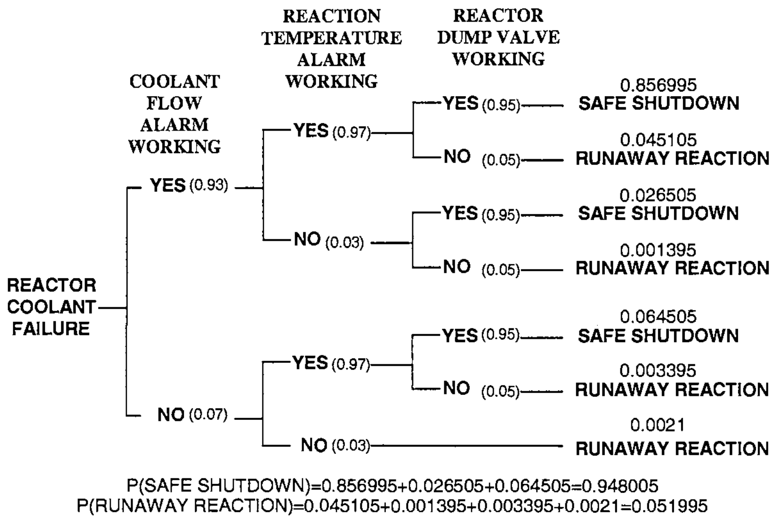

REACTION TEMPERATURE ALARM WORKING
REACTOR DUMP VALVE WORKING
COOLANT FLOW ALARM WORKING
REACTOR COOLANT FAILURE
YES (0.93)
NO (0.07)
YES (0.97)
NO (0.03)
YES (0.95)
NO (0.05)
0.856995 SAFE SHUTDOWN
0.045105 RUNAWAY REACTION
0.026505 SAFE SHUTDOWN
0.001395 RUNAWAY REACTION
0.064505 SAFE SHUTDOWN
0.003395 RUNAWAY REACTION
0.0021 RUNAWAY REACTION
P(SAFE SHUTDOWN)=0.856995+0.026505+0.064505=0.948005
P(RUNAWAY REACTION)=0.045105+0.001395+0.003395+0.0021=0.051995

Figure 2.9. An event tree for a reactor coolant failure (CCPS 2000a)

2.3 The Proposed Fault Model Template

2.3.1 General Structure of the Template

As can be seen from the previous section (Section 2.2), design and selection of mathematical models for monitoring a specific process fault naturally yields itself to:

1) Combining different methods for fault monitoring into a model structure that maps data from the measurement space onto the root cause space;
2) Identifying the parameters of the designed model;
3) Evaluating the model performance.

As was discussed before, finding fault root causes in real time (i.e., performing both fault detection and full fault diagnosis) using fault models alone is too challenging a task. Therefore, in this work, it makes more sense to limit the concept of a fault-monitoring model to a set of mathematical methods that map data from the space of monitored process variables onto the decision space that may include only fault states and a "no fault" state even though there are methods, described in Subsection 2.2.5, which perform identification of fault root causes. As follows from the previous section (Section 2.2) fault models can be represented as a sequence of steps, each step pursuing a specific goal.

At the first step, **Input Variable Selection**, the first part of the mapping marked '①' in Figure 2.4, the set of model input variables is formed out of the set of all the process variables that the operator thinks may be associated with the fault for which the model is being created. At the second step, the second part of the mapping marked '①', **Data Smoothing**, the process data are cleared of noise and outliers. At the third step, **Feature Generation**, the mapping marked '②', a few inputs that concentrate fault-sensitive information, are generated. At the fourth step, **Classification (Interpretation)**, the mapping marked '③', the fault model determines whether there is a fault in the monitored process. A similar generic fault model structure consisting of the noise reduction, residual generation, and classification steps has been proposed previously (Davis, Piovoso et al. 1999), however it is not generic enough to describe all the models that have been used and can be potentially applied in process fault monitoring.

Figure 2.10 below is a graphical illustration of the data processing steps in a fault model with examples of the data processing techniques that can be used at each step. The process parameters that should be used in discovering a fault (or faults) can be selected, e.g., with the Sequential Forward Floating Search (Pudil, Novovicova et al. 1994) abbreviated SFFS or with the exhaustive search. After that, a model can be created to use, e.g., the Haar Transform or a linear filter, to clean noise from the sequential readings of the process variables inside the monitoring window. The PCA or the Canonical Variate Analysis or other similar residual generation techniques can then create a few variables that concentrate fault information out of those cleaned readings of the process variables. Finally, the Hotelling SPC chart (Xie, Goh et al. 2002), the Fuzzy ARTMAP network (Carpenter, Grossberg et al. 1992), or another classifier can be used to monitor these few variables. If the classifier determines that the set of the monitored principal components scores is out of the statistical process control, it is assumed that the process has faults, whereas if the monitored principal components are in the statistical process control, it is assumed that the process is operating normally.

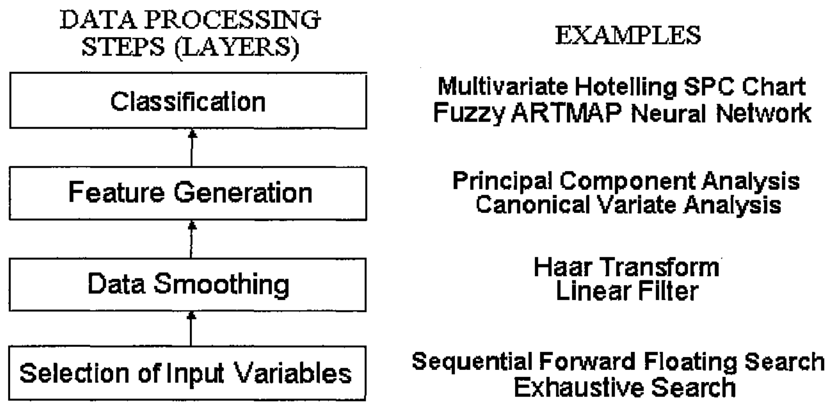

DATA PROCESSING STEPS (LAYERS)
EXAMPLES
Classification
Multivariate Hotelling SPC Chart
Fuzzy ARTMAP Neural Network
Feature Generation
Principal Component Analysis
Canonical Variate Analysis
Data Smoothing
Haar Transform
Linear Filter
Selection of Input Variables
Sequential Forward Floating Search
Exhaustive Search

Figure 2.10. Data processing steps in a fault model with examples

This sequence of data processing steps forms a fault model template. Filling this template with structural components representing methods performing different types of data mapping creates a fault model. Each data processing step (mapping) is optional, except for the Classification.

2.3.2 Further Decomposition of Fault Models

To make it possible to select fault models out of a set of very diverse alternatives that may include all the methods for fault monitoring proposed in the technical literature and to use the minimum possible number of different methods as structural components of the models at the same time, this work partially adopts additional model decomposition similar to the one proposed by many authors, e.g. (Brodley and Smyth 1997). In this approach, methods for data classification are viewed as a combination of three components:

1) Representation
2) Optimized criterion
3) Search technique.

"Representation" is the functional form of a model also known as a basis or link function (different disciplines use different names for this function: statisticians prefer using the term "link" and engineers prefer "basis"). **"Optimized criterion"** is a certain measure of model performance whose maximization or minimization guides the selection of model parameters during the model identification. **"Search"** is a search technique employed by the model to find the values of the model parameters corresponding to the best possible value of the optimization criterion. Thus, under this framework, a conventional logistic regression ($y=exp(x)/(1+exp(x))$) described in the Introductory Categorical Analysis textbooks, e.g., (Agresti 1996), would be represented as a combination of a *logit* basis function (representation), a sum of squared errors (optimized criterion), and the direct calculation of model parameters using the multiple linear regression formula (search). A more complex multi-layer perceptron would be a combination of several (usually, two) consecutive linear functions (representation), a sum of square errors (optimized criterion), and back-propagation (search) (Hagan, Demuth et al. 1995).

Using an approach similar to the one described by Brodley and Smyth, each of the data processing stages can be in turn decomposed into a set of structural components as shown in Figure 3.3. "Selection of Input Variables" is a discrete optimization, so it can be viewed as a combination of a discrete search algorithm ("Search Method"), e.g., the Exhaustive Search, and a "Search Criterion," such as the distance between the data points belonging to different classes. This criterion can be shared with the classification step, and then each step of the discrete optimization would require performing a separate identification of a fault model. Data Smoothing should be viewed as a combination of a representation function and an optimized criterion. In most cases, the optimized criterion is already encoded in that representation as is the case in a simplest moving average filter. Feature Generation methods can also be presented as combinations of a basis function and an optimized criterion, e.g., the PCA can be viewed as a linear transform combined with a minimization of the pairwise cross products of the outputs.

The purpose of the proposed decomposition is to represent a very large pool of various model types suitable for fault monitoring as a much smaller number of model components. These components can make up a fault model component library. Ideally, the decomposition should create as few components as possible out of the set of all the candidate fault models. Therefore, it is proposed not to split the Data Smoothing and Feature Generation steps into components because a combination of randomly chosen structural components for Data Smoothing and Feature Generation is much less likely to produce a valid data processing method than a combination of randomly chosen structural components for Input Variable Selection and Classification. Likewise, it is proposed that "Basis Functions" for Classification methods also include the parameter value search technique.

Another reason for not separating the search techniques from the basis functions is that many search techniques are specifically tailored for certain basis functions and optimized criteria. A good example of such a search technique is the commonly known relationship for finding the vector ***b*** of multiple linear regression parameters:

$$b=\left(X^T X\right)^{-1} X^T y \tag{2.2}$$

In this, ***X*** is the model training set presented in the matrix form and ***y*** is the vector of outputs corresponding to the input data vectors (prototypes) of matrix ***X***. Equation (2.2) is created to find the values of ***b*** that optimize the sum of squared errors between the outputs produced by the regression and the values of ***y*** and not any other criteria, e.g., the sum of absolute values of the errors or the number of erroneous classifications. Hence, in the proposed decomposition, the methods for basis function parameter search are limited only to those that can accommodate a wide range of optimized criteria. Thus, some methods for basis function parameter search cannot be used in the proposed fault model decomposition and this sacrifice is made for the sake of flexibility.

Then the resulting fault model template will look like the one shown in Figure 2.11.

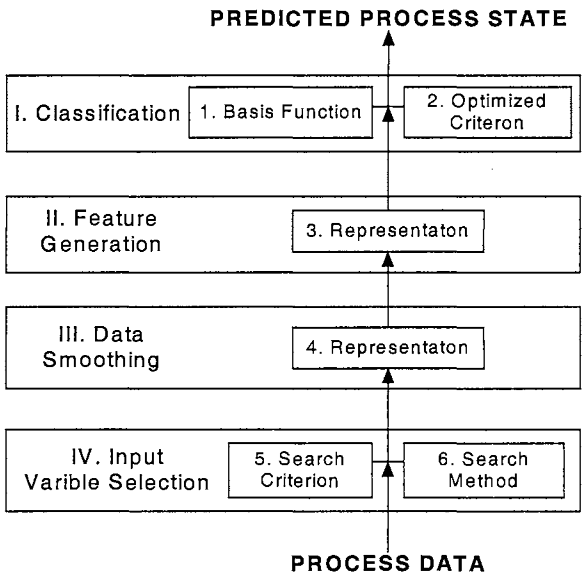


Figure 2.11. The proposed fault model template (generic structure of fault models)

The proposed template may be branched to allow several fault monitoring methods to run in parallel. It was found that application of several methods in parallel significantly increases the performance of automated fault monitoring (Mylaraswamy and Venkatasubramanian 1997). A fault model template in Figure 2.12 uses two classifiers in parallel and averages the classifier outputs. In a similar manner, one can propose a fault model template that uses several residual generation techniques. A more detailed discussion of fault model templates more complex than the one in Figure 2.11 is beyond the scope of this work.

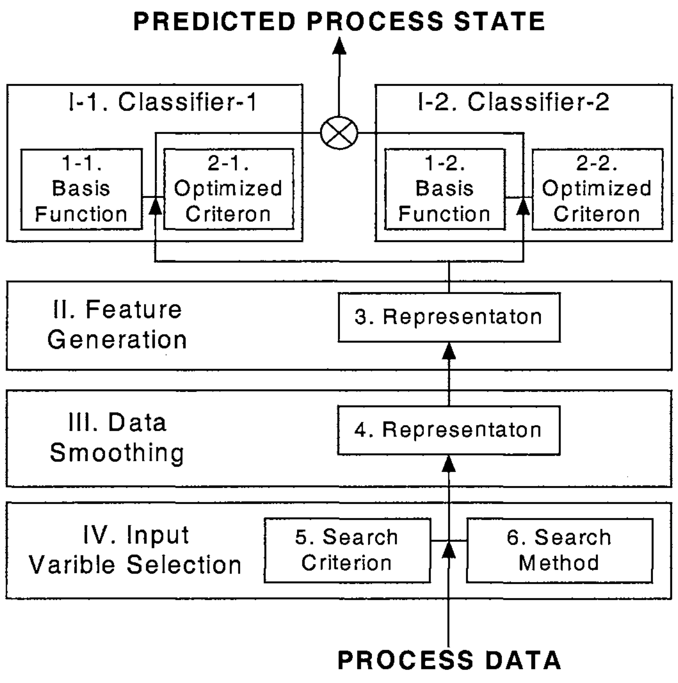


Figure 2.12. A Fault model template with two classifiers

2.3.3 The Need to Try Different Optimized Criteria in Fault Models

The need to try different optimized criteria in the classifiers of designed fault models contradicts common sense. It may seem that the optimized criterion should be the model performance metric: the model parameters should be set to the values that optimize the model performance metric. This is not the case because the number of fault instances available for fault model identification is small, frequently ten or fewer. As a result, any fault model with a classifier basis function with more than **one** parameter will **usually** over-fit the data no matter how large the sample of process data vectors (also known as **prototypes**) used for fault model identification. Minimizing the effect of this over-fitting can be achieved by trying different optimized criteria.

Suppose, that the model performance metric is a combination of the fault detection lag (the time between fault inception and model-based fault detection) and the rate of false alarms. If we created a model for monitoring this fault and set the classifier basis function parameters to minimize the detection lag, the model might effectively mark small regions of the input variable space, where the historicized fault instances started, as faulty. At the same time, the rest of the space of the input variables would be assumed by the model to correspond to the normal operation, which is incorrect.

Figure 2.13 shows process data points connected with arrows in the order in which these data points were observed when three instances of the same fault occurred.

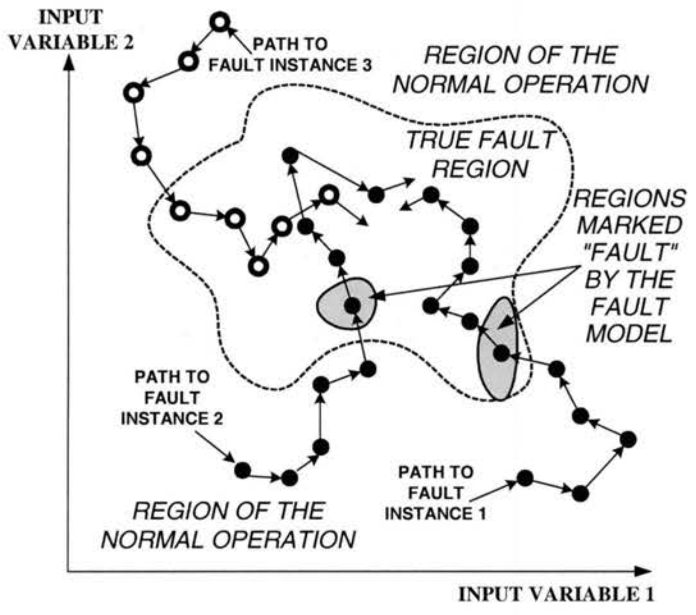


Figure 2.13. Incorrect mapping of faults by a fault model

This fault model cannot detect the third fault instance whose data were not used for model identification because the model marked only two small regions, where the first two fault instances started, as faulty. The reason why the "faulty" regions are small is because the user also wants as few false alarms as possible, so the optimization to reduce the number of false alarms would squeeze the faulty regions around the points where fault inceptions were observed and historicized. Therefore, fault model parameters should be identified using various optimized criteria, e.g., the simple error rate or the conventional Bayesian Risk (Johnson 1998b) and find the one which is the best at alleviating the effects of data over-fitting by fault models.

2.3.4 An Example of Assembling a Fault Model from Structural Components

If a fault model is a set of components filling the above-proposed template, then fault model selection is equivalent to filling the template "slots," at each data processing step, with certain components suitable for these slots. If no method is selected for a certain step, the "slots" at that step remain unfilled.

An example of how model slots can be filled to create a rather complex fault model is given in Figure 2.14. The monitored process is determined to be operating normally ("in control") by using a Hotelling control chart. In this model, the basis function is the Hotelling transform that calculates the T^2 statistics and the associated p-value. The level of significance α is set for the chart to have the lowest estimated probability of misclassifying the faulty and normal operating states. This probability is the optimized criterion. The inputs for the Hotelling chart are the PCA scores that explain 95% of the variation among the process data points of the training set created out of the available historical process data. The PCA fills the only slot of the Residual Generation step. No smoothing of the process data is performed, so the slot of the Data Smoothing method remains empty.

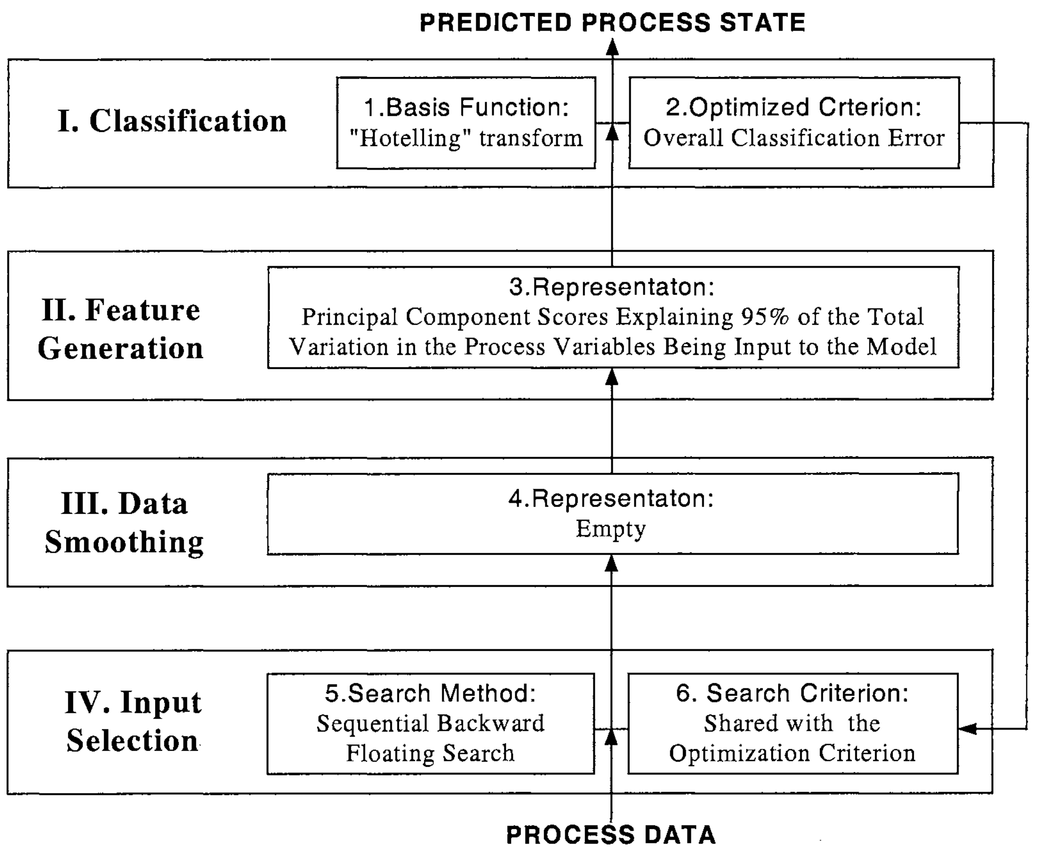

PREDICTED PROCESS STATE
I. Classification
1.Basis Function: "Hotelling" transform
2.Optimized Crterion: Overall Classification Error
II. Feature Generation
3.Representaton: Principal Component Scores Explaining 95% of the Total Variation in the Process Variables Being Input to the Model
III. Data Smoothing
4.Representaton: Empty
IV. Input Selection
5.Search Method: Sequential Backward Floating Search
6. Search Criterion: Shared with the Optimization Criterion
PROCESS DATA

Figure 2.14. An example of the decomposition of a complex fault model

This combination of fault model components is nested inside the Stepwise Floating Forward Search (Pudil, Novovicova et al. 1994) of the process variables that will be optimal inputs to the process model. The Stepwise Floating Forward Search is the "Search Method" used in the model. The optimality of the set of variables selected as fault model inputs is determined by the optimized criterion used in the classifier of the fault model. Thus, the "Search Criterion" slot in the model is shared with the "Optimized Criterion." So, a separate fault model is identified for each set of input process variables tested by the "Search Method." Alternatively, the search for input variables could optimize the pairwise distance between the process data vectors sampled during the occurrence of faults and the process data vectors sampled during the normal process operation, then this distance would be the Search Criterion and the Search Criterion slot would not share the same component with the Optimized Criterion slot.

Note that each data-processing method uses the input data modified by the methods located below it, except the Input Variable Selection that has the lowest position in the hierarchy. In addition, the operator should determine the sampling period for the Residual Generation and Classification from the characteristics of the monitored process while the Data Smoothing uses the sampling period (interval) of the process data historian.

2.4 Construction of Fault Models

From Section 2.3, it may seem that using arbitrary model components to fill slots in the proposed template can generate valid and universally suitable fault models. This is not the case because not all the model components are compatible with each other and not all the resulting fault models are appropriate for solving each specific fault-monitoring problem. Hence, a rule-based approach, an expert system, is needed to determine if a combination of fault model components on the proposed template makes a model design suitable for solving a user-specified fault-monitoring problem.

2.4.1 Attribute-Based Selection of Fault Models

Until the present, all the algorithms proposed for selecting models for process fault monitoring have been limited to those that identify a small group of methods or a single method suitable for predicting the user-specified fault. In these algorithms, the characteristics of the fault-monitoring problem and properties of the data being input to fault models are formalized in terms of attributes. These conventional algorithms apply production rules that determine the compatibility of different methods for fault monitoring with the attributes of the input data and the monitoring problem.

2.4.1.1 Selection of Univariate Control Charts. One such algorithm determines if a univariate control chart is suitable for monitoring a certain process (Brassard and Ritter 1994). This algorithm checks the attributes of the input data and tells what chart should be used to monitor the process producing this input data. The algorithm is shown below in Figure 2.15.

The charts among which the selection is performed are X (Shewhart) that checks if the mean of variable values recorded sequentially is within a certain range, R_m that checks if the modulus if the observed fluctuation in the monitored variable value is within a certain threshold, $\overline{X}$ that checks if the mean of a sample of values of the monitored variable is within the a certain range, R that checks if the average fluctuation between the sample values is within the a certain range, s that checks if the sample variance is within a certain range, c / u charts that check if the number of defects per unit product is within certain limits and p / np charts that check if the number of defective units in a sample is within certain bounds. The attributes of the input data that determine selection of the best control chart are the data scale type (continuous or categorical), sample size, type of the data for the chart, and the variability of the sample size.

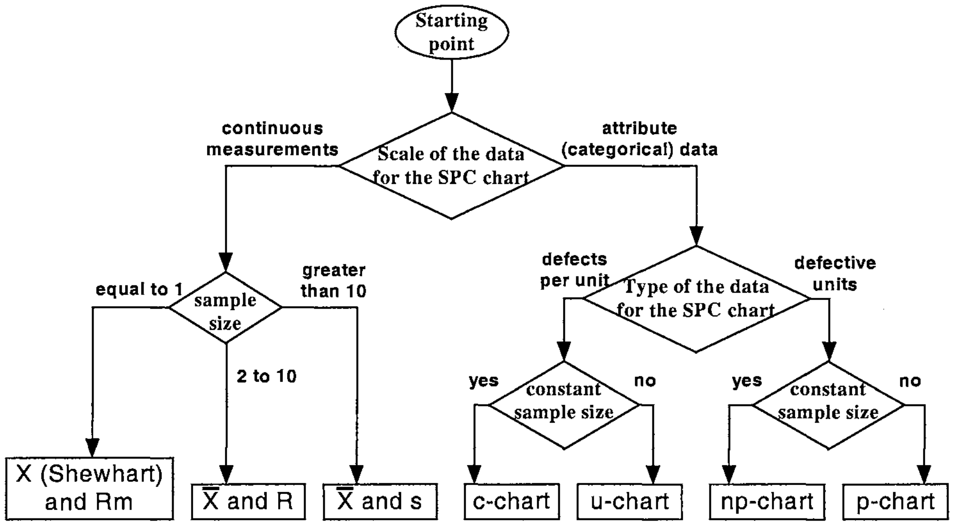


Figure 2.15 An algorithm for selecting an appropriate control chart (Brassard and Ritter 1994)

The algorithm shown in Figure 2.15 is of great utility, but it is limited to a few conventional control charts. A very large amount of experience in using univariate control charts has been accumulated. This is not the case with the other methods for fault monitoring. In addition, characteristics of the monitored process and user preferences should also be considered in model selection. It would be nice to have a register that lists applicability of all known fault model types depending on all possible values of the input data attributes, but as was shown above, the number of different possible fault model types is immense. It would be even better if an algorithm for fault model selection could handle generic attributes of fault models.

2.4.1.2 Selection Among Different Groups of Fault Monitoring Methods. Another algorithm known to the author has been proposed for selecting methods (Dash and Venkatasubramanian 2000) for fault detection and diagnosis based on the attributes of both different method groups and fault monitoring problems. This algorithm is presented as Table 2.1. To apply this algorithm, the user should first determine what attributes of those located in the leftmost column of the table are applicable to his or her problem. Then, a method type with the highest number of '✓'s (a method of this type should be a good solution for a problem with the corresponding attribute) or with the lowest number of '×'s (a method of this type should not be a good solution for a problem with this attribute) corresponding to the selected criteria should be chosen. A method of the type with at least one '—' corresponding to any of the selected criteria should be discarded. A dash in the table means that the methods of the corresponding type cannot be used for solving problems with the corresponding attribute altogether. The question marks in the table mean that the ability of the methods of this type to solve problems with the corresponding attributes is unknown and situation-dependent.

The criteria or attributes of Table 2.1 have the following meanings: "**isolability**" is the ability to distinguish one fault from the other faults possible in the process, "**robustness**" is the lack of sensitivity to the noise and uncertainty in the input variables, "**novelty identifiability**" is the ability to identify new faults, "**adaptability**" is the ability to operate well despite the drift in the process parameters, "**explanation facility**" is the ability to tell the user how the solution was obtained, "**modeling requirement**" is the need to manually identify the component of a fault model based on this method, "**storage and computation**" means that the method does not use a lot of processor time and computer memory, "**multiple fault identifiability**" is the ability of a model to monitor and distinguish several different process faults.

Table 2.1 Table for selecting various types of methods for fault detection and diagnosis (Venkatasubramanian, Rengaswamy et al. 2003b)

Method Attributes	**Methods for Fault Detection and Diagnosis**						
	Parity-space & observers	Digraphs	First-principle hierarchical	Expert systems	QTA	PCA	Neural Networks
Early detection and diagnosis	✓	?	?	✓	✓	✓	✓
Isolability	✓	✗	✗	✓	✓	✓	✓
Robustness	✓	✓	✓	✓	✓	✓	✓
Novelty identifiability	?	✓	✓	✗	?	✓	✓
Classification error low	✗	✗	✗	✗	✗	✗	✗
Adaptibility	✗	✓	✓	✗	?	✗	✗
Explanation facility	✗	✓	✓	✓	✓	✗	✗
Modeling requirement	?	✓	✓	✓	✓	✓	✓
Storage and computation	✓	?	?	✓	✓	✓	✓
Multiple fault identifiability	✓	✓	✓	✗	✗	✗	✗

One problem with the algorithm shown in Table 2.1 is that it is limited to several classes of fault models and does not span all the known methods for fault monitoring. The second problem is that the classes represented by the columns of Table 2.1 may contain a number of very different methods that may have different attributes. For example, Table 2.1 shows that neural networks are not good at identifying multiple faults (the cell in the lower right corner of the table contains symbol '✗'). This is true for most neural networks, but not for the Probabilistic Neural Network or ARTMAP/Fuzzy ARTMAP. The third problem is that it is unrealistic to expect the process operators to specify some attributes listed in Table 2.1 for their monitoring problems. For example, all process faults should preferably be detected and diagnosed early. Likewise, it is always desirable to have a low classification error in the fault models as well as detection robustness. The tradeoff between these three attributes can only be determined numerically by using a statistical analysis of the user-specified fault-monitoring problem and creating a problem-specific performance metric, as will be shown in Chapter III. This third problem also makes the other existing attribute-based algorithms for classifier selection, e.g., (Henery 1994), not quite suitable for finding the best-performing fault models.

2.4.1.3 The Proposed Attribute-Based Selection of Fault Model Components. To overcome the problems with the attribute-based selection of fault models described in Subsection 2.4.1 it is proposed to take advantage of the fault model decomposition proposed in Section 2.3 and assign component compatibility attributes to fault model components that can fill the slots of the template shown in Figure 2.11. These attributes are logical or quantitative variables that do not include information about detection timeliness, robustness and other quantitative characteristics of fault model performance. Quantitative performance of fault models is estimated separately. Different attributes would describe the properties of fault model components, monitoring problem, input data, and user preferences.

Some model component attributes, like minimum required size of the training data set, may depend on the properties of the real-time and historical data that will be used by the model. For those attributes, it is important to provide a method for initializing the attributes depending on the attributes of the monitoring problem, input data, and user preferences.

A list of proposed attributes is presented in Table 2.2 below. The attributes characterizing the properties of the real-time data start with letter ‘D’, the attributes specifying the properties of the historical data start with letter ‘H’, the attributes specifying the solution requirements start with letter ‘S’, and the attributes describing the properties of model components start with letter ‘C’. In addition, there is an attribute describing a characteristic of complete fault model designs, this attribute starts with letter ‘M’. Real-time data attribute have subscripts specifying the data processing step whose input data the attribute describes (the reader is reminded that in the proposed fault model template there are four data processing steps). The attributes with subscript 'A' refer to the data available for the input variable selection method, the attributes with subscript 'B' refer to the data available for the data smoothing method, the attributes with subscript 'C' refer to the data available for the feature generation method, and the attributes with subscript 'D' refer to the data available for the classifier. The pound sign following the attribute indicates that the attribute is a variable that assumes real values while the absence of the pound sign indicates a Boolean-valued attribute. All the listed attributes for a component or a problem must be assigned values before the selection can start.

In addition to the attributes, fault model components added to the component library described above in Subsection 2.3.2 should have data attribute modifiers that reflect changes in the input data properties as a result of being processed by different parts (methods) of the fault model. For example, if an input variable selection method is designed to select one third of the available process variables, then the attribute $D2_B$ specifying the number of variables in the data that will enter the data smoothing should be one-third of $D2_A$, which is the number of process variables related to the fault according to the user.

Table 2.2. The proposed attributes for determining the compatibility of fault model components

Attributes of the data stream *X* processed by the model	Attributes of the process data vector	**D1$_X$#** Average number of single variable measurements in the monitoring window
		D2$_X$# The number of variables in the data
		D3$_X$ Data vector consists of combinations of process variables (**X**=D or E)
		D4$_X$ Monitored process variables are strongly correlated
		D5$_X$ No sensor noise in all the monitored process variables
		D6$_X$# Maximum number of different measurements of a single process variable in the monitoring window
	Attributes of the historical data used for model identification	**H1#** Number of process data vectors (prototypes) selected for model identification
		H2# Smallest number of historicized instances of a monitored fault for all the monitored faults in the specified historical data
		H3 Contains a variable believed to be most closely related to the monitored fault
Attributes of the monitored process and user preferences:	Process model availability	**S1** Monitored process input and output variables specified
		S2 Monitored process model not available
	Solution requirements	**S3** Monitoring multiple fault modes
		S4 Fault model output must be a fault score
		S5 Fault model output must be categorical
		S6# Log$_{10}$ of the upper limit on the CPU time, sec., allowed for model identification
Attributes of fault model components	Basis function	**C1** Requires original process variables
		C2 Requires uncorrelated inputs
		C3 Creates an unbounded region for the normal operation
		C4 Generates binary output
		C5 Fixed number of basis function parameters (excludes **C7**)
		C6 Requires identification of a variable most closely related to the monitored fault
		C7 The size of the input data vector alone determines the number of the parameters
		C8 Generates continuous output
		C9 Generates crisp categorical output
		C10# Minimum number of process data vectors (prototypes) for model identification
		C11 Noise-sensitive basis function
		C12 Univariate basis function
		C13 Outputs fuzzy membership in one of several classes
		C14 Unlimited number of basis function parameters
	Optimized criterion	**C15** Distance-based optimized criterion
		C16 Count-based optimized criterion
		C17 "Parsimonious" optimized criterion that penalizes for the number of basis function parameters
		C18 "Monotonic" optimized criterion: the more basis function parameters the better
	Feature generator	**C19** Based on a process model generating residuals
		C20 Identifies an empirical process model generating residuals
		C21 Useful only when the inputs are correlated
		C22 Feature generator that models data variability
		C23# Minimum number of input variables
		C24# Minimum number of process data vectors for model identification
	Data smoothing	**C25** Smoothens out only noisy variables
	Input variable search method	**C26#** The number of user-selected fault-related process variables must exceed *N*
		C27# The number of user-selected fault-related process variables must not exceed *N*
		C28 All the user-selected fault-related process variables can be chosen as model inputs
		C29 Not designed to retain a specific variable as model input
	Variable selection criterion	**C30** Monotonic search criterion (the more input variables the better)
		C31 The optimized criterion is separate from the variable selection criterion
Fault model attributes		**M1** Minimum CPU time required to identify and evaluate the model

Three different sets of rules would determine compatibility of model components with the monitoring problem and user requirements, with the input data, and with each other. For a certain attribute or group of attributes these rules would specify attributes of fault model components that cannot be used in this design. In addition, fault model components should have data modifiers describing how each sequential data processing step of the fault model changes the input data. Finally, each component should have an associated training time attribute that would allow performing an approximate estimation how long it would take to train a model that includes this component. The proposed rules are listed in Tables 2.3, 2.4, and 2.5. They specify if fault model designs are valid by themselves and valid for solving user-specified problems. If the antecedent of a rule is evaluated as TRUE, then the consequence of this rule does not allow selection of the components with the attributes listed in the rightmost column of the tables. If the proposed algorithm is found useful, additional model component screening rules may be added in the future to the set of the 23 proposed rules. These rules also work for the fault model templates that combine different classifiers, feature generators, and/or data smoothers, like the one shown in Figure 2.12. For those templates, one additional rule that will require different classifiers included in the same model to produce the outputs of the same type is needed.

This approach minimizes the effort needed to specify the necessary attributes for each model, since the set of attributes of a single component will be used in all the designs that include this component. For large component libraries containing dozens of fault model components, this allows reducing the total number of attributes to be specified for the components of this library by several orders of magnitude compared with the case when attributes are specified for each complete model design.

2.4.2 Compatibility of Fault Model Components with the Solution Requirements

Table 2.3 below lists the rules of this class.

Table 2.3. Rules specifying fault model component compatibility with the characteristics of monitored process and user preferences

Rule number	Rule statements		
	Verbal	Symbolic	
		Antecedent(s)	Restrictions on the component attributes
1	If the types (input/output variables) of the monitored process variables are not specified, then the feature generation techniques creating empirical process models and using their residuals are not applicable	*NOT(S1)*	*C20=FALSE*
2	If an input-output model of the monitored process is not available, the techniques that use monitored process models are not applicable	*NOT(S2) AND NOT(C20)*	*C19=FALSE*
3	If a fault model is intended to distinguish several fault modes, then the basis functions that create binary or continuous outputs are not applicable	*S3*	*C4=FALSE AND C8=FALSE*
4	If the user wants the fault scores displayed, the basis functions creating crisp categorical or binary output cannot be used in the fault model	*S4*	*C4=FALSE AND C9=FALSE*
5	If the user wants the fault model to display only if there is a certain fault or there is no fault, the basis functions that create continuous output or output class memberships are not applicable in the fault model	*S5*	*C8=FALSE AND C13=FALSE*
23	Model identification time should not exceed the value specified by the user	*S6 = N*	*M1 < N*

2.4.2.1 Requirements Based on Predefined Conditions. Each fault-monitoring problem has its own characteristics. Likewise, process operators have preferences that should be accounted for when selecting fault models. These characteristics and preferences can be formalized as attributes of **solution requirements**. The base solution requirements may be the availability and ability to use a model of the monitored process for feature generation, the ability of a model to monitor more than one fault, the required form of presenting the model output to the user, and the maximum computation time that can be allotted for training a single fault model. Additional solution requirements are the need to use adaptable fault models due to irreversible process changes, the ability of fault models to monitor batch processes and many others. These additional solution requirements are not essential to illustrate the proposed concept of attribute-based selection of model components and are beyond the scope of this work.

