ADAPTIVE AND PREDICTIVE MODELING FOR REAL-TIME STATISTICAL PROCESS CONTROL

by

Herbert Wiley Huang

Memorandum No. UCB/ERL M96/71

18 November 1996

ADAPTIVE AND PREDICTIVE MODELING FOR REAL-TIME STATISTICAL PROCESS CONTROL

by

Herbert Wiley Huang

Memorandum No. UCB/ERL M96/71

18 November 1996

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Adaptive and Predictive Modeling for Real-Time Statistical Process Control

by
Herbert Wiley Huang

Electrical Engineering and Computer Sciences
University of California at Berkeley

Professor Costas J. Spanos, Advisor

In today's competitive semiconductor industry, manufacturing equipment must be vigilantly monitored so that equipment problems are detected as quickly as possible. Previous work demonstrated a statistical process control (SPC) scheme based on multivariate analysis techniques applied to real-time sensor signals that is effective for the detection of equipment malfunctions.

However, the SPC scheme has several key weaknesses that prevent its use from being more widespread. In particular, the scheme requires training before it can be used. The training can be costly and time-consuming, and must be redone whenever changes occur in the equipment or the process. Moreover, the manufacturer has no easy way of knowing when re-training is necessary.

In this thesis, the SPC scheme's weaknesses will be addressed by introducing some new techniques for modeling signals. A *predictive model* incorporates changes in the machine settings into the scheme; an *adaptive model* tracks the dynamics of the process' real-time behavior. Together, these methods make training unnecessary for the SPC scheme. The methods are illustrated with examples using actual sensor data, and a software implementation of the techniques is described.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Most of all, I would like to thank my research advisor, Professor Costas J. Spanos, for his support of my research and studies. Also deserving thanks is Professor Seth Sanders for serving on my project report committee.

A warm thanks goes out to the members of the Berkeley Computer-Aided Manufacturing (BCAM) group who helped make the experience worthwhile: Roawen Chen, Mark Hatzilambrou, Anna Ison, Nickhil Jakatdar, David Mudie, Xinhui Niu, Manolis Terrovitis, and Crid Yu.

This work was supported by the Semiconductor Research Corporation (SRC), the state of California Micro program, and participating companies (Advanced Micro Devices, Applied Materials, Atmel Corporation, Lam Research, National Semiconductor, Silicon Valley Group, Texas Instruments). Data and consultation with Texas Instruments researchers working on the TI/Sematech J88 project are also acknowledged.

Chapter 1 Introduction

1.1. Motivation

In today's competitive semiconductor industry, companies are constantly trying to increase their wafer yields and throughput. To achieve these goals, manufacturing equipment must be vigilantly monitored to ensure proper processing at each of the thousands of processing steps required to turn a wafer into marketable product. This thesis deals with a *statistical process control* (SPC) scheme that detects equipment malfunctions. The earlier that malfunctions are detected, the less time and money is wasted processing defective wafers. Furthermore, because of the high cost of modern semiconductor equipment, if a scheme can quickly detect malfunctions, this results in a considerable savings due to higher equipment utilization.

The SPC scheme to be discussed in Chapter 2 has been shown to be effective in the detection of equipment malfunctions (or equipment *faults*). One key to its effectiveness is its use of *real-time sensor signals*. These are signals collected automatically and non-invasively during the processing of a wafer, in a way that monitoring is accomplished with little extra cost to the process.

Although the SPC scheme has the potential to reduce the overall cost of ownership of semiconductor equipment by increasing both the wafer yield and throughput of product wafers, several key weaknesses in the scheme prevent its use from being more widespread. In particular, the scheme requires *baseline training* before it can be used. The training allows the scheme to "learn" the behavior of a machine's sensor signals when the machine is operating in a normal state, i.e., its baseline behavior. Later, when the machine is being

monitored, any behavior that deviates from the baseline behavior is flagged as a malfunction.

This training is not completely unwieldy, but can limit the SPC scheme's effectiveness in an actual factory for several reasons. First of all, a baseline training experiment can be costly and time-consuming. This would not be a problem if a manufacturer could perform a single training experiment for all time and never need to do it again. However, once changes occur in the equipment or the process, these changes will affect the real-time sensor signals, and the training must be redone. The changes that occur could be unintentional (due to natural aging of the equipment, for example), or they could be intentional (such as a change in the machine settings to process a different type of product). Moreover, the manufacturer has no easy way of knowing when a new training experiment should be performed. This fact, along with the expense of running a training experiment, has limited the scope of the SPC scheme to small, specialized projects that can be carefully controlled and monitored.

The purpose of this thesis is to address the weaknesses in the scheme, as described in the above paragraph. To this end, some new techniques for modeling signals will be introduced into the scheme. The thesis will also illustrate these methods with examples and applications using sensor data collected from plasma etching equipment.

1.2. Thesis Overview

A two stage model is proposed to model sensor measurements obtained from the processing of silicon wafers (in this thesis, the concentration is on a plasma etching process): a *wafer-wafer predictive stage* incorporates changes in the machine settings into the model; a *real-time adaptive stage* tracks the real-time behavior of the process (see Figure 1.1).

1.2.1. Wafer-Wafer Predictive Stage

A multivariate regression model predicts the wafer average of the output signals based on the machine's input settings (recipe). Previous work did not include the machine's recipe settings as part of the modeling. This meant that separate models had to be created

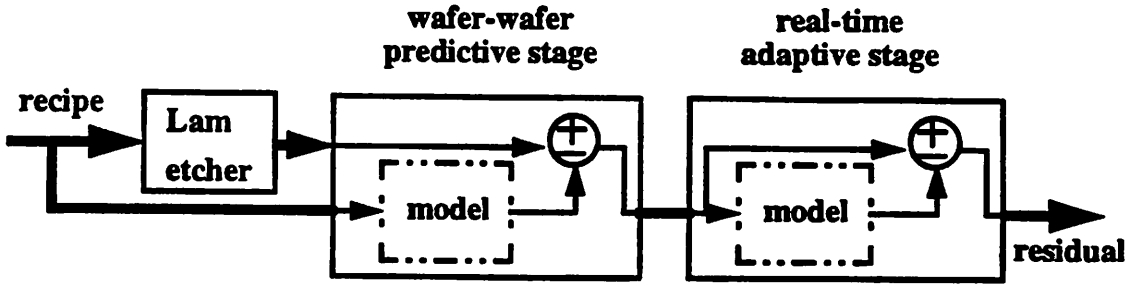

Figure 1.1.
Diagram of the two stage model for a plasma etch process.

for each new recipe. The predictive stage is useful when new products or wafer patterns require recipes to be changed often.

1.2.2. Real-Time Adaptive Stage

An adaptive univariate or multivariate time series algorithm tracks the dynamics of the real-time behavior. Previous work employed static time-series models and needed extensive baseline data. The adaptive stage can adjust to a drifting process, and the sensitivity can be tuned as desired. A baseline experiment is not necessary.

1.3. Thesis Organization

Chapter 2 gives a summary of the previous work upon which this thesis is based; for example, the real-time statistical process control (SPC) methodology that will be referred to throughout this paper is explained there. Chapter 3 describes *predictive* models whose purpose is to predict how recipe changes will affect real-time sensor measurements. Chapter 4, the main chapter of this thesis, shows how *adaptive* models work and how they can be used in the context of SPC; these models allow one to model signals without having to design a separate training experiment. Chapter 5 presents some examples that apply the above techniques to actual sensor data. Chapter 6 describes the software that is available to implement the SPC methodology. Chapter 7 concludes the thesis with a summary and suggestions for future exploration.[1]

1. This document was processed with the FrameMaker® document publishing software [33].

Chapter 2 Background

2.1. Introduction

Since much of this thesis builds upon a wealth of research by previous authors, a brief review of past work will result in a greater understanding of this work. Here is an overview of the background concepts to be presented:

The main application of this work is for real-time statistical process control (SPC) of semiconductor manufacturing equipment. Section 2.2 contains important previous work, including an overview of SPC. The usefulness of real-time equipment sensor data for monitoring the state of a machine is summarized in Section 2.2.1. The concepts of SPC and the methodology used to apply SPC to real-time tool data are summarized in Section 2.2.2. In Section 2.2.3, the use of hierarchical models will be shown to greatly improve the accuracy of the signal models. Section 2.2.4 discusses the automatic generation of time-series models. Finally, the general logistics of collecting real-time signal data are described in Section 2.3.

2.2. Previous Work

Statistical process control (SPC) is a quality control technique employed on a manufacturing line to detect equipment or process problems. Manufacturers want to detect any failures or malfunctions in the process as quickly as possible. Early detection of equipment malfunctions in a production line results in less waste being produced and greater up time of critical process equipment. This in turn equates to higher profitability of the overall process.

Traditional SPC techniques make use of various types of control charts to track important process parameters. Control charts graphically plot parameters as they are col-

lected and use statistical tests to determine when a parameter has significantly departed from what would be considered its usual behavior. The departure occurs—and an alarm is signaled—when a parameter's value falls outside some pre-determined control limits.

Standard practice in industry is to plot control charts based on data measured from occasional "monitor" wafers. The monitor wafers are run perhaps at the start of each working shift or when a process engineer wishes to check the state of the machine. Unfortunately, machine problems which occur between monitor wafers go undetected; moreover, a significant delay can exist between a machine fault and the actual signalling of an alarm.

2.2.1. Real-Time Tool Data

Many researchers have been investigating signals that are more accurate than the machine's input settings in describing wafer states of interest. With the aid of automated sensors and computers, the fault detection delay can be considerably reduced by monitoring real-time data collected non-invasively from equipment, while wafers are still being processed. The usefulness of real-time equipment sensor data for monitoring the state of a machine has been reported in [17][29][19]. The real-time data used in this thesis are a plasma etcher's internal sensor readings, for example electrical signals such as the radio frequency (RF) impedance and D.C. bias, and mechanical signals such as those tracking the coil and throttle positions. These are readily obtained over standard communication ports. Section 2.3 contains more information on how real-time data are collected.

These internal readings should (intuitively) reflect the true state of the equipment much better than the machine's input settings. Because they are closely coupled with the actual state of the chamber, the internal sensors may be able to account for drifts in the machine due to natural aging. Furthermore, equipment malfunctions should manifest themselves first in the values of these internal parameters and much later in off-line wafer measurements, resulting in more timely detection of faults.

2.2.2. Real-Time Statistical Process Control

The nature of the real-time sensor data prevents one from directly applying traditional SPC techniques. This is because traditional techniques assume that each new data sample is independent of all previous samples. This assumption is clearly violated in real-

time sensor data. Data collected sequentially from a machine at a high sampling rate (on the order of one sample per second or so) will likely be correlated from sample to sample (auto-correlation); some of the real-time signals may also be non-stationary. Furthermore, different sensor signals measured from the same equipment will very likely experience correlation between signals (cross-correlation).

The correlated behavior of the real-time data requires modifications to the traditional techniques. A novel scheme was introduced in [9][29] that allows one to use the real-time equipment sensor signals for statistical process control (SPC). The scheme models each real-time signal with a time-series model based on a "baseline" (calibration) set of data. These baseline models characterize what is considered to be the "in-control" behavior of the machine.

Once an equipment's baseline behavior has been established, production wafers can be run through the machine. The baseline time-series models are then used to filter the signals obtained during the subsequent equipment operation into residuals. The residuals (i.e., the differences between the actual and forecast values) from multiple sensor signals are summarized into a single score, known as the Hotelling's T^2 statistic, which can then be graphically displayed using a single-sided control chart. Whenever the signals deviate significantly from their baseline behavior, this will result in large residuals and cause an alarm to be triggered.

This methodology has been implemented in a commercially available software package, known as RTSPC. See Figure 2.1 for a diagram of the real-time SPC data flow. For a more complete description, see [18][29]. More detailed background on time-series models is given in Section 2.2.2.1 and on the T^2 statistic in Section 2.2.2.2.

2.2.2.1. Time-Series Models

Time-series models are used to characterize the real-time sensor data collected from an equipment that is operating under baseline conditions. A time-series model captures the auto-correlation structure among sequential samples. Once a variable is modeled with a univariate time-series model, future values of the variable can be predicted based on past

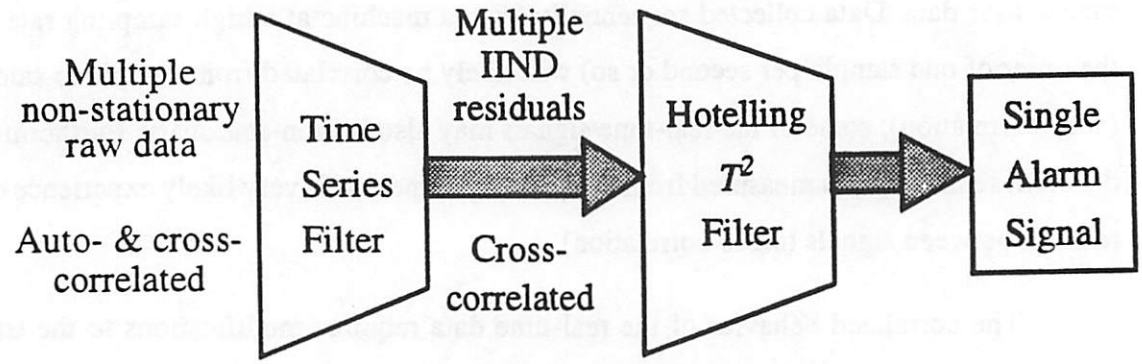

Figure 2.1.
Diagram of the real-time SPC scheme [29][18].

observations. A time-series is concisely described by an ARIMA(P, D, Q) model, where P is the auto-regressive order, D is the integration order, and Q is the moving average order. The general form of the model is:

$$\phi(B)w_t = \theta(B)a_t$$

$$\phi(B) = 1 - \sum_{k=1}^{P} \phi_k B^k \qquad \theta(B) = 1 - \sum_{k=1}^{Q} \theta_k B^k \tag{2.1}$$

$$w_t = \nabla^D z_t, D \geq 0$$

$$\nabla z_t = z_t - z_{t-1} \qquad B^k z_t = z_{t-k}$$

where z_t are the original data, w_t are the differenced data, and a_t are the prediction errors. For more information on time-series models, see the abundant literature: [5][25][34].

2.2.2.2. T^2 Statistic

The reason for using the T^2 statistic is to summarize multiple parameters into a single statistic that can be monitored on a single control chart. This has the advantage of needing to monitor only one control chart and also of reducing the false alarms associated with multiple control charts that are monitoring correlated signals. Before a set of signals can be combined together into a T^2 statistic, they must be filtered with the corresponding baseline models in order to remove any time-series patterns. The formulas for the T^2 statistic and for the *upper control limit (UCL)* to be used on a control chart are given below; they assume that the signals have already been filtered.

The goal of the T^2 control chart is to detect a shift in the operating point of the machine. To do this, a test statistic is calculated based on a set of sequential samples called a *group*; the number of samples in the group is known as the group size n. Each monitored signal is averaged over the n samples of the group and combined into a single vector. The averaging helps to assure that the signals are approximately Gaussian (by the *central limit theorem*). If $\bar{\mathbf{x}}$ is the (column) vector of averaged samples for each signal, then the test statistic is calculated as

$$T^2 = n(\bar{\mathbf{x}} - \bar{\bar{\mathbf{x}}})^T \hat{\mathbf{S}}^{-1} (\bar{\mathbf{x}} - \bar{\bar{\mathbf{x}}}) \tag{2.2}$$

where $\bar{\bar{\mathbf{x}}}$ is the vector of baseline signal averages, and $\hat{\mathbf{S}}$ is their estimated covariance matrix [23]. In (2.2), subtraction by $\bar{\bar{\mathbf{x}}}$ serves to *demean* the signals, and multiplication by $\hat{\mathbf{S}}^{-1}$ serves to scale the signals to unity variance. Multiplication by the scalar n compensates for the reduction in variance due to the group averaging. The vector $\bar{\bar{\mathbf{x}}}$ and matrix $\hat{\mathbf{S}}$ are estimates from a prior set of baseline data. Let the number of baseline observations used in the calculation of $\bar{\bar{\mathbf{x}}}$ and $\hat{\mathbf{S}}$ be called N. Note that the matrix $\hat{\mathbf{S}}$ should be symmetric and positive-definite, so that the T^2 statistic is always positive.

After the test statistic is calculated, it is compared to the UCL; if the statistic is greater than the UCL, an alarm is triggered. The theoretical UCL for the T^2 statistic is related to the F-distribution and depends on the desired Type I error α, the number of monitored variables M, and the number of baseline observations N:

$$UCL_{\alpha, M, N} = \left[\frac{M(N-1)}{N-M}\right] F_{\alpha, M, N-M}. \tag{2.3}$$

As N approaches infinity, then $\bar{\bar{\mathbf{x}}}$ and $\hat{\mathbf{S}}$ should approach their "true" values. In this case, the calculated T^2 statistic should have the same distribution as the sum of squares of M independent standard Gaussian random variables—in other words, a χ^2-distribution with M degrees of freedom. Therefore, if N is relatively large (greater than 20 or 25) then the F-distribution is well-approximated by the χ^2-distribution [23], so that the UCL can be more easily expressed as

$$UCL_{\alpha, M} \cong \chi^2_{\alpha, M}. \tag{2.4}$$

2.2.3. Hierarchical Models (Multiple Time Scale Decomposition)

Although seasonal ARIMA (SARIMA [5]) models were used in the original real-time SPC methodology[29], they are not ideal for modeling semiconductor equipment sensor signals. The reason is that the data is influenced by multiple physical processes that operate on different time scales and have drastically different variances. The different time scales can roughly be separated into three levels: 1) real-time, 2) wafer-wafer, and 3) lot-lot (a "lot" is usually a wafer cassette containing roughly 24 wafers). Signals at the real-time level capture short-term fluctuations during the processing of a single wafer, while the signals at the wafer-wafer level exhibit longer term behavior characteristic of wafers within a single lot. Similarly, at the lot-lot level, one may see even longer term patterns that reflect changes in the machine's state, for example a build-up of film in the chamber due to natural aging.