Some feature generation techniques create empirical models of the monitored processes and output model residuals as the features. However, process models require the knowledge which process variables measure process inputs and which variables measure outputs. In addition, these variables must be in the block of variables selected by the user for fault monitoring. Hence, *Rule 1* says that **without the selection and marking of process input and output variables, the use of process model residual generation techniques is impossible.** The selection and marking of process variables as process inputs and outputs is the attribute *S1* and the feature generation techniques that identify a model of the monitored process are considered to have attribute *C20.*

Some classifier basis functions, e.g., parity-space methods or observers, expect process model residuals as inputs. This means that there should be a model of the monitored process which is a part of the fault model (attribute *C20*) or which runs independent of the fault model (attribute *S2*). Hence, *Rule 2* says: **if an input-output model of the monitored process is not available, the techniques that use monitored process models are not applicable**.

As shown in Figure 2.3, fault models normally tell the operator that something is wrong with the monitored process (see Section 2.3.1). After that, either the operator performs fault diagnosis or a fault diagnosis is performed automatically. In some cases, however, the fault diagnosis can be significantly simplified if the model designed for fault detection distinguishes different faults. Although this will cause a substantial loss of flexibility, a fault model can be trained to distinguish different faults if there is more than one historicized instance of each fault available. If this is the case, the user can assign TRUE to attribute *S3* meaning that the fault model should distinguish multiple faults. However, not all the basis functions can map data into more than two classes: some basis functions output continuous variables (these basis functions have attribute *C8*) that can only be interpreted as membership in one of two possible categories, while some other basis functions create binary outputs (these basis functions have attribute *C4*). Thus, *Rule 3* states that **if a fault model is intended to distinguish several fault modes, then the basis functions that create binary or continuous outputs are not applicable**.

Fault models should present information to the user in the way the user prefers. Fault classifier basis functions can produce either categorical outputs clearly stating existence or non-existence of the monitored faults (these basis functions have either attribute *C4* or attribute *C9*) or a fault score informing the operator of the possibility and extent of the monitored fault (these basis functions have either attribute *C8* or attribute *C13*). The basis functions creating crisp categorical outputs are usually preferred for abrupt clearly defined faults, such as sudden failures of process equipment. Fault score-producing basis functions are better suitable for the cases where a large number of false alarms are inevitable and where the impact of the same fault may vary significantly from one fault instance to another. This consideration gives birth to *Rule 4* saying that **if the user wants the fault scores displayed, the basis functions creating binary or crisp categorical outputs cannot be used in the fault model** and *Rule 5* stating that **if the user wants the fault model to display only if there is a certain fault or there is no fault, the basis functions that create continuous output or class memberships outputs are not applicable in the fault model**.

2.4.2.2 Limits on the Model Identification Time. An analysis involving all the components of the generated model is needed to roughly estimate the identification time for a data-driven model. To provide a simplistic estimate of the identification time for a fault model, it is convenient to view each data processing step of the model as a unit. The number of times (iterations) this unit is run during the model identification is determined by the representation/basis function component except for the Input Variable Selection for which this number is determined by the by the Variable Search Method component. Then the estimate of the order of the amount of time *O(T)* needed to identify a model would be approximately equal to the supreme of the orders of time t_u it takes to run unit u multiplied by the number of times (iterations) n_u that this unit will be run during the model identification:

$$O(T) \approx \sup_{u} \{O[t_u n_u]\} \tag{2.3}$$

If the Variable Search Criterion is the same as the Optimization criterion, then running each evaluation of the Search Criterion requires identifying an entire model with a new set of input variables, so t_u then would be equal to the model identification time.

As an illustration, let us suppose that it takes approximately $t_{u1}=0.002$ seconds to perform one Hotelling transform in the model in Figure 2.14 and it takes $t_{u2}=0.7$ seconds to run an identification of the principal components for this model. Let us also suppose that the classifier parameter identification would take $n_{u1}\approx 10^1$ iterations and the input variable search would take $n_{u3}\approx 10^2$ iterations. The principal components are calculated in a single iteration ($n_{u2}=1$) Then the order of time taken by the identification of the fault model for each tested set of input variables would be calculated as follows:

$$O(T) \approx \sup[O(t_{u1} \cdot n_{u1}), O(t_{u2} \cdot n_{u2})] = \sup[O(0.002 \cdot 10), O(0.7 \cdot 1)] = 10^0 \tag{2.4}$$

By multiplying this expression by the number of iterations that will be taken by the input variable search, we obtain that it will take on the order of 10^2 seconds to identify the fault model in question. Then, if the user-specified limit on the model identification time is below 10^2 seconds, the model in Figure 2.14 is valid or suitable for solving problem, otherwise it is not.

Estimation of the model identification time should be performed only after invoking all the other proposed rules defining fault model component compatibility. This is why the rule limiting the model training time is the last one in the sequence of the proposed rules. This rule number 23 says that **model identification time (attribute *M1*) should not exceed the value (attribute *S6*) specified by the user**.

2.4.3 Rules Specifying Fault Model Component Compatibility with Each Other

Table 2.4 below lists the rules of this class.

Table 2.4. Rules specifying fault model component compatibility with each other

Rule number	**Rule statements**		
	Verbal	**Symbolic**	
		Antecedents	**Restrictions on the component attributes**
6	The basis functions generating categorical (crisp or membership in several classes) or binary outputs are not compatible with the distance –based optimized criteria	*C4 OR C9 OR C13*	*C15=FALSE*
7	The basis functions producing continuous outputs are not compatible with the count-based optimized criteria	*C8*	*C16=FALSE*
8	"Parsimonious" optimized criteria cannot be used if: 1) The basis function has a fixed number of parameters OR 2) The input variable search criterion is separate from the optimized criterion AND the size of the input data vector alone determines the number of basis function parameters	*C5* *OR* *(C31 AND C7)*	*C17=FALSE*
9	"Monotonic" optimized criteria cannot be used if: 1) The number of parameters in the basis function is not limited 2) OR the size of the input data vector alone determines the number of parameters in the basis function AND the input variable search criterion is the same as the optimized criterion AND all the user-selected fault-related process variables can be chosen as model inputs	*C14 OR [NOT(C31) AND C28 AND C7]*	*C18=FALSE*
10	"Monotonic" criteria for searching input variables are not applicable if the input variable search criterion is separate from the optimized criterion AND all the user-selected fault-related process variables can be chosen as model inputs by the selected input variable search method	*C31 AND C28*	*C30=FALSE*
11	The input variable search methods not designed to retain a specific variable as model input are not compatible with the basis functions that require specification of the process variable believed to be most closely related to the monitored fault	*C6*	*C29=FALSE*

2.4.3.1 Count and Distance-Based Optimized Criteria. There are two types of optimized criteria that can be used in mathematical models. The criteria of the first type are distance-based and they are assumed to possess attribute *C15*. They measure how far the values predicted by the model are from the actual ones. This distance can only be defined for basis functions producing the outputs on the continuous scale (attribute *C8*). The other basis functions that can be used in the classifiers, i.e., **the basis functions generating categorical or binary**

outputs (and possessing attributes *C4*, *C9*, and *C13*) are not compatible with the distance-based optimized criteria. This is *Rule 6*.

The optimized criteria of the second type are count-based and they are assumed to possess attribute *C16*. These criteria are functions of the number of correct and incorrect classifications of different states performed by the classifier. This number can only be defined for the basis functions producing categorical or binary outputs. Alternatively, continuous outputs can also be converted into categorical ones; however, this converter would be external to the fault model defined by the template in Figure 2.11. Hence, **the basis functions producing continuous outputs (and possessing attribute *C8*) are not compatible with the count-based optimized criteria.** This is *Rule 7*.

2.4.3.2 "Parsimonious" and "Monotonic" Optimized Criteria. Some optimized criteria, e.g., the Akaike Information Criterion or Divergence Information Criterion, can be considered "parsimonious" and they are assumed to have attribute *C17*. These criteria penalize models for the number of basis function parameters. However, it does not make sense to penalize basis functions for the number of parameters if this number is fixed, like it is in a control chart. The basis functions of this type are characterized by attribute *C5*. For some basis functions, the number of basis function parameters may not be fixed, but may be determined by the number of classifier inputs alone (attribute *C7*), like it is with the first-order multiple linear regression (without the interaction terms). In this case, a "parsimonious" Optimized Criterion does not affect the number of parameters in the identified model unless it can change the number of classifier inputs. The Optimized Criterion can change the number of classifier inputs only when this criterion is also used for selection of process variables for the fault model, which is the case only when the Optimized Criterion is shared with the Input Variable Search Criterion. This means that **"parsimonious" optimized criteria cannot be used if:**

1) **The basis function has a fixed number of parameters OR**

2) **The input variable search criterion is separate from the optimized criterion (attribute *C31*) AND the size of the input data vector alone determines the number of basis function parameters (attribute *C7*).**

This is *Rule 8*.

The other optimized criteria, e.g., error rate, sum of square errors, are monotonic and these optimized criteria have attribute *C18*. With these optimized criteria, the more parameters the classifier has, the better (smaller or larger) is the value of the optimized criterion because with a greater number of parameters a model can fit the training data better. Then, if these optimized criteria are used with a basis function (with attribute *C14*) whose number of parameters is not limited in any manner, the size basis function of the resulting fault model may grow unbounded during model identification and the identification procedure may never end. Even if the number of basis function parameters is determined by the number of classifier inputs alone, but the number of input variables may grow until all the available process variables may be selected as model inputs because the input variable search criterion is the same as the optimized criterion and the input variable search method is such that all the user-selected fault-related process variables can be chosen as model inputs (attribute *C28*). In this case, use of input variable selection is meaningless and the inclusion of an input variable selection technique in the model design makes the design invalid. Hence, *Rule 9* states that **"monotonic" optimized criteria cannot be used if:**

1) **The number of parameters in the basis function is not limited OR**
2) **The size of the input data vector alone determines the number of parameters in the basis function AND the input variable search criterion is the same as the optimized criterion AND all the user-selected fault-related process variables can be chosen as model inputs.**

2.4.3.3 Incompatibilities of the Input Variable Search Criteria. Similar to optimized criteria, input variable search criteria can also be "monotonic," so the more variables are selected, the better is the value of the criterion. These criteria have attribute *C30*. Same as in the case described in the second part of the antecedent of *Rule 9*, such "monotonic" search criteria are incompatible with the input variable search methods that can potentially choose all the available variables as model inputs. Again, this makes the use of input variable selection methods meaningless because in those cases all the available process variables will be selected as model inputs anyway. Thus, *Rule 10* states that **"monotonic" criteria for searching input variables are not applicable if the input variable search criterion is separate from the optimized criterion AND all the user-selected fault-related process variables can be chosen as model inputs by the selected input variable search method**

Some basis functions, e.g., those used in the Quantitative Trend Analysis, may require the process variable believed most closely related to the monitored fault as model input. In the proposed register of attributes, these basis functions have attribute *C6*. However, most input variable search methods are not preprogrammed to surely keep this key process variable as fault model input (and these search methods have attribute *C29*). So, **the input variable search methods not designed to retain a specific variable as model input are not compatible with the basis functions that require specification of the process variable believed to be most closely related to the monitored fault**. This is *Rule 11*.

2.4.4 Rules Specifying Fault Model Component Compatibility with the Input Data

Table 2.5 below lists the rules of this class.

Table 2.5. Rules specifying fault model component compatibility with the properties of input data

Rule number	Rule statements		
	Verbal	Symbolic	
		Antecedents	Restrictions on the component attributes
12	The basis functions that require original process variables for the classifier input data cannot process combinations of these variables	$D3_D$	$C1=FALSE$
13	Certain basis functions cannot be used in fault models to classify strongly correlated data	$D4_D$	$C2=FALSE$
14	If the number of historicized instances of a fault is smaller than four then the basis functions creating unbounded regions for the normal operation and the feature generation techniques that model variability of the input data are not applicable in the model that monitors this fault	$H2<4$	$C3=FALSE$ *AND* $C22=FALSE$
15	Certain basis functions require historicized values of the process variable believed to be most closely connected with the monitored fault for fault model identification	$NOT(H3)$	$C6=FALSE$
16	Basis functions impose a lower limit on the number of process data vectors (prototypes) required for model identification: this number is the minimum number of different process data vectors the fault model must be able to create from the historical data	$H1=N$	$N \geq C10$
17	Noise-sensitive basis functions are not applicable if there is noise in any of the processed variables	$NOT(D5_D)$	$C11=FALSE$
18	The feature generation techniques that de-correlate input variables should not be used if the input variables are not correlated	$NOT(D4_C)$	$C21=FALSE$
19	Some feature generation techniques should not be used if the number of input variables does not exceed the technique-specific number *N*	$D2_C=N$	$N \geq C23$
20	Some feature generation techniques impose a lower limit on the number of process data vectors (prototypes) required for model identification: this number is the minimum number of different process data vectors the fault model must be able to create from the historical data	$H1=N$	$N \geq C24$
21	Data smoothing techniques that remove noise and outliers should only be applied to noisy variables	$D5_B$	$C25=FALSE$
22	Input variable search techniques require the number of user-selected process variables to be within certain limits	$D2_A=N$	$C26 < N < C27$

2.4.4.1. Rules Specifying the Compatibility of Basis Functions. Some fault classifiers with attribute *C1*, e.g., control charts for the process variables believed most closely related to faults, require original process variables in the available input data. These classifiers, or, to be more exact, the basis functions of these classifiers, cannot be used with combinations of process variables (attribute $D3_D$), which may be features generated from the original variables by the feature generation techniques used by the fault model. Hence, *Rule 12* states that **the basis**

functions that require original process variables for the classifier input data cannot process combinations of these variables. Basis functions of some classifiers (possessing attribute *C2*), e.g., those of the classification and regression trees, are usually inefficient if the input variables available to them are correlated, i.e., the input data has attribute $D4_D$. This is due to the fact that they perform partitions on the variables whose values are most closely related to the changes from the normal operation to the faulty ones. If classifier inputs are strongly correlated, all the variables display some, usually small, changes when the monitored process passes between the normal and faulty states. So, in those cases, it would be hard for the classification and regression tree to find the key variables most closely connected to the fault and perform partition of the input space on those variables. Therefore, **certain basis functions cannot be used in fault models to classify strongly correlated data**. This is *Rule 13*.

Some classifiers, e.g., Fisher Discriminant or Multi-layer Perceptron, assign unbounded regions of the input variable space (partitions of Type 1 and Type 4 in Figure 2.6) to the normal process operation. Because of that these classifiers are characterized by attribute *C3*. If only few instances of the monitored fault have been historicized (attribute *H2* less than four), then this extrapolation of the normal operation state to unknown regions of the input space is dangerous and the unbounded region will likely include both faulty and non-faulty states of the monitored process. Similar considerations apply to the feature generation techniques that model data variability, e.g., PCA or ICA. These feature generation techniques are assigned attribute *C22*. MacGregor and Kourti have noted (MacGregor and Kourti 1995) that if the faulty operation is not very well represented in the training set, then most of the variability modeled by these techniques will be the variability within the normal operation and the variability due to the occurrence of faults will be suppressed and not reflected in the features input to the classifier. As a result, the normal operation regions formed by the classifier in the space of inputs will very likely include faults as well and the fault model performance will be low. Hence, *Rule 14* states that **if the number of historicized instances of a fault is smaller than four then the basis functions**

creating unbounded regions for the normal operation and the feature generation techniques that model variability of the input data are not applicable in the model that monitors this fault. Chapter III suggests using the jackknifing for estimating classifier performance. To estimate the performance of the resulting fault models, the proposed jackknifing technique removes one fault instance from the set used for model identification. Therefore, the number of historicized instances of each fault to be monitored should be reduced by one when assigning the value to attribute *H2*.

As was discussed in Subsection 2.4.3.3, some basis functions, e.g., those used in the Quantitative Trend Analysis, may require the process variable believed most closely related to the monitored fault as model input. This means that the process data historian must have logged the values of this variable for the historicized fault instances. This fact is described by attribute *H3*. Thus, *Rule 15* states that **certain basis functions require historicized values of the process variable that the user believes is most closely connected with the monitored fault for fault model identification**.

Sometimes, the historical data that the operator set aside for fault model identification may be very small. The number of different process data vectors that can be extracted from the historical process data allotted by the user for model identification is characterized by attribute *H1*. However, this number must be large enough to allow identification of basis function parameters. The minimum number of different data vectors required for model identification is described by attribute *C10* and it depends on the basis function. For example, linear models require that the number of data vectors for data identification be at least as large as the number of basis function parameters. Control charts typically need about 300 historicized data vectors for parameter identification. As a result, *Rule 16* says that **basis functions impose a lower limit on the number of process data vectors (prototypes) required for model identification: this number is the minimum number of different process data vectors the fault model must be able to create from the historical data**. This rule is true not only for basis functions, but also for feature generators, especially for those that model the monitored process. The lower limit imposed on the number of input for a feature generator is described by attribute *C24*. Consequently, *Rule 20* is that **some feature generation techniques impose a lower limit on the number of process data vectors (prototypes) required for model identification: this number is the minimum number of different process data vectors the fault model must be able to create from the historical data**.

Some classifier basis functions perform very poorly on noisy data. This is particularly true for the basis functions with a large number of parameters, such as neural networks. They require data cleaned of noise and outliers. In this work, "clean" data entering fault classifiers have attribute $D5_D$. The noise-sensitive basis functions are characterized by attribute *C11*. Hence, *Rule 17* precludes the use of the noise-sensitive basis functions with noisy data by stating that **noise-sensitive basis functions are not applicable if there is noise in any of the processed variables**.

2.4.4.2. Rules Specifying the Compatibility of Feature Generation Techniques. A number of feature-generation techniques, like PCA and PLS are based on de-correlating input variables and then operate with a few uncorrelated variables. These feature generators are assumed to have attribute *C21*. However, process variables selected by the user for fault monitoring may not always be correlated (and described by attribute *D4* within this work). This is true if the user-selected variables describe very different points in the monitored process and have very different nature. For example, ambient temperature is usually not correlated with any process flow rates. In those cases, the use of the de-correlating feature generation techniques is meaningless because these feature generators will not reduce the number of variables in the data and will possibly remove features critical for fault detection from the input data. Therefore, **the feature generation techniques that de-correlate input variables should not be used if the input variables are not correlated**. This is *Rule 18*.

The purpose of most feature generation techniques is to reduce the number of variables in the data. However, if this number is small, these feature generators are useless. Hence, in the library of fault model components, each feature generation technique should have an attribute *C23* specifying the minimum number of inputs required for this technique. Therefore, **some feature generation techniques should not be used if the number of input variables does not exceed the technique-specific number** *N*. This is *Rule 19.*

2.4.4.3. Rules Specifying the Compatibility of Data Smoothing and Input Variable Selection Techniques (*O*→*O′* Mappers). Most data smoothing techniques remove noise and outliers from process variables. These data smoothers have attribute *C25.* However, if the variables in the model input data are not noisy and do not have uncertainties (such data are characterized by attribute *D5*), no data smoothing is necessary. In those cases, data smoothing may even hurt by removing the features important for fault detection from the input data. Data noise is primarily due sensor noise. However, with the installation of better sensors, especially those for measuring temperature and pressure, sensor noise becomes negligible. Therefore, **the data smoothing techniques that remove noise and outliers should only be applied to noisy variables.** This is *Rule 21.*

The number of variables selected by the user for fault monitoring and made available to the variable selection technique is the only factor that limits the use of different variable selectors. For example, exhaustive search tries all possible combinations of the input variables. The number of these combinations is equal to the factorial of the number of available process variables. Thus, if the number of available variables exceeds ten, the exhaustive search is not applicable under any circumstances. Likewise, selection of input variables only makes sense when the total number of the user-selected variables is sufficiently large, otherwise there is nothing to select from. In addition, many input variable selection techniques have a lower limit on the required number of input variables. For example, if the sequential floating forward search mentioned in Subsection

2.3.1 requires at least four available variables to select from, otherwise it will not run. Hence, each variable selection method in the library of fault model components should have an attribute *C26* specifying the minimum number of variables among which it can perform the selection and the attribute *C27* specifying the maximum number of variables available for selection. So, *Rule 22* says that **input variable search techniques require the number of user-selected process variables to be within certain limits**.

Data smoothing techniques require a certain number of sequentially recorded data points to work well. For example, the simple averaging requires at least two points to find the average. There is no rule that limits application of data smoothing techniques due to the fact that the monitoring windows may be too narrow because the data smoothing techniques can theoretically operate with monitoring windows of any length. Despite that, when specifying the monitoring window widths, the user should keep in mind that the performance of data smoothing techniques seriously deteriorates if the monitoring window is made too narrow.

2.5 Summary

2.5.1 Statement of the Proposed Algorithm for Fault Model Design

The proposed algorithm for creating model designs appropriate for use in automating the detection of user-specified process faults can be summarized as follows. Before using the algorithm for designing fault models, a developer or an advanced user:

a) Creates a library of fault model components
b) Specifies component attributes, data modifiers, and the time it takes to identify fault models that use these components

After that, for each fault to be monitored:

1) The process operator formalizes the solution requirement in terms of the proposed attributes and the process data historian assigns values to the process data attributes
2) The algorithm creates all the possible combinations of the library components from the proposed fault model template and lists the attributes of the components for each combination
3) For each model design, the algorithm identifies the attributes of real-time data available for each method, which is a part of the design, using the data modifiers of the included components
4) The algorithm invokes all the rules defining model component compatibility rules and finds which combinations of fault model components (model designs) are not suitable for monitoring the fault
5) The algorithm estimates the time required for identification of each suitable model design and marks the designs whose parameter identification time will exceed the user-specified limit as unsuitable
6) Discards all the model designs found not suitable.

After that, the parameters of the remaining fault model designs are identified and the best-performing models are selected according to the guidelines of the next chapter.

2.5.2 Advantages and Disadvantages of the Proposed Algorithm for Fault Model Design

This proposed approach to fault model design makes it possible to:

1) Decompose into components and design out of components any mathematical model whose application to fault detection has been discussed in the technical literature to date (except for those that do not allow separation of the optimized criterion and the basis function),

2) Represent a large number of different fault models as a smaller number of separate model components,

3) Search through large libraries of very diverse mathematical methods (perform mathematical method mining), which is extremely important in the cases when the shape of the modeled relationship is very hard or impossible to determine a priori.

Very unusual fault models can also be designed from the proposed fault model template. A human being may never come up with these models. The conventional wisdom may also say that these "strange" models will never produce good results, but this may not be the case for certain fault-monitoring problems.

The proposed approach to fault model design will also be helpful when the class of the methods most suitable for a certain data processing step in monitoring a specific fault is known. The user would simply add all the available methods of this class to the model component library and remove the other components performing the same function. An example of such a situation would be the one where one or several "good" models of the monitored process have been developed. Then the choice of the feature generation techniques for the model that monitors faults in this process would be limited to these "good" process models.

The downside of the proposed set of the model component compatibility rules is that this set must be amended to allow designing models for monitoring batch processes and the processes that constantly undergo irreversible changes. Unfortunately, no guidelines on how to create new rules can be presented at that point. In addition, as was noted before, the methods that do not allow the separation of the optimized criteria and the basis function linked with the parameter search technique cannot be decomposed for the use in the model component library.

Chapter III. Evaluation of Fault Model Performance

Chapter II has proposed an approach to designing fault models that are operable and formally suitable for user-specified fault-monitoring applications. However, the fault model performance may differ very greatly from one model design to another. This chapter discusses how to estimate the performance of different models in terms of the potential monetary benefit associated with the application of these models in real-time process fault monitoring. Section 3.1 reviews the state of the art in the selection of models for fault monitoring and points out the deficiencies of the conventional approaches. Section 3.2 proposes a net economic benefit-based metric for estimating the performance of fault models. Section 3.3 explains how to evaluate this metric for different combinations of process faults and models intended to monitor these faults.

3.1 State of the Art in Evaluating the Performance of Methods for Fault Monitoring

3.1.1 The Need for Performance-Based Model Selection

Different model types suitable for monitoring the faults forming a set should perform differently when monitoring different faults from this set. For example, the simple methods for fault monitoring, like univariate control charts, are very good at detecting faults characterized by a clear change of the statistical characteristics of a single variable, but they often fail when detecting faults that manifest themselves in small concerted changes of several process variables.

It is not true that complex methods are always better than the simple ones either. Figure 3.1 shows a likely partition of the process variable measurement or residual space by a kernel-based neural network with elliptical kernels, the one presented by Kavuri and Venkatasubramanian (Kavuri and Venkatasubramanian 1993). Figure 3.2 shows how the same space will be partitioned using two univariate X control charts. In both figures, normal operation of the monitored process is inside the dashed ellipsoid and the abnormal operation is outside of it. As seen from the figures, the partition of a residual or measurement space by the control charts is more accurate than the partition of the input space by the neural network.

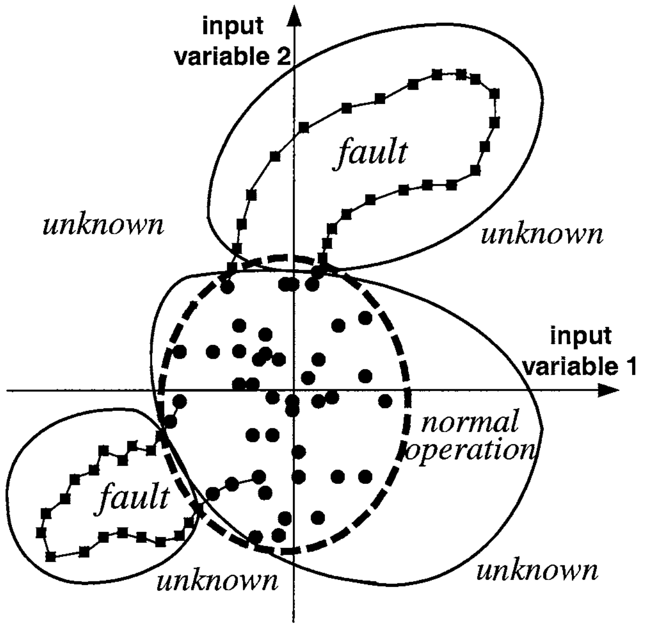


Figure 3.1. Partition of process variable space by a neural network with elliptical kernels

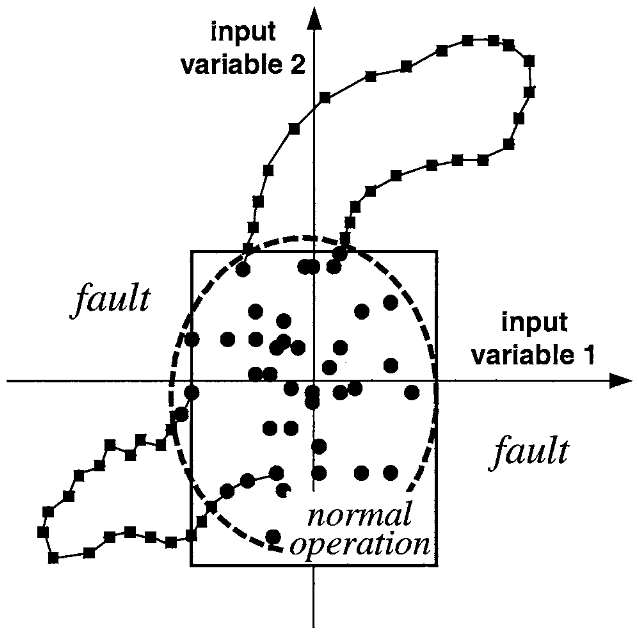


Figure 3.2. Partition of the process variable space by two *X* (Shewhart) charts

In practice, the absolute and even relative performance of a particular class of fault models in monitoring a particular fault is often unknown a priori. Hence, when automating the monitoring of a fault it is important to be able to:

1) Identify fault models of different types,
2) Evaluate the potential performance of these different fault models in monitoring the fault,
3) Select the best-performing model.

3.1.2 Conventional Measures of Fault Model Performance and Their Drawbacks

Several research papers compare the quantitative performance of different types of fault models for fault monitoring and describe how to select the model with the best performance measure. The paper by Kano et al. (Kano and Nagao 2000) measures the performance of fault models using the so-called "reliability index." This index is the ratio of the number of process data points classified as faulty to the total number of process data points sampled during the faulty operation of the process. Two papers (Keyvan, Durg et al. 1993) and (House, Lee et al. 1999) gauged the fault model performance using the percentage of fault instances detected on time by the model. Frank and Ding (Frank and Ding 1997) proposed to measure the performance of fault models based on how far apart the process data points belonging to the normal operation from the data points belonging to the monitored faults are. Finally, the book by Chiang et al. (Chiang, Russell et al. 2001d) used detection lags (i.e., the time from fault inception to fault detection) and misclassification rates (the percentage of times when the classifier confused one fault with another) as measures of fault model performance. Other research papers used similar criteria to evaluate the quality of fault models. While the above performance measures may provide an idea of how good a fault model is, these inducators may not be very useful in practice for the following reasons.

First, some of these performance indicators are applicable only for the fault models based on certain classes of models of the monitored process. For example, the criterion proposed by Frank and Ding (Frank and Ding 1997) assumes that the fault model uses the residuals of a linear model emulating the monitored process. Hence, the performance indicator by Frank and Ding is not suitable for the arbitrary fault model types generated from the template proposed in Section2.3.

Second, use of different measures for selecting fault models may lead to completely dissimilar results. Let us suppose, the fault monitoring performance of two valid fault models: *Model 1* and *Model 2* is measured. Both models try to detect a fault nine times during its existence. *Model 1* correctly detects the fault during six detection attempts and *Model 2* correctly detects the fault during four detection attempts as shown in Figure 3.3. Hence, the error rate of *Model 1* is *1/3* and the error rate of *Model 2* is *5/9*. *Model 2* makes errors more frequently and emulates the fault worse than *Model 1*. However, *Model 2* detected the fault earlier than *Model 1*: *Model 1* detected the fault at *4:20* and *Model 2* detected the fault at *4:00*, hence the detection lag for *Model 2* is shorter than for *Model 1*. In this case, the user would be puzzled which of the proposed performance measures to choose for the selection of fault models.

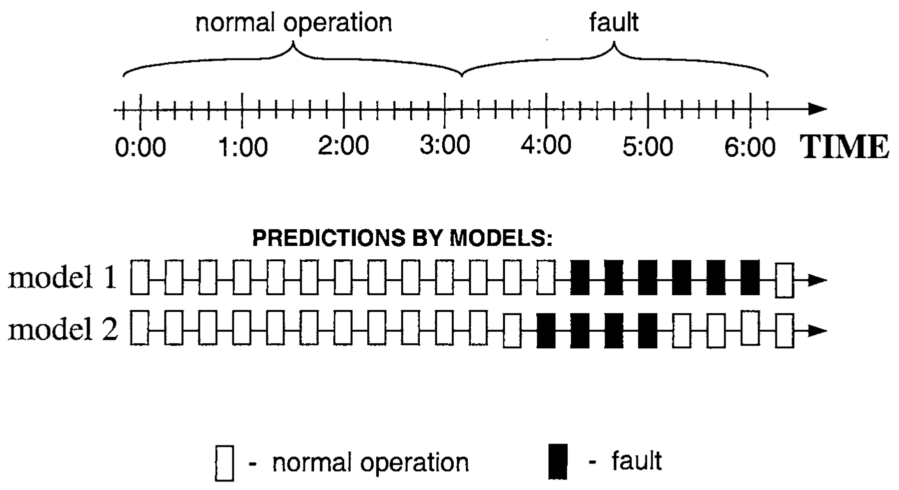


Figure 3.3. Classifications of the process condition (fault or normal operation) by two models

Third, the authors, who evaluated and compared the performance of different fault models in the technical literature, treated the observed measures of fault model performance as deterministic variables. However, the fault model performance measures observed during model tests are random variables affected by a number of factors not accounted for. Such factors may be changes in the quality of the supplies, fluctuations in the ambient conditions, or changes in the monitored process system. As a result, the true performance of a fault model may be somewhat different from the measured one. For example, if a fault model never output false alarms during the test, it does not mean that this model will never tell the user that there is a fault in the monitored process when the process operates normally. In reality, there is still a certain probability that this model will generate false alarms when monitoring the process in real time.

Finally, the conventional measures of fault model performance may not be practical because these measures are some abstract quantities. The relationships of these quantities with the reason for operating the monitored processes, namely, generating as much profit as possible, may seem unclear. The previously mentioned criteria for gauging the performance of fault monitoring are not expressed in terms of economic benefits and costs. The reason is that the conventional methods for economic modeling of continuous processes consider the possibility of faults as constraints that must be met in order to stay in business (Keats, Castillo et al. 1997). Thus, fault monitoring is usually implemented to meet quality operation standards, e.g., the famous Six Sigma (Shina 2002), which sets an upper bound on the probability of out-of-control states, including those caused by faults. This approach completely disregards the fact that, in the process industry, the potential costs of different faults may vary by many orders of magnitude. Fault occurrences may lead only to slight losses in the product output rate or may cause Chernobyl-type disasters. This fact must be accounted for in the evaluation of the performance of fault monitoring.

As was stated in Subsection 2.1.2, the whole purpose of using fault models is to reduce the risk of the losses associated with the fault consequences. Therefore, an applicable measure of model performance should indicate by how much a certain fault model reduces this risk and, more generally, what the incremental net benefit of using a specific fault model for monitoring a specific fault will be. Estimation of such a measure would also give the user an idea whether computerization of fault monitoring using the available methods makes sense or not in each particular case. After selecting the fault model that is expected to add value to the process, the user must also check if the new process setup will meet the quality standards. This procedure is outlined in detail in numerous textbooks and manuals and is outside of the scope of this work.

3.2 Fault Model Performance Metric in a General Form

3.2.1 The Theory of Model Selection

Performance of mathematical models (including those used for real-time monitoring) is their predictive capability. Therefore, as a rule, model performance is calculated in terms of model-data **discrepancy measures**. Discrepancy measures characterize the difference or distance between the actual values of the modeled variables and predictions of these values made by the models. For each set of model arguments, the value of the discrepancy measure is calculated using a problem-specific **loss function** that maps a set of pairs of the predicted and actual values of the modeled variable onto discrepancy measure values. A model with the lowest overall value of the discrepancy measure is considered the best-performing one and thus it should be selected for solving the problem.

Depending on the problem being solved, some discrepancy measures, e.g., the Bayes Risk, may be quantifications of the cost associated with incorrect predictions. The performance measures of this type are called **model performance metrics**. Model performance metrics have a semantically linear nature, i.e., a model with an overall discrepancy measure equal to one is twice as good as a model with an overall discrepancy measure of two. Model performance metrics are preferred over other discrepancy measures. The applicable model performance metric should be used for model selection whenever it can be estimated. Otherwise, the model developer has to come up with a good measure of model-data mismatch to estimate the performance of his or her mathematical model.

3.2.2 The Performance Metric as a Net Benefit of Automating Fault Monitoring

To derive a metric for estimating the performance of different fault models, let us suppose that the user wants to find and utilize a model that monitors fault *X* in real time. The "best" fault model would be the one that adds the maximum possible value to the process (increases the process profitability to the maximum possible extent). Such a model would:

1) Always detect *X* before it causes any undesirable consequences,
2) Produce no false alarms,
3) Never confuse other fault modes with fault *X*,
4) Require no effort to develop and run.

Other requirements to an ideal fault model can be defined, but for now we consider the four listed above to be the most important, base ones and neglect the other possible requirements. In practice, no fault model can be expected to fully satisfy any of these four requirements. In addition, different models will satisfy these requirements to different extents.

As was discussed in Section 3.1.2, evaluation of fault models should be tantamount to the estimation of the potential net economic benefit that the model can add to the monitored process. Then for a fault *X*, the proposed performance metric would be the economic benefit of early detection expected from the model minus the potential costs of an incomplete compliance of with the four base requirements listed previously.

The benefit expected from the automation of monitoring fault *X* is the benefit associated with performing automated detections of fault *X* faster than it takes the operator to detect fault *X* manually. This benefit $B_d(X)$ will be called henceforth the **potential benefit of early detection**. The potential costs associated with the automation of process fault monitoring are:

1) The cost $C_{fa}(X)$ of the false alarms warning the user of the existence of fault X during the normal operation of the monitored process,
2) The cost $C_m(Y,X)$ associated with the possibility of misclassifying other faults Y as fault X,
3) The cost $C_{dm}(X)$ of developing and running a fault model to monitor fault X.

The cost associated with misclassifications of other faults as fault X has two summands: the first one is the cost of a single false alarm of fault X produced in the absence of fault X and in the presence of fault Y and the second summand is the cost of not detecting fault Y on time. The cost $C_m(Y,X)$ includes only the first summand (the cost of a single false alarm) as the second summand is accounted for in the benefit $B_d(Y)$ of early detection of fault Y. For most faults, the cost of a single false alarm should be considered constant regardless of the occurrence of other faults, i.e., $C_m(Y,X)=C_{fa}(X)$ for any $Y \neq X$. The cost of developing and running a fault model to monitor fault X includes the labor that must be spent to develop the model, the expenses associated with the hardware that runs model M, and the cost of collecting additional process measurements (if required). In this work, this cost is the same for all the models because they are all trained on the same labeled (tagged) historical data. The proposed benefit and cost measures are calculated per unit time of fault model operation.

Costs and benefits are semantically linear measures. A fault-incurred cost of two units is two times as bad as a fault-incurred cost of one unit. Moreover, suppose we need to monitor a single fault that occurs once a year. For this case, a model that always detects the monitored fault immediately after its occurrence, but every month issues false alarms that cost *\$500* per alarm is as good as a model that never produces any false alarms, but detects the fault late and the average cost of this lateness is *\$6000* per fault. Thus, for fault X, the benefit and cost-based model performance metric would be the potential net benefit $B(X)$ of automating the monitoring of fault

X:

$$B(X)=B_d(X)-C_{fa}(X)-C_m(X)-C_{dm}(X) \tag{3.1}$$

It is not quite correct to call the potential net benefit *B*(*X*) a "model-data discrepancy" because the net benefit is lower for the models that fit the fault data poorly and is higher for the models that fir the fault data well. In addition, the net benefit may be both positive and negative. However, the proposed measure *B*(*X*) does have the same properties as any other performance metric. The proposed net benefit measure does gauge the model-data fit, and it does quantify the cost associated with incorrect predictions, same as the mean square error and other similar criteria applicable in the simpler cases.

If the net benefit *B*(*X*) is negative or only slightly above zero for all the tested fault models, automation of the fault monitoring that uses any of these models is not justified unless model-based monitoring is absolutely necessary to meet the process quality standards. For the rare cases when a single model *M* is set up to monitor n_f different fault modes, the net benefit of implementing this model will be a sum of the net benefits calculated for each fault monitored by model *M* and the cost $C_{dm}(X)$ of developing and running fault model *M* to monitor fault *X* is calculated as a fraction of the cost $C_{dm}(M)$ to develop and run *M*:

$$C_{dm}(X)=\frac{C_{dm}(M)}{n_f} \tag{3.2}$$

3.2.3 Expectation of the Fault Model Performance Metric

3.2.3.1 Estimating Expectations of Model-Data Discrepancies Model-data discrepancy measures cannot be calculated in a straightforward manner because the actual values of the modeled variables are not known for all the circumstances where the model will be used. Otherwise, there would be no need to use any mathematical models. Because of that, it is often assumed that the best model is the one whose statistically estimated expectation (denoted by the operator $\hat{E}$) of the applicable model-data discrepancy is the optimal. Estimation of the discrepancy expectation for a model requires running the model on a sample of object data consisting of the values of the model inputs (prototypes) and the associated true values of the modeled variable. After the model runs, an "unbiased" statistical estimator (i.e., an estimator whose expectation is equal to the expectation of the estimated variable) of the applicable discrepancy measure maps a set of pairs of the actual and model-predicted values of the modeled variable onto the discrepancy expectation.

In the most common case, these estimators simply average the loss function values calculated for each point of the sample. For example, when the sample is random and the loss function is the square of the distance between the actual and model-predicted values, the discrepancy measure is estimated as the average squared error for the sample. The simple "averaging" estimators are not applicable when the data sample used for model evaluation is not random. This is the case in the evaluation of fault models because selection of historical process data for fault model evaluation at random will not produce a good estimate of how the model will perform in the future. To carry out this evaluation, the operators must select and mark the non-random historical data describing the monitored fault(s) from fault inception to fault rectification. This data must not be omitted because, obviously, fault model behavior during the occurrence of the monitored fault matters most in the evaluation of the model performance.

3.2.3.2 Expanding the Summands of Equation (3.1) Since the historical process data describing faults and normal operation are sampled using different schemes, the expectation of the net benefit of fault monitoring automation $B(X)$ should be estimated as a function of indicators characterizing the model performance only during the occurrence of faults or only during the normal operation. Thus, to obtain an estimator for the model performance metric, the summands of Equation (3.1) should be expanded and the resulting factors classified as random and fixed. Each random factor should describe the model performance during a specific monitored mode of the process operation. For each fault model, each of these factors should be evaluated separately.

The summands of Equation (3.1): the costs and benefits B_d, C_{fa}, and C_m are associated with specific process and model operation events. The benefit $B_d(X)$ is associated with occurrences of fault X, the cost $C_{fa}(X)$ is associated with incorrect warnings of fault X produced by the fault model, the cost $C_m(Y,X)$ is associated with misclassifications of fault Y as fault X by the fault model. These costs and benefits are naturally viewed as products of the average cost or benefit of a single instance of the associated event and the rate of the occurrence of the events of this type during the model operation.

Thus, the potential benefit of early detection of fault X can be calculated as a product of the frequency $r_f(X)$ of occurrences of fault X and the average benefit $\overline{B}_{dX}$ associated with early detection of a single instance of fault X:

$$B_d(X) = r_f(X) \cdot \overline{B}_{dX} = r_f(X) \frac{1}{n_X} \sum_{j=1}^{n_X} \{poslin[L_{du}(X_j) - L_{dm}(X_j)]\} \tag{3.3}$$

In Equation (3.3), $r_f(X)$ is the frequency of occurrences of fault X, $L_{du}(X_j)$ is the cost of the consequences of fault instance X_j when X_j is detected manually by the operator, $L_{dm}(X_j)$ is the cost of the consequences of X_j if X_j is detected by the model, and n_X is the number of historicized instances of fault X. The benefit associated with the detection of a fault instance early using a fault model is estimated as a difference between these costs. Operator *poslin*() performs an identity mapping for positive arguments and returns zero for the other arguments. Equation (3.3) states that the benefit of early detection is zero for the fault instances detected by the operator ahead of the fault model. Evaluation of $L_{du}(X_j)$ and $L_{dm}(X_j)$ using user and model detection lags is described in detail in Subsection 3.3.2.2.

The cost associated with false warning of fault X can be viewed as a product of the frequency $r_{fa}(X)$ of false warnings of fault X produced by the model during normal operation and the average cost $\overline{L}_{fa}(X)$ of a single false warning of fault *X* produced by the fault model during normal operation:

$$C_{fa}(X) = r_{fa}(X) \cdot \overline{L}_{fa}(X) \tag{3.4}$$

The cost associated with misclassifications of fault *Y* as *X* can be considered a product of the average cost $\overline{L}_{fa}(Y,X)$ of a single false warning of fault *X* during the existence of fault *Y*, the frequency $r_f(Y)$ of occurrences of fault *Y*, and the time from fault inception to fault detection of instance Y_i of fault *Y*, $l(Y_i)$, multiplied by the frequency $r_{fa}(Y_i,X)$ of false warnings of fault *X* produced by the model during the existence of fault instance Y_i.

$$C_m(Y,X) = \overline{L}_{fa}(Y,X) \cdot r_f(Y) \cdot \frac{1}{n_X} \sum_{i=1}^{n_X} \left[l(Y_i) \cdot r_{fa}(Y_i,X) \right] \tag{3.5}$$

Using the expansions proposed in Equations (3.3)-(3.5), the net benefit of automating fault monitoring is represented as follows:

$$\begin{aligned} B(X) = {} & r_f(X) \cdot \frac{1}{n_X} \sum_{j=1}^{n_X} \left\{ poslin\left[L_{du}(X_j) - L_{dm}(X_j) \right] \right\} - r_{fa}(X) \cdot \overline{L}_{fa}(X) \\ & - \sum_{Y, Y \neq X} \left\{ \overline{L}_{fa}(Y,X) \cdot r_f(Y) \cdot \frac{1}{n_Y} \sum_{i=1}^{n_Y} \left[l(Y_i) \cdot r_{fa}(Y_i,X) \right] \right\} - C_{dm}(X) \end{aligned} \tag{3.6}$$

Among the variables on the right hand side of Equation (3.6), some variables should be considered fixed while the others should be deemed random.

3.2.3.3 Fixed and Random Variables in Equation (3.6) Even though the occurrences of process faults of types *X*, *Y*, ... are random events, the frequencies of these events ($r_f(X)$, $r_f(Y)$, ...) should be hard set by dividing the number of times N_Z that the fault of type *Z* has been observed by the total time t_{PO} the plant has been in operation:

$$r_f(Z) = \frac{N_Z}{t_{PO}} \tag{3.7}$$

Following the industrial engineering practice, the distribution of the time between faults of the same type can be approximated with the distribution functions, like Exponential, Weibull, Lognormal, etc., commonly used in the survival and reliability analyses. Consequently, fault frequencies can be approximated as the expectations of the distributions of the inverse survival functions (inverse Exponential, inverse Weibull, inverse Lognormal) obtained by inverting the approximations of the time between faults of the same type.

In the reality, the rate of fault occurrences in the process industries is a function of the plant age, maintenance schedule, and current costs of the products generated and raw materials used by the plant. Hence, the survival function approximations proposed here would only represent marginal distributions of the fault rates averaged over the plant life to date, but these approximations would be grossly inaccurate for each particular moment of plant operation. In addition, for any process plant, the number of faults of the same type observed during the plant history is often rather small and not sufficient to accurately approximate any joint distributions of the time between faults, time since last maintenance, plant age, and commodity costs. The very rare cases when process faults of the same type are frequent enough to enable the user to find a good approximation of a joint distribution of the time between faults, plant age, and time since last maintenance are beyond the scope of this work.

The user is also expected to specify the average cost $\overline{L}_{fa}(X)$ of a single false alarm mistakenly warning the user of fault X and the average costs $\overline{L}_{fa}(Y,X)$ of a single misclassification of each fault Y as fault X. Just like the rates of the monitored process faults, the actual costs of fault misclassifications and false alarms depend on the time since the last maintenance, the plant age, the costs of the products generated and raw materials used by the plant at the time of a false alarm or fault misclassification. In addition, these costs depend on the experience of the plant personnel and other human factors changing with time. The cost of fault misclassification may also depend on the time since the inception of the misclassified fault.

Accounting for all these factors may be a challenge for the future, but at this point in time, it is unrealistic to expect either the operation personnel or plant management to come up with the false alarm and misclassification costs as functions of these factors. Hence, the costs of a single false alarm and a single fault misclassification of each type should be specified as fixed deterministic numbers.

Finally, the cost $C_{dm}(X)$ of developing and running the fault model to monitor fault X is a fixed deterministic value for each "fault"-"fault model" pair. This cost can be estimated from prior experience.

The remaining variables on the right hand side of Equation (3.6): $L_{dm}(X_j)$, $L_{du}(X_j)$, $r_{fa}(X)$, $l(Y_i)$, and $r_{fa}(Y,X)$ are viewed as random. One reason for this is that these variables are affected not only by the monitored fault type and the type of the model used to detect the fault, but also by a great number of other factors that cannot be accounted for, e.g., the varying composition of the feeds or changes in ambient conditions. Another reason is that these variables are not hard set by the user and random samples of the values of these variables can be obtained. Because of that, it is possible to estimate statistical characteristics of these five variables.

With the assumptions listed in this subsection, the expectation of fault model performance metric: the net benefit of monitoring automation can be calculated by applying the expectation operator to both sides of Equation (3.6) as follows:

$$
\begin{aligned}
&\hat{E}[B(X)]= \\
&\hat{E}\left(\begin{array}{l} r_f(X)\cdot\sum_{j=1}^{n_X}\left\{poslin\left[L_{du}(X_j)-L_{dm}(X_j)\right]\right\}-r_{fa}(X)\cdot\overline{L}_{fa}(X) \\ -\sum_{Y,Y\neq X}\left\{\overline{L}_{fa}(Y,X)\cdot r_f(Y)\cdot\sum_{i=1}^{n_Y}\left[l(Y_i)\cdot r_{fa}(Y_i,X)\right]\right\}-C_{dm}(X) \end{array} \right) \\
&=\frac{N_Z}{t_{PO}}\cdot\hat{E}\left\{poslin\left[L_{du}(X_j)-L_{dm}(X_j)\right]\right\}-\overline{L}_{fa}(X)\cdot\hat{E}\left[r_{fa}(X)\right] \\
&-\sum_{Y,Y\neq X}\left\langle r_f(Y)\cdot\overline{L}_{fa}(Y,X)\cdot\left\{\hat{E}\left[l(Y_i)\right]\cdot\hat{E}\left[r_{fa}(Y,X)\right]+c\hat{o}v\left[l(Y_i),r_{fa}(Y,X)\right]\right\}\right\rangle-C_{dm}(X)
\end{aligned}
\tag{3.8}
$$

Then, as proposed at the beginning of Subsection 3.2.3.2, estimation of the expectation of fault model performance metric would require:

1) Estimation of the expectation of the benefit of early detection of a single fault instance $E[B_{dX}]=E\{poslin[L_{du}(X_j)-L_{dm}(X_j)]\}$ from the historical data describing the monitored fault X,
2) Estimation of the expectation of the rate of false alarms $E[r_{fa}(X)]$ from the historical data recorded during the normal operation of the monitored process,
3) Estimation of the expectation $E[l(Y_i)]$ of the duration of a single fault instance Y_i of fault *Y*, the expectation $E[r_{fa}(Y,X)]$ of the rate of false alarms during fault *Y*, and the covariance $cov[l(Y_i),r_{fa}(Y,X)]$ of the duration of Y_i and the rate of false alarms during fault *Y* from the historical data describing the faults other than the monitored fault *X* (multiple-fault case).

Since this work is the first one to propose a general method for evaluating the net benefit of automating the process fault monitoring, it is assumed that faults of only one type are possible in the monitored process. If the results of this work are found applicable, future research will consider estimation of the net benefit for more complex cases. For the case when faults of only one type are possible in the process, $r_{fa}(X)=0$, $C_m(Y,X)=0$, and Equation (3.8) simplifies to:

$$\hat{E}[B(X)]=\frac{N_X}{t_{PO}}\cdot\hat{E}\{poslin[L_{du}(X_j)-L_{dm}(X_j)]\}-\overline{L}_{fa}(X)\cdot\hat{E}[r_{fa}(X)]-C_{dm}(X) \qquad (3.9)$$

3.3 Estimating the Expectation of the Fault Model Performance Metric

3.3.1 Sampling the Historical Data to Identify and Evaluate Fault Models

This section describes how to estimate the proposed performance metric for the data-driven fault models. "**Data-driven models**" are those whose identification requires a set of values of the model inputs (prototypes) and the associated true values of the modeled variable. Fault models are almost always data-driven. The models that require no data for the parameter identification are beyond the scope of this work.

The object data used for model identification cannot be reused for estimating the model discrepancy expectation. The reason is that any data-driven model is expected to imitate (fit) the data used for its identification better than the rest of the object data. There are two conventional approaches to sample the available data and estimate the discrepancy expectations for the data-driven models (Linhart and Zuccini 1986).

The first conventional approach is **complexity penalization**. In this approach, the discrepancy expectation is estimated implicitly or explicitly as a sum of the model-data discrepancy due to estimation and the model-data discrepancy due to approximation. The first discrepancy decreases as the observed goodness-of fit of the training data by the model and the model complexity increase. At the same time, the second discrepancy gets larger following increases in the number of the model parameters or the complexity of the basis function shape. The model discrepancy measure is evaluated as a sum or product of these two discrepancies. This approach finds the best compromise between the models that are too simple and inaccurate and the models that are too complex and that over-fit the training data. One famous use of the complexity penalization approach for estimating the performance of simple models consists in the evaluation of the Akaike Information Criterion (Akaike 1973). Complexity penalization is hard to apply in the evaluation of fault models because they are combinations of different mathematical methods and because of the complexity of the profit-based performance measure proposed for these models.

When calculation of discrepancy measures due to the estimation and approximation is complicated, another approach called "**minimization of the generalization error**" is used. This minimization is performed consistently with the "bootstrap" and "cross-validation"-type methods. The **bootstrap**-based methods select several random independent samples of fixed size out of the object data. Each sample is divided into two parts of fixed size. The data in one part of the sample must be independent on the data in the other part. For each sample, the parameters of a data-driven model are identified using the first part of the sample and then the average model-data discrepancy is calculated for the elements of the second part of the sample. The model's discrepancy measure is evaluated as a function of the average model-data discrepancies for the second parts of each selected sample.

The bootstrap-based methods work well when the pool of the data available for model identification and evaluation is large. This is the case with the data describing normal operation of the monitored process. The data collected during this mode of process operation is used to evaluate the expectation of the rate of false alarms $\hat{E}[r_{fa}(X)]$. At the same time, the bootstrap-based methods are not applicable for the evaluation of the other four statistical quantities listed at the end of Section 3.2.3. The reason is that these quantities require the historical data describing faults. This type historical data is usually very limited. The operator can generally be expected to label only a very limited number of instances of the monitored fault and several instances of the other faults in the historical data. Each fault instance is unique and different from other fault instances, plus the process data points describing a single fault instance are collected during a relatively small time interval and hence related to each other. In addition, evaluation of the cost of fault consequences during the automated fault monitoring requires full historical data sets describing each fault instance. The full historical data sets are needed to find out when the fault model would have detected the fault instances used for fault evaluation. These facts seriously complicate creation of two independent random samples of historicized fault data, as required for the bootstrap.

The **cross-validation**-type methods are based on several repeated divisions of the entire set of the modeled object's data into two independent fixed-size parts: the data set for model identification and the holdout data set for calculating the average model-data discrepancy. The divisions are performed in such a way that each available object data point is included in the holdout data set only once. The discrepancy expectation is evaluated as the average over the model-data discrepancies calculated for each division.

The cross-validation-type methods are good when the pool of the data available for model identification and evaluation is small. As was discussed, this is typically the case with the historical data describing faults. If the data describing each fault instance is assumed independent of the other historicized fault instances, the cross-validation can identify several fault models using all the historicized fault instances except for a randomly chosen one placed in the holdout data set. If an instance of the monitored fault *X* is held out, this instance would be used for the evaluation of the benefit of early detection. In addition, the training set (the set of data prototypes) should include randomly sampled sections of the data historicized during the normal operation of the monitored process. The proposed sampling scheme is illustrated in Figure 3.4 where solid rectangles denote the process data historicized during the normal operation and the white rectangles denote the data describing process faults.

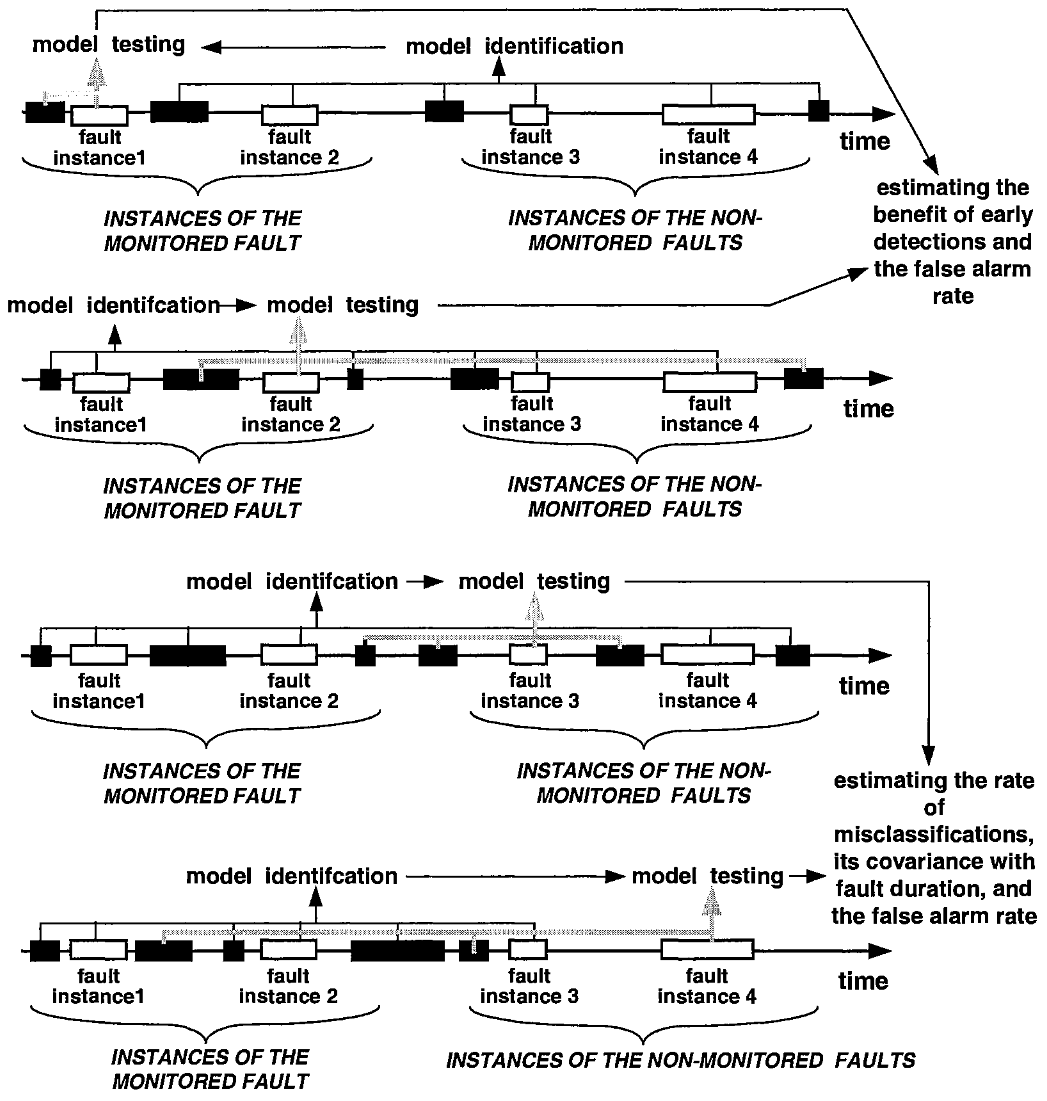

model testing
model identification
fault instance1
fault instance 2
fault instance 3
fault instance 4
time
INSTANCES OF THE MONITORED FAULT
INSTANCES OF THE NON-MONITORED FAULTS
estimating the benefit of early detections and the false alarm rate
model identifcation → model testing
model identifcation → model testing
estimating the rate of misclassifications, its covariance with fault duration, and the false alarm rate
model identifcation → model testing

Figure 3.4. The sampling scheme proposed for fault model identification and evaluation

3.3.2 Estimating the Random Quantities of the Model Performance Metric

3.3.2.1 Types of Fault Consequences. The total damage caused to a process by a process fault can be viewed as an aggregate of the different effects resulting from each fault consequence. With respect to the consequence dynamics, fault consequences can be divided into:

1) Fixed-cost consequences

2) Accumulating-cost consequences.

Each process fault may be associated with a set of consequences of different types.

If a fault is associated with **fixed-cost consequences**, these consequences cannot be avoided once the fault occurs. As suggested by their name, correction of fixed-cost consequences costs a fixed price (effort). For example, a broken valve actuator or compressor impeller will require a fixed amount of cash to purchase a replacement, a fixed amount of labor to replace the actuator (impeller), and a fixed amount of process downtime (if applicable) to perform the replacement. Real-time fault monitoring does not affect the fixed-cost consequences.

The impact of the faults associated with the **accumulating-cost consequences** grows from the moment of fault inception until the fault rectification. With some simplifications, the accumulating-cost consequences can be further subdivided into:

a) Consequences with the costs accumulating at a **constant finite** rate,

b) Consequences with the costs accumulating at a **variable finite** rate,

c) Delayed consequences with the costs accumulating **in one discrete step**.

Common examples of a consequence with the costs accumulating at a constant finite rate are the manufacturing of a substandard product that will be sold at a discounted price or leak-related losses of utilities or chemical reagents. For a process operated out-of-control due to a fault, the rates of product, reactant, and utility losses are usually finite and variable. The most common example of a fault-related loss accumulating in one discrete step is process shutdown due to an out-of control operation caused by faults. Certain process variables must be within the specified limits. When a process is operated out-of-control, the values of these variables may move outside of these limits, as a result, the control system may force the process to shut down. Other examples of a fault-related loss accumulating in one discrete step are failures of the process equipment, conflagrations, explosions and sudden releases of toxic gases or fluids caused by process faults. Figure 3.5 below illustrates how the costs of different consequences of a fault may accumulate.

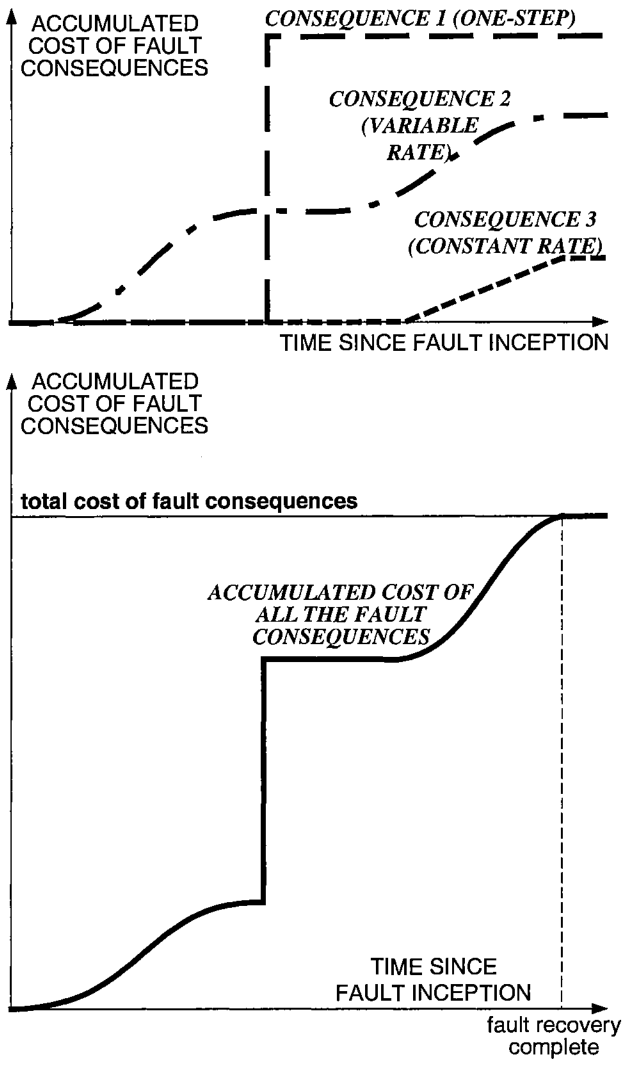


Figure 3.5. Accumulated cost of fault consequences as a function of time since fault inception

3.3.2.2 Tracing the Dynamics of Fault Consequences in the Historical Data

Evaluation of the expectation $E[B_d(X)]$ of the benefit of detecting fault X early requires the operator to specify the total cost $L_{du}(X_j)$ of the consequences associated with each historicized and labeled instance X_j of the monitored fault X. In addition, for each historicized and labeled fault instance X_j, it is necessary to estimate the imaginary cost $L_{dm}(X_j)$ of the associated consequences were these fault instances detected by the evaluated model. Then the expectation of the benefit

$B_d(X_j)$ of early detection of a single instance of the monitored fault X can be estimated as follows:

$$\hat{E}[B_{dX}]=\hat{E}\{poslin[L_{du}(X_j)-L_{dm}(X_j)]\}=\frac{\sum_{j=1}^{n_X} poslin[L_{du}(X_j)-L_{dm}(X_j)]}{n_X} \tag{3.10}$$

In Equation (3.10), n_X is the total number of historicized and labeled instances of fault X. The mean of a sample is an unbiased estimator for the mean of the population from which this sample was obtained. This equation neglects the possibility of irreversible changes in the monitored process and treats all the historicized fault instances equally.

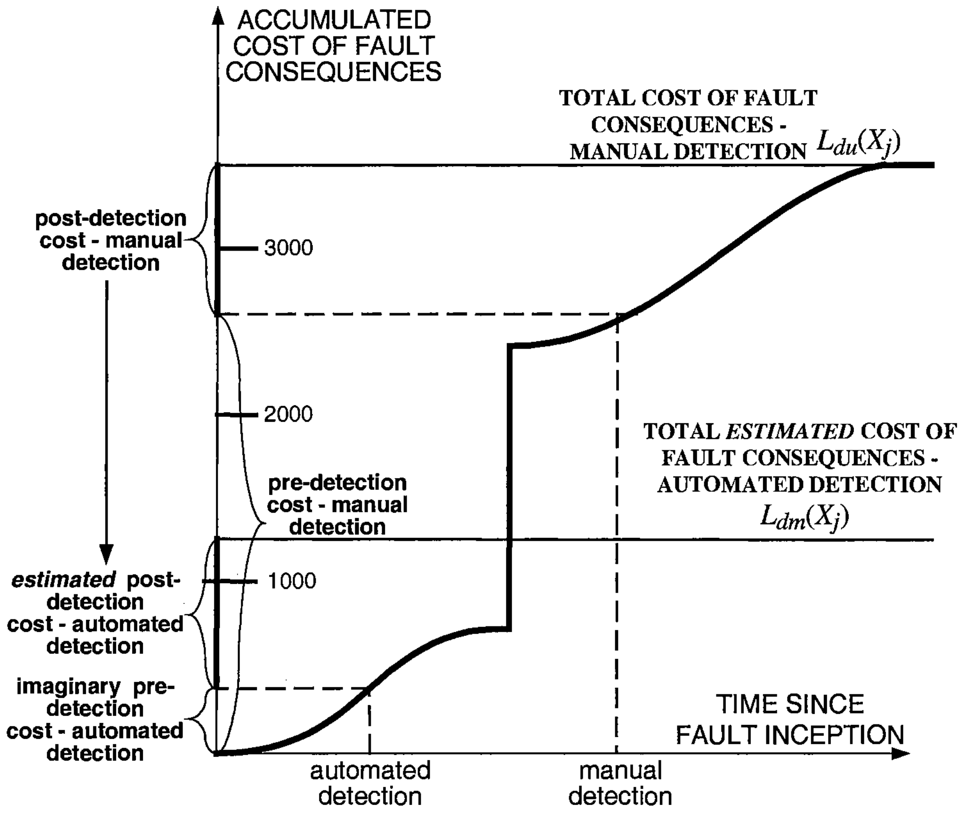
It is proposed to evaluate $L_{dm}(X_j)$ by recovering the dynamics of the consequence cost accumulation for the historicized instances of the monitored fault(s). To accomplish this task, the operator must be able to find in the historical data and mark the moments of time when fault consequences with the costs accumulating in one discrete step have occurred. It is assumed that the operator can also specify the rates of cost accumulation for the fault consequences whose costs grew at finite rates during the historicized faults. If the consequence costs accumulated at variable rates, these rates must be specified for each point of the historical process data describing the monitored fault. This procedure can be automated if the rates of consequence cost accumulation are specified as functions of the historicized process variables. For example, if a fault results in a loss of a reagent, the rate of cost accumulation associated with the reagent loss would be calculated as the difference between the base-case and the historicized rate of the reagent consumption by the process multiplied by the cost of this reagent. The functions relating the costs of different types of fault consequences should be collected in a library. From this library, the operator would pick functions that describe the consequences observed during the historicized fault instances and identify the parameters of these functions according to the existing plant records.

Once the operator provides the historical information on the fault consequences, it is possible to recover the accumulated cost of the consequences of the historicized faults as a function of time. Using the automated (model-based) detection lag as an argument of this function returns the consequence cost that would have been accumulated by the imaginary moment of automated detection. To determine the detection lag t_{DL} for a model-fault instance pair, the moment t_0 of this fault instance inception should be subtracted from the time stamp t_D of the latest process measurements used by the model when the fault was detected. This difference should be further augmented by the time t_P it would take to pass the process data from the sensors or analytical devices to the computer running the fault model and by the time t_C it would take the fault model to perform a single classification of a process data vector plus the average time t_B between model runs:

$$t_{DL} = t_D + t_P + t_C + t_B - t_0 \tag{3.11}$$

Of course, this procedure is valid only for the historicized fault instances placed in the holdout set and hence not used for the model identification.

3.3.2.3 Estimating the Post-Detection Cost. Knowing the consequence cost accumulated by the imaginary moment of fault detection is not sufficient to estimate the total cost $L_{dm}(X_j)$ of the consequences of fault instance X_j if this fault instance were detected by the evaluated model. Unfortunately, faults are rarely corrected instantaneously, so the consequence costs continue accumulating after the detection as well. Estimation of the cost that would have been accumulated after the model-based fault detection for the historicized fault instances is a difficult task. To make this task tractable, the operator must mark the moments when the historicized instances of the monitored faults were detected manually and fault rectification sequences initiated. Then $L_{dm}(X_j)$ can be calculated as a sum of the consequence cost that would have been accumulated until the model-based fault detection and the consequence cost accumulated after the manual detection of fault instance X_j as shown in Figure 3.6 below. This method for estimating $L_{dm}(X_j)$ carries with it a very strong assumption that for each historicized fault instance the cost accumulated after fault detection does not depend on the detection lag. The estimates of $L_{dm}(X_j)$ obtained using this method may be very crude.



ACCUMULATED COST OF FAULT CONSEQUENCES
TOTAL COST OF FAULT CONSEQUENCES - MANUAL DETECTION $L_{du}(X_j)$
post-detection cost - manual detection
3000
2000
1000
pre-detection cost - manual detection
TOTAL *ESTIMATED* COST OF FAULT CONSEQUENCES - AUTOMATED DETECTION $L_{dm}(X_j)$
estimated post-detection cost - automated detection
imaginary pre-detection cost - automated detection
TIME SINCE FAULT INCEPTION
automated detection
manual detection

Figure 3.6. Estimating the consequence cost for the automated detection by assuming the cost accumulated after the detection does not depend on the detection lag

Two approaches for refining the estimation of the imaginary post-detection cost for the case of automated detection are proposed here. In the first approach, the operator approximates the cost accumulated after fault detection (no matter manual or automated) using a relationship known from the first principles of the operation of the monitored process. The operator may know that the post-detection cost does not depend on the detection lag or that certain consequences will not occur after the start of the fault rectification sequence, or that certain consequences will stop accumulating once the fault recovery begins. It is unreasonable to expect the operator to enter any complex functions when identifying fault models. Instead of that, the operator would create a simple table specifying the intervals of detection lags, accumulated consequence costs, occurrence of different consequences, or other process variables and providing either one or more historicized fault instances falling into each specified interval or the expected value of the post-detection consequence cost. For the generic fault instance whose consequence cost dynamics is shown in Figures 3.5 and 3.6, the cost of consequences may be specified as follows. If the consequence cost accumulated before detection is less than \$500 or if *Consequence 1* has occurred, the cost of the post-detection consequences is \$900 with the 95% confidence interval of (\$700, \$1100). Otherwise, the post-detection consequence cost is \$1700 with the 95% confidence interval of (\$1300, \$2100) because in this case *Consequence 1* is not likely to be avoided.

The second proposed approach can be considered empirical and it is used when the operator does not know how the post-detection consequence cost is related to other fault and process parameters. The first step in the second approach is finding which variable has a greater correlation with the post-detection cost: the consequence cost accumulated by the manual fault detection or the manual detection lag. For the faults that can develop at variable rates or are associated with the consequences whose costs accumulate in one single step, the former variable is usually related closer to the consequence cost accumulated after the fault detection than the latter one. For the other faults, the post-detection consequence cost is usually determined by the detection lag except for the cases when the fault inception time cannot be determined accurately. Thus, one advantage of the proposed fault model evaluation method is that it does not require an accurate identification of the time of fault inception to evaluate the performance of fault models.

The second step is dividing the more correlated variable into frequencies spanning all the values that this variable can take for the monitored fault. Each frequency contains at least three detection lags or pre-detection costs, depending on which variable is related closer to the post-detection cost, for three different instances of the monitored fault. The boundaries between the frequencies are averages between the smallest observed value contained in the higher frequency and the largest observed value contained in the lower frequency.

At the third step, the conditional expectation of the consequence cost $L_{dma}(f)$ accumulated after the detection is estimated for each frequency f as the average of the post-detection consequence costs $L_{dma}(X_j)$ for the manual monitoring of the N_f historicized fault instances X_j whose manual detection lag or the consequence cost accumulated before the manual detection belongs to that frequency:

$$\hat{E}[L_{dma}(X_i)|X_i \in f] = \frac{\sum_{\substack{j=1 \\ X_j \in f}}^{N_f} L_{dma}(X_j)}{N_f} \tag{3.12}$$

The same relation is used to calculate the expectation of the post-detection cost for interval specified by the user in the first approach. This idea is visualized in Figure 3.7 below.

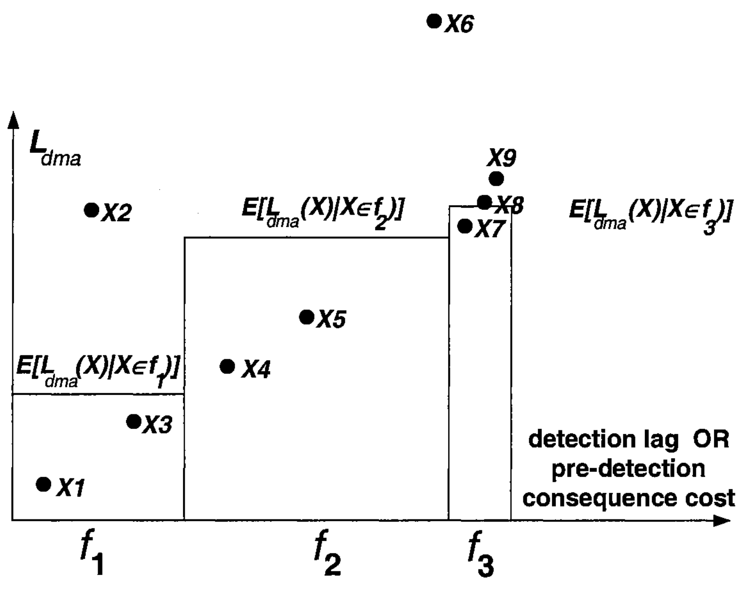

●X6
L_{dma}
●X2
$E[L_{dma}(X)|X\in f_2)]$
X9
●
●X8
●X7
$E[L_{dma}(X)|X\in f_3)]$
●X5
$E[L_{dma}(X)|X\in f_1)]$
● X4
●X3
●X1
detection lag OR pre-detection consequence cost
f_1
f_2
f_3

Figure 3.7. Approximation of the conditional expectations of the pre-detection consequence cost

The expectation of the post-detection consequence cost is assumed to be the same function of the detection lag or pre-detection consequence cost for both manual and automated detection. Then, as required by Equation (3.10), the total consequence cost $L_{dm}(X_j)$ for the case of automated detection of fault instance X_j is estimated as a sum of:

1) The imaginary pre-detection consequence cost $L_{dmb}(X_j)$ for the model-assisted monitoring estimated using the cost dynamics AND
2) The estimated expectation of the post-detection cost $L_{dma}[f(X_j)]$ for the frequency $f(X_j)$ of the model-based detection lags or the imaginary pre-detection consequence costs for the fault instance X_j.

$$L_{dm}\left(X_j\right)=L_{dmb}\left(X_j\right)+\hat{E}\left[L_{dma}\left(X_j\right)\middle|X_j\in f\right] \tag{3.13}$$

The final form of the estimator for evaluating the expectation of the benefit $B_d(X)$ of early detection of fault X will then be as follows:

$$\hat{E}\left[B_d\left(X\right)\right]=\frac{\sum_{j=1}^{n_X} poslin\left\{L_{du}\left(X_j\right)-L_{dmb}\left(X_j\right)-\hat{E}\left[L_{dma}\left(X_j\right)\middle|X_j\in f\right]\right\}}{n_X}\cdot\frac{N_X}{t_{PO}} \tag{3.14}$$

This equation tacitly assumes that *poslin* is a linear function. This is not true for negative values of the argument of *poslin*. However, the well-performing fault models should almost always detect faults ahead of the operator, thus the argument of the *poslin* used in Equation (3.14) should be positive for the overwhelming majority of instances of the monitored fault.