Hierarchical models are created by decomposing the sensor data into multiple components based on different time scales, where each of the components are analyzed separately. This decomposition results in more accurate signal modeling. The better signal models means that SPC will be more accurate, as shown by fewer false alarms. Moreover, the different levels of the hierarchical models supply valuable diagnostic information through the level of the hierarchy that caused a certain alarm. For instance, a slow drift in RF power at the lot-lot level might be interpreted as natural aging of the machine, but a drift at the real-time level might signify a problem with the current wafer. Without the decomposition, the time scale level that exhibited the greatest variance would dominate all other levels, and the useful information that could have been extracted from the other levels would be lost.

See Figure 2.2 for an example of a signal decomposition. The wafer-wafer component is first extracted simply by averaging the signal over each wafer. The real-time component is then produced by subtracting the wafer-wafer component from the original signal. A lot-lot component could similarly be produced by first averaging the signal over each lot, before extracting the wafer-wafer and real-time components. The greater variation of the wafer-wafer component compared to the real-time component is apparent in the Figure 2.2.

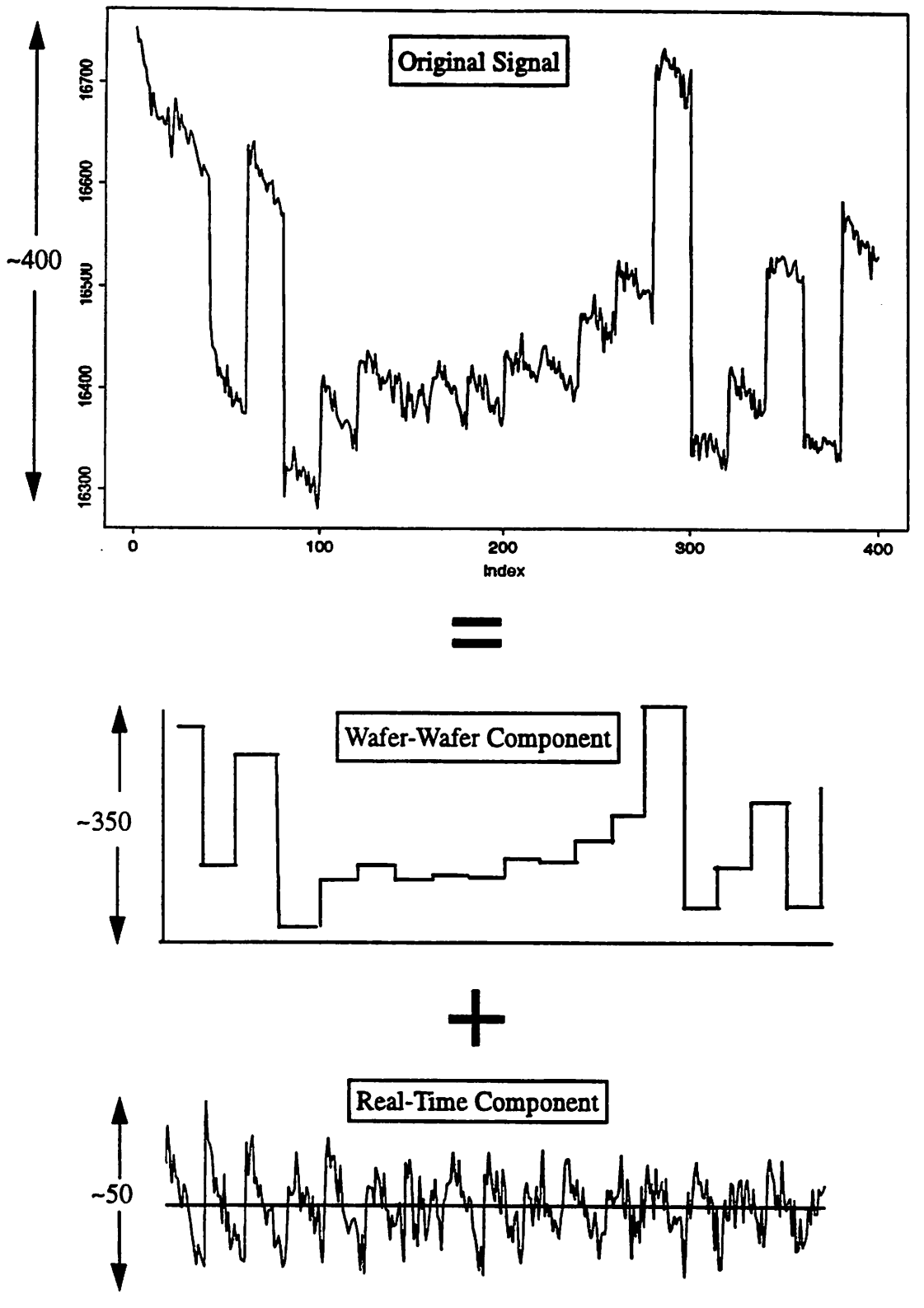

Figure 2.2.
Signal decomposition for the impedance signal.

2.2.4. Automatic Model Generation

The identification of time-series models and estimation of the model parameters have been implemented into an automatic model generation program by [20]. This was an important contribution because manually building time-series models can be a tedious, labor-intensive, and time-consuming endeavor. An automated time-series model generator makes SPC more practical in a manufacturing environment.

A rough description of the model generating procedure is as follows. Note that this procedure relies on known heuristics and experience with the selected signals [20][14]. First the differencing order is determined to ensure that the data series is stationary. Then the autoregressive order is identified and the coefficients are estimated based on solving the modified Yule-Walker equations. Finally, the moving average order and coefficients are calculated using a non-linear optimization program. Once the models are identified and the model coefficients are estimated for each signal, the covariance matrix of the signal residuals is calculated. Together, the models and the covariance matrix characterize the baseline behavior of the system.

2.3. Real-Time Data Collection

This section will describe the general procedure and logistics for collecting real-time data from a piece of equipment. The physical hardware used for the data collection will be described, as well as other practical issues, such as signal selection and pre-filtering of the data.

2.3.1. Hardware

The examples in this thesis use real-time data taken from two different types of state-of-the-art plasma etchers: a parallel-plate Lam 4400 polysilicon etcher and a transformer coupled plasma (TCP) Lam 9600 metal etcher. Data are available from three sources: software that obtains measurements via a standard SECS-II (SEMI Equipment Communication Standard II) port, the Comdel Real Power Monitor (RPM-1), which reads signals through its own RS232 interface, and the Chromex Imaging Spectrograph (optical emission spectroscopy data).

The signals are collected by software running on a PC located next to the etcher. Data is saved to the local hard disk as it arrives. The hard disk is mounted on a local area network (LAN) so that the data are easily available to any host with access to the network. Most of the analysis done for this thesis was run on workstations using UNIX file systems.

2.3.2. Signal Selection

An important part of data collection is the selection of sensor signals: often a machine has hundreds of signals available, but only some of these will be useful for the purposes of SPC. The signals most sensitive to the equipment state are desired, and those signals that are insensitive or have no impact on the equipment state are to be ignored. Usually the desired signals can be chosen with intuition, but one could run a designed experiment to objectively test the significance of individual signals.

This thesis is mostly based on data from the SECS-II signals. These data are collected at a sampling frequency of about 1 Hz. Some of the important signals monitored are: RF load coil position, RF tune vane position, peak-to-peak voltage, load impedance, RF phase error, DC bias, and endpoint. These signals were chosen because they are sensitive to changes in the state of the etch chamber, which directly impacts the wafer. Because these measurements are related electrically or mechanically, some signals are highly correlated with each other.

2.3.3. Pre-processing

As stated in Section 2.3.2, signals that reflect the state of the machine chamber are most useful for SPC. Furthermore, certain intervals of the signal may be more useful than others. In plasma etching, for example, the processing of a single wafer may typically consist of a dozen steps, including steps for stabilization of machine settings and initialization steps, like purging of the etch chamber. In this work, the steps of interest are usually steps in which actual etching of the wafer occurs: the *main etch* step and the *over-etch* step. Of course, a more in-depth analysis would attempt to include all the steps.

In addition to certain steps being more useful than others, certain portions of the signal within a step may be more useful. As an example, a few seconds may be required for the power to stabilize at the beginning of each step. The sensor measurements collected

during this time can be quite erratic, casting doubt on whether or not they contain any information about the state of the machine. Consequently, this transient effect is usually removed to simplify the analysis. For each wafer, the data to be kept for analysis are all concatenated together to form a single stream; this is repeated for each signal that is to be monitored. For further simplification, sometimes the number of samples to be analyzed is constrained to be the same for each wafer, although this is not necessary.

Chapter 3 Predictive Modeling

3.1. Introduction

3.1.1. Motivation

One of the limitations of the real-time SPC methodology described in Chapter 2 and reported in [18] is that training is required to establish the machine's baseline (in-control) behavior. Although the methodology has been shown to be sensitive to such equipment faults as miscalibrated machine settings, improper wafers, and changes in chamber pressure, the sensitivity of the SPC algorithm is derived from its need to be trained on a specific machine running a specific set of input settings. While this may not be a problem for high volume manufacturers who only produce a small number of products, it can be a significant hindrance to manufacturers who may need to change process or equipment settings often.

The goal of this chapter is to enable manufacturers to change equipment settings (also known as recipes) at will without degrading the effectiveness of their real-time SPC. A methodology is proposed to accommodate these users; this is achieved by creating models that predict the effect of new recipes on the sensor readings. Since recipes are only changed once before the processing of a wafer, the models only predict shifts of the sensor readings at the wafer-wafer level; the real-time level is assumed to be unaffected by a recipe switch.

Although only recipe changes are addressed here, rather than equipment or process changes, the methodology could actually be extended to include any kind of alteration to the processing of a wafer. For example, a change of equipment, a change in the amount of exposed surface area on the wafer, or even a change in the mask pattern used in a previous lithography step, could affect the readings of the internal sensors. Any of these could also

be modeled (together with the recipes), provided that the condition can be precisely identified, and provided that enough observations are available to build a reasonable model. For this work we simplify matters by modeling only the equipment settings (i.e., the recipes) of a plasma etcher, such as power settings, gas flows, pressure, and gap spacing.

Furthermore, although only predictive models created at the wafer-wafer level are addressed here, similar models could be built at the lot-lot level. If changes in the equipment or process are made on a lot-lot basis, one might wish to create a predictive lot-lot model to predict their effects on the lot-lot signals. Therefore, throughout this chapter, when a predictive wafer-wafer model is built based on wafer-wafer recipe changes, the reader should remember that a similar procedure could be applied to a predictive lot-lot model based on lot-lot changes.

3.1.2. Modeling

In this chapter, the simplicity of the models is just as important as is their accuracy. Simplicity means that the models can be created and used for prediction with the least possible effort; this is desired since manufacturers may have to build these models quite often and may have little time for verification or refinements. In addition, the amount of data available to produce a model may be limited, which again shows that simple model structures are preferred. The different recipes or processes of interest will cover a vast range of operating points, so the linear models created here to predict changes in the sensor signals can only be approximations.

Moreover, since changes in recipes and equipment are expected to occur at the lot-lot time scale, these models will need to survive through drifts (perhaps due to natural aging) or other intentional or unintentional changes in the state of the equipment; in other words they will need to be adequate for relatively long periods of time—longer than can be justified by the span of observations used to create the models. In conclusion, simple linear models that are easy to generate will be used, since more sophisticated models would not necessarily be more accurate over a long period of time.

3.1.3. Overview

Background on the modeling techniques is given in Section 3.2, and an example of predictive modeling using actual production data is presented in Section 3.3.

3.2. Background

3.2.1. Generalized Model

Here background is presented on modeling the effect of recipe changes on sensor readings. As mentioned previously, recipes are only changed once before the processing of a wafer, so the models of this chapter are for predicting the sensor signals at the wafer-wafer level; the real-time component is captured with the already described time-series models. The combined wafer-wafer and real-time model can be generalized in an *ARIMAX* model, which is an extension of an ARIMA model. The "X" stands for "exogenous", which means the model now contains additional explanatory variables; the additional variables are the recipe settings.

An ARIMAX model, also known as a *transfer function model*, contains two additive parts. The overall ARIMAX model for a stationary process, y_t, based on the stationary processes, $x_{i,t}$, has the following form:

$$y_t = \left[\beta_0 + \sum_{i=1}^{M-1} \beta_i \frac{\omega_i(B)}{\delta_i(B)} x_{i,t}\right] + \left[\frac{\theta(B)}{\phi(B)} a_t\right], \tag{3.1}$$

where a_t are the forecast errors (assumed to be independently and normally distributed with zero mean and variance σ^2), $(M-1)$ is the number of explanatory variables included in the model, and β_i are constants. The polynomials in B ($\omega_i(B)$, $\delta_i(B)$, $\theta(B)$, and $\phi(B)$) are polynomial backshift operators as defined in (2.1). See [5][34] for a more thorough discussion.

In this chapter, only the first square-bracketed term on the right hand side of the ARIMAX model (3.1) is of concern. This first term accounts for the wafer-wafer level effects of the changing machine settings, $x_{i,t}$, on a certain signal, y_t, where i indexes one of the $(M-1)$ recipe parameters being modeled. After the first term of the model has been built, and the wafer-wafer level effects have been subtracted out, the second term of the

model can be built. The second square-bracketed term represents the ARIMA model for the real-time component of the signal as described previously (Section 2.2.2.1).

3.2.2. Simplified Model

Note that the first term is quite general and models not only the influence of the current recipe on the signal, but also the influence of past recipes. However, due to reasons stated in Section 3.1.2, the model will be simplified to only include the current recipe; the influence of past recipes is not modeled. The simplified form is:

$$y_t = \left[\beta_0 + \sum_{i=1}^{M-1} \beta_i x_{i,t}\right] + \left[\frac{\theta(B)}{\phi(B)} a_t\right]. \tag{3.2}$$

The first term (the wafer-wafer term) becomes just a linear combination of the recipe variables (a linear regression). The coefficients β_i for $i = 0, \ldots, M-1$ need to be estimated.

More sophisticated multivariate techniques such as principal component regression (PCR) or partial least squares (PLS) [22][10] regression could be used, but these more complicated methods are difficult to justify due to the increased effort required to use them and due to the limited amount of data available (see Section 3.1.2).

3.2.3. Least Squares Estimation

Classical least squares techniques are used to estimate the coefficients of the linear models. This theory is well known [4][6], but will be repeated here for reference. Our wafer-wafer level prediction model is:

$$y = \beta_0 + \sum_{i=1}^{M-1} \beta_i x_i, \tag{3.3}$$

where the subscript t has been dropped for clarity. Again, the x_i are recipe variables, and the y represents one of the sensor signals. The least squares criterion chooses the β_i values that minimize the sum of squared residuals:

$$\sum_{j=1}^{N} \left\{ y_j - \left[\beta_0 + \sum_{i=1}^{M-1} \beta_i x_{i,j}\right] \right\}^2, \tag{3.4}$$

where j indexes one of the N observations of the variables and response. Note that scaling each variable x_i to unit variance is usually needed in order to avoid numerical precision problems.

The equations can be expressed much more compactly in matrix form. Let $\mathbf{X}$ be the N by M data matrix whose each column contains the observations for one recipe variable, except the first column, which contains all 1's. Let $\mathbf{y}$ be the M by 1 column of observed responses. Let $\hat{\beta}$ be the M by 1 column of estimated coefficients, the first element being the intercept. Then any $\hat{\beta}$ satisfying the *normal equations*:

$$(\mathbf{X}^T\mathbf{X})\hat{\beta} = \mathbf{X}^T\mathbf{y} \tag{3.5}$$

gives a least squares fit. See [6] for a discussion of computational methods for solving this set of equations. Note that the estimates of the coefficients $\hat{\beta}$ are poor if the columns of $\mathbf{X}$ are highly correlated.

3.2.4. Prediction Model

Letting L denote the number of modeled response signals, once $\hat{\beta}$ has been calculated for each signal, an M by L matrix $\mathbf{H}$ can be assembled that contains the corresponding coefficients column $\hat{\beta}$ for each of the signals. Now the full prediction model can be written as:

$$\hat{\mathbf{y}} = \mathbf{H}^T\mathbf{x}, \tag{3.6}$$

where $\hat{\mathbf{y}}$ is the L by 1 column of predicted responses, and $\mathbf{x}$ is the M by 1 column of new recipe settings.

3.3. Example

3.3.1. Experiment

Data for this example are taken from an experiment conducted by Texas Instruments (TI) in Dallas for the Sematech J-88-E project. The experiment (#26) was part of a larger study of various sensors and analysis techniques. The test structure used in this experiment was a multi-layer structure with TiN, Al, TiN, and oxide on silicon, which mimics the via and contact processes TI is developing.