3.3.2.4 Confidence Interval for the Benefit of Early Detection. Often, it is important to know how accurate an estimation of the net benefit of fault monitoring automation is for each evaluated fault model. For the model performance metric proposed in this chapter, it is possible to estimate an approximate standard deviation and a *95%* or *99%* confidence interval. Estimation of this standard deviation and confidence interval would require estimation of these statistical quantities for each random variable included in the metric on the right hand side of Equation(3.9).

For the benefit of early detection it is proposed to estimate an approximate standard deviation as follows. The user specifies standard deviations SC_i for the cost L_i of each consequence C_i. This specification may be performed implicitly, e.g., the operator may state that the cost of a plant shutdown is *\$100,000±20,000* or that the cost of one shutdown is estimated with a *20%* accuracy. This would mean that the shutdown cost has a normal distribution with the mean of *\$100,000* and that the value of the shutdown cost falls into the (*\$80000*; *\$120000*) interval with the probability of *95%*, so the standard deviation for the shutdown cost distribution would be \$*20,000*. Assuming independence of the operator's errors for the cost of each consequence, the variance for the estimated cost of all N_{Cj} consequences actually incurred by fault instance X_j would be approximately estimated as follows:

$$S^2\left[\hat{L}_{du}\left(X_j\right)\right]=\sum_{i=1}^{N_{Cj}} S^2\left(L_{tij}\right) \tag{3.15}$$

In Equation (3.15), S is the standard deviation operator, S^2 is the variance operator, and L_{tij} is the total cost of consequence C_i for fault instance X_j.

The same procedure can be applied to evaluate the standard deviation of the estimated imaginary cost accumulated by the imaginary time of automated fault detection. However, this source of variance can be neglected because for good models that detect faults much earlier than the operator does, the variance of this indicator is much smaller than the variance due to an incorrect specification of the total cost of fault consequences.

As far as the conditional expectation $\hat{E}\left[L_{dma}\left(X_i\right)\middle| X_i\in f\right]$ of the post-detection consequence cost for the imaginary automated detection of fault instance X_j is concerned, its variance can be estimated as the variance of the mean of the sample of the measured post-detection costs for the frequency of detection lags or pre-detection costs where the detection lag or pre-detection cost of X_j belongs:

$$S^2\left\{\hat{E}\left[L_{dma}\left(X_j\right)\middle| X_j\in f\right]\right\}=\frac{S^2\left\{L_{dma}\left[f\left(X_j\right)\right]\right\}}{N_f}=\frac{\sum_{l=1}^{N_f}\left(L_{dma}\left(X_l\right)-\hat{E}\left\{L_{dma}\left[f\left(X_j\right)\right]\right\}\right)^2}{N_f\left(N_f-1\right)} \tag{3.16}$$

In this equation, N_f is the number of historicized fault instances placed in frequency f (see previous section), X_l is an instance of fault X placed in the frequency f, and $S^2\{L_{dma}[f(X_j)]\}$ is the variance of the sample of post-detection consequence costs for the manual detection case and the fault instances included in frequency f.

Equations (3.15) and (3.16) describe the variance in the expectation of the net benefit due to the **specification** error. Another source of variance in this expectation is due to the differences in the **measured** pre-detection cost L_{bk} for the model-based detection for different instances of the same fault estimated as follows:

$$S^2(L_{bk}) = \frac{\sum_{j=1}^{n_X}\left[L_{bkj} - \frac{1}{n_X}\sum_{j=1}^{n_X}(L_{bkj})\right]^2}{(n_X - 1)} \tag{3.17}$$

In Equation (3.17), L_{bkj} is the measured total cost of the consequences of fault instance X_j accumulated by the imaginary time of the automated detection of X_j and n_X is the total number of instances of fault X.

By assuming independence of the deviations calculated in Equations (3.15, 3.16, and 3.17), averaging the variances of L_{du} over different fault instances, and making a correction for the fault instances during which the model detected the fault later than the operator did, the variance of the expectation of the benefit of early detection for a random single instance of fault X can be approximately calculated as a sum of these three variances:

$$S^2\{\hat{E}[B_{dX}]\}$$

$$\approx \frac{\sum_{\substack{j=1 \\ L_{du}(X_j)-L_{dm}(X_j)>0}}^{n_{XN}} \left[\frac{\left[L_{bkj} - \frac{1}{n_{XN}}\sum_{j=1}^{n_{XN}}(L_{bkj})\right]^2}{n_{XN}-1} + \frac{\sum_{i=1}^{n_X} S^2(L_{tij})}{n_X} + S^2\left(\hat{E}\left[L_{dma}(X_j)\right| X_j \in f\right]\right)\right]}{n_{XN}}$$

(3.18)

In this equation, n_{XN} is the number of the instances X_j of fault X for which $L_{du}\left(X_j\right)-L_{dm}\left(X_j\right)$ is non-negative. Equation (3.18) effectively calculates a weighted average of the post-detection costs for each frequency of detection lags or pre-detection costs and assumes that each frequency contains the same number of post-detection costs. The standard deviation of the total benefit $B_d\left(X\right)$ of early detection of fault X is calculated as follows:

$$s\left\{\hat{E}[B_d(X)]\right\}=\frac{n_X}{t_{PO}}s\left\{\hat{E}[B_{dX}]\right\} \tag{3.19}$$

Equation (3.19) assumes that the exact number n_X of faults of type X occurred over the time period t_{PO} is known.

3.3.2.5 False Alarm-Associated Cost. Evaluation of the model performance metric requires estimation of the rate of false alarms $\hat{E}\left[r_{fa}(X)\right]$. To perform this evaluation, the model is run on the historical data recorded during the normal operation of the monitored process and not used for model identification. The interval between the latest data points used during consecutive classifications of the monitored process mode should be the same as the time intervals between the model runs when the fault model will be operating in real time. The fault warnings produced by the tested model are considered false alarms. False alarms separated by the time interval longer than the monitoring window are considered separate false alarms. If the time interval between two false alarms is shorter than the monitoring window, they are considered the same false alarm.

To represent the normal operation well, the test data samples should be collected at different randomly chosen intervals of normal operation of the monitored process. For each interval, the number of false alarms is counted. Then the expectation of the rate of false alarms is estimated by dividing the number of false alarms N_{fa} observed during the test by the total effective length L of the normal operation intervals selected for model evaluation:

$$\hat{E}\left[r_{fa}(X)\right]=\frac{N_{fa}}{L} \tag{3.20}$$

This estimator is unbiased because its expectation matches the definition of the average rate of false alarms.

Each selected interval representing normal operation must be several times longer than the monitoring window width. The first reason for this is that a single run of the tested model requires an interval of historical data as long as the monitoring window. To run more than once during the same test interval, this interval must be even longer. The second reason is that some process data points at the beginning of a data sample collected during the normal operation may be within the monitoring window of a false alarm that would have been produced if the model were run on the data immediately preceding the sample. As a result, false alarms issued during the beginning of some normal data samples may be false alarms initiated earlier. It is proposed not to count false alarms within one monitoring window of the beginning of each selected interval of normal operation of the monitored process. Thus, the total effective length L of the normal operation intervals selected for model evaluation is calculated as follows:

$$L=\sum_{i=1}^{N_N}\left(L_i-2W\right) \tag{3.21}$$

In Equation (3.22), N_N is the number of test intervals selected for model evaluation, L_i is the length of the i^{th} selected interval, and W is the monitoring window width.

To estimate the standard deviation of the expectation $\hat{E}\left[r_{fa}(X)\right]$ of the rate of false alarms for each tested model, the entire test set representing normal operation of the monitored process can be partitioned into intervals of the length equal to the monitoring window width W. Each interval can either contain or not contain a single false alarm. Then the rate of false alarms can be considered as the probability of a randomly chosen interval of historical data recorded during the normal operation and equal in length to the monitoring window width to contain a false alarm. This probability is further divided by the monitoring window width measured in years to obtain the rate of false alarms. Then estimation of the confidence interval for the rate of false alarms will be equivalent to the well-studied problem of estimating the interval for a binomial proportion.

3.3.2.6 Confidence Interval for the Rate of False Alarms. There exist a number of different techniques for estimating this confidence interval; the most common ones are summarized in recent surveys on the subject, e.g., (Brown, Cai et al. 2001). The most recommended technique is the Wilson or score interval. For the rate of false alarms, this confidence interval will be as follows (given the assumptions listed in the previous paragraph):

$$r_{fa}(X)=\frac{1}{W}\cdot\left\{\frac{N_{fa}+0.5Z_{0.5\alpha}^2}{\frac{L}{W}+Z_{0.5\alpha}^2}\pm\frac{Z_{0.5\alpha}\cdot\left(\frac{L}{W}\right)^{0.5}}{\left(\frac{L}{W}\right)+Z_{0.5\alpha}^2}\cdot\left[\frac{W\cdot N_{fa}}{L}\cdot\left(1-\frac{W\cdot N_{fa}}{L}\right)+\frac{W\cdot Z_{0.5\alpha}^2}{4\cdot L}\right]^{0.5}\right\} \tag{3.22}$$

In this equation, Z_R is the R^{th} percentile of the standard normal cumulative distribution function and α is the level of confidence for the interval. The Wilson interval becomes very inaccurate for the values of the number of false alarms N_{fa} close to zero or $n=\frac{L}{W}$. Figure 3.8 below shows apparent downward spikes in the estimated interval coverage probability for the values of $\hat{p}=\frac{W\cdot N_{fa}}{L}$ very close to zero or n. It means that for these values of $\hat{p}=\frac{W\cdot N_{fa}}{L}$ the estimated confidence interval will be much smaller than the actual one. Good fault models should produce false alarms very rarely, so N_{fa} should be close to zero.

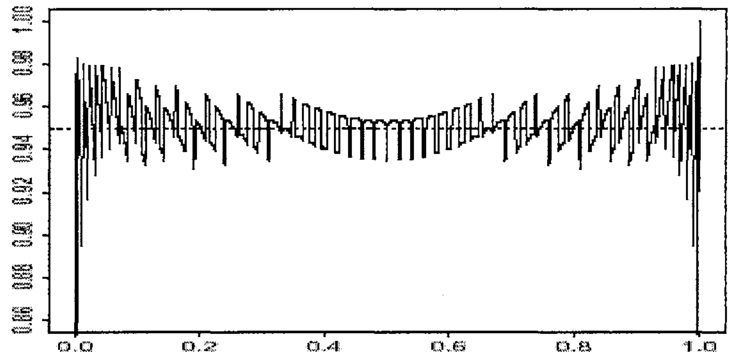


Figure 3.8. Wilson interval coverage probability for $n=50$ (Brown, Cai et al. 2001)

For those cases, it was proposed (Brown, Cai et al. 2001) to replace the lower bound of the confidence interval (3.22) with $\frac{\lambda \cdot W}{L}$. In this ratio, λ solves Equation (3.24):

$$exp(-\lambda) \cdot \sum_{i=0}^{N_{fa}-1} \frac{\lambda^i}{i!} = 1 - \alpha \tag{3.23}$$

It is hard to incorporate this proposed interval into Equation (3.20) to calculate the confidence interval for the proposed performance metric, the net benefit of automating the process fault monitoring. The reason is that the confidence interval for the rate of false alarms specified by Equation (3.23) is asymmetric. However, it is possible to reduce the contribution of the incorrect estimation of the rate of false alarms in the model performance metric estimation error by increasing the size of the part of the test set recorded during the normal operation of the monitored process. If the contribution due to an incorrect estimation of the rate of false alarms is smaller than the contribution of the error in the benefit of early detection by at least an order of magnitude, the former error can be neglected. Then, taking into account Equations (3.22) and (3.23) and assuming that a fault model makes 5-30 false alarms a year, the minimal length L_M of the process data recorded during the normal operation and required for testing the model would be approximately the average cost $5 \cdot \overline{L}_{fa}(X)$ of five false warnings of fault X divided by the standard deviation $S\left[\hat{L}_{du}(X_j)\right]$ of the previous annual loss due to the consequences of fault X:

$$L_M = \frac{5 \cdot \overline{L}_{fa}(X)}{S\left[\hat{L}_{du}\left(X_j\right)\right]} \tag{3.24}$$

Equation (3.24) assumes that the total consequence cost estimation error dominates or is approximately equal to the other errors accounted for by Equation (3.18).

3.4. Summary

This chapter introduces a method for evaluating fault models. This evaluation is the second part of the model selection procedure described in Chapter II. The method proposed in this chapter evaluates the potential monetary benefit of implementing a fault model for monitoring a specific fault. Current procedures for evaluating fault models focus on satisfying the quality standards and do not consider the potential costs associated with this satisfaction. The proposed procedure is based on estimating an expectation of the net benefit of fault model implementation. The procedure assumes that no irreversible changes occur in the process with time. This expectation is represented as a combination of fixed and random variables. The operator's expertise and plant records are used to assign values to the fixed variables. Values of the random variables are estimated by testing the model performance on the available historical data. A scheme for sampling this data is proposed. This custom scheme, a combination of bootstrapping and cross-validation, is tailored for evaluating the performance of process fault models operating in real time given the structure of the process data. This chapter also discusses how to carry out estimation of the consequences of historicized faults and how to find what the cost of these consequences would have been were the monitoring of the historicized faults automated. The estimated net benefit also includes an approximate variance and confidence interval that gives the user an idea of how precise the estimate of the net benefit of implementing different fault models is.

Chapter IV

Implementation and Application of the Algorithm for Designing Process Fault Models

This chapter proposes and demonstrates a complete algorithm for designing optimal mathematical models for process fault monitoring. The proposed algorithm involves constructing valid model designs out of fault model components, as proposed in Chapter II, and evaluating the resulting valid model designs to select the ones that perform best, as described in Chapter III. While Chapters II and III focus on the theory of selecting models for monitoring process faults, this chapter is devoted to the implementation of this theory. Chapter IV illustrates and documents the proposed algorithm's application in designing optimal models for monitoring faults in a commonly known simulation of a chemical process.

4.1 The Sequence of Steps in Constructing Good Monitoring Models

4.1.1 Model Selection Algorithm Outline

The proposed algorithm for designing optimal models for process fault monitoring consists of three parts:

1) Creating a reusable library of fault model components for designing different models for monitoring different faults in different processes,
2) Putting the components together to create fault model designs capable of properly addressing the user preferences, specifics of the monitored process and capable of processing the fault-related variables for the user-specified faults
3) Identifying the parameters of the fault models whose designs have been found suitable for monitoring the user-specified faults, evaluating the performance of the resulting models and selecting the best model.

A flowchart of the proposed algorithm is shown below in Figure 4.1.

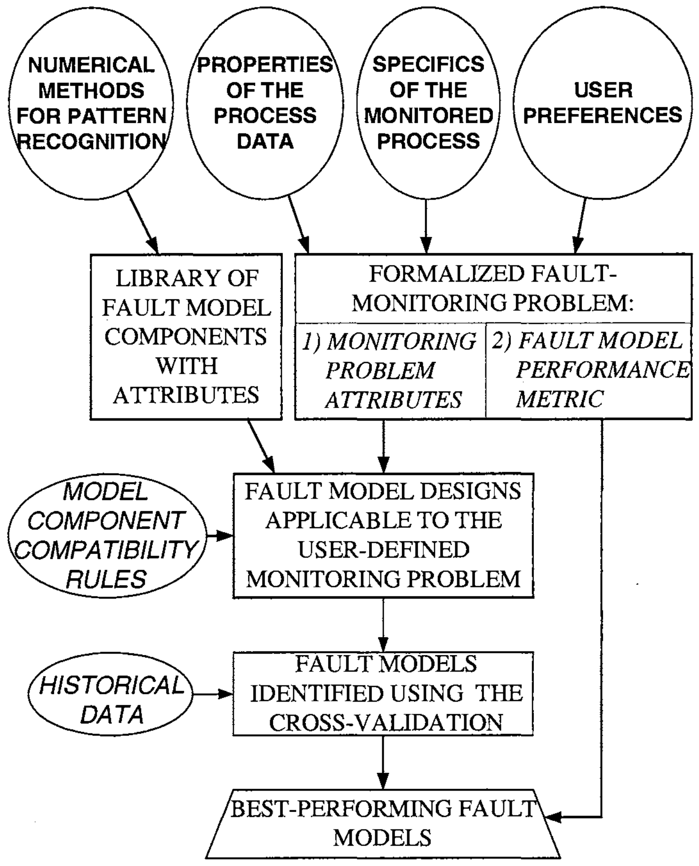

NUMERICAL METHODS FOR PATTERN RECOGNITION
PROPERTIES OF THE PROCESS DATA
SPECIFICS OF THE MONITORED PROCESS
USER PREFERENCES
LIBRARY OF FAULT MODEL COMPONENTS WITH ATTRIBUTES
FORMALIZED FAULT-MONITORING PROBLEM:
1) MONITORING PROBLEM ATTRIBUTES
2) FAULT MODEL PERFORMANCE METRIC
MODEL COMPONENT COMPATIBILITY RULES
FAULT MODEL DESIGNS APPLICABLE TO THE USER-DEFINED MONITORING PROBLEM
HISTORICAL DATA
FAULT MODELS IDENTIFIED USING THE CROSS-VALIDATION
BEST-PERFORMING FAULT MODELS

Figure 4.1. The algorithm for constructing models for fault monitoring

4.1.2 Interaction with the User

In our proposed algorithm, the sequence of steps performed by the user is as follows.

The steps performed by an advanced user one time for all fault models that will be created later:

1) Specifying reusable components for creating fault models and providing the computer code for each fault model component,
2) Specifying the attributes and parameter identification time for each model component placed in the library, using the list of possible attributes provided in Table 3.1,
3) Specifying additional attributes and component compatibility rules for the

components placed in the library (optional)

The steps performed by the process operator for creating a single fault model are as follows:

4) Specifying the monitored fault(s), a list of associated fault consequences, the consequence types, the consequence costs, and the costs of false alarms and fault misclassifications (Type I errors);
5) Selecting the monitored process variables and specifying their characteristics: the historian sampling period, presence of noise and outliers in the variables, process variable type (process input variable, process output variable, or neither), and the need to smooth the variables;
6) Specifying the user preferences for the characteristics of the generated models and the characteristics of the monitored process: the output indication, maximum allowable time for creating and evaluating a single model, the monitoring window width, the maximum expected process delay, the process time constant, and the model execution frequency;
7) Providing optional information about the fault and fault models: specifying the variable most closely related to the monitored fault, providing a first-principle model of the monitored process and custom fault classifiers, and specifying if the user deems there are enough process variables and data to identify a good model of the monitored process empirically;
8) Selecting the historical data required for model identification and specifying the chronology for each fault instance included in this data: the earliest and the latest time of fault inception, the time stamp of fault detection, an approximate time of fault rectification, and the occurrence/duration of fault consequences.

4.1.3 Generating Model Designs Suitable for the User-Specified Problem

Once the Steps 4-8 have been performed for a specific fault-monitoring problem, our proposed algorithm identifies the attributes of the real-time process data (***D1-D6***) using the information obtained from the user and process data historian at Step 5. The attributes of process model availability and solution requirements are initialized using the information specified by the user at Steps 6 and 7, and the historical data attributes are calculated using the input obtained from the user at Step 8.

Once all the problem attributes have been initialized, our proposed algorithm screens out individual model components incompatible with the specifics of the monitored process and user preferences using *Rules 1-5* listed in Table 3.2. After that, the algorithm generates all the possible combinations of the remaining fault model components and uses the rules specifying compatibility of fault model components with each other to screen out the designs where certain components are not compatible with one another. For all the remaining fault model designs, the algorithm identifies the properties of the data streams processed by each fault model component and screens out the designs where the components are found incompatible with the input data. For the remaining designs, an approximate time for identifying each model is estimated and the models whose identification time exceeds the user-specified limit are screened out. The model designs that have passed all the screenings are considered suitable for solving the user-specified fault-monitoring problem.

4.1.4 Evaluating Model Designs Suitable for the User-Specified Problem

Once the suitable fault model designs have been found, it is possible to identify the parameters of these models from the user-specified historical data. The historical data is prepared for model identification by extracting fault model input data vectors at specified sampling periods calculated from the monitored process delay and the longest time constant by the custom methods, similar to those used for calculating sampling periods for the process control. After that, the model expected to be the most efficient in reducing the risk associated with the consequences of the monitored fault should be selected.

The procedure for selecting the best-performing fault model is described in Chapter III. If there are *N* historicized instances of the monitored fault(s), the algorithm identifies *N* models for each suitable fault model design. For each design, the *k*th model is identified using the data from which the *k*th historicized instance of the monitored fault(s) and a randomly selected fraction of the historical data recorded for the normal operation mode is excluded. After that, the expectations of the benefit of earlier fault detection due to the automation of fault monitoring and of the cost of false alarms per unit time are calculated for each resulting model from the historical data not used for the identification of this model. These two expectations are averaged for each model design.

Then, for each model design, the resulting expectation of the cost of false alarms and the cost of creating and running the fault model per unit time are subtracted from the resulting expectation of the benefit of earlier fault detection due to the automation of fault monitoring. This results in the expected net benefit of fault automation for each suitable fault model design. The parameters of the model design associated with the highest expected net benefit of fault automation are then identified with all the user-specified historical data and used for monitoring the user-specified fault.

4.2 Creating a Library of Fault Model Components

In the algorithm for fault model design and selection, the models are assembled out of functional components using the proposed component compatibility rules. These functional components reflect the state of the art in mathematical modeling and process fault detection.

Hence, generating models monitoring user-specified faults will be possible only after creation of a library of fault model components and specifying their attributes.

Ideally, creation of a very good model for monitoring faults in industrial processes would require a very large library of model components shaped as a relational database. An expert system would pick model components for the proposed fault model template and automatically check if the resulting model is a valid one for the fault-monitoring problem. However, so far such an expert system has not been created. Therefore, for this example, the creation of a library of model components and generation of fault models out of these components will be performed manually.

In addition, this chapter is intended to provide a detailed account of implementation and application of the proposed algorithm. Every possible combination of the fault model components added to the component library will be listed and analyzed. Hence, the library of available components should be very small, so that the number of possible combinations of these components would be reasonable and all the steps of the model generation procedure can be fully documented in a reasonable amount of time. For this reason, the library consists only of eleven components. To demonstrate the proposed algorithm better, the library has at least one component of each of the six types. The total number N_C of possible model component combinations can be calculated as a function of the number components of each type placed in the library:

$$N_c = N_b \cdot N_o \cdot (N_f + 1) \cdot (N_{ds} + 1) \cdot (N_{vsm} \cdot N_{vsc} + 1) \tag{4.1}$$

In Equation (4.1), N_b is the number of basis functions, N_o is the number of optimized criteria, N_f is the number of feature generators, N_{ds} is the number of data smoothing techniques, N_{vsm} is the number of variable selection methods, and N_{vsc} is the number of variable selection criteria available in the component library.

Table 4.1 below lists all the components selected for the library, with the associated attributes given in curly parentheses '{}', modifiers of the real-time data attributes given in angle parentheses '<>', and formulas for estimating approximate model identification time. The library contains three basis functions, two optimized criteria, two feature generators, two data smoothing techniques, one variable selection method, and one variable selection criterion. Hence, according to Equation (4.1), the total number of combination of these components is *108*. The remainder of Section 4.2 gives a description of the eleven members of the model component library.

Table 4.1. The demo library of the available fault model components used in Chapter IV

Component type	Component number	Component name	Component {attributes} and <data modifiers>	Minimal estimated CPU time, seconds, needed for model identification,
[1] **Classifier basis function**	1	S control chart for the process variable assumed most closely related to the fault	{C1}, {C4}, {C5}, {C6}, {C9}, {C10=300}, {C12}	$3 \cdot 10^{-4} \cdot D6_D$
	2	Probabilistic neural network identified using the DDE algorithm	{C9}, {C10=$D1_D \cdot D2_D$}, {C11}	$10^{-9} \cdot (C10 \cdot D1_D \cdot D2_D)^2 \approx 10^{-9} \cdot (D1_D \cdot D2_D)^4$
	3	Multilayer perceptron	{C3}, {C8}, {C10=$D1_D \cdot D2_D$}, {C11}	$10^{-8} \cdot (C10 \cdot D1_D \cdot D2_D)^2 \approx 10^{-8} \cdot (D1_D \cdot D2_D)^4$
[2] **Optimized criterion**	4	Mean square error	{C15}, {C18}	
	5	Error rate	{C16}, {C18}	
[3] **Feature generator**	6	PCA, outputs being the principal component scores that explain 95% of the variance in the input process variables	{C21}, {C22}, {C23=10}, <$D1_D=D6_D=D6_C$>, <$D2_D=D2_C/3$>, <$D3_D$=TRUE >, <$D4_D$=FALSE>	$sup(10^{-9} \cdot D2_C{}^2 C10^2; 10^{-10} D2_C{}^5)$
	7	PCA residuals, principal component scores explain 95% of the variance in the input process variables	{C21}, {C23=10}, <$D1_D=D6_D=D6_C$>, <$D2_D=D2_C/3$>, <$D3_D$=TRUE >, <$D4_D$=FALSE>	$sup(10^{-9} \cdot D2_C{}^2 C10^2; 10^{-10} D2_C{}^5)$

Table 4.1. The demo library of the available fault model components used in Chapter IV (continued)

[4] Data smoothing techniques	8	Single-level (scale) Haar function-based wavelet filter for process variables (the number of smoothed process variables is N_S)	{C25}, <D5$_C$=TRUE>	$3 \cdot 10^{-5} \cdot C10 \cdot D1_B$ if {C12=TRUE}, otherwise $3 \cdot 10^{-5} \cdot C10 \cdot D1_B \cdot N_S$
	9	Single-level (scale) biorthogonal **bior3.1** function-based wavelet filter for process variables (the number of smoothed process variables is N_S)	{C25}, <D5$_C$=TRUE>	$10^{-4} \cdot C10 \cdot D1_B$ if {C12=TRUE}, otherwise $10^{-4} \cdot C10 \cdot D1_B \cdot N_S$
[5] Input variable search method	10	Forward Search that selects as model inputs no more than 50% of the user-specified process variables	{C26=4}, {C27=516}, {C29}, <D2$_B$=D2$_A$/2>	$sup\left\{ \sum_{d=1}^{D2_A/2} (D2_A - d) \cdot sup[t_B(d), t_F(d)], t_S \right\}$ Time to identify the basis function parameters with d input variables is appx. $t_B(d)$ seconds, the time to identify parameters for the feature generator with d input variables is appx. $t_F(d)$ seconds, and the time to smooth all the process variables is appx. t_S seconds
[6] Input variable search criterion	11	Same as the optimized criterion		0

4.2.1 Optimized Criteria

The optimized criteria are discussed before the classifier basis functions because the subsequent discussion of the basis functions refers to the optimized criteria. Two optimized criteria are used in our demo library of fault model components:

1) Error rate

2) Sum of square errors.

These two optimized criteria are by far the most commonly used ones in various mathematical models.

Definitions of the error rate may vary. Essentially, the error rate calculated for various classifiers is a statistical estimator of how often the classifier will make errors in the future. For the demo library of fault model components presented in this chapter, the estimated error rate R is defined as the average likelihood (not the probability) of incorrect classification of a training set input data vector (prototype) belonging to each class:

$$\hat{R} = \frac{1}{K} \cdot \sum_{i=1}^{K} \frac{M_i}{N_i} \tag{4.2}$$

In Equation (4.2), K is the number of classes distinguished by the classifier, N_i is the number of the training set data vectors (prototypes), belonging to class i, and M_i is the number of incorrectly classified training set prototypes belonging to class i. Error rate is a count-based optimized criterion: *C16=TRUE* and it is monotonic meaning that it not a function of the number of model parameters: *C18=TRUE.*

The sum of square errors (sometimes called a sum of squared residuals) *SSE* is defined as the sum of squares of the differences between the vector of model outputs z and the vector of actual values y of the variable that the model emulates:

$$SSE = \sum_{i=1}^{H1} (z_i - y_i)^2 \tag{4.3}$$

In Equation (4.3), *H1* is the historical data attribute (defined in Table 2.2) specifying the number of process data vectors used for model identification. The sum of squared errors is a distance-based optimized criterion as it optimizes the distance between the actual and model-predicted values: *C15=TRUE* and it is a "monotonic" criterion as it is not a function of the number of model parameters: *C18=TRUE.*

In this demonstration, the differences in the time required to evaluate different optimized criteria during the model identification is neglected, so this time is included in the model identification time for the classifier basis functions.

4.2.2 Classifier Basis (Link) Functions

The model component library used in this chapter has three classifier basis functions mentioned in Chapter II:

1) S control chart for the process variable assumed most closely related to the fault,
2) Probabilistic neural network trained using the DDE algorithm, and
3) Multi-layer perceptron.

4.2.2.1. S Control Chart. The S control chart (Xie, Goh et al. 2002) is a rather simple univariate method: it is based on measuring a variance of the sample of several recent measurements and establishing if this variance is within the confidence interval determined for the normal operation of the monitored process. In this example, the S chart uses all the measurements of the key fault variable captured by the monitoring window to calculate the variance. However, the lower limit on the variance is set to zero and the upper limit is determined not by the limits of a confidence interval for a specific level of significance, but by minimizing the optimized criterion selected for the other slot of the classifier. The S control chart is by nature very easily compatible with different optimized criteria.

From its description, we can establish that this component requires the original process variables: *C1=TRUE*, generates binary outputs: *C4=TRUE*, has a fixed number of basis function parameters: *C5=TRUE* (the only parameter is the upper limit on the variance), requires identification of a variable believed most closely related to the monitored fault *C6=TRUE* (this requirement has been imposed voluntarily), generates crisp categorical output *C9=TRUE*, and it is a univariate basis function: *C12=TRUE*. Furthermore, the upper limit on the sample variance for the normal operation should be calculated using at least *300* process data vectors: *C10=300* according to the control chart parameter identification guidelines. All the other basis function-related attributes for this component are set to "*FALSE*." For this demo library, the upper limit for the sample variance calculated by the control chart is determined by the Golden Section univariate optimization with 10 iterations. Hence, according to Subsection 2.4.2.2, the minimal time required to identify the parameter of the control chart is proportional to the size of the input data vector $D6_D$ times the minimal number of data vectors required for model identification times ten. It was found approximately equal to $3 \cdot 10^{-4} \cdot D6_D$ seconds.

4.2.2.2 Probabilistic Neural Network Trained with a Dynamic Decay Adjustment. Specht (Specht 1990) introduced neural networks of this type fifteen years ago. These networks have two hidden layers. When presented with an input data vector $\boldsymbol{x}$, the first hidden layer calculates the probability that this data vector belongs to class k according to Equation (4.4):

$$p_k(x)=\sum_{j=1}^{m_k}\frac{w_{kj}\cdot exp\left[-0.5\cdot\sigma^{-2}\cdot\left(x-\mu_j^k\right)^T\cdot\left(x-\mu_j^k\right)\right]}{\left(2\pi\right)^{0.5d}\cdot\sigma^d} \tag{4.4}$$

In this equation, m_k is the number of network neurons belonging to class k, w_{kj} is the weight of jth neuron belonging to class k, σ is the neuron spread, μ_j^k is the center of jth neuron belonging to class k, and d is the dimensionality of the network inputs (in fault models, d is the dimensionality of process data vectors serving as classifier inputs). After that, the second "competitive" layer selects the class where the data vector $\boldsymbol{x}$ has the highest probability of membership calculated by the first layer.

In the original work by Specht, the network neurons centers μ_j^k are the data points of the training set, each neuron weight w_{kj} is set to the reciprocal of the number of data points of class k in the training set, and the spread is specified by the user. This makes the total number of neural network parameters equal to the number of data vectors in the training set times the number of dimensions in the input data vectors. Obviously, the algorithm for identifying neural network parameters proposed by Specht is not acceptable to create a classifier for real-time fault monitoring from training sets that contain several thousands data points or more. In those cases, the number of data neurons should be several times smaller than the number of data points available for model identification. Then the neural network parameters must be identified using the algorithms that create neurons sequentially, starting from a single neuron and ending when the neural classifier performance does not improve significantly with the addition of new neurons. A training algorithm of this type, the Dynamic Decay Adjustment (DDA), is used with the probabilistic neural network in the proposed library of fault model components. The work by Berthold and Diamond (Berthold and Diamond 1998) gives the details on what the DDA algorithm is and how it is used to train probabilistic neural networks. For this library, the two sensitivity thresholds are set to *0.4* and *0.2* respectively. The MATLAB code for the probabilistic neural networks trained with the DDA is available in the public domain library of additional MATLAB applications. This code is given in the Appendix.

The probabilistic neural network creates crisp categorical output: *C9=TRUE*, it is sensitive to the noise in the input data (unlike the control charts): *C11=TRUE*, and the minimum number of data vectors (prototypes) required for the training set is set to the size of the input data vectors: $C10=D1_D \cdot D2_D$. All the other basis function-related attributes for the probabilistic neural network were set to "*FALSE*." It was found empirically that the time required for model identification grows as the square of the size of input data vectors times the square of the number of training data vectors (prototypes) times 10^{-9}. Given the restriction on the minimum number of data vectors (prototypes) in the training data set, the minimum time required for identifying the probabilistic neural network is approximately $10^{-9} \cdot (D1_D \cdot D2_D)^4$ seconds.

One important issue for the probabilistic neural network is its compatibility with different optimized criteria. The DDA algorithm is tailored to minimize the error rate, not any other quantities. Combining the DDA-trained probabilistic neural network with the count-based optimized criteria (count and distance-based optimized criteria are defined in Subsection 2.4.3.1) other than the error rate is achieved by modifying the training data set. Suppose, the component library contains the criterion specifying that the cost of misclassifying faults as normal operation is ten times as high as the cost of misclassifying normal operation as faults. Then the training set for the probabilistic neural network will be modified to contain ten times as many process data points sampled during faults as the number of process data points sampled during normal operation. If the optimized criterion is the error rate, the number of "faulty" and "normal" process data points in the training set should be equal. As far as the distance-based optimized criteria are concerned, they are incompatible with all the basis functions producing categorical outputs, including the probabilistic neural network. This fact is stated by *Rule 6* introduced in Subsection 2.4.3.1.

4.2.2.3 Multi-layer Perceptron. Multi-layer perceptrion is the oldest and most commonly used neural network and the first one explained in the neural network textbooks (Hagan, Demuth et al. 1995). It consists of several layers of parallel linear transformations with additional transform functions for each layer. The multi-layer perceptron used as a component of the demo library consists of two layers and does not use any nonlinear transforms. The output scalar y of this perceptron is calculated from input data vector $\boldsymbol{x}$ using the following equation:

$$y = log\ sig\{W_2 [log\ sig(W_1 x + b_1)] + b_2\} \tag{4.5}$$

In this equation, *logsig* is the log-sigmoid function, $\boldsymbol{W_1}$ and $\boldsymbol{W_2}$ are the matrices of network weights for the first and second layers respectively, and vectors $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ contain network biases for the first and second layer respectively.

The network parameters $\boldsymbol{W_1}$, $\boldsymbol{W_2}$, $\boldsymbol{b_1}$, and $\boldsymbol{b_2}$ are identified using the commonly accepted technique called the Steepest Descent Optimization. Compliant with the requirements to fault model components outlined in Subsection 2.3.2, the Steepest Descent search technique can optimize any distance-based criteria, e.g., the Eulerian Distance, for which derivatives can be estimated. As far as the count-based criteria are concerned, they are not compatible with the multi-layer perceptron according to *Rule 7* introduced in Subsection 2.4.3.1 because the perceptron produces continuous outputs and hence the attribute *C8* for the multi-layer perceptron is equal to *TRUE.*

The other attributes of the multi-layer perceptron are creation of unbounded regions for the normal operation: $C3=TRUE$, and sensitivity to noise in the input data $C11=TRUE$. For this component, the minimum number of historicized process data vectors (prototypes) required to identify the multi-layer perceptron parameters is set to the input vector size (same as for the probabilistic neural network): $C10=D1_D \cdot D2_D$. The relationship between the size of the training data set and the time required to identify the network parameters is the same as for the probabilistic neural network. However, the steepest descent-based training of a multi-layer perceptron is about ten times as long as the DDA-based training of a probabilistic neural network given the same dimensionality of the input data vectors and the number of prototypes in the training set. Hence, the parameter identification time for the multi-layer perceptron used in the demo library is approximately $10^{-8} \cdot (D1_D \cdot D2_D)^4$ seconds.