The experiment consisted of 38 8-inch wafers processed on a Lam TCP 9600 metal etcher; the etchant gases are BCl_3 and Cl_2. The first 21 wafers were run in the first lot, and the rest were run in the second lot. A total of seven *centerpoints* and three *checkpoints* were run. A centerpoint is a "center" of the experimental design; i.e., a wafer run with the nominal recipe from which other experimental runs deviate from. The centerpoint settings are shown in Table 3.1. On the other hand, the checkpoints are wafers run with recipes different from any of the experimental runs. These wafers were not used by TI for modeling, but were used to verify models built using the other wafers in the experiment.

top power (Watt)	bot power (Watt)	pressure (mTorr)	Cl2+BCl3 (sccm)	Cl2 / BCl3
350.0	130.0	13.5	150.0	1.000

Table 3.1.
Centerpoint etch recipe.

The designed recipe parameters were: top power (Watt), bottom power (Watt), chamber pressure (milliTorr), total gas flow rate (Cl_2 flow + BCl_3 flow), and gas flow ratio (Cl_2 flow / BCl_3 flow); gas flow rates are measured in units of sccm, standard cubic centimeters per minute. Note that total gas flow and gas flow ratio are varied in the experiment rather than the individual gas flows, since these have more physical significance. See Table 3.2 for the complete design of experiment (DOE).

wafer #	run #	top power	bot power	press. (mT)	Cl2 (sccm)	BCl3 (sccm)	Cl2+ BCl3	Cl2/ BCl3
1	ctr 1	350.0	130.0	13.5	75.0	75.0	150.0	1.000
2	5	392.7	121.5	16.3	68.4	73.1	141.5	0.936
3	5	392.7	121.5	16.3	68.4	73.1	141.5	0.936
4	18	307.3	138.5	16.3	81.7	76.8	158.5	1.064
5	10	307.3	138.5	10.7	72.9	68.5	141.5	1.064
6	8	307.3	121.5	16.3	72.9	68.5	141.5	1.064
7	7	392.7	138.5	10.7	68.4	73.1	141.5	0.936
8	6	307.3	138.5	16.3	68.4	73.1	141.5	0.936
9	17	392.7	121.5	16.3	81.7	76.8	158.5	1.064
10	15	392.7	138.5	16.3	76.6	81.9	158.5	0.936
11	ctr 2	350.0	130.0	13.5	75.0	75.0	150.0	1.000
12	13	392.7	121.5	10.7	76.6	81.9	158.5	0.936
13	14	307.3	138.5	10.7	76.6	81.9	158.5	0.936
14[a]	9	392.7	121.5	10.7	72.9	68.5	141.5	1.064
15	19	392.7	138.5	10.7	81.7	76.8	158.5	1.064
16	4	307.3	121.5	10.7	68.4	73.1	141.5	0.936
17	11	392.7	138.5	16.3	72.9	68.5	141.5	1.064
18	chk 1	350.0	132.0	10.0	75.0	75.0	150.0	1.000

Table 3.2. (Sheet 1 of 2).
Design of experiment. In the "run #" column, "ctr" denotes a centerpoint, and "chk" denotes a checkpoint.

wafer #	run #	top power	bot power	press. (mT)	Cl2 (sccm)	BCl3 (sccm)	Cl2+ BCl3	Cl2/ BCl3
19	12	307.3	121.5	16.3	76.6	81.9	158.5	0.936
20	16	307.3	121.5	10.7	81.7	76.8	158.5	1.064
21	ctr 3	350.0	130.0	13.5	75.0	75.0	150.0	1.000
22	ctr 4	350.0	130.0	13.5	75.0	75.0	150.0	1.000
23	24	450.0	130.0	13.5	75.0	75.0	150.0	1.000
24	29	350.0	130.0	13.5	65.0	65.0	130.0	1.000
25	chk 2	325.0	140.0	12.0	71.0	69.0	140.0	1.029
26	28	350.0	130.0	13.5	80.2	69.8	150.0	1.150
27[b]	31	350.0	130.0	7.0	75.0	75.0	150.0	1.000
28	ctr 5	350.0	130.0	13.5	75.0	75.0	150.0	1.000
29	ctr 6	350.0	130.0	13.5	75.0	75.0	150.0	1.000
30	30	350.0	130.0	13.5	85.0	85.0	170.0	1.000
31	23	250.0	130.0	13.5	75.0	75.0	150.0	1.000
32	26	350.0	150.0	13.5	75.0	75.0	150.0	1.000
33	27	350.0	125.0	13.5	68.9	81.1	150.0	0.850
34	chk 3	425.0	135.0	18.0	78.0	82.0	160.0	0.951
35	32	350.0	130.0	20.0	75.0	75.0	150.0	1.000
36	25	350.0	110.0	13.5	75.0	75.0	150.0	1.000
37	ctr 7	350.0	130.0	13.5	75.0	75.0	150.0	1.000
38[c]	31	350.0	130.0	7.0	75.0	75.0	150.0	1.000

Table 3.2. (Sheet 2 of 2).
Design of experiment. In the "run #" column, "ctr" denotes a centerpoint, and "chk" denotes a checkpoint.

a. The SECS-II data for this wafer were cut short for some reason.

b. The pressure setting of 7.0 mTorr was lower than allowed by the equipment. causing a malfunction alarm.

c. The pressure setting of 7.0 mTorr was lower than allowed by the equipment, causing a malfunction alarm.

The *main etch* step is the fourth step of the etching process, in which the RF power is applied to begin the etch cycle. During the main etch step, the intensity of a particular wavelength of light emitted by the plasma is monitored[1]. An abrupt change in this monitor

1. The wavelength used is 261.8 nm. This line corresponds to the species AlCl, a byproduct of Al etching.

signal, known as the *endpoint* signal, indicates that the underlying substrate has begun to be exposed, and therefore that the step has completed [21].

At this point, although much of the film has been etched away, some material still remains due to the process non-uniformity. To allow for this, etching is continued in a fifth step, the *over-etch* step. In practice, this step lasts for an amount of time equal to a percentage of the main etch step's duration. In this case, the over-etch was run for 100% of the main etch time. Data from the SECS-II interface[2] were collected at a sampling rate of about 1 Hz during the second through sixth steps.

3.3.2. Procedure

The SECS-II software collects a large number of real-time signals, only some of which are useful for SPC. Thus the first step was to pick a subset of the signals to analyze. The criteria for selecting signals was somewhat subjective: signals should be stationary, or be able to be made stationary via differencing. Signals should be continuous and "smooth" (not flat, spiky, or containing steps). Signals that are flat (constant) obviously do not contain any useful information, and signals that have spikes (large, abrupt jumps) will trigger many false alarms. The signals chosen for this example are listed in the results found in Section 3.3.3.

Next each real-time signal was averaged over the duration of each wafer to produce wafer-wafer level signals (the first few seconds of data are ignored in order to remove any transient effects; see Section 2.3.3). A prediction model for each wafer-wafer signal was built by regressing the data from all wafers onto the designed recipe parameters[3] (models for the main etch step and over-etch step are created separately). Apparently, the following three wafers had either processing or data collection problems and could not be used in the analysis: wafers #14, 27, and 38.

---

2. TI used a custom collection software designed to run on a workstation.
3. The statistical software package S-PLUS was used to build the regression models [3].

3.3.3. Results

The results of the regressions are given in Figure 3.1 (the results in the figure are only given for the main etch models, but the results for the over-etch models are similar). The figure displays the R^2 value for each signal's wafer-wafer prediction model[4]. Each residual estimate has 29 degrees of freedom, since 35 observations (wafers) were available for analysis, and each model has six parameters (5 recipe parameters and an intercept). For simplicity, no elimination of insignificant model parameters was performed[5].

As can be seen by the R^2 values, many of the wafer-wafer level signals correlate very well with the machine settings, while other signals correlate very poorly. Note that a few of the sensor signals are almost perfectly correlated to certain machine controls. For instance, the "Chamber_pressure" sensor is almost perfectly correlated to the chamber pressure input setting, as is the "RF_Gen_#3_TCP_FWD_PWR" sensor to the top power setting. To see the prediction models in use, an example plotting the actual real-time signal for "RF_match_#1_tuning_position" alongside the predicted wafer-wafer values (for the main etch step) is shown in Figure 3.2.[6] Of course the predictions are never perfect; the actual wafer-wafer values (real-time signals averaged over each wafer) will differ from the predictions. The difference between them are the wafer-wafer residuals; these will be passed to the SPC algorithm.

4. The R^2 statistic is a percentage value telling the amount of total variance that is explained by the model; a value of unity is the best [26].

5. However. closer examination reveals that out of the six model parameters, the top power, bottom power, chamber pressure. and intercept were significant for almost all signals, while the total gas flow and gas flow ratio were only significant for a few of the signals.

6. Only 35 wafers are shown in the figure, rather than 38, since wafers #14, 27, and 38 were removed from the analysis.

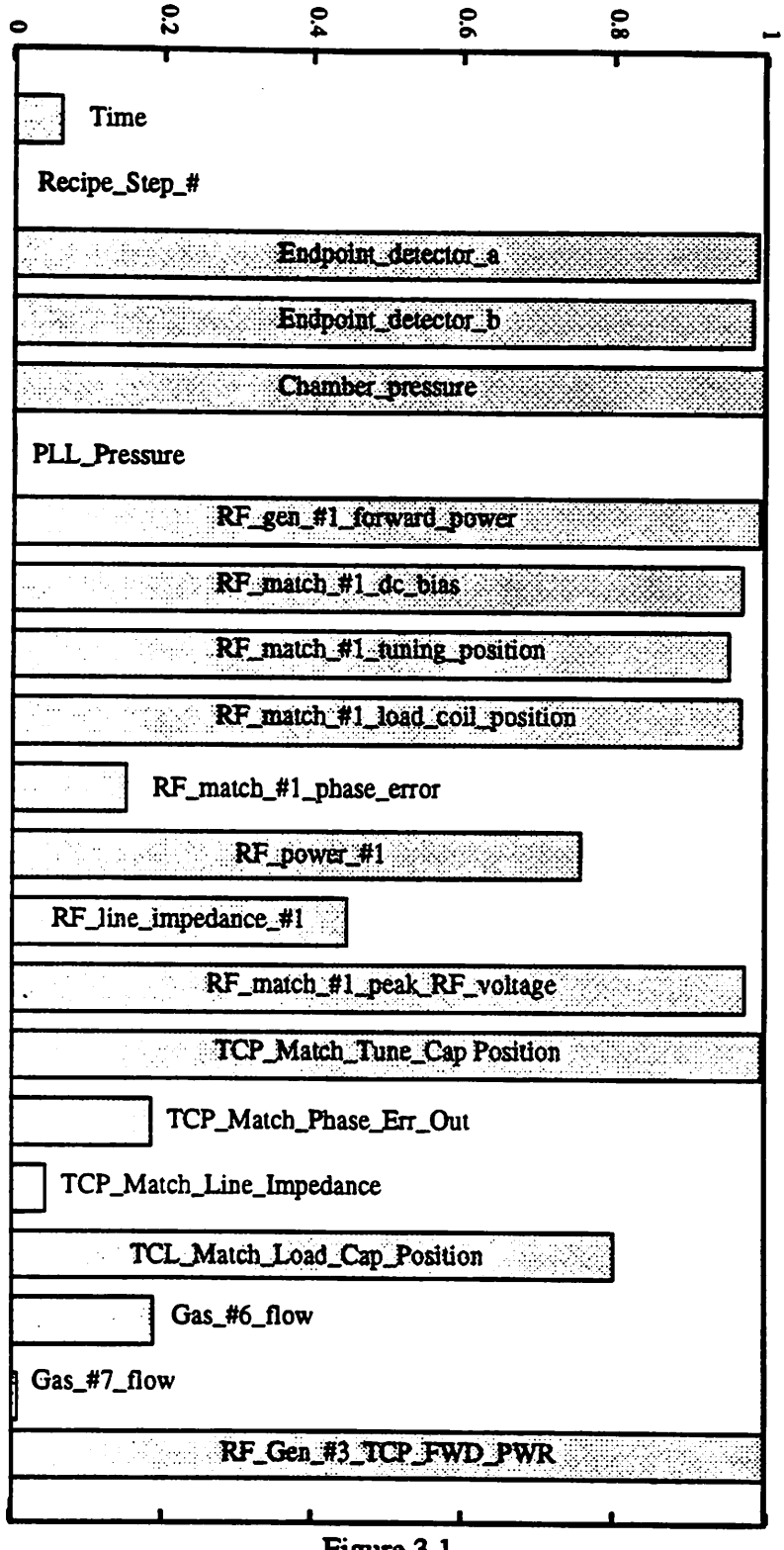

Figure 3.1.
R^2 values for various wafer-wafer signal prediction models.

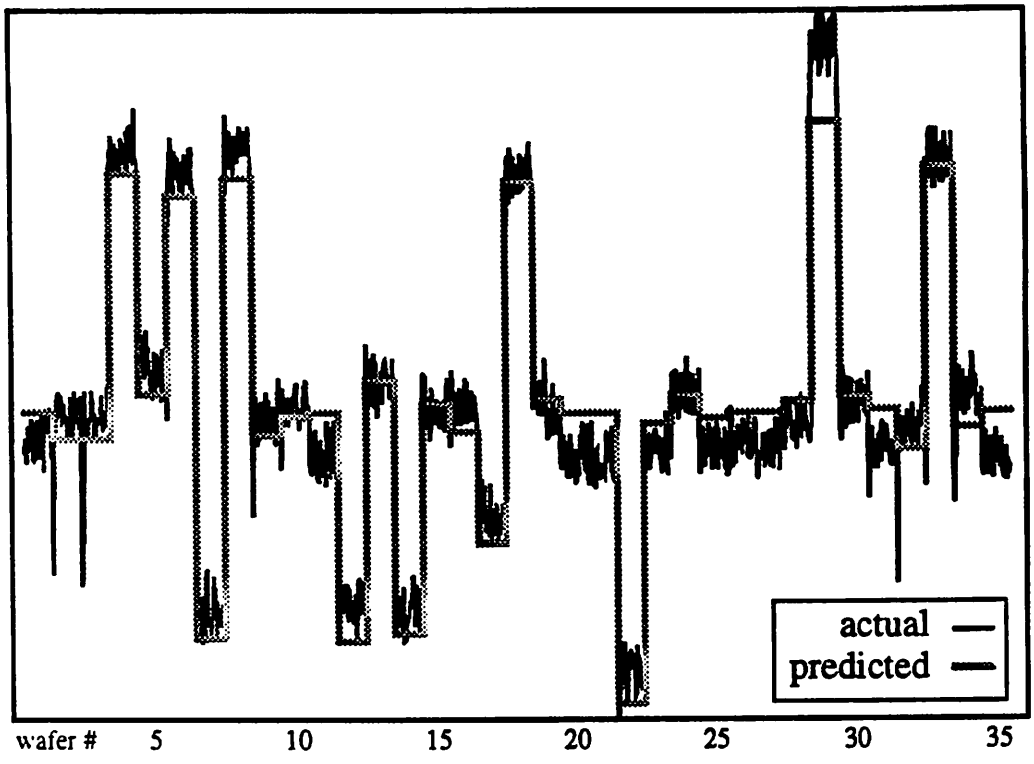

Figure 3.2.
Predicted wafer means and actual real-time signal for
"RF_match_#1_tuning_position".

Chapter 4 Adaptive Modeling

4.1. Introduction

4.1.1. Motivation

As discussed in Section 3.1.1, the real-time SPC methodology described in Chapter 2 requires training in order to establish an equipment's baseline behavior. The need for a training experiment places a considerable prerequisite on those who wish to use the methodology, possibly costing much time and effort. The work in Chapter 3 alleviates this requirement somewhat, by avoiding the methodology's need to be trained on a specific machine running a specific set of input settings; although this accommodates users who change machine or process settings often, it still does not obviate the necessity for training.

In this chapter, *adaptive* modeling techniques are introduced with the goal of eliminating the need for baseline training experiments. Adaptive models that can adjust to the statistics of a signal and also track its statistical variations offer greater flexibility in the model building process. Not only do they remove the need for baseline training, but they are also the answer to the problem of machine drift —i.e., slow changes in the state of the equipment, perhaps due to natural aging. Adaptive models are able to detect these changes and update themselves in real-time; the updates are automatic, without any need for intervention by the user.

Before extolling the potential merits of adaptive models too far, note that practical matters will unavoidably temper their success. Issues such as estimation noise and the amount of data available will impose inevitable tradeoffs. Note that adaptive models can only track variations in the data that change slowly relative to the model's speed of convergence. Abrupt changes that are intentional, like recipe changes, will still require predictive

models in order to anticipate their effects on the sensor readings; abrupt changes that are unintentional will be interpreted as equipment malfunctions.

4.1.2. Adaptive and Predictive Modeling

As a further motivation, the techniques of this chapter can be combined with the predictive modeling of Chapter 3 and the hierarchical modeling ideas of Section 2.2.3 to model signals that are non-stationary and, in addition, are affected by abrupt recipe, machine, or process changes. For example, for an SPC scheme that detects faults in *real-time*, in addition to an adaptive model that operates at the real-time level to track statistical variations, a predictive model is also needed at the wafer-wafer level to anticipate the effect of recipe changes. More specifically, the predictive models use the processing recipe to predict the mean values for each of the sensor signals; these mean values are used to center the data for the adaptive algorithm at the real-time level.

Another possibility is to use both predictive modeling and adaptive modeling at the wafer-wafer level in order to dynamically correct a prediction model. Suppose a wafer-wafer prediction model is built that predicts the effect of recipe changes on the mean value of each signal. In order to keep this model up-to-date, it should be adjusted whenever new data is available. Thus, the prediction model itself can be made adaptive by simply using the recipe settings as inputs to an adaptive filter, and starting the initial filter with the original prediction model that was built.