4.2.3 Feature Generation Techniques

The feature generation techniques in this library are based on the Principal Component Analysis (PCA). Karl Pearson introduced this technique more than a hundred years ago (Pearson 1901). For an arbitrary multivariate data sample $\boldsymbol{X}$, this technique creates a matrix $\boldsymbol{V}$ of loading vectors. Matrix $\boldsymbol{V}$ is created to transform $\boldsymbol{X}$ into principal component scores $\boldsymbol{Y}=\boldsymbol{Z}\boldsymbol{V}$ where $\boldsymbol{Z}$ is a normalized data matrix $\boldsymbol{X}$. Geometrically, the vector in the first column of $\boldsymbol{Z}$ specifies the direction of the greatest variance in matrix $\boldsymbol{X}$; the second column of $\boldsymbol{Z}$ specifies the direction of the greatest variance in $\boldsymbol{X}$ perpendicular to the first column of $\boldsymbol{V}$, the third column of $\boldsymbol{Z}$ specifies the direction of the greatest variance in $\boldsymbol{X}$ perpendicular to the first and second columns of $\boldsymbol{Z}$, etc. The resulting number of principal component scores is equal to the number of input variables. The sample variance of the principal component scores monotonically decreases from the first principal component score to the last. Many practical applications of the PCA reduce the dimensionality of the original data by using only the first $n_{x\%}$ component scores that "explain" $x\%$ of the variance in the original data matrix:

$$\sum_{i=1}^{n_{x\%}} S_i \approx \sum_{i=1}^{d} S_i \tag{4.6}$$

In Equation (4.6), S_i is the sample variance for the set of measurements of the i^{th} principal component score obtained from the data matrix $\boldsymbol{X}$ and d is the number of variables in $\boldsymbol{X}$.

Component 6 of the proposed demo library creates a PCA loading matrix $\boldsymbol{V}$ out of a sequence of process data points in the training set selected by the user to identify the fault model parameters. *Component 6* performs feature generation by multiplying separate process data points by matrix $\boldsymbol{V}_{95\%}$ that includes only the first $n_{95\%}$ columns of the full loading matrix $\boldsymbol{V}$. As a result, the resulting principal component scores explain *95%* of the variance in $\boldsymbol{X}$. *Component 7* of the proposed demo library generates features by multiplying separate process data points by matrix $\boldsymbol{C}_{95\%}$ that consists of the columns of the full loading matrix $\boldsymbol{V}$ not included in $\boldsymbol{V}_{95\%}$. *Component 7* exemplifies a numerical method not typically used in process monitoring. In the models that include *Components 6* and *7*, process data vectors are formed not out of the measured process data points, but out of the process data points transformed using the PCA.

Components 6 and *7* are useful only when the process variables used as fault model inputs are correlated: *C21=TRUE*, the outputs of *Component 6* model process data variability: *C22=TRUE* (while the outputs of *Component 7* model deviations from the data variability observed for the training set), and *Components 6* and *7* need at least five input variables to perform an efficient data dimensionality reduction: *C23=5*. *Components 6* and *7* apply PCA-based conversions to all the process variables available to the feature generators, as a result, after this transformation, all the PCA-converted variables have the same sampling period. Consequently, the number of different measurements in the monitoring window is the same for all the converted variables, hence $D1_D=D6_D=D6_C$. It was found empirically that the proposed PCA conversion cuts the number of input variables approximately in half: $D2_D=D2_C/2$. This applies to both *Component 6* and *Component 7* because the number of output variables produced by the former plus the number of output variables produced by the latter component is equal to the number of the original variables. *Component 6* and *Component 7* create combinations of the original variables: $D3_D=TRUE$ and the outputs produced by these components are not correlated: $D4_D=FALSE$.

In the MATLAB implementation of the PCA, the loading matrix $\boldsymbol{V}$ is calculated as follows. First, the correlation matrix $\boldsymbol{R}$ of the input data matrix $\boldsymbol{X}$ is calculated. Then the eigenvalues λ of matrix $\boldsymbol{R}$ are found by solving the following equation:

$$|\boldsymbol{R}-\lambda\boldsymbol{I}| = 0 \tag{4.7}$$

For each eigenvalue λ_i, an eigenvector $\boldsymbol{v}_i$ of matrix $\boldsymbol{R}$ is estimated by solving the equation:

$$\boldsymbol{R}\boldsymbol{v}_i = 0 \tag{4.8}$$

The columns of the loading matrix $\boldsymbol{V}$ are the calculated eigenvectors $\boldsymbol{v}_i$ of the correlation matrix $\boldsymbol{R}$.

If the number of process data points in the training data set is much greater than the number of the converted variables, then the limiting stage in the calculation of the loading matrix is the estimation of the correlation matrix $\boldsymbol{R}$. Since this estimation involves multiplication of the data matrix $\boldsymbol{X}$ by itself, the time needed to calculate the correlation matrix grows proportional to product of the squares of the dimensions of $\boldsymbol{X}$: $D2_C{}^2 \cdot H1^2$. Equations (4.7) and (4.8) are solved by the Singular Value Decomposition. Solution of these equations involves $d+1$ multiplications of the d-dimensional matrix $\boldsymbol{R}$ by itself, d being the number of the converted input variables. Hence, the time needed to calculate the eigenvectors of $\boldsymbol{R}$ is proportional to $D2_C{}^5$. With these considerations, it was found that the time required for identifying the parameters of *Component 6* and *Component 7* is approximately equal to $10^{-9} \cdot D2_C{}^2 \cdot C10^2$ seconds or of $10^{-10} \cdot D2_C{}^5$ seconds, whichever is greater.

4.2.4 Data Smoothing Techniques

In the library of fault model components used in this work, the data smoothing techniques are represented by two wavelet transforms: the Haar transform and the bi-orthogonal **bior3.1.** A wavelet is a waveform of effectively limited duration that has an average value of zero. Wavelet analysis is decomposing a signal into shifted and scaled versions of the original (or *mother*) wavelet. Wavelet shifting and scaling is illustrated in Figures 4.2 and 4.3 respectively.

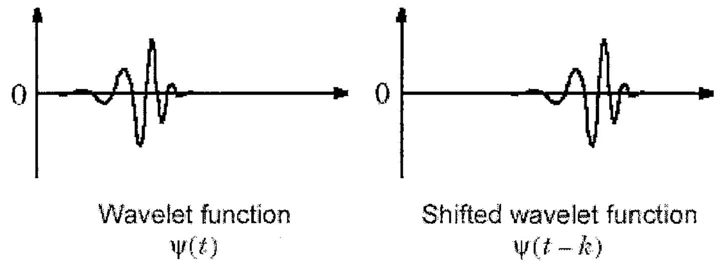


Figure 4.2. Wavelet shifting (from the MATLAB User Manual)

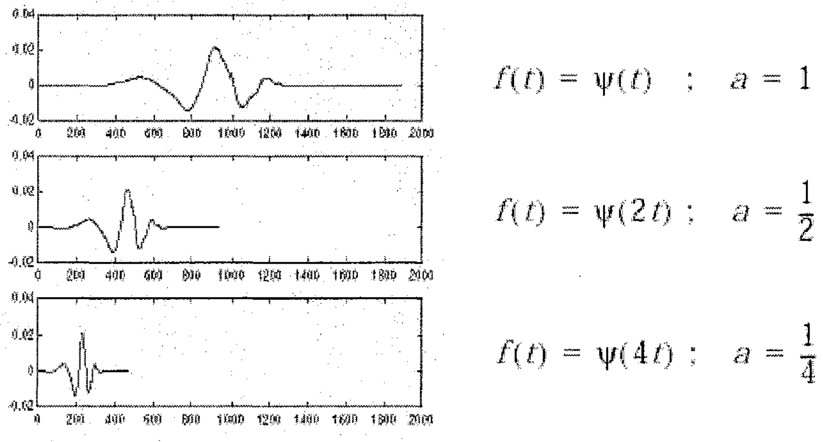


Figure 4.3. Wavelet scaling (from the MATLAB User Manual)

Usually, the signal is divided into intervals of equal length. Then signal at each interval is approximated by a wavelet function. Then each interval is divided in halves and the residuals of the first approximation are again approximated on each of the resulting smaller intervals with the same wavelet condensed twofold. The procedure is repeated *n* times and results in an *n*-level wavelet decomposition of the original signal. Interval length at the first level of the decomposition is 2^{n-1}. The decomposition results in a set of coefficients specifying the wavelet amplitudes for each interval at each scale. The output of the wavelet smoothing techniques is the original signal restored using only the wavelet coefficients and disregarding the residuals obtained at the last level of the decomposition. The procedure of signal decomposition and restoration using wavelet transforms is called wavelet filtering.

The demo library has two data-smoothing components: *Component 8* and *Component 9. Component 8* is a 3-level wavelet filter that uses the Haar wavelets shown in Figure 4.4, and the *Component 9* is a 3-level wavelet filter that uses the **bior3.1** wavelets shown in Figure 4.5. The time scale for data filtering is measured in seconds. *Component 8* and *Component 9* perform wavelet filtering of the readings of each noisy variable inside the monitoring window plus *8* consecutive readings collected after the last time stamp inside the monitoring window. As a result, fault detection is delayed by *8* sampling intervals of the noisy variable with the longest sampling period. The reason is that the marginal values of the monitored variables are distorted by the wavelet transform.

The proposed filters should only be applied to noisy process variables; otherwise these filters may erase important fault information from the smoothed readings. Hence for *Components 8* and *9*, *C25=TRUE.* These components almost completely remove noise from the process variables, so $D5_C$*=TRUE.* For both *Component 8* and *Component 9* the time it takes to create "filtered" process data vectors is proportional to the number of consecutive measurements of the smoothed variable inside the monitoring window and the number of process data vectors. Thus, if N_S is the number of process variables smoothed by the model, then for *Component 8* this time will be approximately $3{\cdot}10^{-5}{\cdot}\ N_S\ C10{\cdot}D1_B$ seconds and for *Component 9* this time will be approximately $10^{-4}{\cdot}N_S{\cdot}C10{\cdot}D1_B$ seconds.

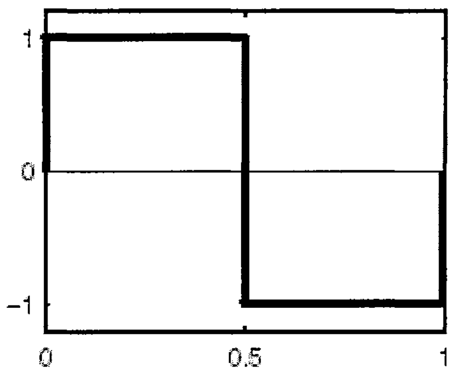


Figure 4.4. Haar wavelet (from the MATLAB User Manual)

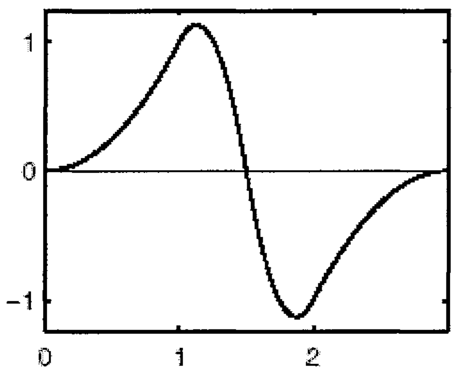


Figure 4.5. **Bior3.1** wavelet (from the MATLAB User Manual)

4.2.5 Input Variable Search

The demo library presented in this work has only one input variable search method component (*Component 10*): a forward search that selects as model inputs no more than 50% of the user-specified process variables. No specific input variable search criteria are added to the library, it is assumed that whenever input variable search is used, the input variable search criterion is the same as the optimized criterion. Hence, each iteration of the input variable search requires identification of a new fault model. The forward search technique is commonly used for selecting model inputs and it is described in many textbooks, e.g., (Johnson 1998a). The technique is so simple that it is described below in full.

At the first step, each candidate input variable is evaluated individually as the only model input. After that, the variable whose model resulted in the best value of the variable selection criterion is selected. Then this selected variable is used in combinations with all the remaining variables to create models with two input variables. After that, the variable whose addition to the one selected at the first step results in the best value of the variable selection criterion is added to the pool of model inputs. The procedure is repeated until either the value of the variable selection criterion starts getting worse with the addition of any new input variable to the model or the number of input variables selected for the model exceeds one half of the total number of the available variables.

Application of this variable selection technique makes sense when the number of available input variables is at least four: *C26=4*. If n input variables are available, the number of iterations I for this variable selection technique is calculated as follows:

$$I=n+(n-1)+(n-2)+\ldots+int(n/2)\approx 0.375n^2 \tag{4.9}$$

It is reasonable to set the upper limit on the number of iterations performed during the forward search at 10^5, therefore *Component 10* can be used if no more than *C27=516* input variables are available. In addition, *Component 10* is not designed to retain a user-specified variable in the pool of selected input variables. The number of process variables selected by the component is one half of the total number of the available variables: $D2_B=D2_A/2$. The time it takes to identify a model that uses forward selection is hard to estimate because the time required for model identification at each iteration of the search is different. This time depends on the number of input variables already added to the pool of input variables. If we assume that the smoothing of input variables precedes the input variable search (i.e., the selection is performed among smoothed process variables), the time to identify the basis function parameters with d input variables is estimated at $t_B(d)$ seconds, the time to identify parameters for the feature generator with d input variables is estimated to be $t_F(d)$ seconds, and the time to smooth all the process variables is approximately t_S seconds, then an approximate time t_I to identify a model that uses forward search will be as follows:

$$t_I \approx sup\left\{ \sum_{d=1}^{D2_A/2} (D2_A - d) \cdot sup[t_B(d), t_F(d)], t_S \right\} \tag{4.10}$$

4.3 Formalization of a Fault Monitoring Problem for a Simulated Chemical Process

4.3.1 Tennessee-Eastman Process Simulation

The remainder of this chapter demonstrates how the proposed algorithm finds the best-performing model for monitoring a fault in a simulated chemical process called Tennessee-Eastman (TE) challenge problem (Downs and Vogel 1993). The code emulating the TE process is available in the public domain. This code, provided by Downs and Vogel, runs the TE process simulation in the open-loop mode. The user must add a regulatory control system. A number of academic workers used this simulation to test various approaches to process control. Also, several research papers reported applications of the TE simulation for testing various methods of fault detection (Akbaryan and Bishnoi 2001; Chen and Howell 2002; Chiang, Russell et al. 2001b; Gertler, Li et al. 1999; Kano and Nagao 2000; Oh, Mo et al. 1997; Raich and Cinar 1997; and Wilson and Irwin 2000).

As shown in Figure 4.6, the process involves coordination of five unit operations: an exothermic two-phase reactor, a product condenser, a flash separator, a stripper, and a recycle compressor. The process consists of producing two products from four reactants by the following reactions:

$\mathbf{A}(g) + \mathbf{C}(g) + \mathbf{D}(g) \rightarrow \mathbf{G}(liq)$	Product 1
$\mathbf{A}(g) + \mathbf{C}(g) + \mathbf{E}(g) \rightarrow \mathbf{H}(liq)$	Product 2
$\mathbf{A}(g) + \mathbf{E}(g) \rightarrow \mathbf{F}(liq)$	Byproduct
$3\mathbf{D}(g) \rightarrow 2\mathbf{F}(liq)$	Byproduct

Streams *1*, *2*, and *3* supply pure substances **A**, **D**, and **E** respectively. Stream *4* supplies a nearly equimolar mixture of **A** and **C**. Stream *9* removes the unconverted reactants and byproducts, and stream 11 removes the products **G** and **H** from the process.

In Figure 4.6, (*FI*) denotes a flow rate indicator, (*XI*) stands for a composition analyzer, (*LI*) is a level indicator, (*PI*) is a pressure indicator, (*TI*) is a temperature indicator, (*JI*) is an ampermeter, and (*SC*) is an agitator speed controller.

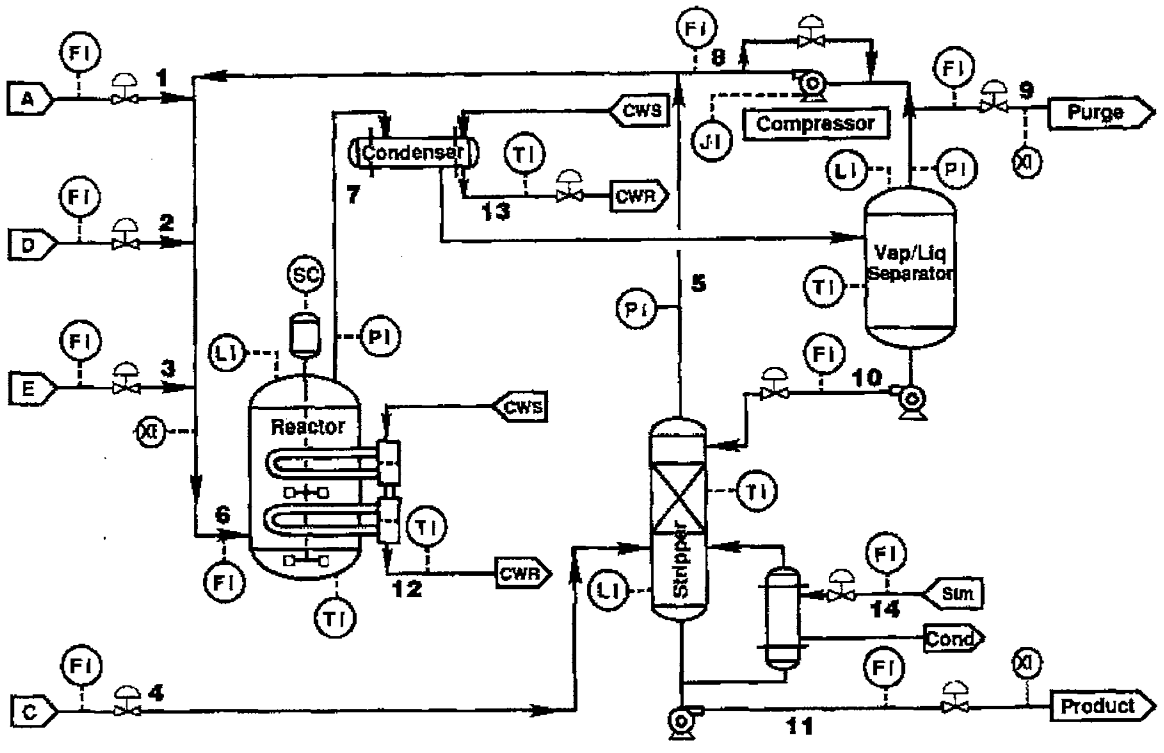


Figure 4.6. Flowchart of the simulated TE process (Downs and Vogel 1993)

In the process, there are 41 measured process variables with added measurement noise. In addition, 19 composition measurements from gas chromatographs are sampled at two different rates with delays due to the composition analyses. There are 12 manipulated variables: flow though 11 control valves, and the reactor agitator speed. Downs and Vogel provide a steady-state material and energy balance, some physical property data, qualitative information on the reaction kinetics and list specifications for the setpoint tracking. Downs and Vogel also list the costs of the components, steam, and unit time of compressor operation. The simulation includes 20 potential disturbances listed in Table 4.2, which can be invoked by changing the corresponding disturbance flag from zero to one. Some of these disturbances can be assumed caused by faults, so the ability of applicable fault models to detect those faults can be tested. Downs in Vogel provided a more detailed description of the TE simulation in their original work (Downs and Vogel 1993).

Table 4.2. Disturbances preprogrammed in the TE Process

Number	**Disturbance description**	**Disturbance type**
1	**A/C** Feed Ratio Changes, **B** Feed Constant in Stream 4	Step
2	Molar Fraction of **B** Changes, **A/C** Ratio Constant in Stream 4	Step
3	**D** Feed Temperature Changes in Stream 2	Step
4	Reactor Cooling Water Inlet Temperature	Step
5	Condenser Cooling Water Inlet Temperature	Step
6	Feed Loss in Stream 1	Step
7	Pressure Loss in Stream 4	Step
8	Composition of Stream 4	Random variation
9	Temperature of Stream 2	Random variation
10	Temperature of Stream 4	Random variation
11	Reactor Cooling Water Inlet Temperature	Random variation
12	Condenser Cooling Water Inlet Temperature	Random variation
13	Reaction Kinetics	Slow drift
14	Reactor Cooling Water Valve	Sticking
15	Condenser Cooling Water Valve	Sticking
16	Unknown	Unknown
17	Unknown	Unknown
18	Unknown	Unknown
19	Unknown	Unknown
20	Unknown	Unknown

The authors of the TE simulation have also included the costs of the reagents and products involved in the TE process. These costs are summarized in Table 4.3.

Table 4.3. Costs of the substances involved in the Tennessee-Eastman Process

Component	**A**	**C**	**D**	**E**	**F**	**G**	**H**
Cost, $/kgmol	2.206	6.177	22.06	14.56	17.89	30.44	22.94

4.3.2 Proposed Control and Fault Monitoring in the TE Process

This work uses the control strategy proposed for the TE simulation by Lyman and Georgakis (Lyman and Georgakis 1995). The code implementing this control scheme called by the authors *"Control Structure 2"* was written by the authors of (Chiang, Russell et al. 2001b) and it is available in the public domain at *http://brahms.scs.uiuc.edu/*. The original TE process simulation by Downs and Vogel is contained in the file named **TEPROB.F**. The code that implements the *"Control Structure 2"* is contained in **TEMAIN_MOD.F**. A flowsheet of the TE process incorporating this structure is shown in Figure 4.7 below. In this figure, (*YC*) denotes a device controlling quantity *Y* where *Y* can be *T* (temperature), *P* (pressure), *L* (level), *X* (composition), and *F* (flow rate).

Most disturbances in Table 4.2 do not produce serious economic consequences. *Disturbances 1* to *12* are external to the TE simulation. *Disturbances 16-20* are not documented, so no meaningful analysis of these disturbances can be performed. Monitoring the cases of the reactor cooling water valve sticking (*Disturbance 14*) is fairly straightforward. This failure almost immediately causes very significant changes in the reactor temperature. No other disturbance out of the 20 causes similar effects. *Disturbance 15* by itself is very unlikely to cause any serious consequences. It was observed that this disturbance has a limited effect because the condenser cooling water valve is almost always fully open. Hence, out of the disturbances preprogrammed in the TE simulation, the reaction kinetics drift (*Disturbance 13*) is the most suitable one for demonstrating the proposed algorithm for fault model selection. This fault can be due to impurities suddenly appearing in the process feeds or, much less likely, failures of the reactor agitator. Reaction kinetics drifts affect many process variables simultaneously. Most affected are the concentrations of the products **G** and **H** in Stream 11.

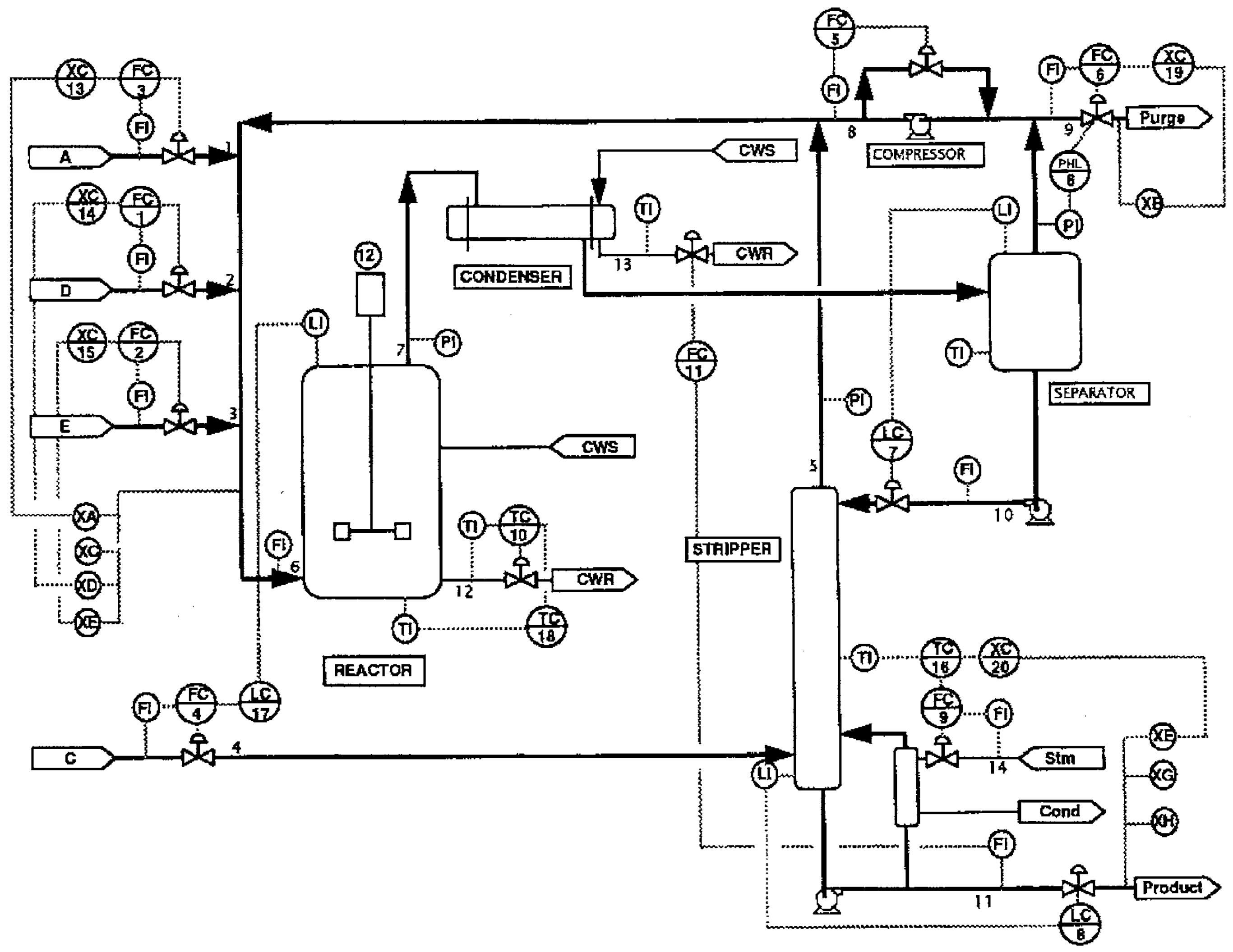


Figure 4.7. Simulated control scheme for the TE process (Lyman and Georgakis 1995)

It is assumed that the product Stream 11 is directed to a "day tank" with a capacity on the order of 1000 cubic meters and the contents of this tank are supplied to a product purification process located downstream of the TE process. Because of this, the concentrations of **G** and **H** supplied to the downstream process can be kept close to the specified values by blending any substandard product in the mixture of **G** and **H**. However, the kinetics drifts also reduce the rates of production of either **G** or **H** and this insufficient production cannot be corrected by dilution.

Reaction kinetics drift may also cause the reactor pressure, not controlled directly in the control structure proposed by Lyman and Georgakis, to increase to the shutdown threshold and stop the operation of the TE process. The kinetics drifts simulated in the original code of the TE process cause process shutdowns only very rarely; the expected time from fault inception to a shutdown is in the order of months. The original TE code was modified as shown in Figure 4.8 to create a process where the risk of shutdowns due to the kinetics drifts is higher and the expected time from fault inception to a shutdown is about 1.1 days. In addition, when emulating each historicized fault instance, the TE code is run using a new value of the random seed *G* initialized in the function *TEINIT*.

Original code:

```
Line 1529:   S1=SSPAN*TESUB7(I)*IDVFLAG+SZERO
Line 1530:   S1P=SPSPAN*TESUB7(I)*IDVFLAG
```

Modified code:

```
Line 1529:  S1=SSPAN*TESUB7(I)*1.475*IDVFLAG+SZERO
Line 1530:  S1P=SPSPAN*TESUB7(I)*1.475*IDVFLAG
```

Figure 4.8. Modification of the TE simulation

To avoid a process shutdown due to kinetics drift, the operator starts decreasing the Stream 11 flow rate (XMEAS(17)) set point in 1% increments every 15 minutes, as soon as a reaction kinetics drift is detected, by a total value of 10% of the original set point. This action reduces the reaction rates, decreases the reactor pressure, and effectively reduces the risk of process shutdown. After detecting the reaction kinetics drift, the plant personnel also perform a root-cause analysis of the fault, following which the fault is rectified. It takes from 10 to 24 hours from fault detection to fault rectification. At the end of fault rectification the flag of *Disturbance 13* in the TE computer code is changed back to zero and the set point for the flow rate of Stream 11 is returned to its original value in 1% increments. These actions are programmed in a modified version of the control code **`TEMAIN_MOD.F`** given in Appendix 1.

4.3.3 Specifying the Monitored Faults and Fault Consequences

Before our proposed algorithm starts generating models to monitor the "*Reaction Kinetics Drift*", the conditions of this problem must be specified and formalized in terms of the attributes proposed in Chapter II and listed in Table 3.1.

At the starting point (Step 4 in Section 4.1.2), the user specifies the faults whose monitoring he or she wants to automate and names the models whose design and parameters must be identified. In addition, the user specifies possible consequences associated with these faults and their costs. Finally, the user specifies the costs of a single false alarm for each monitored fault. These costs may vary depending on when the false alarm is produced: during the normal operation of the monitored process or during the occurrence of other listed faults.

In the problem of monitoring the reaction kinetics drift, there is only one fault ("*Reaction kinetics drift*"). There are two possible consequences of this fault: one is the loss of the products and reagents plus an incomplete process capacity utilization, and the other one is a possible process shutdown. The "*Process shutdown*" cost is fixed at *$25,000* and the cost of a single false alarm of the Reaction Kinetics Drift is assumed to be *$200*. The cost C_{Rj} of the "*Reagent & product loss plus an incomplete capacity utilization*" for instance *j* of the "*Reaction Kinetics*

Drift" is calculated as the fault-related change in the cost of the raw materials and utilities consumed plus the market value of the pure products not manufactured as a result of the fault. The resulting equation for calculating this cost will consist of a difference of two costs:

$$C_{Rj} = C_{nj} - C_{aj} \tag{4.11}$$

The first part of the difference is the expected cost C_{nj} of the final products that would have been manufactured minus the cost of the inputs consumed during the time span of the fault instance j if the TE process had operated normally during that period:

$$C_{nj} = \left(K_{out}^{m} - K_{in}\right)\left(t_{rj} - t_{ij}\right) \tag{4.12}$$

In Equation 4.12, t_{ij} is the earliest time when the fault instance j started; t_{rj} is the time the fault instance j was fully corrected, K_{in} is the average cost of reagents and utilities supplied per unit time to the TE process during the normal operation, and K_{out}^{m} is the average market price of pure **G** and **H** produced by the TE process per unit time.

The second part C_{aj} of Equation (4.11) is the difference of the cost of the final products actually manufactured during the time span of the fault instance j and the cost of the inputs actually consumed during the fault instance j:

$$C_{aj} = \int_{t_{ij}}^{t_{rj}} \left\{ \frac{V_{11}(t)}{\sum\limits_{k=D}^{H} \frac{x_k(t)}{\rho_k}} \left[x_G(t) K_G^m + x_H(t) K_H^m\right] - \left[\sum_{k=1}^{4} Q_k(t) \cdot CF_k \cdot K_k\right] \right\} dt \tag{4.13}$$

In Equation 4.13, $V_{11}(t)$ is the volumetric flow rate of the output *Stream 11*, x_k is the molar fraction of component k in *Stream 11*, ρ_k is the molar density of component k, K_G^m and K_H^m are the costs of one molar unit of pure products **G** and **H** respectively, $Q_k(t)$ is the flow rate of input stream k, CF_k is the factor to convert the flow rate, and K_k is the cost of a unit of mixture supplied to the process by input Stream k. The integral in Equation 4.13 is estimated using the method of rectangles with the step equal to the difference between the sample interval of product concentration measurements (15 minutes).

This algorithm for calculating the cost of the "*Reagent & product loss plus an incomplete capacity utilization*" assumes:

1) Full recovery of the products **G** and **H** during purification

2) Constant cost of the utilities at the purification stage that follows the TE process.

These two assumptions are purely due to the lack of information about the purification process. These assumptions would not be needed if the historicized values of the purification process variables were available. The characteristics of the variables of Equations 4.6 and 4.7 are given in Table4.4.

Table 4.4. Summary of the variables in Equation 4.13

Variable	$Q_A(t)$	$Q_B(t)$	$Q_C(t)$	$Q_D(t)$	CF_AK_A	CF_BK_B
TE variable (if applicable)	*XMEAS(1)*	*XMEAS(2)*	*XMEAS(3)*	*XMEAS(4)*		
Constant value (if applicable)					98.62	0.6894
Units	kscmh	Kg/h	Kg/h	kscmh	$·Kmole/kscm	$·Kmole/kg

Variable	CF_CK_C	CF_DK_D	K_{in}	K^m_{out}	$V_{11}(t)$	x_D	x_E
TE variable (if applicable)					*XMEAS(17)*	*XMEAS(37)*	*XMEAS(38)*
Constant value (if applicable)	0.3164	188.5	5745	7379			
Units	$·Kmole/kg	$·Kmole/kscm	$/h	$/h	m^3/h		

Variable	x_F	x_G	x_H	ρ_D	ρ_E	ρ_F	ρ_G
TE variable (if applicable)	*XMEAS(39)*	*XMEAS(40)*	*XMEAS(41)*				
Constant value (if applicable)				9.3438	7.935	6.833	9.871
Units				Kmole/kg	Kmole/kg	Kmole/kg	Kmole/kg

Variable	ρ_H	K^m_G	K^m_H
TE variable (if applicable)			
Constant value (if applicable)	8.118	40.59	30.58
Units	Kmole/kg	$/Kmole	$/Kmole

Figure 4.9 shows how a software implementation of our proposed algorithm would prompt the user to specify the information on the monitored faults and their consequences and how the user would respond to the "*Reaction Kinetics Drift.*"

List of Monitored Faults

Fault Number	Fault Name	Automated monitoring
1	Reaction kinetics drift	○ No [✓ Yes, using model **A**]
2	All the other possible faults	[✓ No] ○ Yes
3	ADD ANOTHER FAULT	

List of Consequences and Their Costs

	Consequence name	Consequence cost accumulates:	Accuracy of estimation	This consequence is associated with faults:
1	Reagent & product loss plus an incomplete capacity utilization	☐ In one discrete step with one-time cost: \$ ☐ At a constant rate of \$/s [✓ As the following function of process variables:] $C_{Rj} = C_{nj} - C_{aj}$	5%	[✓ Reaction kinetics drift]
2	Process shutdown	[✓ In one discrete step with one-time cost: \$25,000] ☐ At a constant rate of \$/s ☐ As the following function of process variables: ____________________________	20%	[✓ Reaction kinetics drift]

Costs of a Single False Alarm

Cost of model implementation and operation (\$/yr.):_\$300_			
Cost of false alarms	During the normal operation		\$100
	During the occurrence of:	All the other faults	[✓ Same as during the normal operation] ☐ \$____

Figure 4.9 User menus for specifying the monitored faults and their possible consequences

4.3.4 Identifying the Monitored Process Variables and Their Characteristics

After the user specifies the faults he wants to monitor and possible fault consequences, he or she should select a block of the monitored process variables that will serve as inputs to the fault model and the characteristics of these variables. For this block, the user should specify the variables related to the monitored fault (Step 5 in Section 4.1.2).

Since the *"Reaction kinetics drift"* occurs in the reactor, it makes sense to select the reactor variables (pressure, temperature, level, input flow rate, input component concentrations, and the reactor cooling water temperature) as the fault model inputs. In addition, the reaction kinetics drift affects the reactor outputs, therefore, it also makes sense to monitor process outputs: the concentrations of the components in the output Streams 9 and 11 as well as the flow rate of the stream exiting the reactor. The reactor exit flow rate is approximately a sum of the flow rates in Streams 8, 9 and 10. The block of the monitored process variables is contoured in Figure 4.10 below.

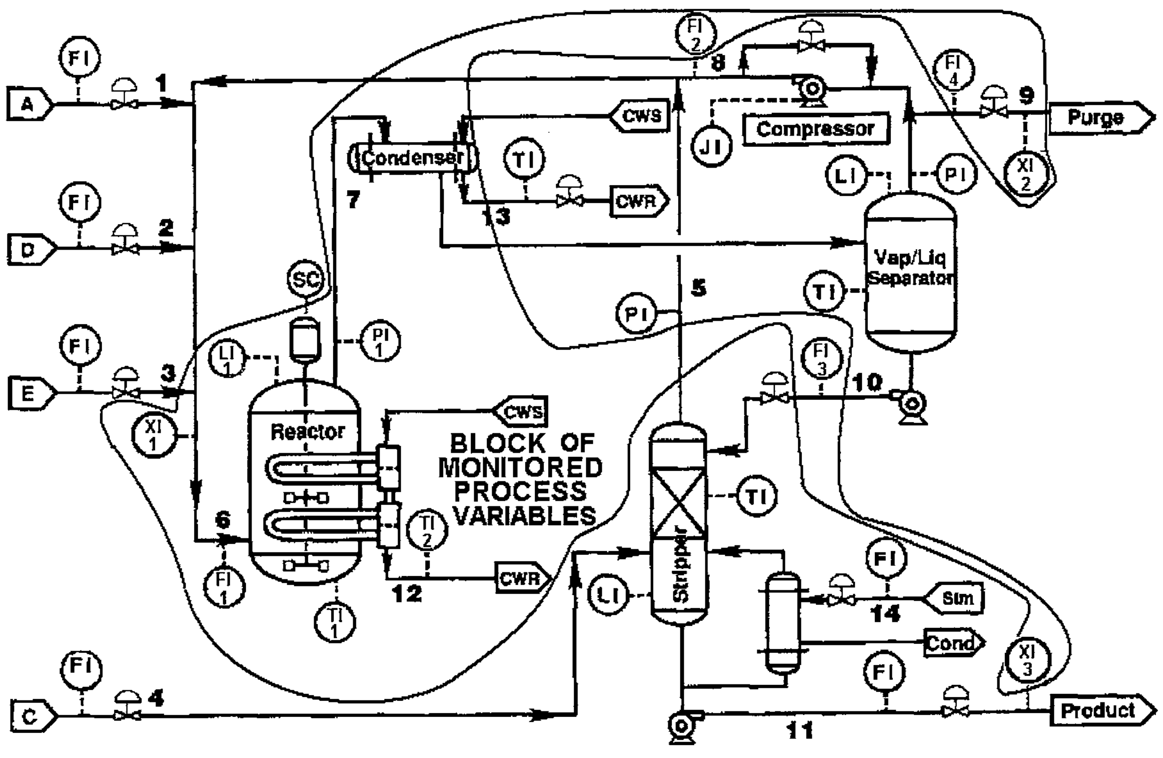


Figure 4.10. Process variables believed to be affected by Disturbance 13

Table 4.5. The list of monitored process variables and their characteristics

Instrument tag and stream number		Text descriptor		TE variable	Algorithm variable number	Process data historian information on the selected process variables			Smooth the process variable measurements	Process variable type
						Units	Historian sampling period, T_S seconds	Noisy signal		
XI-1	6	Reactor feed: molar fractions of the components	A	XMEAS(23)	1	mol%	360	YES	[✓ Yes] No	input
			B	XMEAS(24)	2	mol%	360	YES	[✓ Yes] No	input
			C	XMEAS(25)	3	mol%	360	YES	[✓ Yes] No	input
			D	XMEAS(26)	4	mol%	360	YES	[✓ Yes] No	input
			E	XMEAS(27)	5	mol%	360	YES	[✓ Yes] No	input
			F	XMEAS(28)	6	mol%	360	YES	[✓ Yes] No	input
XI-2	9	Purge gas: molar fractions of the components	A	XMEAS(29)	7	mol%	360	YES	[✓ Yes] No	output
			B	XMEAS(30)	8	mol%	360	YES	[✓ Yes] No	output
			C	XMEAS(31)	9	mol%	360	YES	[✓ Yes] No	output
			D	XMEAS(32)	10	mol%	360	YES	[✓ Yes] No	output
			E	XMEAS(33)	11	mol%	360	YES	[✓ Yes] No	output
			F	XMEAS(34)	12	mol%	360	YES	[✓ Yes] No	output
			G	XMEAS(35)	13	mol%	360	YES	[✓ Yes] No	output
			H	XMEAS(36)	14	mol%	360	YES	[✓ Yes] No	output
XI-3	11	Product: molar fractions of the components	D	XMEAS(37)	15	mol%	900	YES	[✓ Yes] No	output
			E	XMEAS(38)	16	mol%	900	YES	[✓ Yes] No	output
			F	XMEAS(39)	17	mol%	900	YES	[✓ Yes] No	output
			G	XMEAS(40)	18	mol%	900	YES	[✓ Yes] No	output
			H	XMEAS(41)	19	mol%	900	YES	[✓ Yes] No	output
FI-1	6	Reactor feed rate		XMEAS(6)	20	kscmh	15	YES	[✓ Yes] No	input
FI-2	8	Recycle flow		XMEAS(5)	21	kscmh	15	YES	[✓ Yes] No	
FI-3	10	Product separator underflow		XMEAS(14)	22	m^3/h	15	YES	[✓ Yes] No	output
FI-4	9	Purge rate		XMEAS(10)	23	kscmh	15	YES	[✓ Yes] No	output
TI-1		Reactor temperature		XMEAS(9)	24	°C	15	YES	[✓ Yes] No	
TI-2		Reactor cooling water outlet temperature		XMEAS(21)	25	°C	15	YES	[✓ Yes] No	
LI-1		Reactor level		XMEAS(8)	26	%	15	YES	[✓ Yes] No	
PI-1		Reactor pressure		XMEAS(7)	27	Kpa	15	YES	[✓ Yes] No	

For each selected variable, the process data historian provides the measurement units, the sampling period, and whether the variable is noisy or not. After viewing this information, the user would specify the process variable types and the need to smooth the variables. Table 4.5 below shows the list of the process variables selected for monitoring the *"Reaction Kinetics Drift"* and their characteristics. In addition to the characteristics listed in Table 4.5, it can be added that the selected process variables are strongly correlated. For the normal operation of the monitored process, about 95% of the variance in the selected process variables can be explained by at most *14* principal components.

4.3.5 Entering User Preferences and the Characteristics of the Monitored Process

The next step after specifying the monitored process variables and their characteristics is indicating the user preferences and identifying the characteristics of the monitored process (Step 6 in Section 4.1.2).

For this version of the fault model selection algorithm the user is expected to indicate in what form he or she expects to receive the model output: as a fault score or as a binary signal assuming values *"fault"* and *"no fault."* In the TE simulation the Reaction Kinetics Drift is initiated in an abrupt fashion by setting a fault flag to one, hence, the output should also be in a binary form. In addition, this version of the algorithm expects the user to specify the maximum time for creating and evaluating a single fault model and the frequency of fault model execution during the real-time fault monitoring. For the model monitoring the *"Reaction kinetics drift"* these two values are set at one hour and ten minutes, respectively.

The characteristics of the monitored process consist of the monitoring window width, maximum expected process delay and the longest time constant of the monitored process. It is recommended to set the former value to the sum of the two latter ones. The maximum expected process delay and the longest time constant of the monitored process are usually known from the tests performed to tune the process control system. Figure 4.11 shows a graph reproduced from (Lyman and Georgakis 1995). This graph indicates the response of the monitored process to the step change in the set points for the product composition (ratio of variables 18 and 19). These variables produce the slowest response to the set point change among those selected for the monitored variable block. The required characteristics of the monitored process have been identified using this step response as shown in Figure 4.11. Figure 4.12 shows how the user would fill an interactive software menu to specify the user preferences and monitored process characteristics to monitor the *"Reaction kinetics drift."*

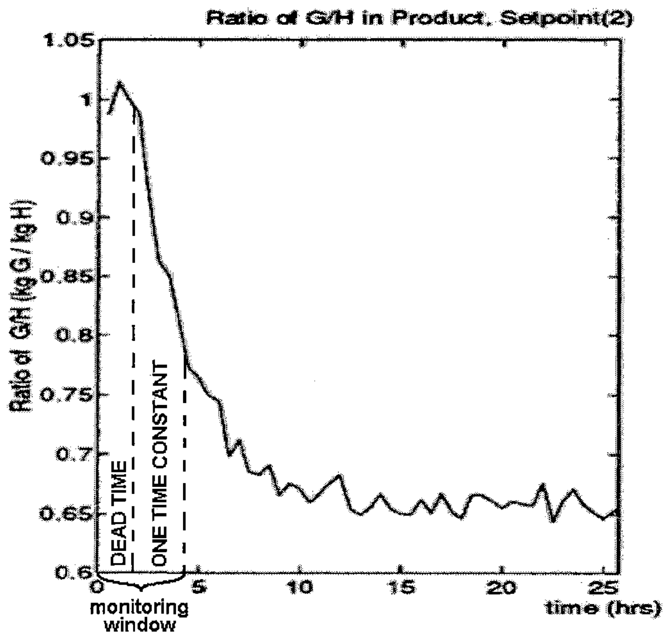


Figure 4.11. Heuristic identification of the monitoring window width using a step response (Lyman and Georgakis 1995)

Detection and Reporting of "Reaction kinetics drift"

Fault name	Kinetics Drift
Desired output indication	[✓ Fault / No Fault] ☐ Fault Score
Maximum allowable time for creating and evaluating a single model	○ 5 minutes [✓ 1 hour] ○ 10 hours ○ 3 days
Application execution frequency	[✓ Every 10 minutes] ○ As often as possible
Monitoring window width	14400 seconds
Maximum expected process delay	5400 seconds
Longest time constant for the monitored process	9000 seconds
Number of process data points sampled during the longest time constant	20

Figure 4.12. User menu for specifying the user preferences and process characteristics for monitoring the *"Reaction Kinetics Drift"*

4.3.6. Providing Optional Information about the Fault and Fault Models

At this step (Step 7 in Section 4.1.2), the user provides a priori information about the monitored process and fault. The user may want to provide a first-principle process model whose residuals reflect the monitored fault better than the selected process variables by themselves. The user may also provide a specific classifier, as an additional fault model component that he or she believes will perform better than the existing library components in monitoring the fault. Some fault model components require specifying the process variable that the user believes is most closely related to the monitored fault. Finally, the user should indicate whether the block of the process variables selected at Step 5 could be used by the fault model components to create an empirical process model.

Figure 4.13 below shows how the user would specify the optional information for monitoring the *"Reaction Kinetics Drift"* for a software implementation of our fault model selection algorithm. In this example, no first-principle model of the monitored process is available and no user-specified classifier that may perform very well when monitoring this fault is available. However, since the process variable types have been specified for the monitored process, it is assumed that an input-output process model is identifiable. The user also believes that the concentration of product ***H*** in the product stream (Variable 19) is most closely related to the monitored fault. Indeed, reaction kinetics changes should primarily affect the concentrations of the products in the product streams.

Optional Information on Fault-Monitoring Models

Adding a user-specified fault classifier	[✓ No] ○ Yes, empirical ○ Yes, first-principle
Using a monitored process model	[✓ No] ○ Yes (provide the module, inputs, and outputs)
Can identify input-output model of the monitored process	○ No [✓ Yes]
Process variable assumed most closely related to the fault	[✓ _19_] ○ Unknown

Figure 4.13. Providing optional information about the fault and fault models for the *"Reaction Kinetics Drift"*

4.3.7 Selecting and Marking Historical Data for Model Identification

Fault model identification usually requires historical data that describes the fault to be monitored. With the help of a process data historian, the operator should find and label the historical data intervals corresponding to the previous instances of the fault(s) to be monitored. As was described in Chapter III, estimation of fault model performance also requires that the user identify points in time associated with fault inception, fault detection, fault rectification, and fault consequences for each instance of the monitored fault(s). Since the exact time of fault inception is usually unknown, the user should specify the earliest and the latest possible time of fault inception for each historicized instance of the monitored fault. The average of these two values will be assumed to correspond to time of fault inception. Also, fault model identification requires the user to specify several additional intervals of historical data recorded during the absence of the monitored faults. It is very desirable to include the instances of the faults not monitored by the generated model in these additional historical data; however, this work considers only a single-fault case.

Historicized Intervals of Normal Operation

	Interval beginning		Interval end
1	Date/Time		Date/Time
2	Date/Time		Date/Time
3			
Total length of the selected normal operation data		**Required**	0.31 years
		Actual	

Tagging Historicized Instances of "Reaction kinetics drift"

	Chronology of faults historicized over the period of (yrs.): 10				**Fault consequences**	
	Earliest possible time of fault inception	**Latest possible time of fault inception**	**Fault detection occurred at**	**Fault was mostly rectified after**	"Off-spec product & reagent loss" **started to occur at**	"Process shutdown" **occurred at**
1	Date/Time	Date/Time	Date/Time	Date/Time	Date/Time	Date/Time
2	Date/Time	Date/Time	Date/Time	Date/Time	Date/Time	Date/Time
3						

Figure 4.14. Specification of historical data for creating models for automated monitoring of the "*Reaction kinetics drift*"

Figure 4.14 on the previous page shows a menu that the operator will need to complete to prepare the historical data for identifying fault models that monitor the "*Reaction kinetics drifts*" in the TE process. For the example described in this chapter, six instances of the "*Reaction kinetics drift*" have been simulated. These instances assumed to have occurred over the period of ten years are summarized in Table 4.6. Hence, according to the scheme described in Section 3.3.1, six different models should be identified for each valid and suitable model design. The training data for each of these six models includes a unique combination of five of the six available historicized instances of the "*Reaction Kinetics Drift.*"

Table 4.6. Simulation of "*Reaction Kinetics Drifts*" in the TE process

Fault instance	**Random seed G in *TEINIT***	**Time (s) since the beginning of the simulation until:**				**Detection lag, s**	**Accumulated Cost of Fault Consequences ($)...**			
		fault inception	**fault detection**	**fault rectification**	**process shutdown**		***...from fault inception till fault detection***	***...from fault detection till fault rectification***	***...total***	***Standard deviation***
1	4651207995	18000	80000	210000	–	62000	449	5508	5957	298
2	9873223412	36000	58750	–	58750	22665	25053	0	25053	5000
3	3485834345	24000	90000	210000	–	66000	1773	6051	7824	391
4	4346024432	45000	103200	–	103200	58200	25770	0	25770	5000
5	7984782901	51000	90000	240000	–	39000	326	4425	4751	238
6	8934302331	6000	207750	–	207750	201750	26553	0	26553	5001

For each of the six fault instances, the costs of fault consequences have been calculated using the methodology described in Section 4.3.3. For the cases when the process shutdown preceded the detection, the pre-detection consequence cost included the cost of a process shutdown. The average standard deviation of the accumulated cost of fault consequences is *$2655* or *$1593* when calculated on the annual basis. This is *1600%* of the cost of a single false alarm. Hence, according to Equation (3.24), for the error of estimation of the cost associated with false alarms to be sufficiently small, the test data set should include at least *5/16=0.31* years or approximately *six million* seconds of the normal operation. To comply with this requirement, twenty simulations *300,000* second long were performed for the normal operation of the TE process. For each model, two of these simulations were randomly chosen for model identification and the remaining eight were held out for model evaluation. The total length of the historical data is *7,029,700* seconds and it contains *468,647* different process data points recorded during the operation of the TE process. Of these points, *374,000* are used for the model testing and the remaining *95,000* are used for model testing. Figure 4.15 below shows the accumulated cost trajectory for *Instance 1* of the "*Reaction Kinetics Drift.*"

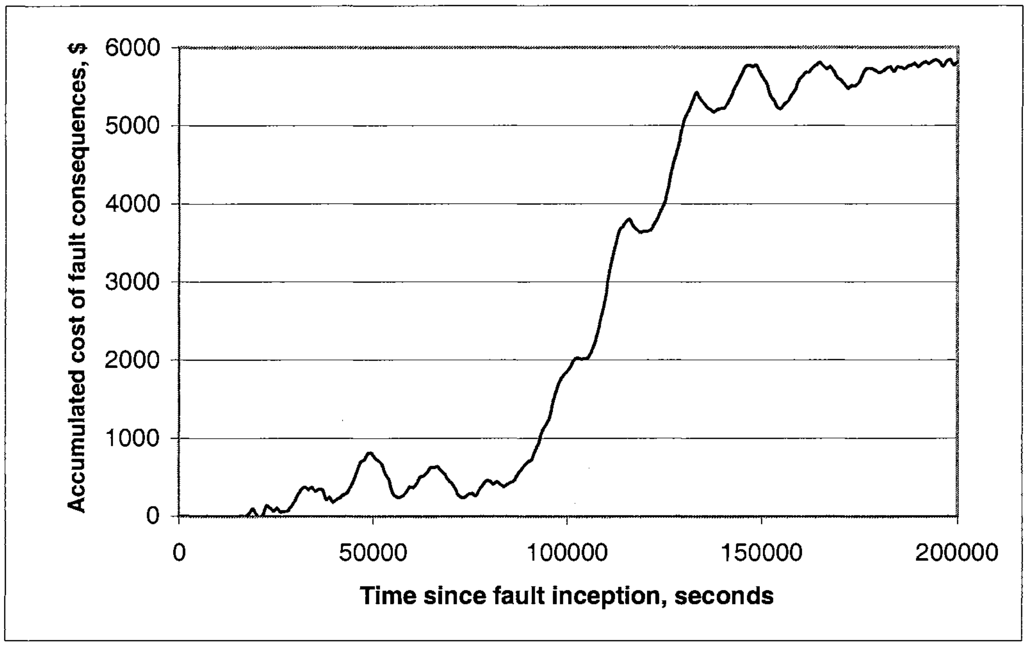


Figure 4.15. Accumulated cost of the consequences of *Instance 1* of the "*Kinetics Drift*"

4.3.8 Estimating the Expectation of the Benefit of Early Detection of a Single Instance of "*Reaction Kinetics Drift*"

As outlined in Chapter III, the benefit of early fault detection is estimated as an average difference between the costs that have actually been incurred for the historicized fault instances and the costs that would have been incurred if these instances had been detected using the proposed models. The latter is hard to estimate even if estimates of the consequence costs are available for every moment of the historicized fault instances. The reason is that the costs incurred from the time of fault detection until fault rectification are not fixed and they may depend on the timeliness of fault detection and the actions of the operators. As a rule, the later a fault is detected, the greater is the expected consequence cost that will be incurred from fault detection till fault rectification.

To estimate the consequence costs incurred during fault recovery, it can be assumed that these costs do not depend on the detection lag if the detection occurs early enough to avoid the process shutdown. Furthermore, no process shutdowns have been observed during the fault rectification procedures, so, it is assumed that the only consequence occurring during the recoveries is the "*Off-spec product & reagent loss.*" It is also assumed that the ability of fault models to detect the "*Reaction Kinetics Drift*" ahead of the operator is not affected by the variations in the potential costs incurred during the fault recovery from one fault instance to another. Then, if a fault model detected a historicized instance of the "*Reaction Kinetics Drift*" earlier than the operator, the expected cost of the consequences to be incurred after the model-based detection would be the average post-detection cost for the historicized instances with no process shutdown (instances 1, 3, and 5), which is *$5328* according to Table 4.6. The standard deviation of the post-detection cost for instances 1, 3, and 5 is *$828*. For the cases when a fault model did not detect a historicized fault instance before, the benefit of fault model-based early detection is assumed zero.

The after-the-detection costs can also be estimated using the empirical frequency histogram-based "quick and dirty" method outlined in Chapter III. Since the post-detection consequence cost depends more on the pre-detection cost than on the detection lag, an empirical distribution of the pre-detection costs (and not detection lags for manual fault detection) for the "*Reaction Kinetics Drift*" is approximated using the proposed histogram.

Each frequency of the histogram includes the costs incurred between fault inception and fault detection for at least three fault instances. Hence, two frequencies are created: the first one includes the pre-detection costs for fault instances 1, 3 and 5 and the second one includes the pre-detection costs for kinetics drift instances 2, 4, and 6. The boundary between these frequencies is the average between the largest pre-detection cost of the first frequency and the smallest pre-detection cost of the second frequency, which is *$13,413*. The lower limit of the first frequency is the smallest possible pre-detection cost, which is zero and the upper limit of the second frequency is the largest possible pre-detection cost, which is positive infinity. The expectation of the post-detection cost for cases when the pre-detection cost is in the first frequency, i.e., below *$13,413*, is the average of the post-detection costs for the kinetics drift instances 1, 3, and 5 falling into this first frequency. This average is identical to the analytical result shown in the previous paragraph. For the six fault instances in Table 4.6, the pre-detection cost being in the second frequency effectively means that the model did not detect the fault before the process shutdown, and for those cases the benefit of fault model-based early detection is assumed zero.

4.3.9 Attributes of the Reaction Kinetics Drift Monitoring Problem

At this point, all the preparations for generating models for monitoring "*Reaction Kinetics Drift*" are complete. Table 4.7 below summarizes the information introduced in Sections 4.3.2 to 4.3.7 in terms of the attributes (listed in Table 2.3) that determine the compatibility of the fault model components.

In Section 4.3.5, the user has set the sampling period for the process data variables at *450* seconds according to the process characteristics. The feature generation and classification parts of the fault model use this sampling period. However, the data smoothing parts of the fault models use the highest possible resolution available for this data in the process data historian, which is *15* seconds. In addition, for variables 15-19 the sampling period of the data historian is *900* seconds, which is greater than the *450* seconds inferred from the process characteristics. To avoid the redundancies in the process data vector, these variables are supplied to the feature generator at the sampling interval of *900* seconds. Table 4.8 summarizes how the monitored process variables are sampled for the fault model and explains the origin of the attributes $D1_A$, $D1_C$, $D6_A$, and $D6_C$.

Table 4.7. Attributes identified for the fault "*Reaction kinetics drift*"

Attribute types	**List of attributes**	
	Symbolic form	**Verbal form**
Attributes of the process data vector	$D1_A$=358	Average number of different measurements of a single variable in the monitoring window available for data smoothing is 358
	$D1_C$=29	Average number of different measurements of a single variable in the monitoring window available for feature generation and classification is 29
	$D2_A$=27	The number of variables in the data is 27
	$D3_A$=FALSE	Data vector consists of original process variables
	$D4_A$=TRUE	The elements of the process data vector are strongly correlated
	$D5_A$=FALSE	There is noise in the user-specified process variables
	$D6_A$=960	Maximum number of different measurements of a single variable in the monitoring window available for data smoothing is 960
	$D6_C$=32	Maximum number of different measurements of a single variable in the monitoring window available for feature generation and classification is 32
Historical data properties	H1=95000	The number of different process variable measurements in the training data is 95000
	H2=4	The smallest number of instances is historicized for the "Reaction Kinetics Drift," this number is four
	H3=TRUE	Contains a variable believed to be most closely related to the monitored fault
Specifics of the monitored process and user preferences	S1=TRUE	Monitored process model not identifiable (input & output variables not specified completely)
	S2=TRUE	Monitored process model not available
	S3=FALSE	Designed fault model monitors a single fault
	S4=FALSE	The fault score-type output is not acceptable
	S5=TRUE	Fault model output must be categorical
	S6=3.5	Log_{10} of the upper limit on the computer time required for model identification is 3.5
	N_S=27	The number of process variables to smooth is 27

Table 4.8. Sampling process measurements for the fault model

Process variable	**Data smoothing**		**Feature generation and classification**	
	Sampling period, seconds	**Number of measurements in the monitoring window**	**Sampling period, seconds**	**Number of measurements in the monitoring window**
1	360	40	450	32
2	360	40	450	32
3	360	40	450	32
4	360	40	450	32
5	360	40	450	32
6	360	40	450	32
7	360	40	450	32
8	360	40	450	32
9	360	40	450	32
10	360	40	450	32
11	360	40	450	32
12	360	40	450	32
13	360	40	450	32
14	360	40	450	32
15	900	16	900	16
16	900	16	900	16
17	900	16	900	16
18	900	16	900	16
19	900	16	900	16
20	15	960	450	32
21	15	960	450	32
22	15	960	450	32
23	15	960	450	32
24	15	960	450	32
25	15	960	450	32
26	15	960	450	32
27	15	960	450	32

4.4. Creation and Selection of Fault Models for Monitoring Reaction Kinetics Drift

As discussed previously, selection of fault models consists of two stages: generation of valid fault model designs and selection of the best-performing valid design.

4.4.1 Generating Valid and Suitable Fault Model Designs

First, *Rules 1-5* screen out individual model components incompatible with the process specifics and user requirements. At this stage, *Rule 5* screens out *Component 3* because its outputs are continuous and hence it possesses attribute *C8*. After that, Rules 6 to 11 screen out the designs where the components are incompatible with each other. Finally, Rules 12 to 23 invalidate the designs incompatible with the properties of the input data.

Table 4.9 summarizes the screening of unsuitable fault model designs using *Rules 6* to *23*. The first column of this table shows the order numbers of the fault model designs that passed the check for the compatibility with the process specifics and user requirements. The second, fifth, seventh, and ninth columns list the attributes of the real-time data used by the input variable selection methods, data smoothing techniques, feature generators and fault classifiers respectively. The attributes of the real-time data change as the input data are processed by fault model components as specified by the data modifiers of the fault model components listed in Table 4.1. The attributes equal to FALSE are omitted, the attributes equal to TRUE are listed without the '=' sign following the attribute. The third and fourth columns list the input variable search criteria and search method numbers respectively used in the model designs undergoing the compatibility screening. These numbers are order numbers of the fault model components listed in Table 4.1 above and used in the screened model designs. The sixth column lists the numbers of the data smoothing components, the eighth column lists the numbers of the residual generators, the tenth column lists the numbers of the optimized criteria, and the eleventh column lists the numbers of the basis functions. The 12th column lists the decimal logarithm of the estimated time in seconds required to identify the fault models for monitoring the Reaction Kinetics Drift. This time is estimated only for the designs where the components are compatible with each other and the input data. The 13th column shows if the corresponding model design is found suitable, and the subsequent columns show the reasons why the model was found unsuitable. The 14th column shows the number of the incompatible components, the 15th column shows the attribute of those components found incompatible, and the 16th column shows the number of the rule (as listed in Tables 2.3, 2.4, and 2.5) that specifies the incompatibility of the component possessing the attribute listed in column 15.

Table 4.9. Screening out invalid fault model designs using the component compatibility rules

Model design number	Attributes of data stream A	Input variable search criterion	Input variable search method	Attributes of data stream B	Data smoothing	Attributes of data stream C	Feature generator	Attributes of data stream D	Optimized Criterion	Basis Function	***Log_{10} of estimated CPU time, s., needed for model identification***	**Model design valid**	**Incompatibility**		
													Component	*Component or model attribute*	*Rule*
1	{D1$_A$ =358} {D2$_A$ =27} {D4$_A$} {D6$_A$ =960}	–	–	{D1$_B$ =358} {D2$_B$ =27} {D4$_B$} {D6$_B$ =960}	–	{D1$_C$ =29} {D2$_C$ =27} {D4$_C$} {D6$_C$ =32}	–	{D1$_D$ =29} {D2$_D$ =27} {D4$_D$} {D6$_D$ =32}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
2		–	–		–		–		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
3		–	–		–		–		**5**	**1**	-2.0 (1)	**YES**			
4		–	–		–		–		**5**	**2**	N/A	**NO**	**2**	**C11**	**17**
5		–	–		–		**6**	{D1$_D$ =32} {D2$_D$ =9} {D3$_D$} {D6$_D$ =32}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
6		–	–		–		**6**		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
7		–	–		–		**6**		**5**	**1**	N/A	**NO**	**1**	**C1**	**12**
8		–	–		–		**6**		**5**	**2**	N/A	**NO**	**2**	**C11**	**17**
9		–	–		–		**7**		**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
10		–	–		–		**7**		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
11		–	–		–		**7**		**5**	**1**	N/A	**NO**	**1**	**C1**	**12**
12		–	–		–		**7**		**5**	**2**	N/A	**NO**	**2**	**C11**	**17**
13		–	–		**8**	{D1$_C$ =29} {D2$_C$ =27} {D4$_C$} {D5$_C$} {D6$_C$ =32}	–	{D1$_D$ =29} {D2$_D$ =27} {D4$_D$} {D5$_D$} {D6$_D$ =32}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
14		–	–		**8**		–		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
15		–	–		**8**		–		**5**	**1**	0.5 (**8**)	**YES**			
16		–	–		**8**		–		**5**	**2**	3.6 (**2**)	**NO**	–	**M1**	**23**
17		–	–		**8**		**6**	{D1$_D$ =32} {D2$_D$ =9} {D3$_D$} {D5$_D$} {D6$_D$ =32}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
18		–	–		**8**		**6**		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
19		–	–		**8**		**6**		**5**	**1**	N/A	**NO**	**1**	**C1**	**12**
20		–	–		**8**		**6**		**5**	**2**	1.9 (**8**)	**YES**			
21		–	–		**8**		**7**		**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
22		–	–		**8**		**7**		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
23		–	–		**8**		**7**		**5**	**1**	N/A	**NO**	**1**	**C1**	**12**
24		–	–		**8**		**7**		**5**	**2**	1.9 (**8**)	**YES**			
25		–	–		**9**		–	{D1$_D$ =29} {D2$_D$ =27} {D4$_D$} {D5$_D$} {D6$_D$ =32}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
26		–	–		**9**		–		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
27		–	–		**9**		–		**5**	**1**	1.0 (**9**)	**YES**			
28		–	–		**9**		–		**5**	**2**	3.6 (**2**)	**NO**	–	**M1**	**23**
29		–	–		**9**		**6**	{D1$_D$ =32} {D2$_D$ =9} {D3$_D$} {D5$_D$} {D6$_D$ =32}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
30		–	–		**9**		**6**		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
31		–	–		**9**		**6**		**5**	**1**	N/A	**NO**	**1**	**C1**	**12**
32		–	–		**9**		**6**		**5**	**2**	2.4 (**9**)	**YES**			
33		–	–		**9**		**7**		**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
34		–	–		**9**		**7**		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
35		–	–		**9**		**7**		**5**	**1**	N/A	**NO**	**1**	**C1**	**12**
36		–	–		**9**		**7**		**5**	**2**	2.4 (**9**)	**YES**			
37		**11**	**10**	{D1$_B$ =358} {D2$_B$ =14} {D4$_B$} {D6$_B$ =960}	–	{D1$_C$ =29} {D2$_C$ =14} {D4$_C$} {D6$_C$ =32}	–	{D1$_D$ =29} {D2$_D$ =14} {D4$_D$} {D6$_D$ =32}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
38		**11**	**10**		–		–		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
39		**11**	**10**		–		–		**5**	**1**	N/A	**NO**	**10**	**C29**	**11**
40		**11**	**10**		–		–		**5**	**2**	N/A	**NO**	**2**	**C11**	**17**
41		**11**	**10**		–		**6**	{D1$_D$ =32} {D2$_D$ =5} {D3$_D$} {D6$_D$ =32}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
42		**11**	**10**		–		**6**		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
43		**11**	**10**		–		**6**		**5**	**1**	N/A	**NO**	**10**	**C29**	**11**
44		**11**	**10**		–		**6**		**5**	**2**	N/A	**NO**	**2**	**C11**	**17**
45		**11**	**10**		–		**7**		**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
46		**11**	**10**		–		**7**		**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
47		**11**	**10**		–		**7**		**5**	**1**	N/A	**NO**	**10**	**C29**	**11**
48		**11**	**10**		–		**7**		**5**	**2**	N/A	**NO**	**2**	**C11**	**17**

Table 4.9. Screening out invalid fault model designs using the component compatibility rules (continued)

Model number	Attributes of data stream A	Input variable search criterion	Input variable search method	Attributes of data stream B	Data smoothing	Attributes of data stream C	Feature generator	Attributes of data stream D	Optimized Criterion	Basis Function	***Log_{10} of estimated CPU time, s., needed for model identification***	**Model design valid**	**Incompatibility**		
													Component	*Attribute*	*Rule*
49	{$D1_A$ =358}	**11**	**10**	{$D1_B$ =358}	**8**	{$D1_C$ =29}	–	{$D1_D$ =29}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
50	{$D2_A$ =27}	**11**	**10**	{$D2_B$ =14}	**8**	{$D2_C$ =14}	–	{$D2_D$ =14}	**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
51	{$D4_A$} {$D6_A$	**11**	**10**	{$D4_B$} {$D6_B$	**8**	{$D4_C$} {$D5_C$}	–	{$D4_D$} {$D5_D$}	**5**	**1**	N/A	**NO**	**10**	**C29**	**11**
52	=960}	**11**	**10**	=960}	**8**	{$D6_C$ =32}	–	{$D6_D$ =32}	**5**	**2**	4.3 **(10)**	**NO**	–	**M1**	**23**
53		**11**	**10**		**8**		**6**	{$D1_D$	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
54		**11**	**10**		**8**		**6**	=32}	**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
55		**11**	**10**		**8**		**6**	{$D2_D$	**5**	**1**	N/A	**NO**	**10**	**C29**	**11**
56		**11**	**10**		**8**		**6**	=5}	**5**	**2**	2.6 (10)	**YES**			
57		**11**	**10**		**8**		**7**	{$D3_D$}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
58		**11**	**10**		**8**		**7**	{$D5_D$}	**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
59		**11**	**10**		**8**		**7**	{$D6_D$	**5**	**1**	N/A	**NO**	**10**	**C29**	**11**
60		**11**	**10**		**8**		**7**	=32}	**5**	**2**	2.6 **(10)**	**YES**			
61		**11**	**10**		**9**		–	{$D1_D$ =29}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
62		**11**	**10**		**9**		–	{$D2_D$ =14}	**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
63		**11**	**10**		**9**		–	{$D4_D$} {$D5_D$}	**5**	**1**	N/A	**NO**	**10**	**C29**	**11**
64		**11**	**10**		**9**		–	{$D6_D$ =32}	**5**	**2**	4.3 **(10)**	**NO**	–	**M1**	**23**
65		**11**	**10**		**9**		**6**	{$D1_D$	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
66		**11**	**10**		**9**		**6**	=32}	**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
67		**11**	**10**		**9**		**6**	{$D2_D$	**5**	**1**	N/A	**NO**	**10**	**C29**	**11**
68		**11**	**10**		**9**		**6**	=5}	**5**	**2**	2.6 (10)	**YES**			
69		**11**	**10**		**9**		**7**	{$D3_D$}	**4**	**1**	N/A	**NO**	**4**	**C15**	**6**
70		**11**	**10**		**9**		**7**	{$D5_D$}	**4**	**2**	N/A	**NO**	**4**	**C15**	**6**
71		**11**	**10**		**9**		**7**	{$D6_D$	**5**	**1**	N/A	**NO**	**10**	**C29**	**11**
72		**11**	**10**		**9**		**7**		**5**	**2**	2.6 (10)	**YES**			

Table 4.10 below lists the valid fault model designs. These designs are presented both as a sequence of numbers and as verbal descriptions. Figure 4.16 summarizes the results of applying the selection rules proposed in Chapter II to all the 108 possible combinations of fault model components listed in Table 4.1. This figure shows how many fault model designs have been deselected by each group of rules.

Table 4.10. Valid model designs suitable for monitoring the "*Reaction Kinetics Drift*"

Candidate model #	Input variable search criterion	Input variable search method	Data smoothing	Feature generator	Optimized criterion	Basis function	Verbal description of the model
3)	–	–	–	–	5	1	S control chart with the out-of-control bounds determined by minimizing the rate of erroneous identification of the monitored process modes
15)	–	–	8	–	5	1	S control chart with the out-of-control bounds determined by minimizing the rate of incorrect identification of the monitored process modes; the process variable for the chart is smoothed using the Haar function-based wavelet transform
20)	–	–	8	6	5	2	Probabilistic neural network created by minimizing the rate of incorrect identification of the monitored process modes; the process variables for the network are smoothed using the Haar function-based wavelet transform and converted to PCA scores accounting for 95% of the variance in the original smoothed variables
24)	–	–	8	7	5	2	Probabilistic neural network created by minimizing the rate of incorrect identification of the monitored process modes; the process variables for the network are smoothed using the Haar function-based wavelet transform and converted to PCA residuals with the PCA scores accounting for 95% of the variance in the original smoothed variables
27)	–	–	9	–	5	1	S control chart with the out-of-control bounds determined by minimizing the rate of incorrect identification of the monitored process modes; the process variable for the chart is smoothed using the **bior3.1** wavelet transform
32)	–	–	9	6	5	2	Probabilistic neural network created by minimizing the rate of incorrect identification of the monitored process modes; the process variables for the network are smoothed using the **bior3.1** wavelet transform and converted to PCA scores accounting for 95% of the variance in the original smoothed variables
36)	–	–	9	7	5	2	Probabilistic neural network created by minimizing the rate of incorrect identification of the monitored process modes; the process variables for the network are smoothed using the **bior3.1** wavelet transform and converted to PCA residuals with the PCA scores accounting for 95% of the variance in the original smoothed variables
56)	11	10	8	6	5	2	Probabilistic neural network created by minimizing the rate of incorrect identification of the monitored process modes; the network uses the PCA scores that explain 95% of the variance in the process variables selected using the floating forward search for the optimal fault model and smoothed using the Haar-function based wavelet filter
60)	11	10	8	7	5	2	Probabilistic neural network created by minimizing the rate of incorrect identification of the monitored process modes; the network uses the residuals for the PCA scores that explain 95% of the variance in the process variables selected using the floating forward search for the optimal fault model and smoothed using the Haar-function based wavelet filter
68)	11	10	9	6	5	2	Probabilistic neural network created by minimizing the rate of incorrect identification of the monitored process modes; the network uses the PCA scores that explain 95% of the variance in the process variables selected using the floating forward search for the optimal fault model and smoothed using the **bior3.1** wavelet filter
72)	11	10	9	7	5	2	Probabilistic neural network created by minimizing the rate of incorrect identification of the monitored process modes; the network uses the residuals for the PCA scores that explain 95% of the variance in the process variables selected using the floating forward search for the optimal fault model and smoothed using the **bior3.1** wavelet filter

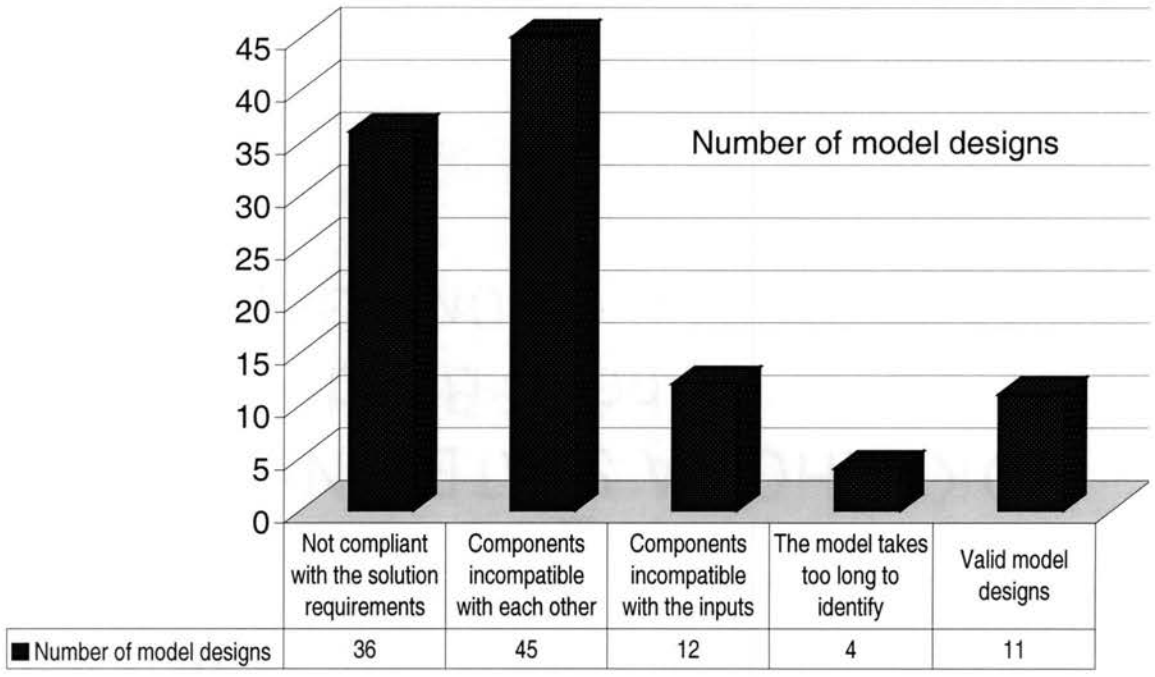


Figure 4.16. A summary of the screening of the candidate fault model designs

Figure 4.17 graphically illustrates how a fault model design was found incompatible due to the incompatibility of the components used in this design with each other. This figure shows that *Rule 11* has found the "S Control Chart" and "Forward Search" incompatible. This rule specifies incompatibility of fault model components whose attribute *C6* is equal to *TRUE* (this is the case for the "S Control Chart" that requires the variable believed most closely related to the monitored fault) with the components whose attribute C29 is equal to *TRUE* (this is the case for the "Forward Search" that is not designed to retain this variable in the input data).