4.1.3. Chapter Overview

Extensive background on adaptive filters is given in Section 4.2, beginning with an introduction to optimum static filters. Then the section continues towards a development of adaptive filters, and ends with a summary, which outlines some important issues, including stability and convergence rate. Section 4.3 details how the actual adaptive algorithms are implemented in a computer. A discussion of several important implementation issues is found in Section 4.4, including tradeoffs that need to be considered when using adaptive algorithms, and how to use adaptive modeling to implement real-time SPC. An example of how adaptive modeling can be used on actual production data is presented in Section 4.5.

4.2. Background

The term *filter* is used to describe a tool that is applied to *observations* of a signal in order to extract certain information about the signal. The design of an *optimum* filter is based on some *a priori* assumptions about the statistics of the signal. However, if the signal is unknown, difficult to characterize, or is time-varying, the use of an *adaptive* filter is often advantageous. An adaptive filter starts from a predetermined set of initial conditions and, based on some recursive algorithm, tries to converge to the optimum solution. In a time-varying environment, the adaptive filter offers the ability to track variations in the statistics of the input data, provided that the variations are sufficiently slow.

Adaptive filters have been successfully used in a vast variety of applications. Some of these applications include: adaptive equalization for digital communication channels, system identification, speech compression, signal detection, and echo cancellation. Although these applications are quite diverse, they each use adaptive filters in a similar way: the input signal is used to estimate a *desired response*, and the estimation error is used to adjust the filter.

However, the essential difference among the various applications of adaptive filtering arises in the manner in which the desired response is extracted. For example, in a system identification application, the desired response is the actual output of the unknown system being modeled, whereas in an echo cancellation application, the desired response is the input signal with echoes removed.

In this paper, adaptive filtering will used as a *modeling* technique. The technique works by assuming that the observed signal is produced as the output of a *white (uncorrelated) noise process* fed through a linear transfer function. With this assumption, the signal can be input into an adaptive filter that attempts to find the inverse of the transfer function. The filter's goal is to predict the *next value* of the signal, based on past values. Therefore, the desired response is the actual next value of the signal. The filter works by continually adjusting and correcting itself, until it reproduces the original white noise process; thus, it is a *whitening filter* (also known as a *prediction-error filter*).

If the adaptive filter is successful, it will have found a model for the observed signal (or an approximation to it). Any observed deviations from the model can be easily detected and flagged as equipment malfunctions. In particular, adaptive models can replace the time series models needed in the SPC methodology of Section 2.2.2.

This section is relatively lengthy, but is necessary for a good understanding of the subject matter. For further references, the reader is encouraged to consult [11][12][37][7]. The section is divided into a number of sub-sections. Section 4.2.1 begins by introducing non-adaptive linear filters and solving the optimum linear filtering problem. Section 4.2.2 extends this theory by showing an iterative solution to the problem based on the method of steepest descent; the iterative solution forms the basis for a simple adaptive filtering algorithm. This algorithm is a simplified version of another adaptive algorithm based on the method of least squares, discussed in Section 4.2.3. In both of the above two sections, the important issues of stability and convergence are addressed. This discussion concludes with a summary in Section 4.2.4.

4.2.1. Wiener Filters

The theory of Wiener filtering is concerned with extracting information about a signal given some observations of that signal. More precisely, a Wiener filter seeks to make an *optimum* estimate of some desired response based on a *linear* function of the observations. Filtering is also a kind of modeling, since the optimum filter will represent the correlation structure of the input signal. The optimality criterion used here is to minimize the *mean square* error of the filter estimates.

4.2.1.1. Modeling Assumptions

The theory behind the solution of an optimum Wiener filter necessarily assumes certain model structures for both the input signal and the filter. For the problem presented here, the input signal is modeled as a ***real***, ***discrete-time***, and ***wide-sense (second-order)*** *stationary* stochastic process. A wide-sense stationary process, $\{x(t)\}$, is characterized by its first and second moments:[1]

1. Usually processes will be assumed to have zero mean, but not necessarily unit variance. If a process does not have zero mean, it can be easily centered by subtracting the mean from each value of the process.

1. The mean value of the process, m, is constant for all times t:

$$m \equiv E[x(t)] = \text{constant} \tag{4.1}$$

2. The *autocorrelation function* of the process, defined by

$$r_x(t_1, t_2) \equiv E[x(t_1)x(t_2)], \tag{4.2}$$

depends only on the difference between the observation times t_1 and t_2, as shown by

$$r_x(t_1, t_2) = r_x(t_2 - t_1). \tag{4.3}$$

This way of characterizing a stochastic process is practical since it lends itself to measurements and is well suited for linear analysis. Note that a *Gaussian* process is completely characterized by its first and second moments: if it is wide-sense stationary, then it is also *strictly* stationary (i.e., its statistical properties are invariant to a shift in time origin).

In this analysis, the filter is assumed to be a linear, *finite-duration impulse response (FIR)* filter. An FIR filter contains only forward paths and no feedback loops, thus making it inherently *stable* and, in addition, mathematically tractable. On the other hand, an *infinite-duration impulse response (IIR)* filter contains both feedforward and feedback paths. Consequently, unless it is properly controlled, feedback can cause an IIR filter to become unstable. Although the stability problem is manageable for static filters, it complicates the situation for adaptive filters. Therefore, in most applications for which adaptivity is required, the use of FIR filters is preferred, even though an IIR filter can sometimes provide better performance (see Section 4.3.3 for more on adaptive IIR filtering).

4.2.1.2. Problem Formulation

Consider the block diagram of Figure 4.1. The filter input is $x(t)$, a realization of a discrete-time wide-sense stationary stochastic process, or more simply, a *time series*. The filter output is $y(t)$, a linear function of the input:

$$y(t) \equiv \sum_{k=0}^{M-1} h_k x(t-k), \tag{4.4}$$

where $[h_0, h_1, \ldots, h_{M-1}]$ is the impulse response of the filter (the filter coefficients or *filter taps*), and M is the order of the filter (the number of degrees of freedom). The output $y(t)$ attempts to estimate the desired response, denoted by $d(t)$. The estimation error is the difference between the desired response and the filter estimate:

$$e(t) \equiv d(t) - y(t). \tag{4.5}$$

Recall that in this paper adaptive filters are used as prediction-error (whitening) filters. Consequently, the filter output $y(t)$ is a prediction of the next value of the signal $x(t+1)$ before it is known, and the desired signal $d(t)$ is the actual value $x(t+1)$. This notation might be slightly confusing at first, but is preferred over always using $x(t+1)$ or $\hat{x}(t)$. In addition, it keeps the derivation applicable to the general case.

The optimality criterion for the Wiener filter is to minimize the mean square error of the estimation error, denoted by

$$MSE \equiv E[e^2(t)]. \tag{4.6}$$

Substituting (4.5) into (4.6) we have

$$\begin{aligned} MSE &= E[(d(t) - y(t))^2] \\ &= E[d^2(t)] - 2E[d(t)y(t)] + E[y^2(t)] \end{aligned} \tag{4.7}$$

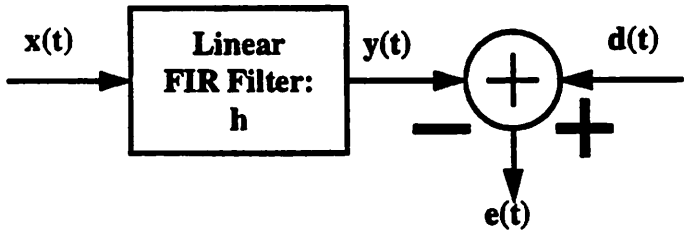

Figure 4.1.
Block diagram of the linear filtering problem.

4.2.1.3. Matrix Form

To solve for the optimum filter, the next step would be to substitute (4.4) into (4.7) and then set up a system of M equations by differentiating the resulting expression with respect to each of the filter coefficients. However, a more elegant solution is gained by first expressing (4.7) in matrix form. Therefore, an M-by-1 *column vector* of the current and $(M-1)$ previous observations is defined as

$$\mathbf{x} \equiv \left[x(t)\ x(t-1)\ \ldots\ x(t-M+1)\right]^T, \tag{4.8}$$

and an M-by-1 vector of the filter taps is defined as

$$\mathbf{h} \equiv \left[h_0\ h_1\ \ldots\ h_{M-1}\right]^T. \tag{4.9}$$

Now (4.4) can be written (dropping the indices)

$$y = \mathbf{h}^T\mathbf{x} = \mathbf{x}^T\mathbf{h}, \tag{4.10}$$

and (4.5) can be written

$$e = d - \mathbf{h}^T\mathbf{x}, \tag{4.11}$$

and (4.7) can be written

$$MSE = E[d^2] - 2E[d\mathbf{x}^T]\mathbf{h} + \mathbf{h}^T E[\mathbf{x}\mathbf{x}^T]\mathbf{h}. \tag{4.12}$$

Further define the *M-by-M correlation matrix* of the input signal $x(t)$ as

$$\mathbf{R}_x \equiv E[\mathbf{x}\mathbf{x}^T]. \tag{4.13}$$

The fact that the input signal is a real, wide-sense stationary process means that $\mathbf{R}_x$ is a *symmetric, Toeplitz, positive semi-definite* matrix. It is related to the input signal's autocorrelation function $r_x(k)$ as follows:

$$\mathbf{R}_x = \begin{bmatrix} r_x(0) & r_x(1) & \dots & r_x(M-1) \\ r_x(1) & r_x(0) & \dots & \dots \\ \dots & \dots & \dots & r_x(1) \\ r_x(M-1) & \dots & r_x(1) & r_x(0) \end{bmatrix}. \tag{4.14}$$

Since the matrix $\mathbf{R}_x$ is positive semi-definite, then

$$\mathbf{x}^T\mathbf{R}_x\mathbf{x} \geq 0 \text{ for all } \mathbf{x} \neq \mathbf{0}. \tag{4.15}$$

The equality only occurs when $\mathbf{R}_x$ is singular[2]; however, in practice $\mathbf{R}_x$ is almost always nonsingular, in which case it is *positive definite* so that all its eigenvalues are positive [2].

The desired signal and the input signal are assumed to be *jointly wide-sense stationary*, both with zero mean. Let the M-by-1 *cross-correlation vector* between the desired signal and the input signal be

$$\mathbf{p} \equiv \begin{bmatrix} p(0) & p(1) & \dots & p(M-1) \end{bmatrix}^T \tag{4.16}$$

$$p(k) \equiv E[d(t)x(t-k)], \tag{4.17}$$

so that

$$\mathbf{p} = E[d\mathbf{x}]. \tag{4.18}$$

---

2. Such a situation arises essentially only when the process consists of the sum of K sinusoids with $K \leq M$.

Since the desired signal is assumed to be wide-sense stationary and of zero mean, its *variance* can be defined as

$$\sigma_d^2 \equiv E[d^2] = r_d(0). \tag{4.19}$$

Now (4.12) can be re-written in matrix form as

$$MSE = \sigma_d^2 - 2\mathbf{p}^T\mathbf{h} + \mathbf{h}^T\mathbf{R}_x\mathbf{h}. \tag{4.20}$$

4.2.1.4. Optimum Filter Solution

The cost function, MSE, is a quadratic (second-order) function of the unknown filter taps $\mathbf{h}$. This function can be visualized as a bowl-shaped $(M+1)$-dimensional surface (a paraboloid) with M degrees of freedom represented by the elements of $\mathbf{h}$. This surface is referred to as the *error performance surface* of the filter. The importance of this particular surface is that it is characterized by a *unique* minimum (as long as $\mathbf{R}_x$ is non-singular). Figure 4.2 depicts the error performance surface for a 2-tap filter.

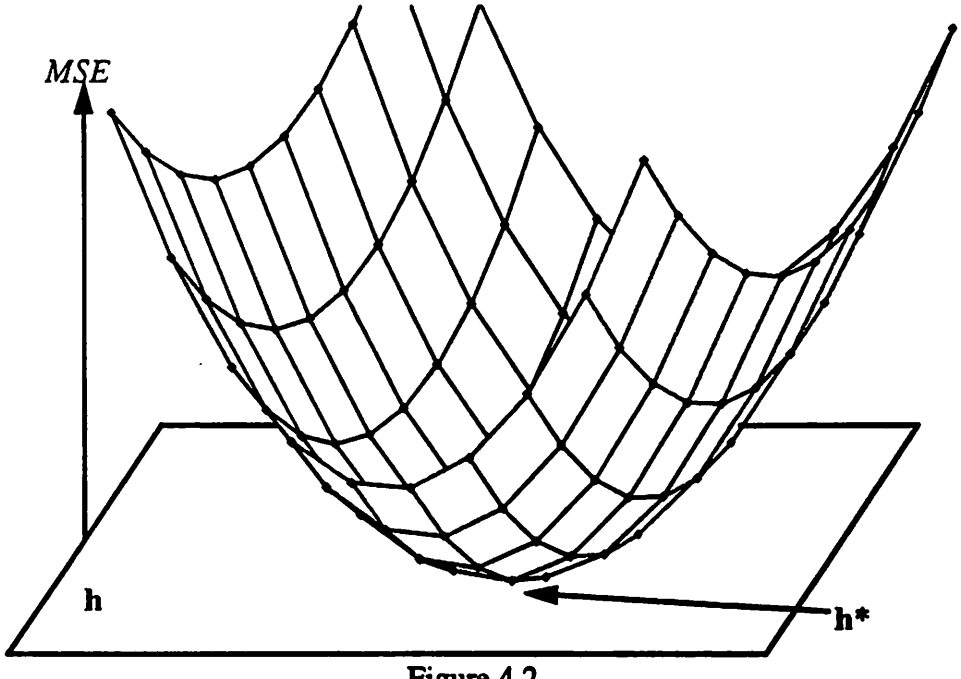

Figure 4.2.
Visualization of an error performance surface.

At the lowest point of the error performance surface, the cost function MSE attains its minimum value, denoted by MSE^*. This point corresponds to the optimum filter tap vector. At this point, the *gradient* vector must be identically zero; in other words,

$$\nabla_k MSE \equiv \frac{\partial}{\partial h(k)} MSE = 0 \text{ for } k = 0, 1, \ldots, M-1, \tag{4.21}$$

or in vector form:

$$\nabla MSE = \mathbf{0}. \tag{4.22}$$

The gradient vector can be calculated to be

$$\nabla MSE = -2\mathbf{p} + 2\mathbf{R}_x\mathbf{h}, \tag{4.23}$$

so that putting together (4.22) and (4.23), the system of M equations can be solved in matrix form:

$$\begin{aligned} -2\mathbf{p} + 2\mathbf{R}_x\mathbf{h}^* &= \mathbf{0} \\ \mathbf{R}_x\mathbf{h}^* &= \mathbf{p} \\ \mathbf{h}^* &= \mathbf{R}_x^{-1}\mathbf{p} \end{aligned} \quad . \tag{4.24}$$

The solution shows the optimum filter taps denoted by $\mathbf{h}^*$. This vector of filter taps is *optimum in the mean square sense*; no other M-order linear filter can be designed that has a smaller mean square error. The minimum value of the MSE function can be found using (4.20) and (4.24) as

$$MSE^* = MSE|_{\mathbf{h}=\mathbf{h}^*} = \sigma_d^2 - \mathbf{p}^T\mathbf{R}_x^{-1}\mathbf{p}. \tag{4.25}$$

Equations (4.21) are known as the *Wiener-Hopf equations* and also as the *normal equations*. The term "normal equations" refers to the geometric development of the optimum filtering problem, commonly known as the *principle of orthogonality*. Geometrically, the minimum point of a quadratic error function occurs when the error signal is *orthogonal* (normal) to the hyperplane spanned by the space of the M filter inputs. The geometric development of the solution can be shown to be equivalent to the minimum mean square error method.

4.2.1.5. Canonical Form of the Error Performance Surface

An alternate form for (4.20), the formula for the error performance surface, will be useful both theoretically and conceptually in subsequent sections. By using (4.24) in both (4.20) and (4.25) to remove the presence of $\mathbf{p}$, then subtracting the former by the latter, one arrives at the expression

$$MSE = MSE^* + (\mathbf{h} - \mathbf{h}^*)^T \mathbf{R}_x (\mathbf{h} - \mathbf{h}^*). \tag{4.26}$$

This equation shows explicitly the unique optimality of the minimizing filter tap vector $\mathbf{h}^*$. Recall that $\mathbf{R}_x$ is positive semi-definite, so that MSE can never be less than MSE^*.

The quadratic form on the right side of (4.26) is quite informative; however, the formula can be further simplified by a change of basis. Let $\mathbf{Q}$ be an M-by-M matrix:

$$\mathbf{Q} \equiv \left[\mathbf{q}_1 \; \mathbf{q}_2 \; \cdots \; \mathbf{q}_M\right], \tag{4.27}$$

where each $\mathbf{q}_k$ is a unit-length eigenvector of $\mathbf{R}_x$ with corresponding eigenvalue λ_k. Also define the M-by-M diagonal matrix of eigenvalues as

$$\Lambda \equiv diag(\lambda_1, \lambda_2, \ldots, \lambda_M). \tag{4.28}$$

With these definitions, the matrix $\mathbf{R}_x$ can be expressed in terms of its eigenvalues and eigenvectors with a *similarity transformation*, where $\mathbf{Q}$ is a *unitary* matrix [2]:

$$\begin{aligned} \mathbf{R}_x &= \mathbf{Q}\Lambda\mathbf{Q}^T \\ \mathbf{Q}^T\mathbf{Q} &= \mathbf{I} \end{aligned} \tag{4.29}$$

After using (4.29) in (4.26) and defining the *transformed* version of $(\mathbf{h} - \mathbf{h}^*)$ as

$$\mathbf{v} \equiv \mathbf{Q}^T(\mathbf{h} - \mathbf{h}^*), \tag{4.30}$$

the formula (4.26) can be expressed in its *canonical form* as

$$\begin{aligned} MSE &= MSE^* + \mathbf{v}^T\Lambda\mathbf{v} \\ &= MSE^* + \sum_{k=1}^{M} \lambda_k v_k^2 \end{aligned} \tag{4.31}$$

This formulation uses the *principal axes* of the error performance surface as its new basis so that all cross-product terms disappear. The usefulness of this result will become apparent in later sections.