Figure 4.18 graphically illustrates how a fault model design was found incompatible due to the incompatibility of the components used in this design with the input data. Application of *Rule 17* has revealed that the "Probabilistic Neural Network" is an invalid component in the Design number *44* because this component's attribute *C11=TRUE* (i.e., this basis function is noise-sensitive) and the inputs to the "Probabilistic Neural Network" ("*data stream D*") are noisy ($D5_D$=*FALSE*).

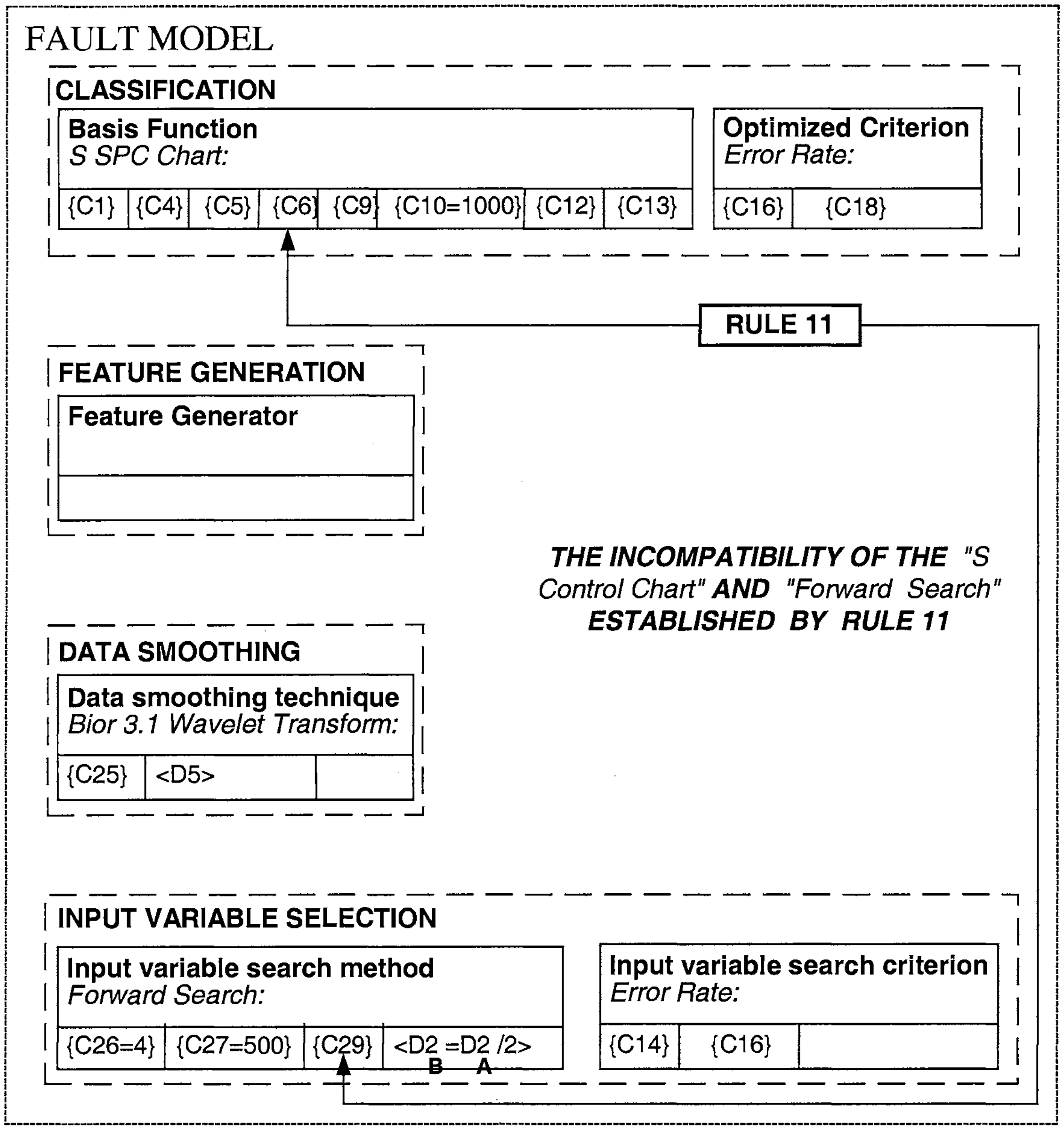

FAULT MODEL
CLASSIFICATION
Basis Function
S SPC Chart:
{C1} {C4} {C5} {C6} {C9} {C10=1000} {C12} {C13}
Optimized Criterion
Error Rate:
{C16} {C18}
RULE 11
FEATURE GENERATION
Feature Generator
THE INCOMPATIBILITY OF THE "S Control Chart" AND "Forward Search" ESTABLISHED BY RULE 11
DATA SMOOTHING
Data smoothing technique
Bior 3.1 Wavelet Transform:
{C25} <D5>
INPUT VARIABLE SELECTION
Input variable search method
Forward Search:
{C26=4} {C27=500} {C29} <D2 =D2 /2>
B A
Input variable search criterion
Error Rate:
{C14} {C16}

Figure 4.17. Invalidating fault model design number 63 because of the fault model component incompatibility with each other

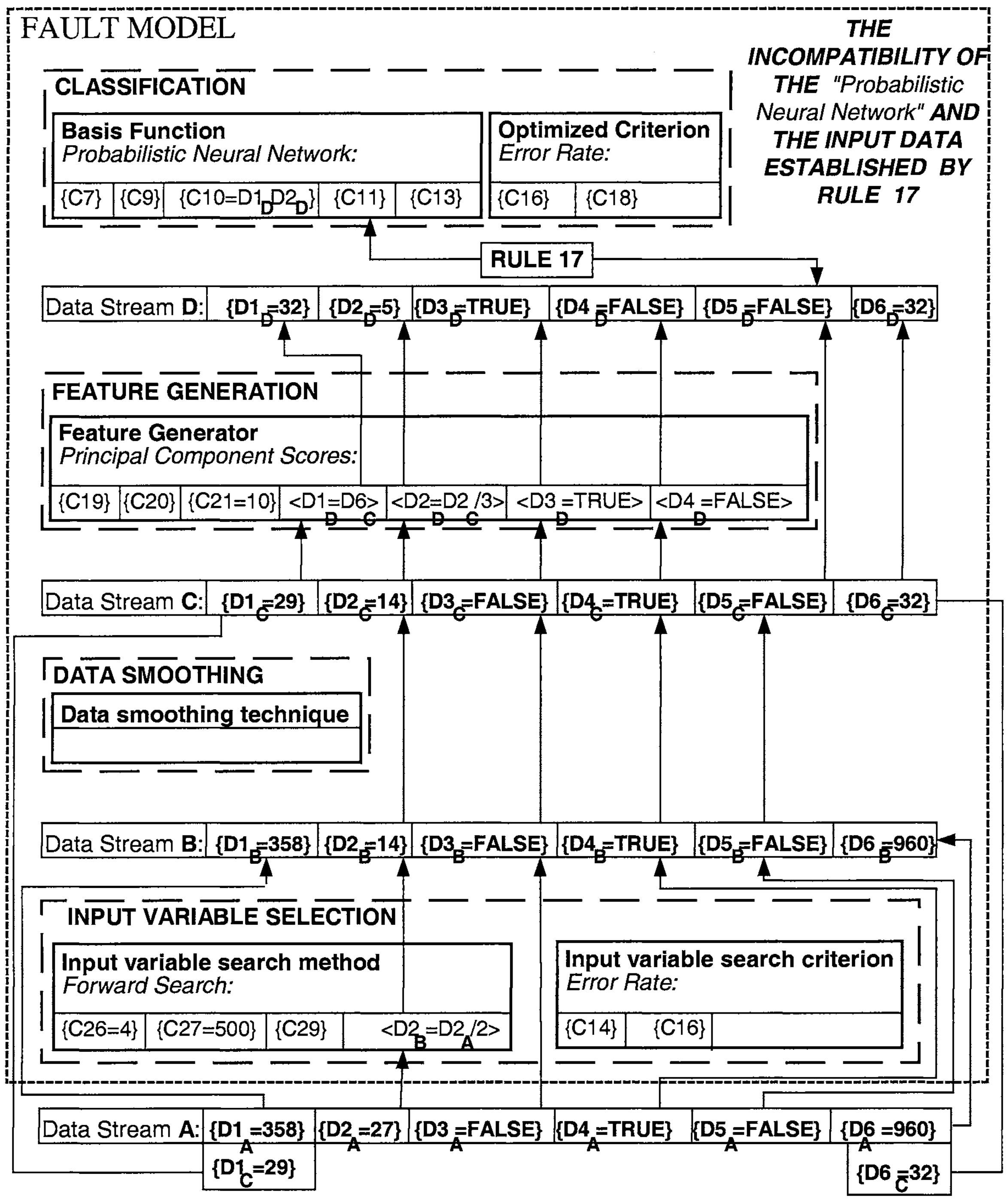

FAULT MODEL
THE INCOMPATIBILITY OF THE "Probabilistic Neural Network" AND THE INPUT DATA ESTABLISHED BY RULE 17
CLASSIFICATION
Basis Function
Probabilistic Neural Network:
{C7} {C9} {C10=D1D D2D} {C11} {C13}
Optimized Criterion
Error Rate:
{C16} {C18}
RULE 17
Data Stream D: {D1D=32} {D2D=5} {D3D=TRUE} {D4D=FALSE} {D5D=FALSE} {D6D=32}
FEATURE GENERATION
Feature Generator
Principal Component Scores:
{C19} {C20} {C21=10} <D1D=D6C> <D2D=D2C/3> <D3D=TRUE> <D4D=FALSE>
Data Stream C: {D1C=29} {D2C=14} {D3C=FALSE} {D4C=TRUE} {D5C=FALSE} {D6C=32}
DATA SMOOTHING
Data smoothing technique
Data Stream B: {D1B=358} {D2B=14} {D3B=FALSE} {D4B=TRUE} {D5B=FALSE} {D6B=960}
INPUT VARIABLE SELECTION
Input variable search method
Forward Search:
{C26=4} {C27=500} {C29} <D2B=D2A/2>
Input variable search criterion
Error Rate:
{C14} {C16}
Data Stream A: {D1A=358} {D2A=27} {D3A=FALSE} {D4A=TRUE} {D5A=FALSE} {D6A=960}
{D1C=29}
{D6C=32}

Figure 4.18. Invalidating fault model design number 44 because of the incompatibility of a fault model component with the input data

4.4.2 Estimation of the Performance of Suitable Fault Model Designs

Identification of the parameters for the fault model designs found suitable for monitoring the "*Reaction Kinetic Drifts*" has been performed according to the guidelines of Chapter III. As was described in Section 4.3.7, the historical data selected for model identification and evaluation covers six fault instances and four intervals of normal process operation. Following the cross-validation procedure described in Chapter III, six models have been identified for each suitable model design. Each model used five historicized instances the "*Reaction Kinetic Drifts*" out of the six available and two intervals of normal operation out of the four available for identifying the model parameters. Each of the six models used a unique combination of five out of the six available historicized fault instances and randomly chosen two intervals of normal operation (out of the four available) for the parameter identification.

The performance of the valid fault designs was evaluated with the **test data sets**: the fractions of the historical data selected by the operator for model identification and not used for identifying fault model parameters. For each model, this evaluation data consisted of measuring:

1) The model-based fault detection lag for the fault instance not used for the model identification
2) The rate of false alarms for the normal operation data not used for model identification.

The results of estimating the performance of the suitable fault model designs and the standard deviation of the performance measure using Equations (3.8) and (3.19) respectively are given in Table 4.11. The expected performance indicators of different fault model designs in monitoring the "*Reaction Kinetics Drift*" are compared in Figure 4.19 below.

Table 4.11. Evaluation of fault model designs suitable for monitoring "*Reaction Kinetics Drifts*"

Fault model designs	**3**	**5**	**19**	**24**	**27**	**32**	**36**	**56**	**60**	**68**	**72**
Expected benefit of early detection, $/yr	561	573	1657	2031	866	695	1270	1942	2381	2289	2221
Expected cost of false alarms. $/yr	4081	3726	1315	1536	0	657	219	657	439	1753	876
Estimated net benefit of fault monitoring automation, $/year	-720	-853	42	195	-1066	262	752	985	1642	1236	1345
Approximate standard deviation of the net benefit estimate, $/year	1651	1609	1295	1209	1563	1376	1324	1303	1481	1267	1271
Performance rank	11	10	7	6	5	9	4	3	1	8	2

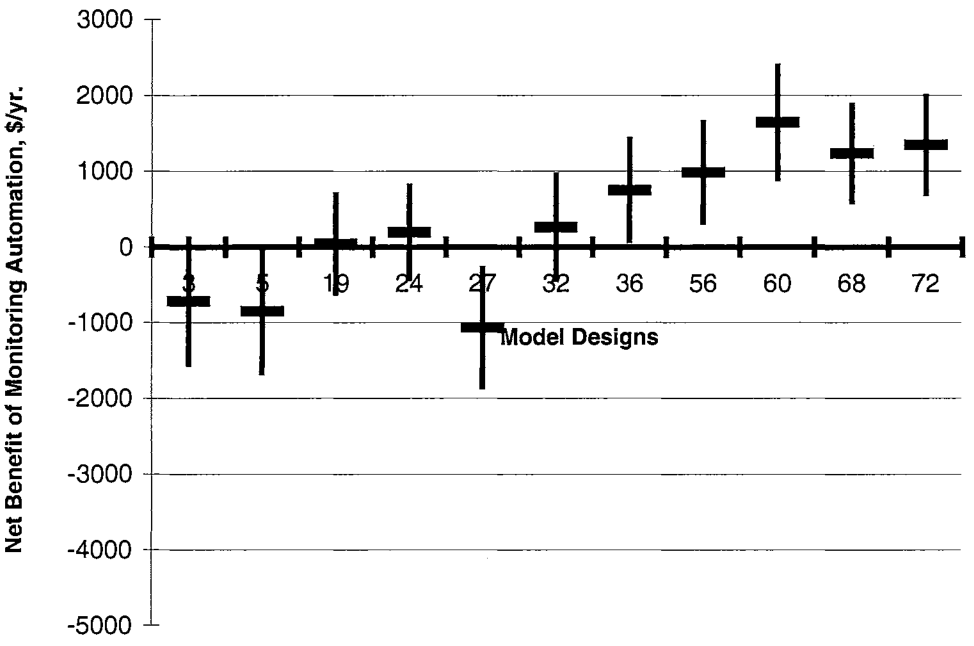


Figure 4.19. Estimated performance of different valid problem-compatible fault model designs

4.4.3 A Brief Discussion of the Results

Most combinations of fault model components were suitable for monitoring the "Reaction Kinetics Drift". Out of *108* possible combinations, only *11* survived the screening. In Figure 4.16, most designs were deselected due to the incompatibility of the participating model components. In fact, these designs can be screened out at the time when the library of model components is being created.

An analysis of the numerical results reveals that the control chart-based fault models do not perform very well in monitoring the "*Reaction Kinetics Drift*" and an implementation of these models will probably not have a positive economic effect on the monitored process. Most probabilistic neural network-based methods ended up in the positive territory. The errors of the net benefit estimates are quite large (as shown by the error bars in Figure 4.19). This is mostly due to the realistically chosen *20%* error in the estimation of the cost of plant shutdowns. Design 60 "*probabilistic neural network created by minimizing the rate of incorrect identification of the monitored process modes; the network uses the residuals for the PCA scores that explain 95% of the variance in the process variables selected using the floating forward search for the optimal fault model and smoothed using the Haar-function based wavelet filter*" was found the best-performing among the eleven tested fault model designs.

4.5 Summary

Chapter IV has illustrated in detail a practical application of the theory of selecting models for automating process monitoring. It guided the reader step-by-step through a complete procedure of fault model design and evaluation. Chapter IV has shown by example how to perform the four separate steps in creating and selecting fault models:

1) Creation of a reusable library of fault model components out of the numerical methods described in the literature;
2) Specification of the attributes of the user-specified fault monitoring problems;
3) Screening out fault model designs not suitable for the user-specified fault monitoring problems;
4) Evaluation of the performance of suitable fault model designs.

The demonstration in Chapter IV describes an automation of the monitoring of only one fault and uses the fault model template with the slots for only one data processing method of each type used in fault models. Despite this fact, the illustration can be easily extended to the case where several faults are monitored simultaneously and fault models are designed from more complex templates allowing the use of different data processing methods in parallel. The illustration provided in Chapter IV is very important for understanding the proposed approach, encoding this approach in software, and applying this software in the industry.

Chapter IV does not contain any evaluations of the proposed approach, as these evaluations must be performed using real-life process data that have a proprietary nature, so these data cannot be published in the open-source literature. These evaluations must be separate for each fault. The evaluations should be carried out by comparing the estimated model performance with the model performance measured on the fault data not included in the training and testing sets. It is also important to note that the relative and absolute performance indicators estimated in this work cannot be projected to, extrapolated to or factored in the expected performance of the tested fault models in the industrial practice. Each real-world problem will require a separate estimation and comparison of the performance of different fault models.

Chapter V

Conclusions

5.1 An Outline of the Contributions Made by this Work

This work has made four important contributions.

First, this work has documented the reasons why despite the superabundance of different computer-based techniques that can take over many routine human activities, the automation of the real time fault monitoring in the process industries has been so difficult. This work has also systematized different existing methods for fault monitoring as different stages in the mapping from the space of the monitored variables to the space of faults.

Second, this work has proposed a new expert system-type approach for designing fault models suitable for solving user-specified fault-monitoring problems. This approach decomposes fault models into a library of components that perform specific functions. In the library, each component has a set of characteristic attributes. This work has also defined a set of rules determining the compatibility of different model components with the user requirements, characteristics of the monitored process, monitored faults, properties of the input data, and the other components participating in the fault model. By placing different subsets of model components on the proposed template and verifying the validity of the resulting combinations, various fault models, both conventional and new, can be designed.

Third, this work has proposed a method for selecting the best fault model design. This method specifies how to sample the available historical data and how to quantify the cost of fault consequences and their dynamics. The proposed measure of fault model performance is the potential increase in the profitability of the monitored process due to the implementation of this model. In addition, evaluation of the confidence interval for this performance measure has been described.

Fourth, this work has included a comprehensive example of an application of the proposed methodology. This application has used a small library of fault model components to design models suitable for monitoring faults in a well-known public-domain simulation of a chemical process. The example has also included evaluation of the suitable model designs. Using this demonstration it will be easier to implement the proposed algorithm in the industry.

5.2 Advantages and Disadvantages of the Proposed Method

Modeling process faults is a very difficult problem because each fault-monitoring problem has specific, sometimes unique conditions. In addition, it is difficult to create a good data sample for the fault model identification as all the process data points historicized during the occurrence of the same fault instance are related to each other and the number of historicized fault instances is usually small. Hence, the suitability of different models for monitoring different faults in the process industries is often unknown a priori. This work has proposed an efficient method for designing and evaluating mathematical models for real-time process fault monitoring. Although other simple algorithms for selecting methods for fault detection and diagnosis have already been proposed, this approach is the first to attempt covering all the possible methods for process fault monitoring (except for the few classifiers that do not allow the separation of the basis function linked with the parameter search technique and the optimized criterion), including those that have never been discussed and those that will be proposed in the future. As was stated in Chapter II, this approach allows:

1) Potential use of components from any mathematical model whose application to fault detection has been discussed in the technical literature to date (including even the aggregated models),
2) Making the problem of fault model search tractable by representing a large number of different fault models as a much smaller number of separate model components,

3) Searching through large libraries of very diverse mathematical methods, which is extremely important in the cases when the shape of the modeled relationship is very hard or impossible to determine a priori.

Even for the cases when the applicable model design is approximately known, (e.g., it may be known that monitoring specific faults is done best with control charts) the proposed method can be used for refining the design further (e.g., it may find out how to select the input variables, smooth the data, and generate the residuals for these charts best).

Although this proposed approach for fault model selection is so universal, its practical application may seem difficult because of the requirement to decompose the known methods for process fault monitoring into components. However, creation of libraries of fault model components is a straightforward task as long as the methods to be decomposed allow separation of the optimized criteria and the basis functions linked with the parameter search methods. Sometimes, one must perform this task creatively, as was done in Subsection 4.2.2.2 to decompose a probabilistic neural network that, in its original implementation, can only minimize the misclassification rate for the training set. It is also important to initialize all the component attributes. This procedure may be complicated for some "opaque" numerical methods supplied with various software packages. For the libraries containing many fault model components, the number of possible combinations of these components is extremely large. However, as was shown in Chapter IV, the proposed component compatibility rules select only very few model designs that are valid and suitable for the specified problem.

This work is also the first one to propose to evaluate the economic effects of implementing real-time fault monitoring in the process (continuous-stream) industry. Unlike the conventional approaches for estimating the fault model performance based on how likely these models are to help in meeting the quality and safety standards, this proposed estimator evaluates how different implementations of fault monitoring reduce the fault-related economic losses. This new approach also has a clear advantage over the other methods for estimating the performance of fault models in that it does not require the operators to find the exact inception time of the historicized fault instances. However, for the sake of the simplicity and clarity, the proposed approach is based on the assumption of no irreversible changes in the monitored process. At this point it is unknown how valid this underlying assumption is for evaluating the economic effect of monitoring the real-world processes most of which do undergo irreversible changes during their lifetime. In addition, tracing fault consequences in the historical data may sometimes be difficult.

5.3 The Impact of this Work Beyond the Scope of the Process Fault Monitoring

Advances in computer hardware allow creation of very large mathematical models with a great number of parameters, massively parallel processing and a great degree of structural redundancy. These models can often imitate very complex phenomena, e.g., meteorological events, nuclear explosions, etc. However, in many cases, simpler models have a clear advantage over the complex ones. This is the case when:

1) The amount of independent data points to train and validate a model is very limited
2) A transparent and/or adaptable model is preferred
3) The model has to operate in real time using a limited amount of computing resources.

All these three conditions apply to the models designed to detect and identify faults in the process industries in real time.

To perform the task of modeling faults, this work has proposed to optimize a selection of model components, or to be more exact, the components of the algorithm according to which a fault model is trained and run. Different from the conventional approaches where the number of the degrees of freedom in a model is equal to the number of parameters, the proposed approach uses additional degrees of freedom in the selection of model components making it possible to build smaller, simpler, and well-generalizing models. The suitability of such an approach is confirmed by the fact that the number of model parameters did not determine model performance in monitoring the TE process: some neural-network based models did not perform well. In addition, the model selection algorithm proposed in this work can be extended to allow model selection for the other application where the above-mentioned conditions 1) to 3) exist. This is the case primarily in the analyses of small samples or samples with interdependent data often encountered in the Bioinformatics, predicting major stock and commodity market events, marketing research and many others.

5.4 Future Work

There are at least three directions for the future work that will improve the method for designing optimal models for fault detection.

The first direction is automation of the proposed method for fault model design. This requires creation of a relational database containing different model components with attributes required by the production rules that design fault model out of these components. In this work, fault model design was performed manually. In the future, an expert system-approach should be used to generate fault models suitable for solving specific user-defined process-monitoring problems. Experienced industrial users are expected to add a lot more fault model design rules in addition to those proposed in this work. If the number of components is relatively large, it is impossible to estimate the performance of every problem-suitable combination of these components, so this proposed method should also incorporate an appropriate discrete search algorithm. This algorithm would allow finding the best-performing fault model by testing only a subset of all the possible combinations of the available fault model components.

The second direction is extending this approach to the cases when no historical data for the fault to be monitored are available. This would require extrapolation of the historical data recorded for other similar faults onto the fault that the user wants to monitor. This method would rely on the "data farms" that share recorded fault data among different factories and enterprises, such as the Process Equipment Reliability Database (Thomas and Moosemiller 2001).

The third direction is the inclusion of the cost of fault misclassifications and the effects of slow irreversible changes in the monitored process in the proposed estimator of fault model performance. In addition, it may be important to be able to use the statistical inference to decide whether an adaptable fault model is needed to monitor a specific process and when the model updates should occur.

References

1. Agresti, A. Introduction to Categorical Data Analysis. New York, NY, Wiley (1996).

2. Akaike, H. Information Theory and an Extension to the Maximum Likelihood Principle. Second International Symposium on Information Theory (1973).

3. Akbaryan, F. and P. R. Bishnoi. "Fault Diagnosis of Single-Variate Systems Using a Wavelet-based Pattern Recognition Technique." Industrial & Engineering Chemistry Research **40**(16): 3612-3622 (2001).

4. Aldrich, C., D. W. Moolman, et al. "Monitoring and Control of Hydrometallurgical Processes with Self-Organizing and Adaptive Neural Net Systems." Computers & Chemical Engineering **v19**(nSuppl): S803-S808 (1995).

5. Al-Ghanim, A. and J. Jordan. "Automated Process Monitoring Using Statistical Pattern Recognition Techniques on X-bar Control Charts." Journal of Quality in Maintenance Engineering **2**(1): 25-49 (1996).

6. Al-Qurashi, F., G. Sharma, et al. "Application of Relational Chemical Process Safety Databases for Lowering Mean Failure Rates." Process Safety Progress **20**(4): 280-285 (2001).

7. Aretakis, N. and K. Mathioudakis. Wavelet Analysis for Gas Turbine Fault Diagnosis. Proceedings of the 1996 International Gas Turbine and Aeroengine Congress & Exhibition (1996).

8. Bakshi, B. "Multiscale PCA with Application to Multivariate Statistical Process Monitoring." AIChE Journal **44**(7): 1596-1610 (1998).

9. Berthold, M. R. and J. Diamond. "Constructive Training of Probabilistic Neural Networks." Neurocomputing **19**: 167-183 (1998).

10. Brassard, M. and D. Ritter. *Control Charts*. A Pocket Guide of Tools for Continuous Improvement and Effective Planning. First. Salem, NH, GOAL/QPC (1994).

11. Brodley, C. E. and P. Smyth. "Applying Classification Algorithms in Practice." *Statistics and Computing* **7**: 45-56 (1997).

12. Brown, L. D., T. T. Cai, et al. "Interval Estimation for a Binomial Proportion." *Statistical Science* **16**(2): 101-133 (2001).

13. Carpenter, G. A., S. Grossberg, et al. "Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps." *IEEE Transactions on Neural Networks* **3**(5): 698-713 (1992).

14. CCPS. *Guidelines for Improving Plant Reliability through Data Collection and Analysis*. New York, NY, AIChE (1998).

15. CCPS. *Event Tree Analysis*. Guidelines for Chemical Process Quantitative Risk Analysis. New York, NY, AIChE (2000a).

16. CCPS. *Fault Tree Analysis*. Guidelines to Chemical Process Quantitative Analysis. New York, NY, AIChE: 304-322 (2000b).

17. CCPS. *Guidelines for Chemical Process Quantitative Risk Analysis*. New York, NY, American institute of Chemical Engineers, Center for Chemical Process Safety (2000c).

18. Chen, J. and J. Howell. "Towards Distributed Diagnosis of the Tennessee Eastman Process Benchmark." *Control Engineering Practice* **10**(9): 971-987 (2002).

19. Chiang, L. H., E. L. Russell, et al. *Canonical Variate Analysis*. Fault Detection and Daignosis in Industrial Systems. London, UK, Springer: 85-99 (2001a).

20. Chiang, L. H., E. L. Russell, et al. *Fault Detection and Diagnosis in Industrial Systems*. London, Springer Verlag (2001b).

21. Chiang, L. H., E. L. Russell, et al. *Partial Least Squares*. Fault Detection and Diagnosis in Industrial Systems. London, UK, Springer: 71-84 (2001c).

22. Chiang, L. H., E. L. Russell, et al. *Results and Discussion*. Fault Detection and Diagnosis in Industrial Systems. London, Springer (2001d).

23. Clark, R. *The Dedicated Observer Approach to Instrument Fault Detection.* Proceedings of the 15th IEEE-CDC. Fort Lauderdale, FL: 237-241 (1979).

24. Crowe, C. "Data Reconciliation - Progress and Challenges." *Journal of Process Control* **6**(2/3): 89-98 (1996).

25. Dash, S. and V. Venkatasubramanian. "Challenges in the Industrial Applications of Fault Diagnostic Systems." *Computers and Chemical Engineering* **24**: 785-791 (2000).

26. Davis, J. F., M. J. Piovoso, et al. "Process Data Analysis and Interpretation." *Advances in Chemical Engineering* **25**: 1-103 (1999).

27. Downs, J. J. and E. F. Vogel. "A Plant-Wide Industrial Process Control Problem." *Computers and Chemical Engineering* **17**(3): 245-255 (1993).

28. Dunia, R., S. J. Qin, et al. "Identification of Faulty Sensors Using the Principal Component Analysis." *AIChE Journal* **42**(10): 2797-2812 (1996).

29. Fantoni, P. F. and A. Mazzola. "Pattern Recognition-Artificial Neural Network-Based Model for Signal Validation in Nuclear Power Plants." *Annals of Nuclear Energy* **23**(13): 1069-1076 (1996).

30. Frank, P. M., S. X. Ding, et al. "Model-Based Fault Diagnosis in Technical Processes." *Transactions of the Institution of Measurement and Control* **22**(1): 57-101 (2000).

31. Frank, P. M. and X. Ding. "Survey of Robust Residual Generation and Evaluation Methods in Observer-Based Fault Detection Systems." *Journal of Process Control* **7**(6): 403-424 (1997).

32. Gertler, J., X. Fang, et al. "Detection and Diagnosis of Plant Failures: the Orthogonal Parity Equation Approach." *Control and Dynamic Systems* **37**: 159-216 (1990).

33. Gertler, J., W. Li, et al. "Isolation-Enhanced Principal Component Analysis." *AIChE Journal* **45**(2): 323-334 (1999).

34. Gertler, J. and R. Monajemy. "Generating Directional Residuals with Dynamic Parity Relations." *Automatica* **31**: 627-635 (1995).

35. Guglielmi, G., T. Parisini, et al. "Keynote Paper: Fault Diagnosis and Neural Networks: a Power Plant Application." *Control Engineering Practice* **3**(5): 601-620 (1995).

36. Hagan, M. T., H. B. Demuth, et al. *Neural Network Design*. Boston, MA, PWS Publishing Company (1995).

37. Healy, J. D. "A Note on Multivariate CUSUM." *Technometrics* **29**: 409-412 (1987).

38. Henery, R. J. *Review of Previous Empirical Comparisons*. Machine Learning, Neural, and Statistical Classification. D. Michie, D. J. Spiegelhalter and C. C. Taylor. Chichester, England, Ellis Horwood LTD: 124-130 (1994).

39. Hoskins, J. C. and D. M. Himmelblau. "Fault Diagnosis in Complex Chemical Plants Using Artificial Neural Networks." *AIChE Journal* **16**(4): 241-251 (1991).

40. House, J. M., W. Y. Lee, et al. "Classification Techniques for Fault Detection and Diagnosis of an Air-Handling Unit." *ASHRAE Transactions* **105 part 1**: 1087-1100 (1999).

41. Hyvarinen, A. and E. Oja. "Independent Component Analysis: Algorithms and Applications." *Neural Networks* **13**: 411-430 (2000).

42. Isermann, R. "Supervision, Fault Detection, and Fault Diagnosis Methods. An Introduction." *Control Engineering Practice* **5**(5): 639-652 (1997).

43. Isermann, R. and P. Balle. "Trends in the Application of Model-Based Fault Detection and Diagnosis of Technical Processes." *Control Engineering Practice* **5**(5): 709-719 (1997).

44. Jackson, J. E. *A User's Guide to Principal Components*. New York, N.Y., John Wiley and Sons Inc. (1991).

45. Janusz, M. E. and V. Venkatasubramanian. "Automatic Generation of Qualitative Descriptions of Process Trends for Fault Detection and Diagnosis." *Engineering Applications of Artificial Intelligence* **4**(5): 329-339 (1991).

46. Johnson, D. E. *7.5 Variable Selection Procedures*. Applied Multivariate Methods for Data Analysis. Pacific Grove, CA, Brooks/Cole**:** 245-255 (1998a).

47. Johnson, D. E. *Applied Multivariate Methods for Data Analysis*. Pacific Grove, CA, Brooks/Cole (1998b).

48. Kano, M. and K. Nagao. "Comparison of Statistical Process Monitoring methods: Application to the Eastman Challenge Process." *Computers and Chemical Engineering* **24**: 175-181 (2000).

49. Kassidas, A., P. A. Taylor, et al. "Off-Line Diagnosis of Deterministic Faults in Continuous Dynamic Multivaraible Processes Using Speech Recognition Methods." *Journal of Process Control* **8**(5-6): 381-393 (1998).

50. Kavuri, S. N. and V. Venkatasubramanian. "Combining Pattern Classification and Assumption-Based Techniques for Process Fault Diagnosis." *Computers and Chemical Engineering* **16**(4): 299-312 (1992).

51. Kavuri, S. N. and V. Venkatasubramanian. "Representing Bounded Fault Classes Using Neural Networks with Ellipsoidal Activation Functions." *Computers and Chemical Engineering* **17**(2): 139-163 (1993).

52. Keats, J. B., E. Castillo, et al. "Economic Modeling for Statistical Process Control." *Journal of Quality Technology* **29**(2): 144-147 (1997).

53. Kelly, B. E. and F. P. Lees. "The Propagation of Faults in Process Plants: Modeling Fault Propagation." *Reliability Engineering and System Safety* **16**(3) (1986).

54. Keyvan, S., A. Durg, et al. *Evaluation of the Performance of Various Artificial Neural Networks to the Signal Fault Diagnosis in Nuclear Reactor Systems*. 1993 IEEE International Conference on Neural Networks, IEEE, Piscataway, NJ, USA (1993).

55. Koh, C. K. H., J. Shi, et al. "Multiple Fault Detection and Isolation Using the Haar Transform." *Journal of Manufacturing Science and Engineering, Transactions of the ASME* **121**(2): 290-299 (1999).

56. Korbicz, J. and J. Kus. *A Fault Detection and Isolation System Using GMDH Neural Networks*. UKACC International Conference on Control' 98, IEE (1998).

57. Kourti, T., P. Nomikos, et al. "Analysis, Monitoring, and Fault Diagnosis of Batch Processes Using Multiblock and Multiway PLS." *Journal of Process Control* **5**(4): 277-284 (1995).

58. Kramer, M. A. "Nonlinear Principal Component Analysis Using Autoassociative Neural Networks." *AIChE Journal* **37**(2): 233-243 (1991).

59. Kramer, M. A. and R. S. Mah. *Model-Based Monitoring*. Proceedings of the 2nd International Conference on Foundattions of Computer-Aided Process Operations. D. Rippin, J. Hale and J. Davis**:** 45-68 (1993).

60. Kramer, M. A. and B. L. Palowitch. "A Rule-Based Approach to Fault Diagnosis Using the Signed Directed Graph." *AIChE Journal* **33**(7): 1067-1078 (1987).