4.2.2. Method of Steepest Descent

Now that the optimum filter solution has been established for a stationary environment, this section proceeds to develop an adaptive solution. A straightforward approach is to develop an *iterative* procedure for solving the optimum filter problem. An iterative procedure begins with an initial "guess" for the filter taps, which is located some distance away from the minimum point of the error performance surface. After each iteration, an appro-

priate correction is applied to the filter in such a way that it moves closer to the minimum point. Thus, starting from an arbitrary point on the error performance surface, the filter adapts in a step-by-step fashion, always moving closer to the optimum solution. Of course, in a time-varying environment, the optimum point is constantly changing, so the filter never exactly reaches it.

An old optimization technique, the *method of steepest descent*, suggests a simple way of iterating towards the optimum solution: start with an initial guess for the filter taps (the guess could be arbitrary, or could be based on some prior knowledge), then compute the *gradient* vector and update the filter taps in the negative direction of the gradient (i.e., in the direction of steepest descent). The iteration update is as follows:

$$\mathbf{h}(n+1) = \mathbf{h}(n) + \frac{1}{2}\mu[-\nabla MSE(n)] \tag{4.32}$$

where the indices denote the iteration number, μ is a positive real-valued constant called the *step size*, and the factor 1/2 is added to simplify notation later. Figure 4.3 exhibits an example of the ideal path travelled by a filter tap vector as it moves towards the optimum point (the plot is a two-dimensional contour plot of an error performance surface). Notice that each point of the steepest descent path is always perpendicular to the contour curves, even if the contours are elliptical.

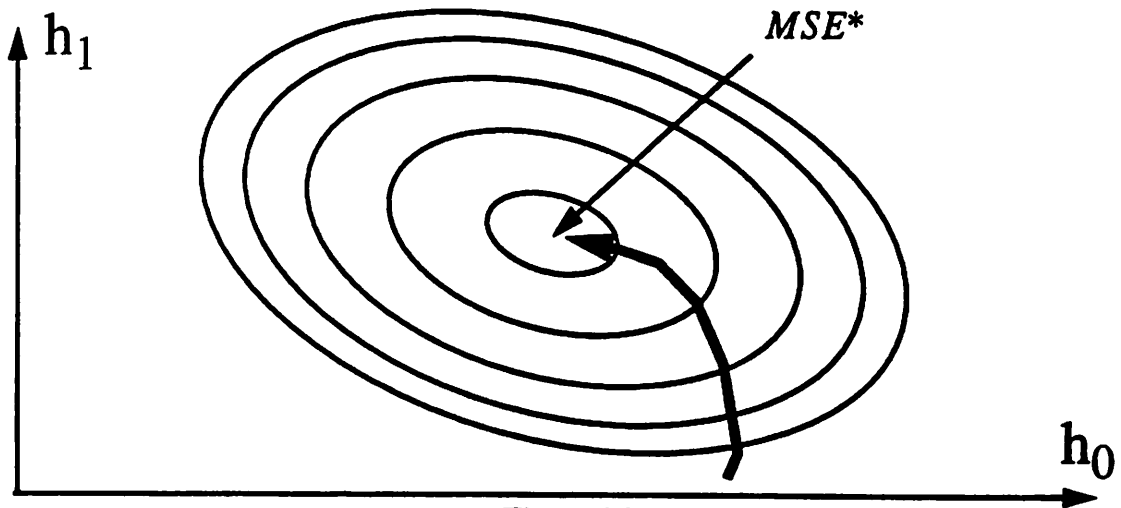

Figure 4.3.
Example of a steepest descent path.

4.2.2.1. Stability and Convergence of Steepest Descent

Substituting (4.23) into (4.32) results in the recursion relation

$$\begin{aligned} \mathbf{h}(n+1) &= \mathbf{h}(n) + \mu[\mathbf{p} - \mathbf{R}_x\mathbf{h}(n)] \\ &= (\mathbf{I} - \mu\mathbf{R}_x)\mathbf{h}(n) + \mu\mathbf{p} \end{aligned} \tag{4.33}$$

The presence of feedback in (4.33) means that the algorithm must be carefully designed for stability. Its stability depends on the input correlation matrix $\mathbf{R}_x$ and the step size μ; of these two, the step size is under the designer's control.

For a more complete stability analysis, the transformation of Section 4.2.1.5 will prove useful. First subtract $\mathbf{h}^*$ from both sides of (4.33) and use (4.24) to eliminate $\mathbf{p}$:

$$\begin{aligned} [\mathbf{h}(n+1) - \mathbf{h}^*] &= (\mathbf{I} - \mu\mathbf{R}_x)\mathbf{h}(n) + \mu\mathbf{R}_x\mathbf{h}^* - \mathbf{h}^* \\ &= (\mathbf{I} - \mu\mathbf{R}_x)\mathbf{h}(n) - (\mathbf{I} - \mu\mathbf{R}_x)\mathbf{h}^* . \\ &= (\mathbf{I} - \mu\mathbf{R}_x)[\mathbf{h}(n) - \mathbf{h}^*] \end{aligned} \tag{4.34}$$

Then using the similarity relation of (4.29) and the transformation of (4.30), the following is obtained:

$$\mathbf{v}(n+1) = (\mathbf{I} - \mu\Lambda)\mathbf{v}(n) . \tag{4.35}$$

The transformation allows for the decoupling of (4.35) into M scalar-valued first-order difference equations (recall that λ_k is an eigenvalue of $\mathbf{R}_x$):

$$v_k(n+1) = (1 - \mu\lambda_k)v_k(n) \text{ for } k = 1, 2, \ldots, M . \tag{4.36}$$

Each of these equations corresponds to a different *mode* of the algorithm. The solution to each is a simple *geometric series*:

$$v_k(n) = (1 - \mu\lambda_k)^n \cdot v_k(0) , \tag{4.37}$$

where $v_k(0)$ is the initial value for the k-th mode. A time constant for the convergence of the k-th mode can be approximated (for small μ) as

$$\tau_k \cong \frac{1}{\mu\lambda_k} . \tag{4.38}$$

This reflects the fact that a small step size μ will result in a large time constant (slow convergence), and a large step size will result in a small time constant (fast convergence).

Although a time constant for the convergence of each mode has been presented in (4.38), relating these back to the overall time constant $\tau_{\mathbf{h}}$ for the convergence of the filter

tap vector $\mathbf{h}$ is not straightforward. However, a worst-case (upper bound) for $\tau_{\mathbf{h}}$ can be given as

$$\tau_{\mathbf{h}} < \frac{1}{\mu\lambda_{min}}. \tag{4.39}$$

Clearly, a large step size is desired for fast convergence. Furthermore, the slowest mode of convergence is determined by the *minimum* eigenvalue λ_{min} of $\mathbf{R}_x$.

The desire for a large step size, however, conflicts with the limits that must be maintained for stability. To ensure the stability of the steepest descent algorithm, irrespective of initial conditions, the geometric series corresponding to each mode of the algorithm must converge. For the sum of a geometric series to converge, its geometric ratio must have absolute value less than one. Therefore, the stability condition is

$$|1 - \mu\lambda_k| < 1 \text{ for } k = 1, 2, \ldots, M. \tag{4.40}$$

Or, since the step size μ and the eigenvalues λ_k are real and positive, (4.40) simplifies to

$$0 < \mu < \frac{2}{\lambda_{max}}. \tag{4.41}$$

The stability condition places an upper limit on the step size μ, where the limit is determined by the *maximum* eigenvalue λ_{max} of $\mathbf{R}_x$.

Looking at (4.39) and (4.41), one sees immediately that a large *eigenvalue spread* hampers the performance of steepest descent algorithms. In other words, the worst-case convergence rate of the algorithm is poor if the eigenvalue ratio $\lambda_{max}/\lambda_{min}$ (also known as the condition number of $\mathbf{R}_x$) is large.

4.2.2.2. Least Mean Square (LMS) Algorithm

The steepest descent algorithm discussed in Section 4.2.2.1 is obviously an idealization, because in reality one does not know the exact correlation matrix $\mathbf{R}_x$ nor the cross-correlation vector $\mathbf{p}$. Consequently, the gradient vector must be *estimated* from the available data. Estimates based on data will be noisy, so that even in steady-state, the filter taps will continue to fluctuate around the optimum point. Steepest descent methods of this sort belong to the family of *stochastic gradient algorithms.*

The Least Mean Square (LMS) algorithm is a widely used algorithm because of its simplicity; it does not require matrix inversions nor other expensive operations. The LMS algorithm uses simple, *instantaneous* estimates for $\mathbf{R}_x$ and $\mathbf{p}$. (Compare these with (4.13) and (4.18), respectively):

$$\hat{\mathbf{R}}_x = \mathbf{x}\mathbf{x}^T \tag{4.42}$$

$$\hat{\mathbf{p}} = d\mathbf{x}. \tag{4.43}$$

Replacing the actual variables in (4.23) with their estimates, the instantaneous estimate of the gradient vector is

$$\begin{aligned} \hat{\nabla} MSE &= -2d\mathbf{x} + 2\mathbf{x}\mathbf{x}^T\hat{\mathbf{h}} \\ &= -2\mathbf{x}[d - \mathbf{x}^T\hat{\mathbf{h}}] \\ &= -2e\mathbf{x} \end{aligned} \tag{4.44}$$

where e represents the estimation error. Using this gradient estimate in (4.32) gives

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mu[e(n)\mathbf{x}(n)]. \tag{4.45}$$

As discussed in Section 4.2.2.1, the choice of step size μ requires particular attention. The condition given by (4.41) ensures *mean* convergence of the filter taps. However, in practice a more conservative condition is necessary [7]. This condition ensures *mean-squared convergence* of the error signal and is simpler to use since it does not require calculation of λ_{max}:

$$\begin{aligned} 0 < \mu < \frac{2}{M\sigma_x^2} \\ \sigma_x^2 \equiv E[x^2] = r_x(0) \end{aligned} \tag{4.46}$$

Note that applying the analysis of Section 4.2.2.1 to the LMS algorithm is not completely correct without a number of assumptions. In particular, consecutive input vectors $\mathbf{x}(n)$ must constitute a sequence of statistically independent vectors, and both $\mathbf{x}(n)$ and the desired response $d(n)$ at time n must be independent of all previous samples of $d(n)$. Furthermore, $\mathbf{x}(n)$ and $d(n)$ must consist of jointly Gaussian distributed random variables for all n. Clearly, these assumptions are often far from true; nevertheless, experience with the LMS algorithm has shown that the results of the analysis are usually found to be in agreement with experiments and computer simulations. Even if the results are not exactly correct, they serve as reliable filter design guidelines [11].

4.2.3. Method of Least Squares

The method of least squares is a classical method that seeks to fit a model—linear, in our case—to some observed data by minimizing the sum of square differences between the model predictions and the actual data. This theory is fundamentally different than the Wiener filtering theory presented previously, because the least squares theory does not presuppose a probabilistic framework—for instance, time averages are used, instead of expectations. Nevertheless, many analogous ideas exist between the two frameworks, like the principle of orthogonality and the normal equations. This section will try to present an intuitive understanding of the concepts, rather than divulge rigorous proofs, by extending the ideas from the previous section.

The discussion of Section 4.2.2.1 showed that the method of steepest descent can suffer from poor performance if its correlation matrix is *ill-conditioned*. Visually, this corresponds to a highly elliptical error performance surface whose major axis is significantly longer than its minor axis. This causes the algorithm to follow a highly indirect path towards the optimum point. An algorithm that leads an arbitrary initial condition directly to the optimum point would be ideal (compare Figure 4.4 with Figure 4.3).

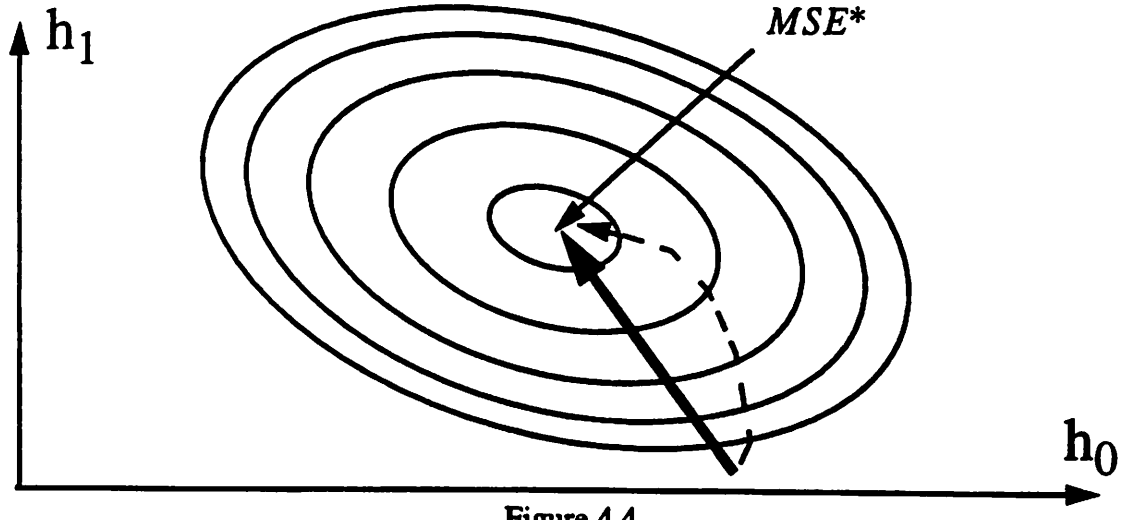

Figure 4.4.
Example of a "direct" path.

The update formula for the method of steepest descent (4.32) can be slightly modified to result in an improved algorithm. To see this, first multiply (4.23) on the left by $\mathbf{R}_x^{-1}$ and use the result of (4.24) to obtain

$$\mathbf{h}^* = \mathbf{h} + \frac{1}{2}\mathbf{R}_x^{-1}[-\nabla MSE], \tag{4.47}$$

which immediately suggests that the following algorithmic update would be ideal:

$$\mathbf{h}(n+1) = \mathbf{h}(n) + \frac{1}{2}\mu\mathbf{R}_x^{-1}[-\nabla MSE(n)]. \tag{4.48}$$

The formula now has the same form as (4.32) except for the presence of $\mathbf{R}_x^{-1}$. This algorithm is sometimes known as the *orthogonalized steepest descent* algorithm, since the $\mathbf{R}_x^{-1}$ term transforms an elliptical error performance surface into a circular one by orthogonalizing its axes. If the instantaneous gradient estimate of (4.44) were used in (4.48), the result would be an algorithm analogous to (4.45):

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mu\mathbf{R}_x^{-1}[e(n)\mathbf{x}(n)]. \tag{4.49}$$

The update of (4.48) is idealized since it requires knowledge of $\mathbf{R}_x^{-1}$, as well as the gradient vector ∇MSE. Furthermore, no clues are given as to how the step size μ should be chosen. Next a different approach will be pursued that will lead to a practical algorithm similar in form to (4.48).

4.2.3.1. Recursive Least Squares (RLS) Algorithm

In order to develop a realistic algorithm using the method of least squares, the time-averaged estimates for the correlation matrix $\mathbf{R}_x$ and the cross-correlation vector $\mathbf{p}$ are necessary. For operation in non-stationary environments, an exponential weighting factor (or *forgetting factor*) λ is also introduced so that recent data is more heavily weighted than past data. The estimates are:

$$\hat{\mathbf{R}}_x(n) = \sum_{k=0}^{n} \lambda^k \mathbf{x}(n-k)\mathbf{x}^T(n-k) \tag{4.50}$$

$$\hat{\mathbf{p}}(n) = \sum_{k=0}^{n} \lambda^k d(n-k)\mathbf{x}(n-k) \tag{4.51}$$

The positive constant λ should be close to, but less than (or equal to) one. The quantity $1/(1-\lambda)$ is a rough measure of the *memory* of the algorithm. The special case $\lambda = 1$ corresponds to infinite memory, i.e., the entire history of data is utilized.

One can show that the normal equations, derived in a manner similar to Section 4.2.1.4, are given by

$$\hat{\mathbf{R}}_x(n)\hat{\mathbf{h}}(n) = \hat{\mathbf{p}}(n). \tag{4.52}$$

Note that the estimated correlation matrix $\hat{\mathbf{R}}_x$ cannot be assumed to be Toeplitz, although it is still symmetric and positive semi-definite.

Calculating the estimates (4.50) and (4.51) for every iteration would be computationally expensive. The following simple recursions incur the least possible computation for each update:

$$\hat{\mathbf{R}}_x(n) = \lambda\hat{\mathbf{R}}_x(n-1) + \mathbf{x}(n)\mathbf{x}^T(n) \tag{4.53}$$

$$\hat{\mathbf{p}}(n) = \lambda\hat{\mathbf{p}}(n-1) + d(n)\mathbf{x}(n). \tag{4.54}$$

The recursions (4.53) and (4.54) could be used, theoretically, as the basis for an adaptive algorithm by using the estimates to solve $\hat{\mathbf{h}}(n)$ in (4.52). However, solving (4.52) requires the inversion of $\hat{\mathbf{R}}_x(n)$ and having to invert a matrix at each iteration would make the algorithm computationally expensive since matrix inversion requires on the order of M^3 operations, where M is the dimension of the matrix. This would make the algorithm practically useless for real-time estimation.

Since actually $\hat{\mathbf{R}}_x^{-1}(n)$ is desired rather than $\hat{\mathbf{R}}_x(n)$, the *matrix inversion lemma* (not proved here) is used. This result reduces computational complexity by providing a recursive update for the inverse of $\hat{\mathbf{R}}_x(n)$. For notational convenience, let

$$\mathbf{P}(n) \equiv \hat{\mathbf{R}}_x^{-1}(n) . \tag{4.55}$$

Applying the matrix inversion lemma to (4.53) results in the recursive update

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mu(n)\mathbf{P}(n-1)\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{P}(n-1) \tag{4.56}$$

$$\mu(n) = \frac{1}{\lambda + \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}(n)} . \tag{4.57}$$

Note that $\mu(n)$ is a positive scalar and can be thought of as a variable step size. Note that (4.56) contains no matrix inversions.

Using (4.56) and (4.54) with (4.52) to solve for $\hat{\mathbf{h}}(n)$, one can show that

$$\hat{\mathbf{h}}(n) = \hat{\mathbf{h}}(n-1) + \mu(n)\mathbf{P}(n-1)[e(n)\mathbf{x}(n)] \tag{4.58}$$

$$e(n) = d(n) - \hat{\mathbf{h}}^T(n-1)\mathbf{x}(n) \tag{4.59}$$

where $e(n)$ is the prediction error based on the old filter estimate. Together, equations (4.56), (4.57), (4.58), and (4.59) comprise the ***Recursive Least Squares (RLS)*** algorithm. Notice that (4.58) is of the same form as (4.49), except that the step size is time-varying and the derivation is exact, rather than using an instantaneous estimate of the gradient.

4.2.3.2. Stability and Convergence

The least squares estimate of the filter taps $\hat{\mathbf{h}}(n)$ possesses some important properties that relate it back to the optimum Wiener filter of Section 4.2.1.4. Setting $\lambda = 1$ for the moment so that no data is "forgotten", if the input signal $\mathbf{x}(n)$ and the desired response $d(n)$ are *jointly stationary ergodic processes*, then the least squares estimate $\hat{\mathbf{h}}(n)$ approaches the optimum Wiener filter $\mathbf{h}^*$ as n goes to infinity. In other words, the estimate is *consistent*. Furthermore, if the error signal $e(n)$ has zero mean, then $\hat{\mathbf{h}}(n)$ is an ***unbiased*** estimate [12]. Loosely, a process being ergodic means that it is asymptotically uncorrelated, so that two samples taken at distant lags become more and more uncorrelated.

In a non-stationary environment, the factor λ must be set to a value less than one so that the importance of old data gradually diminishes. By doing so, the algorithm attains the capability to track statistical variations in the environment in which it operates (as long as the variations are slow with respect to the convergence time of the algorithm). However, the use of $\lambda < 1$ changes the behavior of RLS drastically; the estimate of the filter taps $\hat{\mathbf{h}}(n)$ becomes no longer consistent because the memory of the algorithm becomes finite. In general, a fast adaptation must be traded off for a more noisy adaptive process.

4.2.4. Summary

When evaluating a particular adaptive algorithm, various factors need to be taken into account, for example: rate of convergence, accuracy, computational requirements, and numerical stability.

- The RLS algorithm takes a more direct path to the solution, since it uses an estimate of $\mathbf{R}_x^{-1}$. This makes it independent of the eigenvalue spread.

- The number of iterations to convergence using the RLS algorithm is about an order of magnitude less than for the LMS algorithm.
- The computational complexity of the RLS algorithm increases as the square of the filter order, whereas for the LMS algorithm, the complexity increases linearly with the filter order.

The major advantage of RLS over LMS lies in faster convergence and reduced sensitivity to eigenvalue spread for stationary inputs. On the other hand, the RLS algorithm retains less advantage over LMS in low eigenvalue spread situations, in cases where the *signal-to-noise-ratio (SNR)* is low, and in tracking non-stationary data.

The RLS algorithm converges in a mean square sense in about $2M$ iterations, where M is the number of taps. This means that the rate of convergence for RLS is typically an order of magnitude faster than for LMS [11]. Also, the RLS algorithm, in theory, converges to the exact optimum filter when operating in a stationary environment (with $\lambda = 1$), whereas the LMS algorithm necessary involves some residual noise power due to its use of an instantaneous gradient estimate.

Some of the disadvantages of the RLS algorithm are its much higher computational requirements and implementational complexity. Although "fast" RLS algorithms exist that reduce the required computing power, the numerical sensitivity of these algorithms remains problematic. In addition, the superiority of RLS over LMS is often lost with non-stationary data. Even with exponential weighting for tracking, it is unclear how to choose the weighting factor. Additionally, the exponential weighting tends to increase numerical problems in the algorithm.

4.3. Adaptive Algorithms

In this section, details about the adaptive algorithms used in this paper are given. For most of this work, the Recursive Least Squares (RLS) algorithm is preferred over the Least Mean Square (LMS) algorithm due to its faster convergence rate and reduced sensitivity to eigenvalue spread. However, the performance disparity between the two algorithms is not always so great, and in some cases the LMS algorithm may be advantageous (most notably in non-stationary environments). Although the computational requirements of RLS is greater than that of LMS, this is not really a concern here, since the algorithm is

not being implemented directly in hardware. The use of modern computers is assumed, so that computation speed is plenty adequate. Moreover, the models that are used are usually of fairly small order.

Section 4.3.1 and Section 4.3.2 outline the steps of the RLS and LMS algorithms, respectively. The following two sections, Section 4.3.3 and Section 4.3.4, touch only briefly on the issues of adaptive IIR filtering and numerical stability; these issues are important, but an in-depth discussion of them would be beyond the scope of this work.

4.3.1. Recursive Least Squares (RLS)

The RLS algorithm (Section 4.2.3.1) requires initialization before the recursions can begin. In particular, starting values for the filter tap vector $\hat{\mathbf{h}}(n)$ and the inverse of the correlation matrix $\mathbf{P}(n)$ are needed. For $\hat{\mathbf{h}}(n)$, unless a better starting point is known, setting the initial filter taps to all zeroes is customary:

$$\hat{\mathbf{h}}(-1) = \mathbf{0}. \tag{4.60}$$

For $\mathbf{P}(n)$, if some prior data were available, an estimate based on the data could be precomputed and used for the initial value $\mathbf{P}(-1)$. Otherwise, if no prior knowledge is available, the matrix can be initialized with

$$\mathbf{P}(-1) = \delta^{-1}\mathbf{I} \tag{4.61}$$

where δ is a small positive constant. The initialization procedure consisting of (4.60) and (4.61) is referred to as *soft-constrained initialization*, with δ being the only parameter. The recommended choice of δ is that it should be small compared to $0.01\sigma_x^2$, where σ_x^2 is the variance of the input signal $x(n)$ [11]. The effect of initializing with (4.61) is to introduce a *bias* into the estimate of the filter taps $\hat{\mathbf{h}}(n)$. However, as the number of iterations n becomes large (the amount of observed data increases), the bias diminishes to zero, so that the estimate of $\hat{\mathbf{h}}(n)$ is *asymptotically unbiased*. Therefore, for long data lengths, the exact value of δ is unimportant.

After the algorithm has been initialized, the steps of the recursion can proceed as shown in Figure 4.5.

For each iteration $n = 0, 1, \ldots$, compute

1. $\mathbf{s}(n) = \mathbf{x}^T(n)\mathbf{P}(n-1)$
2. $\mathbf{k}(n) = [\mathbf{P}(n-1)\mathbf{x}(n)]/[\lambda + \mathbf{s}(n)\mathbf{x}(n)]$
3. $e(n) = d(n) - \hat{\mathbf{h}}^T(n-1)\mathbf{x}(n)$
4. $\hat{\mathbf{h}}(n) = \hat{\mathbf{h}}(n-1) + \mathbf{k}(n)e(n)$
5. $\mathbf{P}(n) = \frac{1}{\lambda}[\mathbf{P}(n-1) - \mathbf{k}(n)\mathbf{s}(n)]$

Figure 4.5.
The RLS algorithm [11].

4.3.2. Normalized LMS

The LMS algorithm (Section 4.2.2.2) also requires initialization of its filter tap vector $\hat{\mathbf{h}}(n)$. Setting it to zero is convenient, unless some prior knowledge results in a better initial guess:

$$\hat{\mathbf{h}}(-1) = \mathbf{0}. \tag{4.62}$$

Figure 4.6 shows the steps in the *normalized* LMS algorithm, which is a variation on the standard LMS algorithm. In the standard algorithm, a constant step size is used, but in the normalized version, the step size $\mu(n)$ is allowed to vary inversely with the squared Euclidean norm of the input vector $\mathbf{x}(n)$. The normalized step size helps to dampen the problem of *gradient noise amplification*, which can occur when noisy data causes the algorithm's estimate of the gradient to be especially bad.

For each iteration $n = 0, 1, \ldots$, compute

1. $e(n) = d(n) - \hat{\mathbf{h}}^T(n-1)\mathbf{x}(n)$
2. $\mu(n) = \tilde{\mu}/[a + \|\mathbf{x}(n)\|^2]$
3. $\hat{\mathbf{h}}(n) = \hat{\mathbf{h}}(n-1) + \mu(n)e(n)\mathbf{x}(n)$

Figure 4.6.
The normalized LMS algorithm [11].

For the algorithm to be convergent in a mean square sense, the dimensionless parameter $\tilde{\mu}$ should satisfy the following inequality (compare to (4.46)):

$$0 < \tilde{\mu} < 2. \tag{4.63}$$

The positive parameter a is included to prevent division by a small value in the event $\mathbf{x}(n)$ becomes close to zero. Setting $a = 0$ and fixing the step size $\mu(n)$ (by skipping the second step in Figure 4.6) reduces the algorithm to the standard LMS.

4.3.3. Adaptive IIR Filtering

An adaptive IIR filter can provide significantly better performance than an adaptive FIR filter having the same number of filter taps; alternatively, for a given level of performance, an IIR filter generally requires fewer taps than the corresponding FIR filter, making it more computationally efficient. These advantages are due to the feedback paths in an IIR filter which allows it to generate an infinite-duration impulse response with only a finite number of coefficients. The actual benefit of using an IIR filter depends on the signal to be modeled; for instance, if a signal is well-modeled by a few moving average terms, the benefit can be large.

However, the potential gains to be realized by adaptive IIR filters are offset by augmented problems with stability and convergence. Unlike FIR filters, IIR filters have the possibility of becoming unstable, meaning that the filter coefficients could grow without bound. Also unlike FIR filters, the error performance surface of an IIR filter is not guaranteed to have a unique minimum. Thus, even if the algorithm converges, the convergence may be at a local rather than a global minimum.

The poles of an IIR filter's transfer function can be located at positions other than at the origin of the z-plane[3]; instability problems arise when one or more of these poles migrate onto or outside the unit circle of the z-plane and remain there for a significant length of time. This occurrence is not uncommon, especially if the application requires that the poles be near the unit circle. Several ad hoc schemes exist for combatting this problem: one possible course of action is to ignore updates that move the filter's poles too close to the unit circle, or to reduce the step size. Another possibility is to use an exponential weighting factor to push unstable poles towards the origin [7].

3. When referring to the z-transform of the filter at a particular instant of time, the coefficients must be assumed to be fixed.

One can see already that the properties of an adaptive IIR filter are considerably more complex than those of an adaptive FIR filter. In fact, relatively few analytical results are known regarding their behavior because of the inherent non-linearities. This paper will not discuss the subject much further, although future work in this area is encouraged. For an overview on adaptive IIR filtering, see [28] or [32].

As a final note, an adaptive filter can be made IIR simply by feeding back present or past filter outputs (signal predictions) as extra inputs to the filter; this algorithm is often referred to as *pseudolinear regression* and can be derived as a steepest descent method using an approximate gradient [8][28]. The resulting output of this filter is no longer linear, so that the error performance surface is no longer quadratic and can have multiple local minima. Moreover, the algorithm is not even guaranteed to converge to a local minimum; depending on the input, it may approach a biased solution [13][36].

However, despite these drawbacks, the algorithm has been shown to be useful in practice [36][8]. On the positive side, stability monitoring is not required for this algorithm; it has a self-stabilizing feature whereby unstable poles have a tendency to migrate back into the stable region [28]. Also, the problem of the algorithm not converging to a minimum (local or global) can sometimes be solved by differencing the signal [28].

4.3.4. Numerical Stability

In the digital implementation of an adaptive filtering algorithm, one must be aware of numerical stability problems due to the finite precision of the system. Essentially two sources of error exist: finite-precision arithmetic errors and quantization errors incurred during analog-to-digital conversion. If these errors accumulate without bound, they can lead to instability.

Any adaptive algorithm is vulnerable to numerical stability problems, but an important one to be aware of is "explosive divergence" in the RLS algorithm. This problem occurs when the **P** matrix (Section 4.2.3.1) loses its property of positive definiteness, causing filter taps to increase without bound. See [11] for a simple cure, and also for more numerically robust ways of implementing RLS, including a procedure based on the QR

decomposition based recursive least squares (QRD-RLS) algorithm and other procedures based on alternative filter structures.

Any further discussion of the numerical properties of adaptive algorithms would be straying from the scope of this paper. However, as in Section 4.3.3, future work in this area is encouraged.

4.4. Implementation

In this section, a number of issues will be discussed regarding the implementation of adaptive algorithms for the purpose of real-time SPC. The first sub-section explains how to initialize and configure the adaptive algorithms described in Section 4.3; the most important issue, discussed in Section 4.4.1, is the fundamental tradeoff between estimation noise and tracking ability when choosing algorithm parameters. Section 4.4.2 describes a scheme for combining adaptive algorithms. Section 4.4.3 introduces multivariate modeling and how it can be used.

4.4.1. Algorithm parameters

4.4.1.1. Exponential Weighting Factor in RLS

The exponential weighting factor λ allows the RLS algorithm to track slow (compared to the algorithm's convergence time) statistical variations in a non-stationary environment by weighting recent data more heavily than past data (Section 4.2.3.1). The influence of past data becomes less and less prominent until eventually its contribution to the adaptive filter becomes insignificant. More precisely, the λ factor applies a geometric weighting to each input sample, so that the quantity

$$\frac{1}{1-\lambda} \tag{4.64}$$

is a rough measure of the "memory" of the algorithm, i.e., the number of past samples that the algorithm uses in its estimation of the filter taps.

If the adaptive filter is to be used in a stationary environment, the special case of $\lambda = 1$ should be applied, since this corresponds to infinite memory, i.e., the entire history of data is utilized. Otherwise, the positive constant λ should be less than one, so the algo-

rithm can react to changes quicker; however, the value should be kept above 0.95 to prevent instability.

As noted in Section 4.2.3.2, with a λ value less than one, the RLS algorithm's estimates are no longer consistent. In other words, its outputs become stochastic, and some excess error variance will always exist. A smaller value of λ enhances the RLS algorithm's ability to track non-stationary signals by increasing the speed at which young data is incorporated into the algorithm. However, it also causes more noise to appear in the filter taps and output, thus increasing the steady-state mean square error (also known as *misadjustment*).

The exact choice of λ should depend on the time scale at which one wishes to apply SPC. For example, to detect equipment faults at the real-time level, the memory of the algorithm should be approximately set to the number of input samples available in a few wafers. Assuming 50 samples are available per wafer, then using (4.64) as a rule of thumb, a value for λ of about 0.99 would be appropriate. Similarly, to detect faults at the wafer-wafer level (using real-time data), the algorithm memory should be set to the number of input samples in an entire lot; if one lot consists of about 20 wafers, then λ should be around 0.999.

4.4.1.2. Step Size in LMS

A similar tradeoff between estimation noise and tracking ability also exists in the choice of the step size μ for the LMS algorithm. Recall that μ contributes directly to the time constant (4.38) for the convergence of each mode of the algorithm. A larger step size accelerates the rate of adaptation, but at the expense of an increase in the average excess mean squared error during steady-state. On the other hand, a smaller value of μ results in a slower adaptation, but decreases the excess mean squared error after adaptation. Typically, values of μ on the order of a tenth of the upper bound given in (4.46) are used [37].

4.4.1.3. Model order

As mentioned in Section 4.2.4, the RLS algorithm converges (in a mean square sense) in about $2M$ iterations, where M is the order of the model, i.e., the number of filter taps. In a stationary environment with the weighting factor λ set to 1.0, the algorithm will

eventually converge to the optimum filter, regardless of M. However, in a non-stationary environment, no such guarantee can be made; in fact increasing the model order will increase estimation noise in the adaptation process.

For the LMS algorithm, the upper bound in (4.46) for the step size μ is inversely proportional to the model order M. Thus, increasing the model order requires μ to be made smaller. Although the estimation noise normally decreases with the reduced step size, the effect is cancelled out, since the noise also increases with M; so in this case, the reduced step size does not result in an attendant reduction in the overall estimation noise.