61. Leonardt, S. and M. Ayoubi. "Methods of Fault Diagnosis." *Control Engineering Practice* **5**(5): 683-692 (1997).

62. Lezanski, P. "An Intelligent System for Grinding Wheel Condition Monitoring." *Journal of Materials Processing Technology* **109**: 258-263 (2001).

63. Linhart, H. and W. Zuccini. *Model Selection*. New York, John Wiley and Sons (1986).

64. Lowry, C. A., W. H. Woodall, et al. "A Multivariate EWMA Control Chart." *Technometrics* **34**: 46-53 (1992).

65. Luo, R., M. Misra, et al. "Sensor Fault Detection via Multiscale Analysis and Dynamic PCA." *Industrial and Engineering Chemistry Research* **38**(4): 1489-1495 (1999).

66. Lyman, P. R. and C. Georgakis. "Plant-Wide Control of the Tennessee-Eastman Problem." *Computers & Chemical Engineering* **19**(3): 321-331 (1995).

67. MacGregor, J. F. and T. Kourti. "Statistical Process Control of Multivariate Processes." *Control Engineering Practice* **3**(3): 403-414 (1995).

68. Marcu, T. and L. Mirea. "Robust Detection and Isolation of Process Faults Using Neural Networks." _IEEE Control Systems Magazine_ **17**(5): 72-79 (1997).

69. Mastrangelo, C. M. and J. M. Porter. _Data Mining in a Chemical Process Application_ (1998).

70. Mylaraswamy, D. and V. Venkatasubramanian. "A Hybrid Framework for Large Scale Process Diagnosis." _Computers and Chemical Engineering_ **21S**: S935-S939 (1997).

71. Nimmo, I. "Adequately address abnormal situation operation." _Chemical Engineering Progress_ **91**(9): 36-45 (1995).

72. Nomikos, P. and J. F. MacGregor. "Monitoring Batch Processes Using the Multi-way Principal Component Analysis." _AIChE Journal_ **40**(8): 1361-1373 (1994).

73. Oh, Y. S., K. J. Mo, et al. "Fault Diagnosis Based on Weighted Symptom Tree and Pattern Matching." _Industrial & Engineering Chemistry Research_ **36**(7): 2672-2678 (1997).

74. Oyeleye, O. and E. A. Lehtihet. "A Classification Algorithm and Optimal Feature Selection Methodology for Automated Solder Joint Defect Inspection." _Journal of Manufacturing Systems_ **17**(4): 251-262 (1998).

75. Pearson, K. "Lines and Planes of Closest Fit to Systems of Points in Space." _Philosophy Magazine_ **2**(11): 559-572 (1901).

76. Phatak, A. and R. Sparks. "New Trends in Process Monitoring: SPC for Continuous Process Industries." _Computers in Control_ **48**(4): 38-40 (1995).

77. Pudil, P., J. Novovicova, et al. "Floating Search Methods in Feature Selection." _Pattern Recognition Letters_ **15**(11): 1119-1125 (1994).

78. Qiu, J., X. Wen, et al. _Development of Automatic Discriminating Approach for FMC Condition Monitoring and Fault Diagnosis_. Proceedings of SPIE (1995).

79. Raich, A. and A. Cinar. "Statistical Process Monitoring and Disturbance Diagnosis in Multivariable Continuous Processes." _AIChE Journal_ **42**(4): 995-1009 (1996).

80. Raich, A. and A. Cinar. "Diagnosis of Process Disturbances by Statistical Distance and Angle Measures." *Computers & Chemical Engineering* **21**(6): 661-673 (1997).

81. Rengaswamy, R. and V. Venkatasubramanian. "A Syntactic Pattern Recognition Approach for Process Monitoring and Fault Diagnosis." *Engineering Applications in Artificial Intelligence* **8**(1): 35-51 (1995).

82. Rich, S. H. and V. Venkatasubramanian. "Model-Based Reasoning in Diagnostic Expert Systems for Chemical Process Plants." *Computers and Chemical Engineering* **11**: 111-122 (1987).

83. Roberts, S. W. "Control Chart Tests Based on Geometric Moving Averages." *Technometrics* **1**: 239-250 (1959).

84. Roehl, N. M. P., C.E.; De Azevedo, H.R. Teles. *Fuzzy ART Neural Network Approach for Incipient Fault Detection and Isolation in Rotating Machines*. Proceedings of the 1995 IEEE International Conference on Neural Networks (1995).

85. Sadegh, P., H. Madsen, et al. *Optimal Input Design for Fault Detection and Diagnosis*. Proceedings of the 1995 American Control Conference (1995).

86. Shao, R., F. Jia, et al. "Wavelets and Nonlinear Principal Component Analysis for Fault Monitoring." *Control Engineering Practice* **7**: 865-879 (1999).

87. Shina, S. *Six Sigma for Electronics Design and Manufacturing*. New York, McGraw Hill (2002).

88. Specht, D. F. "Probabilistic Neural Networks." *Neural Networks* **3**: 109-118 (1990).

89. Srinivasan, R., V. D. Dimitriadis, et al. "Safety Verification Using a Hybrid Knowledge-Based Mathematical Programming Framework." *AIChE Journal* **44**(2): 361-371 (1998).

90. Steinbach, J. *Safety Assessment for Chemical Processes*. Weinheim, Germany, Wiley-VCH (1999).

91. Tham, M. T. and A. Parr. "Succeed at Online Validation and Reconstruction of Data." *Chemical Engineering Progress* **90**(5): 46-53 (1994).

92. Thomas, H. and M. Moosemiller. "Establishing a Data Farm to Harvest Quality Reliability Information." _Process Safety Progress_ **20**(3): 173-182 (2001).

93. Tsai, C. S. and C. T. Chang. "Dynamic Process Diagnosis via Integrated Neural Networks." _Computers and Chemical Engineering_ **19 Suppl.**: S747-S752 (1995).

94. Tzafestas, S. G. and P. J. Dalianis. "Artificial Neural Networks in the Fault Diagnosis of Technological Ssystems: a Case Study in Chemical Engineering Process." _Engineering Simulation_ **13**(6): 939-954 (1996).

95. Vedam, H. and V. Venkatasubramanian. _Wavelet-Theory-Based Adaptive Trend Analysis System for Process Monitoring and Diagnosis_. Proceedings of the 1997 American Control Conference, Albuquerque, IEEE (1997).

96. Vedam, H., V. Venkatasubramanian, et al. "A B-Spline-Based Method for Data Compression, Process Monitoring, and Diagnosis." _Computers and Chemical Engineering_ **22**: S827-S830 (1998).

97. Venkatasubramanian, V., R. Rengaswamy, et al. "A Review of Process Fault Detection and Diagnosis. Part II: Quantitative Models and Search Strategies." _Computers and Chemical Engineering_ **27**: 313-326 (2003a).

98. Venkatasubramanian, V., R. Rengaswamy, et al. "A Review of Process Fault Detection and Diagnosis. Part III: Process History-Based Methods." _Computers and Chemical Engineering_ **27**: 327-346 (2003b).

99. Venkatasubramanian, V., R. Rengaswamy, et al. "A Review of Process Fault Detection and Diagnosis. Part I: Quantitative Model-Based Methods." _Computers and Chemical Engineering_ **27**: 293-311 (2003c).

100. Wang, D. D. and J. Xu. _Fault Detection Based on Evolving LVQ Neural Networks_. Proceedings of the 1996 IEEE International Conference on Systems, Man and Cybernetics (1996).

101. Whiteley, J. R. and J. F. Davis. "A Similarity-Based Approach to Interpretation of Sensor Data Using Adaptive Resonance Theory." _Computers and Chemical Engineering_ **18**(7): 637-661 (1994).

102. Willsky, A. S. "A Survey of Design Methods for Failure Detection in Dynamic Systems." _Automatica_ **12**: 601-611 (1976).

103. Wilson, D. J. H. and G. W. Irwin. "PLS Modelling and Fault Detection on the Tennessee Eastman Benchmark." _International Journal of Systems Science_ **31**(11): 1449-1457 (2000).

104. Wise, B. M. and N. B. Gallagher. "Process Chemometrics Approach to Process Monitoring and Fault Detection." _Journal of Process Conrol_ **6**(6): 329-348 (1996).

105. Woodward, R. H. and P. L. Goldsmith. _Cumulative Sum Techniques_. London, UK, Oliver and Boyd (1964).

106. Xie, M., T. N. Goh, et al. _Statistical Models and Control Charts for High Quality Processes_. Norwell, MA, Kluwer Academic Publishers (2002).

107. Yamashita, Y., H. Komori, et al. "Running Multiple Neural Networks for Process Trend Interpretation." _Journal of Chemical Engineering of Japan_ **32**(4): 552-556 (1999).

108. Zhao, J., B. Chen, et al. "Multidimensional Non-Orthogonal Wavelet-Sigmoid Basis Function Neural Network for Dynamic Process Fault Diagnosis." _Computers and Chemical Engineering_ **23**: 83-92 (1998).

Appendix. Modified Code of the Tennessee-Eastman Control Algorithm

```
C    Appendix. Modified Code of the Tennessee-Eastman Control Algorithm
C    Last modification March 31, 2004
C
C
C               Tennessee Eastman Process Control Test Problem
C
C                        Original codes written by
C
C                       James J. Downs and Ernest F. Vogel
C
C                    Process and Control Systems Engineering
C                           Tennessee Eastman Company
C                                  P.O. Box 511
C                             Kingsport, Tennessee 37662
C
C-----------------------------------------------------------------------
C
C  New version is a closed-loop plant-wide control scheme for
C  the Tennessee Eastman Process Control Test Problem
C
C  The modifications are by:
C
C             Evan L. Russell, Leo H. Chiang and Richard D. Braatz
C
C                     Large Scale Systems Research Laboratory
C                        Department of Chemical Engineering
C                    University of Illinois at Urbana-Champaign
C                        600 South Mathews Avenue, Box C-3
C                            Urbana, Illinois 61801
C                          http://brahms.scs.uiuc.edu
C
C The modified text is Copyright 1998-2002 by The Board of Trustees
C of the University of Illinois.  All rights reserved.
C
C Permission hereby granted, free of charge, to any person obtaining a copy
C of this software and associated documentation files (the "Software"), to
C deal with the Software without restriction, including without limitation
C the rights to use, copy, modify, merge, publish, distribute, sublicense,
C and/or sell copies of the Software, and to permit persons to whom the
C Software is furnished to do so, subject to the following conditions:
C              1. Redistributions of source code must retain the above copyright
C                 notice, this list of conditions and the following disclaimers.
C              2. Redistributions in binary form must reproduce the above
C                 copyright notice, this list of conditions and the following
C                 disclaimers in the documentation and/or other materials
C                 provided with the distribution.
C              3. Neither the names of Large Scale Research Systems Laboratory,
C                 University of Illinois, nor the names of its contributors may
C                 be used to endorse or promote products derived from this
C                 Software without specific prior written permission.
C
C THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
C OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
C FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
C THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
C OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
C ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
C DEALINGS IN THE SOFTWARE.
C-----------------------------------------------------------------------
C
C  Users should cite the original code using the following references:
C
C    J.J. Downs and E.F. Vogel, "A plant-wide industrial process control
C    problem." Presented at the AIChE 1990 Annual Meeting, Session on
C    Industrial Challenge Problems in Process Control, Paper #24a
C    Chicago, Illinois, November 14, 1990.
C
C    J.J. Downs and E.F. Vogel, "A plant-wide industrial process control
C    problem," Computers and Chemical Engineering, 17:245-255 (1993).
C
C  Users should cite the modified code using the following references:
C
C    E.L. Russell, L.H. Chiang, and R.D. Braatz. Data-driven Techniques
C    for Fault Detection and Diagnosis in Chemical Processes, Springer-Verlag,
C    London, 2000.
C
C    L.H. Chiang, E.L. Russell, and R.D. Braatz. Fault Detection and
```

```
C     Diagnosis in Industrial Systems, Springer-Verlag, London, 2001.
C
C     L.H. Chiang, E.L. Russell, and R.D. Braatz. "Fault diagnosis in
C     chemical processes using Fisher discriminant analysis, discriminant
C     partial least squares, and principal component analysis," Chemometrics
C     and Intelligent Laboratory Systems, 50:243-252, 2000.
C
C     E.L. Russell, L.H. Chiang, and R.D. Braatz. "Fault detection in
C     industrial processes using canonical variate analysis and dynamic
C     principal component analysis," Chemometrics and Intelligent Laboratory
C     Systems, 51:81-93, 2000.
C
C
C   Main program for demonstrating application of the modified Tennessee Eastman
C   Process Control Test Problem
C
C
C  this code requires a configuration file te_config.dat to run
C
C   the configuration file contains:
C   the number of seconds to run the simulation
C       (last byte's offset=7)
C   fault inception time since the start of the simulation, seconds
C       (last byte's offset=14)
C   disturbance number
C       (last byte's offset=21)
C   sampling period
C       (last byte's offset=28)
C   time of manual fault detection
C       (last byte's offset=35)
C   file to output the simulated values of process variables
C       (last byte's offset=47)
C   example:
C
C  (START OF FILE)450000   6000     13     150 300000 faul130.dat(END OF FILE)
C  remove the words in the parentheses
C
C  The simulation also logs the cost of "product loss and incomplete capacity
C  utilization" in file cost.dat that must be erased each time before the
C  start of the simulation
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C    DISTURBANCE VECTOR COMMON BLOCK
C
      INTEGER IDV
      COMMON/DVEC/ IDV(20)
C
C    CONTROLLER COMMON BLOCK
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      INTEGER FLAG
      COMMON/FLAG6/ FLAG
C
      DOUBLE PRECISION GAIN1, ERROLD1
      COMMON/CTRL1/ GAIN1, ERROLD1
      DOUBLE PRECISION GAIN2, ERROLD2
      COMMON/CTRL2/ GAIN2, ERROLD2
      DOUBLE PRECISION GAIN3, ERROLD3
      COMMON/CTRL3/ GAIN3, ERROLD3
      DOUBLE PRECISION  GAIN4, ERROLD4
      COMMON/CTRL4/ GAIN4, ERROLD4
      DOUBLE PRECISION GAIN5, TAUI5, ERROLD5
      COMMON/CTRL5/ GAIN5, TAUI5, ERROLD5
      DOUBLE PRECISION GAIN6, ERROLD6
      COMMON/CTRL6/ GAIN6, ERROLD6
      DOUBLE PRECISION GAIN7, ERROLD7
      COMMON/CTRL7/  GAIN7, ERROLD7
      DOUBLE PRECISION GAIN8, ERROLD8
      COMMON/CTRL8/ GAIN8, ERROLD8
      DOUBLE PRECISION GAIN9, ERROLD9
      COMMON/CTRL9/ GAIN9, ERROLD9
      DOUBLE PRECISION GAIN10, TAUI10, ERROLD10
      COMMON/CTRL10/ GAIN10, TAUI10, ERROLD10
      DOUBLE PRECISION GAIN11, TAUI11, ERROLD11
      COMMON/CTRL11/ GAIN11, TAUI11, ERROLD11
      DOUBLE PRECISION GAIN13, TAUI13, ERROLD13
      COMMON/CTRL13/ GAIN13, TAUI13, ERROLD13
      DOUBLE PRECISION GAIN14, TAUI14, ERROLD14
      COMMON/CTRL14/ GAIN14, TAUI14, ERROLD14
      DOUBLE PRECISION GAIN15, TAUI15, ERROLD15
      COMMON/CTRL15/ GAIN15, TAUI15, ERROLD15
      DOUBLE PRECISION GAIN16, TAUI16, ERROLD16
      COMMON/CTRL16/ GAIN16, TAUI16, ERROLD16
      DOUBLE PRECISION GAIN17, TAUI17, ERROLD17
      COMMON/CTRL17/ GAIN17, TAUI17, ERROLD17
      DOUBLE PRECISION GAIN18, TAUI18, ERROLD18
      COMMON/CTRL18/ GAIN18, TAUI18, ERROLD18
      DOUBLE PRECISION GAIN19, TAUI19, ERROLD19
      COMMON/CTRL19/ GAIN19, TAUI19, ERROLD19
      DOUBLE PRECISION GAIN20, TAUI20, ERROLD20
      COMMON/CTRL20/ GAIN20, TAUI20, ERROLD20
      DOUBLE PRECISION GAIN22, TAUI22, ERROLD22
      COMMON/CTRL22/ GAIN22, TAUI22, ERROLD22
C
C  Local Variables
C
      INTEGER I,NN,NPTS,TEST,TEST1,TEST3,TEST4,DN,SMPI,SSPTS,DET
C
      DOUBLE PRECISION TIME, YY(50), YP(50), LOSS, DX1, DX2
C
      CHARACTER*11 IFL
C
      NN = 50
      LOSS=0.0
C
C   Read simulation parameters from a file:
C
      OPEN(UNIT=140,FILE='te_config.dat',STATUS='old')
      READ(140,55) NPTS,SSPTS,DN,SMPI,DET,IFL
      CLOSE(UNIT=140)
      PRINT *, NPTS,SSPTS,DN,SMPI,DET,IFL

 55   FORMAT(1X,5(I6,1X),A11)
      DELTAT = 1. / 3600.
C
C  Initialize Process
C  (Sets TIME to zero)
C
      CALL TEINIT(NN,TIME,YY,YP)
C
C  Set Controller Parameters
C  Make a Stripper Level Set Point Change of +15%
C
      SETPT(1)=3664.0
      GAIN1=1.0
      ERROLD1=0.0
      SETPT(2)=4509.3
      GAIN2=1.0
      ERROLD2=0.0
      SETPT(3)=.25052
      GAIN3=1.
      ERROLD3=0.0
      SETPT(4)=9.3477
      GAIN4=1.
      ERROLD4=0.0
      SETPT(5)=26.902
      GAIN5=-0.083
      TAUI5=1./3600.
      ERROLD5=0.0
      SETPT(6)=0.33712
      GAIN6=1.22
      ERROLD6=0.0
      SETPT(7)=50.0
      GAIN7=-2.06
      ERROLD7=0.0
      SETPT(8)=50.0
      GAIN8=-1.62
      ERROLD8=0.0
      SETPT(9)=230.31
      GAIN9=0.41
      ERROLD9=0.0
      SETPT(10)=94.599
      GAIN10= -0.156* 10.
      TAUI10=1452./3600.
      ERROLD10=0.0
      SETPT(11)=22.949
      GAIN11=1.09
      TAUI11=2600./3600.
      ERROLD11=0.0
      SETPT(13)=32.188
      GAIN13=18.
      TAUI13=3168./3600.
      ERROLD13=0.0
      SETPT(14)=6.8820
      GAIN14=8.3
      TAUI14=3168.0/3600.
      ERROLD14=0.0
      SETPT(15)=18.776
      GAIN15=2.37
      TAUI15=5069./3600.
      ERROLD15=0.0
      SETPT(16)=65.731
      GAIN16=1.69/ 10.
      TAUI16=236./3600.
      ERROLD16=0.0
      SETPT(17)=75.000
      GAIN17=11.1/ 10.
      TAUI17=3168./3600.
      ERROLD17=0.0
      SETPT(18)=120.40
      GAIN18=2.83* 10.
      TAUI18=982./3600.
      ERROLD18=0.0
      SETPT(19)=13.823
      GAIN19=-83.2 / 5. /3.
      TAUI19=6336./3600.
      ERROLD19=0.0
      SETPT(20)=0.83570
      GAIN20=-16.3/ 5.
      TAUI20=12408./3600.
      ERROLD20=0.0
      SETPT(12)=2633.7
      GAIN22=-1.0 * 5.
      TAUI22=1000./3600.
      ERROLD22=0.0
C
C    Example Disturbance:
C    Change Reactor Cooling
C
      XMV(1) = 63.053 + 0.
      XMV(2) = 53.980 + 0.
      XMV(3) = 24.644 + 0.
      XMV(4) = 61.302 + 0.
      XMV(5) = 22.210 + 0.
      XMV(6) = 40.064 + 0.
      XMV(7) = 38.100 + 0.
      XMV(8) = 46.534 + 0.
      XMV(9) = 47.446 + 0.
      XMV(10)= 41.106 + 0.
      XMV(11)= 18.114 + 0.
C
C       SETPT(6)=SETPT(6) + 0.2
C
C  Set all Disturbance Flags to OFF
C


      DO 100 I = 1, 20
      IDV(I) = 0
 100  CONTINUE
C
      OPEN(UNIT=2111,FILE=IFL,STATUS='new')
      OPEN(UNIT=2112,FILE='cost.dat',STATUS='new')
C
C
C  Simulation Loop
C
      COST=0.0
      COST1=0.0
      COST2=0.0
      COST3=0.0
      LL=0
      DO 1000 I = 1, NPTS
        IF (I.GE.SSPTS) THEN
          IDV(DN)=1
        ENDIF
        IF (I.GE.(NPTS-80000)) THEN
          IDV(DN)=0
```

```
         ENDIF
         TEST=MOD(I,3)
         IF (TEST.EQ.2) THEN
           CALL CONTRL1
           CALL CONTRL2
           CALL CONTRL3
           CALL CONTRL4
           CALL CONTRL5
           CALL CONTRL6
           CALL CONTRL7
           CALL CONTRL8
           CALL CONTRL9
           CALL CONTRL10
           CALL CONTRL11
           CALL CONTRL16
           CALL CONTRL17
           CALL CONTRL18
        ENDIF
           TEST1=MOD(I,360)
        IF (TEST1.EQ.2) THEN
           CALL CONTRL13
           CALL CONTRL14
           CALL CONTRL15
           CALL CONTRL19
        ENDIF
        TEST1=MOD(I,900)
C
C Fault recovery actions
C
        IF (TEST1.EQ.2) THEN
           CALL CONTRL20
           IF ((I.GE.DET).AND.(I.LE.(NPTS-80000))) THEN
                 IF (SETPT(11).GE.20.65) THEN
                   SETPT(11)=SETPT(11)*.99
                 ENDIF
           ELSE
                 IF (SETPT(11).LE.22.94) THEN
                   SETPT(11)=SETPT(11)/.99
                 ELSE
                   SETPT(11)=22.949
                 ENDIF
           ENDIF
C
C Calculation of the cost of "reagent loss and incomplete cap. utilization"
C
       IF ((I.GE.SSPTS).AND.(XMEAS(7).LE.0.3E+04)) THEN
       CIN=98.62*XMEAS(1)+0.6894*XMEAS(2)+0.3164*XMEAS(3)+188.5*XMEAS(4)
       FR=XMEAS(38)/7.935+XMEAS(39)/6.833+XMEAS(40)/9.871+XMEAS(41)/8.118
       FR=FR+XMEAS(37)/9.3438
       COST1=COST1+XMEAS(40)*XMEAS(17)*30.44/FR
       COST2=COST2+XMEAS(41)*XMEAS(17)*22.94/FR
       COST3=COST3+CIN
       COST=(COST3-5745.0)/4.+(5534.5-COST1-COST2)/3.
       PRINT *, XMEAS(40)*XMEAS(17)*30.44/FR,XMEAS(41)*XMEAS(17)*22.94/FR
       WRITE(2112,*) COST
       ENDIF
       ENDIF

         TEST3=MOD(I,5000)
        IF (TEST3.EQ.0) THEN
          PRINT *, 'Simulation time (in seconds) = ', I
        ENDIF

         TEST4=MOD(I,SMPI)
         IF (TEST4.EQ.0) THEN
         CALL OUTPUT
         ENDIF
C
         CALL INTGTR(NN,TIME,DELTAT,YY,YP)
C
         CALL CONSHAND
C
 1000 CONTINUE
C
          CLOSE(UNIT=2112)
        STOP


       END
C
C==================================================================
C
      SUBROUTINE CONTRL1
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN1, ERROLD1
      COMMON/CTRL1/ GAIN1, ERROLD1
C
      DOUBLE PRECISION ERR1, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR1 = (SETPT(1) - XMEAS(2)) * 100. / 5811.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN1 * ( ( ERR1 - ERROLD1 ) )
C
      XMV(1) = XMV(1) + DXMV
C
      ERROLD1 = ERR1
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE CONTRL2
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN2, ERROLD2
      COMMON/CTRL2/ GAIN2, ERROLD2
C
      DOUBLE PRECISION ERR2, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR2 = (SETPT(2) - XMEAS(3)) * 100. / 8354.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN2 * ( ( ERR2 - ERROLD2 ) )
C
      XMV(2) = XMV(2) + DXMV
C
      ERROLD2 = ERR2
C
      RETURN
      END
C
```

```
C=============================================================================
C
      SUBROUTINE CONTRL3
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN3, ERROLD3
      COMMON/CTRL3/ GAIN3, ERROLD3
C
      DOUBLE PRECISION ERR3, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR3 = (SETPT(3) - XMEAS(1)) * 100. / 1.017
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN3 * ( ( ERR3 - ERROLD3 ) )
C
      XMV(3) = XMV(3) + DXMV
C
      ERROLD3 = ERR3
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE CONTRL4
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN4, ERROLD4
      COMMON/CTRL4/ GAIN4, ERROLD4
C
      DOUBLE PRECISION ERR4, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR4 = (SETPT(4) - XMEAS(4)) * 100. / 15.25
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN4 * ( ( ERR4 - ERROLD4 ) )
C
      XMV(4) = XMV(4) + DXMV
C
      ERROLD4 = ERR4
C
      RETURN
      END
```

```
C
C=============================================================================
C
      SUBROUTINE CONTRL5
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN5, TAUI5, ERROLD5
      COMMON/CTRL5/ GAIN5, TAUI5, ERROLD5
C
      DOUBLE PRECISION ERR5, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR5 = (SETPT(5) - XMEAS(5))  * 100. / 53.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
C       PRINT *, 'GAIN5= ', GAIN5
C       PRINT *, 'TAUI5= ', TAUI5
C       PRINT *, 'ERR5= ', ERR5
C       PRINT *, 'ERROLD5= ', ERROLD5
C
      DXMV = GAIN5 * ((ERR5 - ERROLD5)+ERR5*DELTAT*3./TAUI5)
C
      XMV(5) = XMV(5) + DXMV
C
      ERROLD5 = ERR5
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE CONTRL6
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
      INTEGER FLAG
       COMMON/FLAG6/ FLAG
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN6, ERROLD6
      COMMON/CTRL6/ GAIN6, ERROLD6
C
      DOUBLE PRECISION ERR6, DXMV
C
C  Example PI Controller:
C     Stripper Level Controller
      IF (XMEAS(13).GE.2950.0) THEN
        XMV(6)=100.0
        FLAG=1
        ELSEIF (FLAG.EQ.1.AND.XMEAS(13).GE.2633.7) THEN
        XMV(6)=100.0
        ELSEIF (FLAG.EQ.1.AND.XMEAS(13).LE.2633.7) THEN
        XMV(6)=40.060
        SETPT(6)=0.33712
        ERROLD6=0.0
      FLAG=0
      ELSEIF (XMEAS(13).LE.2300.) THEN
      XMV(6)=0.0
      FLAG=2
      ELSEIF (FLAG.EQ.2.AND.XMEAS(13).LE.2633.7) THEN
      XMV(6)=0.0
      ELSEIF (FLAG.EQ.2.AND.XMEAS(13).GE.2633.7) THEN
      XMV(6)=40.060
      SETPT(6)=0.33712
      ERROLD6=0.0
      FLAG=0
      ELSE
      FLAG=0
C
C    Calculate Error
C
      ERR6 = (SETPT(6) - XMEAS(10)) * 100. /1.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN6 * ( ( ERR6 - ERROLD6 ) )
C
      XMV(6) = XMV(6) + DXMV
C
      ERROLD6 = ERR6
      ENDIF
C
      RETURN
      END
C
C=======================================================================
C
      SUBROUTINE CONTRL7
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN7, ERROLD7
      COMMON/CTRL7/ GAIN7, ERROLD7
C
      DOUBLE PRECISION ERR7, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR7 = (SETPT(7) - XMEAS(12)) * 100. / 70.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN7 * ( ( ERR7 - ERROLD7 ) )
C
      XMV(7) = XMV(7) + DXMV
C
      ERROLD7 = ERR7
C
      RETURN
      END
C
C=======================================================================
C
      SUBROUTINE CONTRL8
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
```

```
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN8, ERROLD8
      COMMON/CTRL8/ GAIN8, ERROLD8
C
      DOUBLE PRECISION ERR8, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR8 = (SETPT(8) - XMEAS(15)) * 100. / 70.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV =  GAIN8 * ( ( ERR8 - ERROLD8 ) )
C
      XMV(8) = XMV(8) + DXMV
C
      ERROLD8 = ERR8
C
      RETURN
      END
C
C=======================================================================
C
      SUBROUTINE CONTRL9
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN9, ERROLD9
      COMMON/CTRL9/ GAIN9, ERROLD9
C
      DOUBLE PRECISION ERR9, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR9 = (SETPT(9) - XMEAS(19)) * 100. / 460.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN9 * ( ( ERR9 - ERROLD9 ) )
C
      XMV(9) = XMV(9) + DXMV
C
      ERROLD9 = ERR9
C
      RETURN
      END
C
C=======================================================================
C
      SUBROUTINE CONTRL10
C
C  Discrete control algorithms
C
C
```

```
C    MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C    CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN10, TAUI10, ERROLD10
      COMMON/CTRL10/ GAIN10, TAUI10, ERROLD10
C
      DOUBLE PRECISION ERR10, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR10 = (SETPT(10) - XMEAS(21)) * 100. / 150.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN10*((ERR10 - ERROLD10)+ERR10*DELTAT*3./TAUI10)
C
      XMV(10) = XMV(10) + DXMV
C
      ERROLD10 = ERR10
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE CONTRL11
C
C  Discrete control algorithms
C
C
C    MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C    CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN11, TAUI11, ERROLD11
      COMMON/CTRL11/ GAIN11, TAUI11, ERROLD11
C
      DOUBLE PRECISION ERR11, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR11 = (SETPT(11) - XMEAS(17)) * 100. / 46.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN11*((ERR11 - ERROLD11)+ERR11*DELTAT*3./TAUI11)
C
      XMV(11) = XMV(11) + DXMV
C
      ERROLD11 = ERR11
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE CONTRL13
C
C  Discrete control algorithms
C
```

```
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN13, TAUI13, ERROLD13
      COMMON/CTRL13/ GAIN13, TAUI13, ERROLD13
C
      DOUBLE PRECISION ERR13, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR13 = (SETPT(13) - XMEAS(23)) * 100. / 100.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN13 * ((ERR13 - ERROLD13)+ERR13*DELTAT*360./TAUI13)
C
      SETPT(3) = SETPT(3) + DXMV * 1.017 / 100.
C
      ERROLD13 = ERR13
C
      RETURN
      END
C
C=======================================================================
C
      SUBROUTINE CONTRL14
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN14, TAUI14, ERROLD14
      COMMON/CTRL14/ GAIN14, TAUI14, ERROLD14
C
      DOUBLE PRECISION ERR14, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR14 = (SETPT(14) - XMEAS(26)) * 100. /100.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN14*((ERR14 - ERROLD14)+ERR14*DELTAT*360./TAUI14)
C
      SETPT(1) = SETPT(1) + DXMV * 5811. / 100.
C
      ERROLD14 = ERR14
C
      RETURN
      END
C
C=======================================================================
C
      SUBROUTINE CONTRL15
C
C  Discrete control algorithms
```

```
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN15, TAUI15, ERROLD15
      COMMON/CTRL15/ GAIN15, TAUI15, ERROLD15
C
      DOUBLE PRECISION ERR15, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR15 = (SETPT(15) - XMEAS(27)) * 100. / 100.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN15 * ((ERR15 - ERROLD15)+ERR15*DELTAT*360./TAUI15)
C
      SETPT(2) = SETPT(2) + DXMV * 8354. / 100.
C
      ERROLD15 = ERR15
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE CONTRL16
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN16, TAUI16, ERROLD16
      COMMON/CTRL16/ GAIN16, TAUI16, ERROLD16
C
      DOUBLE PRECISION ERR16, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR16 = (SETPT(16) - XMEAS(18)) * 100. / 130.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN16 * ((ERR16 - ERROLD16)+ERR16*DELTAT*3./TAUI16)
C
      SETPT(9) = SETPT(9) + DXMV * 460. / 100.
C
      ERROLD16 = ERR16
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE CONTRL17
C
```

```
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN17, TAUI17, ERROLD17
      COMMON/CTRL17/ GAIN17, TAUI17, ERROLD17
C
      DOUBLE PRECISION ERR17, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR17 = (SETPT(17) - XMEAS(8)) * 100. / 50.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV =GAIN17*((ERR17 - ERROLD17)+ERR17*DELTAT*3./TAUI17)
C
      SETPT(4) = SETPT(4) + DXMV * 15.25 / 100.
C
      ERROLD17 = ERR17
C
      RETURN
      END
C
C=======================================================================
C
      SUBROUTINE CONTRL18
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN18, TAUI18, ERROLD18
      COMMON/CTRL18/ GAIN18, TAUI18, ERROLD18
C
      DOUBLE PRECISION ERR18, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR18 = (SETPT(18) - XMEAS(9)) * 100. / 150.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN18 * ((ERR18 - ERROLD18)+ERR18*DELTAT*3./TAUI18)
C
      SETPT(10) = SETPT(10) + DXMV * 150. / 100.
C
      ERROLD18 = ERR18
C
      RETURN
      END
C
C=======================================================================
C
      SUBROUTINE CONTRL19
```

```
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN19, TAUI19, ERROLD19
      COMMON/CTRL19/ GAIN19, TAUI19, ERROLD19
C
      DOUBLE PRECISION ERR19, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR19 = (SETPT(19) - XMEAS(30)) * 100. / 26.
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN19*((ERR19 - ERROLD19)+ERR19*DELTAT*360./TAUI19)
C
      SETPT(6) = SETPT(6) + DXMV * 1. / 100.
C
      ERROLD19 = ERR19
C
      RETURN
      END
C
C=======================================================================
C
      SUBROUTINE CONTRL20
C
C  Discrete control algorithms
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN20, TAUI20, ERROLD20
      COMMON/CTRL20/  GAIN20, TAUI20, ERROLD20
C
      DOUBLE PRECISION ERR20, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR20 = (SETPT(20) - XMEAS(38)) * 100. / 1.6
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN20*((ERR20 - ERROLD20)+ERR20*DELTAT*900./TAUI20)
C
      SETPT(16) = SETPT(16) + DXMV  * 130. / 100.
C
      ERROLD20 = ERR20
C
      RETURN
      END
C
C=======================================================================
C
      SUBROUTINE CONTRL22
C
C  Discrete control algorithms
```

```
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
C   CONTROLLER COMMON BLOCK
C
      DOUBLE PRECISION SETPT, DELTAT
      COMMON/CTRLALL/ SETPT(20), DELTAT
      DOUBLE PRECISION GAIN22, TAUI22, ERROLD22
      COMMON/CTRL22/  GAIN22, TAUI22, ERROLD22
C
      DOUBLE PRECISION ERR22, DXMV
C
C  Example PI Controller:
C    Stripper Level Controller
C
C    Calculate Error
C
      ERR22 = SETPT(12) - XMEAS(13)
C
C    Proportional-Integral Controller (Velocity Form)
C         GAIN = Controller Gain
C         TAUI = Reset Time (min)
C
      DXMV = GAIN22*((ERR22 - ERROLD22)+ERR22*DELTAT*3./TAUI22)
C
      XMV(6) = XMV(6) + DXMV
C
      ERROLD22 = ERR22
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE OUTPUT
C
C
C   MEASUREMENT AND VALVE COMMON BLOCK
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
      WRITE(2111,100) XMEAS
 100  FORMAT(41(E13.5,1X))
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE INTGTR(NN,TIME,DELTAT,YY,YP)
C
C  Euler Integration Algorithm
C
C
      INTEGER I, NN
C
      DOUBLE PRECISION TIME, DELTAT, YY(NN), YP(NN)
C
      CALL TEFUNC(NN,TIME,YY,YP)
C
      TIME = TIME + DELTAT
C
      DO 100 I = 1, NN
C
          YY(I) = YY(I) + YP(I) * DELTAT
C
 100  CONTINUE
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE CONSHAND
C
C  Euler Integration Algorithm
C
```

```
C
      DOUBLE PRECISION XMEAS, XMV
      COMMON/PV/ XMEAS(41), XMV(12)
C
      INTEGER I
C
      DO 100 I=1, 11
        IF (XMV(I).LE.0.0) XMV(I)=0.
        IF (XMV(I).GE.100.0) XMV(I)=100.
 100    CONTINUE
C
      RETURN
      END
```

VITA 2

Roman Shapovalov

Candidate for the Degree of

Doctor of Philosophy

Thesis: CONSTRUCTION AND EVALUATION OF MATHEMATICAL MODELS FOR REAL-TIME PROCESS FAULT MONITORING

Major Field: Chemical Engineering

Biographical:

Education:
Graduated from Secondary School # 218 of the City of Moscow in June, 1990; graduated with honor from Mendeleyev University of Chemical Technology of Russia with Diploma in Chemical Cybernetics in March, 1996; completed the requirements for the Master of Science in Chemical Engineering at Oklahoma State University in December, 1999; completed the requirements for the Doctor of Philosophy in Chemical Engineering at Oklahoma State University in May 2004.

Experience:
Worked as an interpreter and project manager with Joint-Stock Company "Special Bio-Physical Technologies" from 1995 to 1997 and as a Research and Teaching Assistant at the School of Chemical Engineering of Oklahoma State University from 1997 to 2003. Participated in developing a data handling software and control hardware for the Smart Bridge project.

Professional Memberships:
Member of American Institute of Chemical Engineers and American Statistical Association.