The actual choice of the model order will depend on the characteristics of the input signal and the amount of data available. Suppose that the input signal is stationary and its statistics are known. Recall that FIR filters perform best when the input is an auto-regressive signal. Therefore, if the input signal were known to be well-characterized by a third order auto-regressive model, then M should be set to three. Theoretically, larger values of M would not result in any degradation of performance. However, if the amount of data available to the adaptive filter is small, then one might wish to reduce the model order in order to increase the rate of convergence.

Of course, usually the statistics of the input signal will not be known, or the signal may not be stationary. Then the adaptive filter should be configured to track statistical variations in the signal; in other words, λ should be decreased for the RLS algorithm, or the step size μ should be increased for the LMS algorithm. Either of these actions increases noise in the adaptation process, resulting in a higher steady-state error variance.

In the application of SPC, the amount of data available will most likely be relatively limited and the statistics of the input signal will be unknown or the signal will be non-stationary. Both of these qualities suggest that smaller model orders will perform better. In practice, different filter configurations can be tested by running computer simulations to determine the best overall algorithm parameters. This author has found that model orders less than five seem to achieve good results, at least when operating on real-time plasma sensor measurements.

4.4.2. Combining Adaptive Algorithms

Thus far, the RLS and LMS algorithms have been compared and contrasted, but another possibility is to use them *together* to filter a signal. For example, to take advantage of the faster convergence rate of RLS and the good tracking abilities of LMS, one might first employ the RLS algorithm and then switch over to the LMS algorithm. The way this would work is as follows: in the "adapting" mode, the RLS algorithm is running with $\lambda = 1$ so as to converge rapidly to an approximate solution for the filter taps; then in the "tracking" mode, the taps are transferred over to the LMS algorithm, which runs in place of RLS. In this way, a feasible solution is found quickly, and then the good tracking ability of LMS is applied to follow slow statistical variations. Another important attribute of this scheme is that data collected during the "adapting" mode can be used to help choose a good step size μ for the "tracking" mode.

4.4.3. Multivariate Modeling

For improved modeling performance, correlations between different signals can be modeled to produce a *multivariate* time-series model. The difference between a univariate and multivariate time-series model is simply that multivariate models express a variable as a function of past values of that variable, as well as past values of other variables. This has the advantage of adding extra explanatory variables to a model, perhaps for the purpose of adding redundancy. As an example, the following equation shows a multivariate model for the variable u, where the next value of u is a linear function of past values of both u and v:

$$u(n+1) = \alpha_0 u(n) + \beta_0 v(n) + \alpha_1 u(n-1) + \beta_1 v(n-1) + \ldots \quad (4.65)$$

Note, however, that using a multivariate model quickly increases the model order. For instance, if a variable is modeled by three past values of itself and two other explanatory variables, then the overall model order becomes nine. This has the effect of increasing the convergence time and increasing estimation noise, as discussed in Section 4.4.1.3.

4.5. Example

4.5.1. Experiment

The data for this example was taken from the same experiment as in Section 3.3.1. Refer there for details on the experimental design and data collection.

4.5.2. Procedure

A software program was written in C++ [30][31] to implement the RLS and LMS algorithms; see Appendix C for a code listing. The program is configured with the necessary parameters: number of feedforward taps, number of feedback taps, λ or μ, number of signals, number of data points per wafer, and initial **P** matrix. Also, the option to use multivariate modeling can be turned on or off, either the RLS or LMS algorithm can be used, and different input data formats can be selected. Real-time data are read into the program one wafer at a time and filtered through the RLS algorithm. Relevant output data are saved to files (the user can specify the filenames), including the prediction errors (residuals), predicted outputs, filter tap values, and estimated variances.

If a wafer-wafer prediction model has been created (see Section 3.3.2 for the procedure), the predictions for all the wafers should be written to a file, with each line of the file being the predicted signal means for one wafer. Before the software reads in the real-time data for a wafer, it will first read in the wafer-wafer prediction values. These values will be used to center the real-time data for that wafer. If no wafer-wafer predictions are available, then the program centers the real-time signals for each wafer by simply subtracting the average values over that wafer; in this case, the algorithm is not operating in real-time, because the average values are not known until after each wafer has finished processing.

4.5.3. Results

In this example, the RLS algorithm at the real-time level is combined with a prediction model at the wafer-wafer level (see Section 4.1.2). A plot of the actual signal for "RF_match_#1_load_coil_position" alongside the predictions of the combined wafer-wafer and real-time model (for the main etch step) is shown in Figure 4.7.[4] The wafer-wafer prediction model was described in Section 3.3. The real-time adaptive model used an order of five and an exponential weighting factor of $\lambda = 0.99$.

4. Only 35 wafers are shown in the figure, rather than 38, since wafers #14, 27, and 38 were removed from the analysis due to processing or data collection problems.

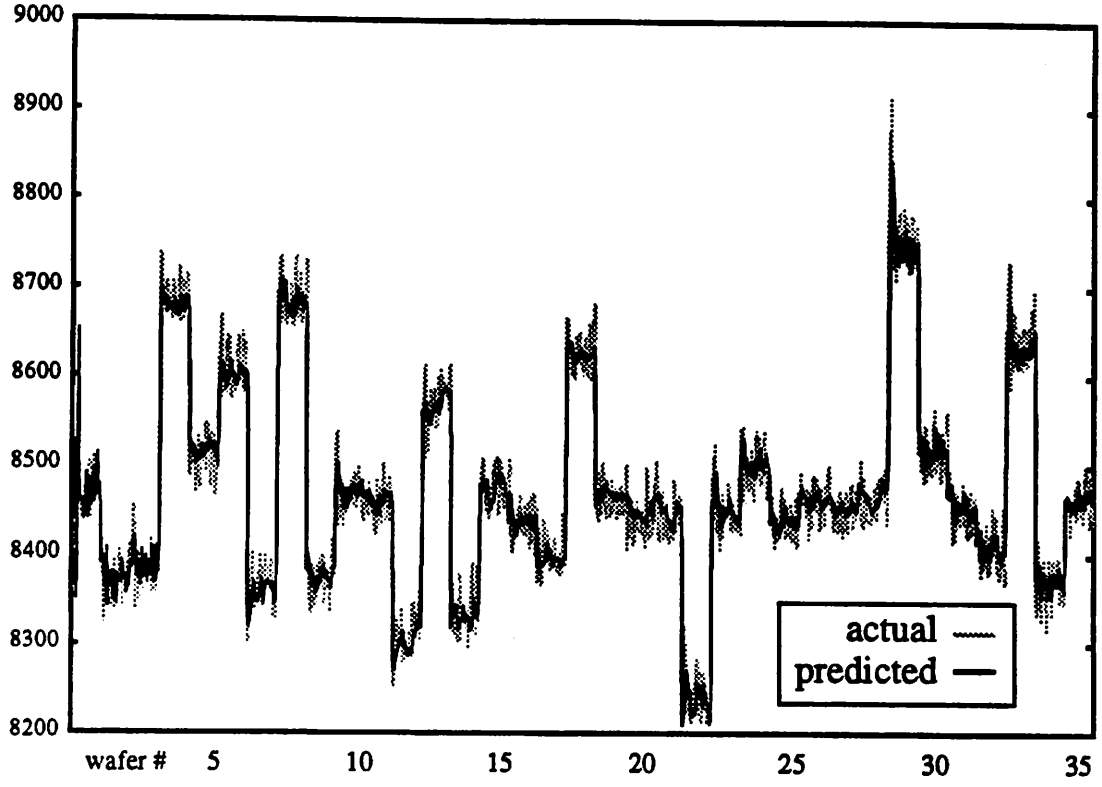

Figure 4.7.
Predicted and actual signal for "RF_match_#1_load_coil_position".

The residuals (actual minus predicted values) are displayed in Figure 4.8.[5] These can be passed to an SPC scheme for fault detection. The residuals look "in control" except for those at wafer #1 and wafer #29. The large residuals at wafer #1 are due to transients in the convergence of the adaptive algorithm; these always occur during the first few iterations of the algorithm and can be ignored. The large residuals at wafer #29 are probably due to an abnormally low power setting for that wafer.[6] Note that occasionally the residuals show *runs*, i.e., consecutive values whose signs are identical. This is because prediction errors from the wafer-wafer model cause the real-time signal to be centered incorrectly; when the error is large, the real-time adaptive filter is not able to completely compensate.

5. Note that the vertical scale in this figure is smaller than in the previous figure.

6. See wafer #31 of Table 3.2. (The actual wafer number is 31, rather than 29, since wafers #14 and 27 are not included in the figure.)

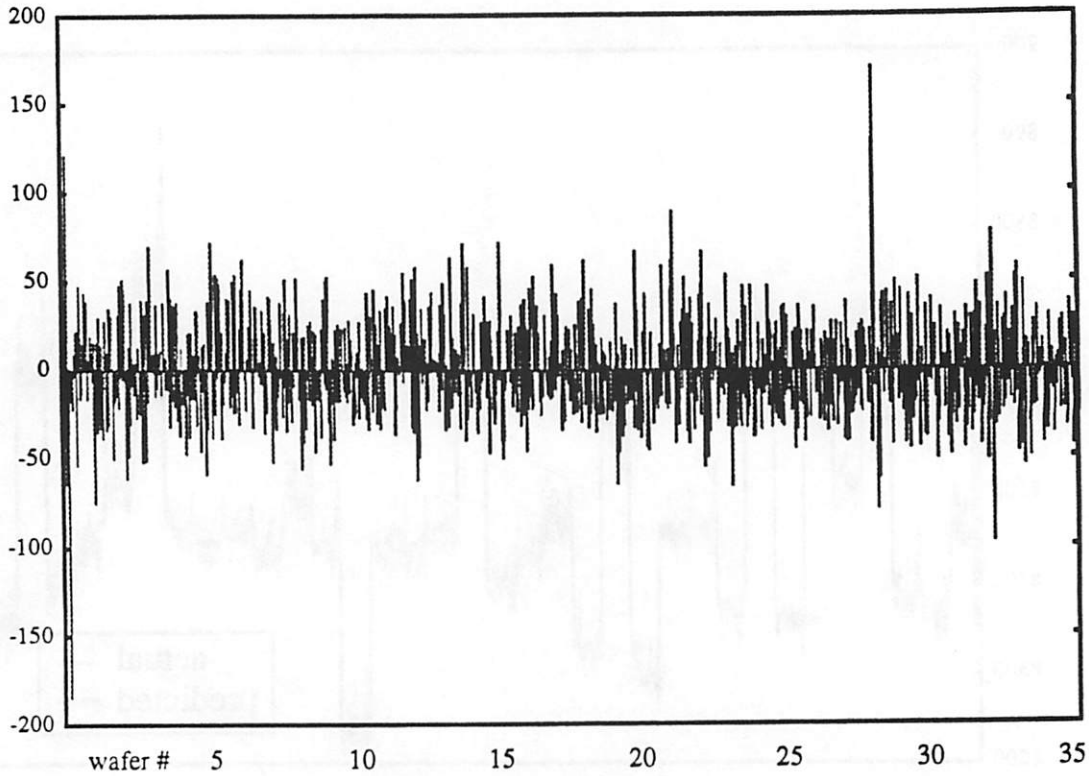

Figure 4.8.
Residuals for "RF_match_#1_load_coil_position".

Chapter 5 Experimental Analysis

5.1. Introduction

In this chapter, the predictive modeling techniques of Chapter 3 and the adaptive modeling techniques of Chapter 4 are applied to the statistical process control (SPC) methodology described in Chapter 2. The analysis will use sensor data collected from experiments conducted at Texas Instruments (TI) in Dallas for the Sematech J-88-E project.

5.2. Adaptive Calculation of the T^2 Statistic

This section shows how the T^2 statistics can be calculated adaptively; this is required to effectively apply adaptive models to the real-time SPC methodology described in Section 2.2.2. In Section 5.2.1, the method for estimating the error covariance matrix is presented. Section 5.2.2 uses the estimate to calculate T^2 values that are normalized so that any value greater than one denotes an equipment alarm. The implementation of these calculations is described in Section 5.2.3.

5.2.1. Estimation of the Error Covariance Matrix

In order to calculate the T^2 statistic, an estimate of the error (residual) covariance matrix $\hat{\mathbf{S}}$ (see Section 2.2.2.2) is needed. In non-stationary environments, the estimate should have a finite memory, so that signals can be tracked. A general methodology is to use a moving window to collect the data, followed by an estimation of the covariance matrix using the windowed data. For example, in a simple moving average, the window has a length N and constant height $1/N$.

A computationally simpler way is to use an exponential weighting factor λ, similar to the factor used in the RLS algorithm (Section 4.2.3.1):

$$\tilde{\mathbf{S}}(k) = \sum_{i=0}^{k} \lambda^i \bar{\mathbf{e}}(k-i)\bar{\mathbf{e}}^T(k-i), \tag{5.1}$$

where $\tilde{\mathbf{S}}$ is an intermediate matrix to be used in the calculation of $\hat{\mathbf{S}}$, and $\bar{\mathbf{e}}$ is a (column) vector of signal residuals averaged over a specified group size. The following recursive update can be used in place of (5.1) in order to reduce the computational expense:

$$\tilde{\mathbf{S}}(k) = \lambda\tilde{\mathbf{S}}(k-1) + \bar{\mathbf{e}}(k)\bar{\mathbf{e}}^T(k). \tag{5.2}$$

Note that (5.1) and (5.2) are analogous to (4.50) and (4.53), respectively. Since calculation of the T^2 statistic actually requires the inverse of $\hat{\mathbf{S}}$, rather than $\hat{\mathbf{S}}$ itself, the recursive techniques of Section 4.2.3.1 can be directly applied (see (4.56) and (4.57)). The recursion is also initialized in a manner identical to what is done in Section 4.3.1.

The value of $\hat{\mathbf{S}}$ is just $\tilde{\mathbf{S}}$ multiplied by a scale factor. For the case $\lambda = 1$, the scale factor is $1/(k+1)$, where k is the iteration count; this case results in a consistent estimate of the covariance matrix:

$$\hat{\mathbf{S}}(k) = \frac{1}{k+1}\tilde{\mathbf{S}}(k). \tag{5.3}$$

For the case $\lambda < 1$, the scale factor is $(1-\lambda)$:

$$\hat{\mathbf{S}}(k) = (1-\lambda)\tilde{\mathbf{S}}(k). \tag{5.4}$$

5.2.2. Normalized T^2 Statistic

Once an estimate of the error covariance matrix is available, the T^2 statistic can be readily calculated with

$$T^2 = n\bar{\mathbf{e}}^T\hat{\mathbf{S}}^{-1}\bar{\mathbf{e}}, \tag{5.5}$$

where n is the group size, and $\bar{\mathbf{e}}$ is the (column) vector of signal residuals averaged over the group. The control limits are determined by looking up the chi-squared distribution value with number of degrees of freedom equal to the number of signals used in the statistic (see Section 2.2.2.2). Each calculated T^2 value is then divided by the control limit to produce a "normalized" T^2 statistic. The normalized values have the convenient property that values greater than one are alarms, and values less than (or equal) to one are in-control.

5.2.3. Implementation

A computer program was written in C++ to implement the above algorithm. The inputs to the program are the signal residuals, and the outputs are the normalized T^2 values. Parameters to the program, which can be set by the user, are the exponential weighting factor λ, the group size n, and also the value δ, which is used to initialize the recursion (see Section 4.3.1); the false alarm probability (Type I error) is set at 1%.

The covariance matrix estimate $\hat{\mathbf{S}}$ can be quite poor when the algorithm is first started. For $\lambda = 1$, this can result in missed alarms for the first several iterations. For $\lambda < 1$, this can result in false alarms for the first several iterations; a larger value of δ can help alleviate this problem.

The group size n can be increased to dampen the effect of noisy estimates. This is often necessary when analyzing real-time data, especially if a predictive wafer-wafer model is used with real-time data: the wafer-wafer predictions tend not to be very accurate, causing false alarms at the points where recipe changes occur. For wafer-wafer or lot-lot signals, good results can usually be produced with the group size set to one.

5.3. Experiment 1: Wafer-Wafer Data

In this experiment, the adaptive techniques for estimating the T^2 statistic, as detailed in Section 5.2, will be applied to some actual data. Data were collected from a series of four lots of wafers that were all processed on a Lam TCP 9600 metal etcher (the etchant gases are BCl_3 and Cl_2) using TI's "Recipe 44", shown in Table 5.1. The lots consisted of a total of 62 wafers run on October 24, 25, and 26, 1995. Note that these wafers were not necessarily processed as 62 consecutive wafers. An arbitrary number of wafers could have been processed between any two of the lots in the experiment. See Table 5.2 for a summary of the experiment.

top power (Watt)	bot power (Watt)	pressure (mTorr)	Cl_2 (sccm)	BCl_3 (sccm)
350	132	12	75	75

Table 5.1.
Recipe 44.

Date (1995)	Begin time	End time	Lot ID	# of wafers	Wafer #'s	Notes
10/24	17:32	19:41	5660	23	1 to 23	Recipe 44
10/24	20:12	20:36	6215	6	24 to 29	Recipe 44
10/25	02:30	04:38	6076	23	30 to 52	Recipe 44[a]
10/26	09:55	10:43	1111	10	53 to 62	Recipe 44[b]

Table 5.2.
Summary of Experiment 1.

a. The first half of Lot 6076 was run with different TiN film thicknesses than the second half (unfortunately, experimental records do not indicate at which wafer in the lot the thicknesses changed).

b. The first wafer of Lot 1111 was a dummy oxide wafer. The fifth, sixth, and seventh wafers have induced faults.

In the fourth (last) lot, the recipes were altered on three of the wafers in order to simulate actual machine faults. The recipes used for each wafer in Lot 1111 are described in Table 5.3. Wafers 5, 6, and 7 are the wafers that contain induced faults. Note that wafer 1 was a dummy oxide wafer (the rest are production wafers).

Wafer ID	Recipe
1	Recipe 44 (dummy oxide wafer)
2	Recipe 44
3	Recipe 44
4	Recipe 44
5	TCP power +10% = 385 W
6	BCl_3 flow -10% = 67.5 sccm
7	Bottom RF power +20% = 158 W
8	Recipe 44
9	Recipe 44
10	Recipe 44

Table 5.3.
Recipes used for each wafer of Lot 1111.

5.3.1. Procedure

The real-time data for each wafer were averaged to produce wafer-wafer data for the main etch step (step 4); averaging ignored the first ten samples of each wafer. All the wafer-wafer data were concatenated into one file and adaptively filtered using the RLS

algorithm with an order of three and $\lambda = 0.99$. The ten SECS-II signals listed in Table 5.4 were used in the analysis.

LamStation Signals
RF_gen_#1_forward_power
Endpoint_detector_a
Chamber_pressure
RF_match_#1_tuning_position
RF_match_#1_load_coil_position
RF_match_#1_peak_RF_voltage
TCP_Match_Tune_Cap_Position
RF_Gen_#3_TCP_FWD_PWR
TCP_Match_Load_Cap_Position
AC2_valve_angle

Table 5.4.
Signals used in the analysis of Experiment 1.

The residuals produced by the adaptive filtering were used to calculate a normalized T^2 statistic for each wafer by the methodology of Section 5.2. All ten signals were included in the statistic. The exponential weighting factor was set to $\lambda = 1$, and the group size was set to $n = 1$.

5.3.2. Results

Figure 5.1 shows the normalized T^2 values for each wafer. The dotted line at the vertical value of one is the control limit (recall that the T^2 values have been divided by the control limit, so that a value greater than one denotes an alarm). Note that the values for about the first ten data points are not very accurate due to the small number of data on which to base the estimates; the estimates improve as more data becomes available.

Alarms are evident at wafers #30, 42, and for most of wafers #53 to 62. Examination of Table 5.2 reveals that the first two lots were processed consecutively without intervening wafers (since little time passed between wafers #23 and 24). On the other hand, a gap of time (of about six hours) existed between wafer #29 (the last wafer of the second lot) and wafer #30 (the first wafer of the third lot); this means that other wafers were processed

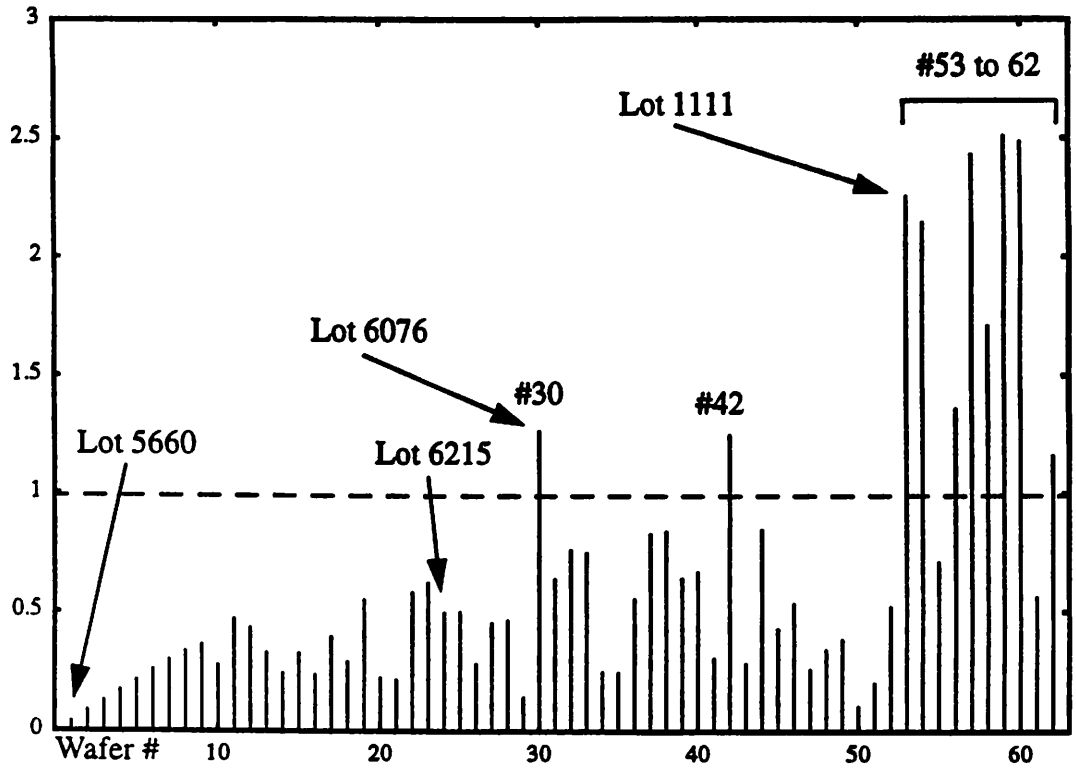

Figure 5.1.
Normalized T^2 values.

between the second and third lots, probably with recipes different from Recipe 44. These intervening wafers, or the gap in time, may have shifted the equipment's operating point slightly. This is the most likely reason for an alarm at wafer #30 (the first wafer of the third lot), but none at wafer #24 (the first wafer of the second lot). This seems to suggest that alarms occurring at the first wafer of a lot can be ignored, especially if a large span of time exists between the processing of that wafer and the previous wafer.

As mentioned in the footnotes to Table 5.2, the film thicknesses on the wafers of the third lot (Lot 6076) changed sometime in the middle of the lot. Unfortunately, the specific wafer number was not recorded in the experimental records. However, the alarm at wafer #42 is good evidence that the film thickness changed on the thirteenth wafer of the lot (wafer #30 is the first wafer of the lot).

A large gap of time (of over 29 hours) also existed between the third and fourth lots of the experiment. This explains the large alarm at wafer #53 (in addition to the fact that

wafer #53 was a dummy oxide wafer). All three of the induced fault wafers (wafers #57, 58, and 59) exhibit alarms as well. However, wafers #54, 56, 60, and 62 apparently cause false alarms. This may be an indication of "memory" in the equipment, whereby the processing of one wafer can have some influence on the processing of the next wafer in the batch.

5.4. Experiment 2: Real-Time Data

The purpose of this experiment is to demonstrate the advantages that adaptive models have over static models. The experiment uses a set of data in which the statistics of the signals vary over the length of the data. The results of the data analysis will show that adaptive models are able to track the variations, whereas static models become useless once the statistics of the signals have changed.

Data were collected from a single lot of wafers processed on a Lam TCP 9600 metal etcher using TI's "Recipe 44", shown in Table 5.1. The lot consisted of 23 consecutive wafers run on September 26 and 27, 1995. The three SECS-II signals listed in Table 5.5 were used in the analysis.

LamStation Signals	Abbreviation
TCP_Match_Tune_Cap_Position	tcpTune
RF_Gen_#3_TCP_FWD_PWR	tcpPwr
TCP_Match_Load_Cap_Position	tcpLoad

Table 5.5.
Signals used in the analysis of Experiment 2.

5.4.1. Procedure

The analysis used real-time data from the main etch step (step 4) of each wafer. The first fifteen samples of the main etch step were skipped from each wafer in order to ignore transient responses; the next 30 samples were kept for analysis. All the real-time data were then demeaned within each wafer (by subtracting the wafer average) and concatenated into a single stream. Figure 5.2 shows plots of each of the signals listed in Table 5.5. Notice the time-varying nature of the signals; in particular, each signal's variance exhibits fluctuations over the length of the plot (even though the data are all from a single lot of wafers).

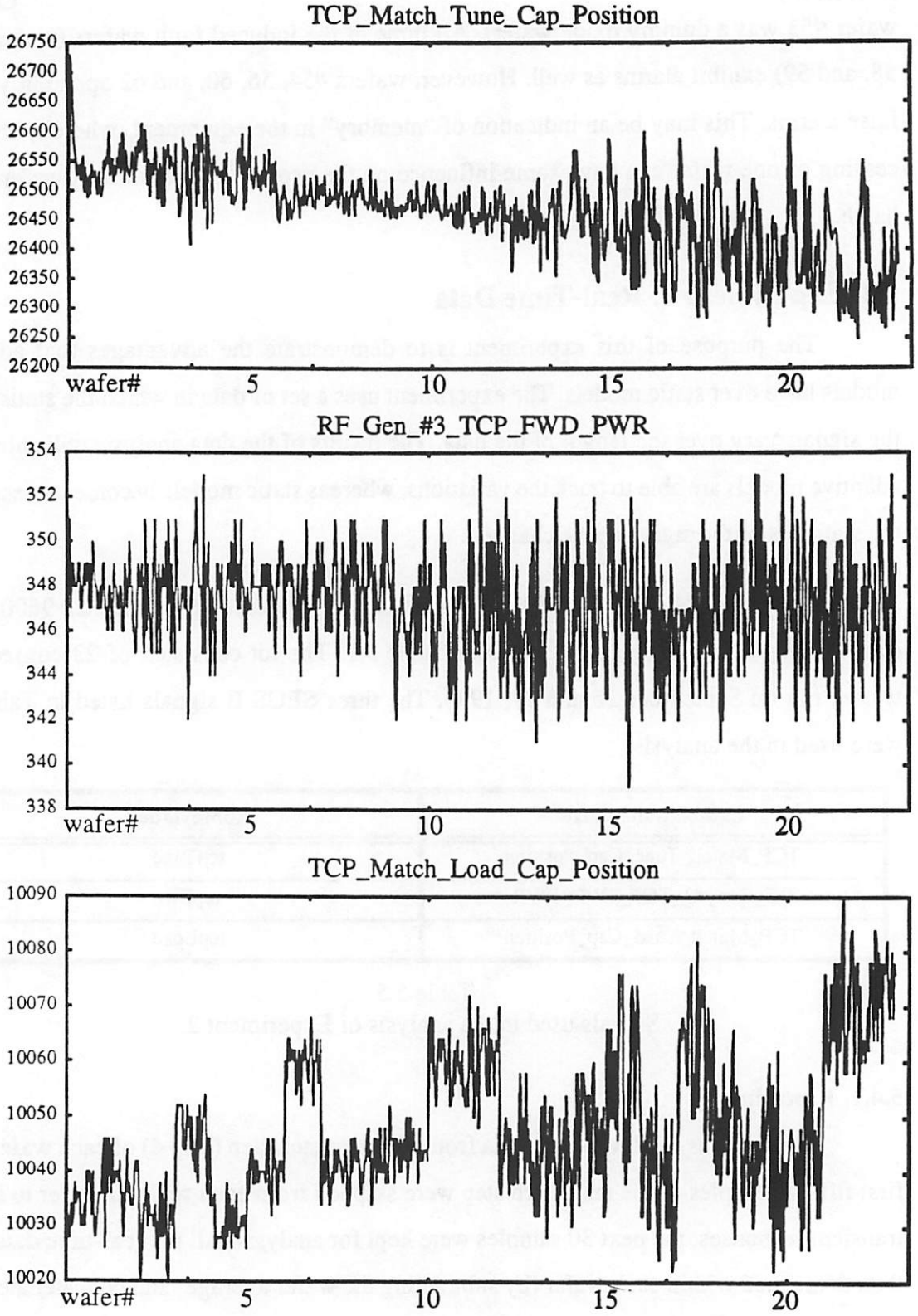

Figure 5.2.
Real-time signals in Experiment 2.

Both static models and adaptive models will be applied to this set of data in order to form a comparison between the two. The procedure is as follows: first, suppose that one wishes to create static models for this data in order to monitor the equipment with a control chart. A set of baseline wafers must be chosen for the modeling; suppose that the first half of the data set (containing data for half a wafer lot) is chosen for the baseline and that the models created from this baseline are used to monitor subsequent wafers. Next, the same set of data are monitored using adaptive models. The adaptive models do not require a baseline model, although they do require some time for the models to converge to an approximate solution.

5.4.2. Results

The software program RTSPC[1] was used to generate ARIMA models[2] for each of the signals based on the first half of the data set (the first 11 of the 23 wafers). Figure 5.3 shows the residuals of the models[3], along with the "three-sigma" control limits ("three-sigma" refers to three times the standard deviation of the residuals). For all three signals, the variance of the residuals is significantly larger in the second half of the lot compared to the first half. This results in many alarms (points that cross the control limits), even though there is no record of any problems with the process or equipment. At this point, one would need to generate a new baseline model in order to continue monitoring the process.

The same set of data were then adaptively filtered using the RLS algorithm with an order of three and $\lambda = 0.99$; the three-sigma control limits were adaptively estimated with a forgetting factor whose value was also $\lambda = 0.99$. The residuals and adaptive control limits are shown in Figure 5.4. The difference between Figure 5.3 and Figure 5.4 is clear: the adaptive estimation of the control limits allows the chart to adjust to the changing variances of the residuals. A new baseline model is not necessary in order to monitor subsequent wafers.

---

1. See Chapter 6 and Appendix B for more information on this software utility.
2. The following are the models identified by the RTSPC software. For "tcpTune": third-order auto-regressive after one level of differencing; for "tcpPwr": third-order auto-regressive (no differencing); for "tcpLoad": no model (only a mean value).
3. In Figure 5.3, the titles use the abbreviated names for the signals, as shown in Table 5.5; also, the words "short-term" in the titles are synonymous with "real-time".

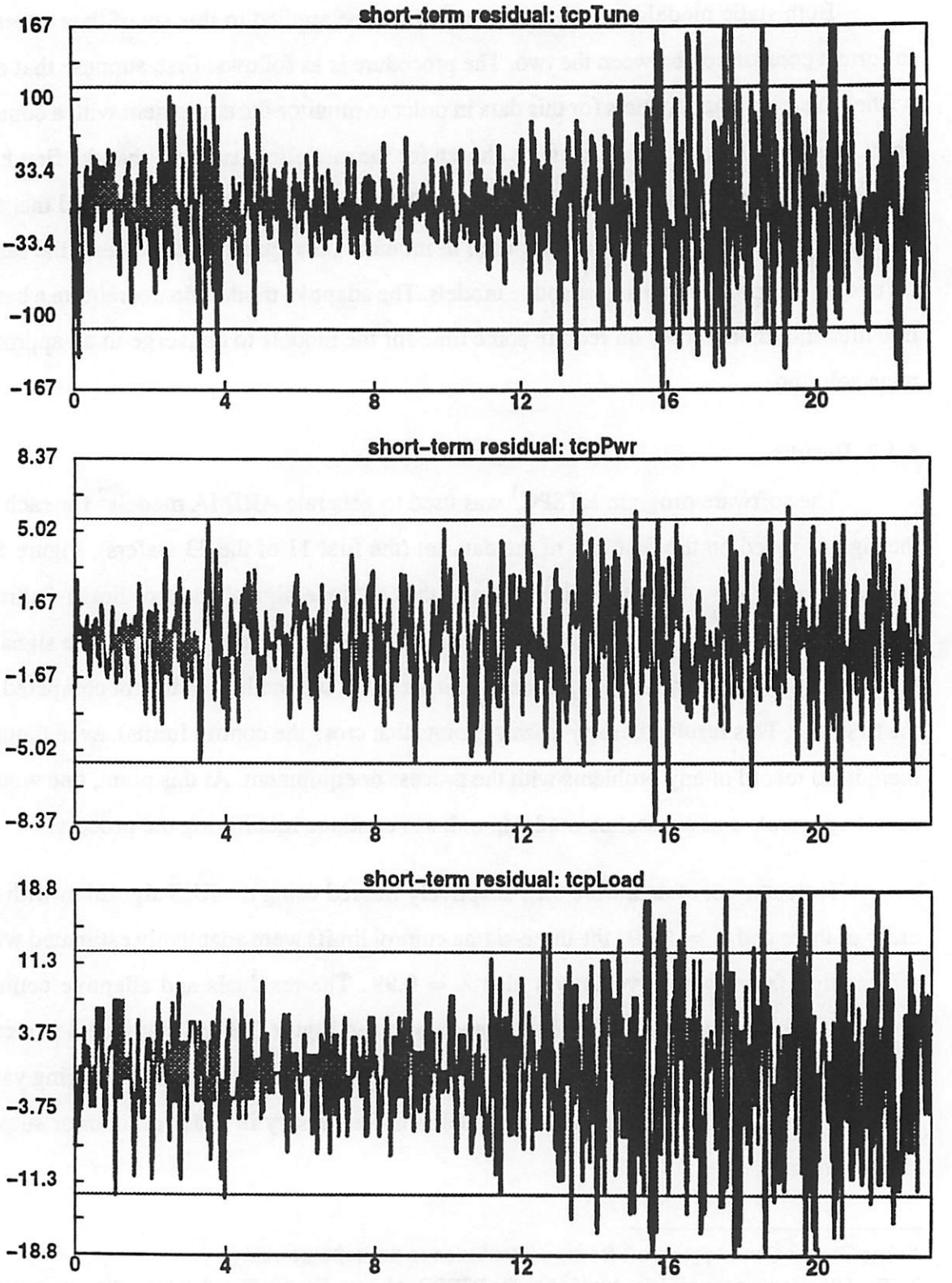

Figure 5.3.
Static models: residuals and control limits.

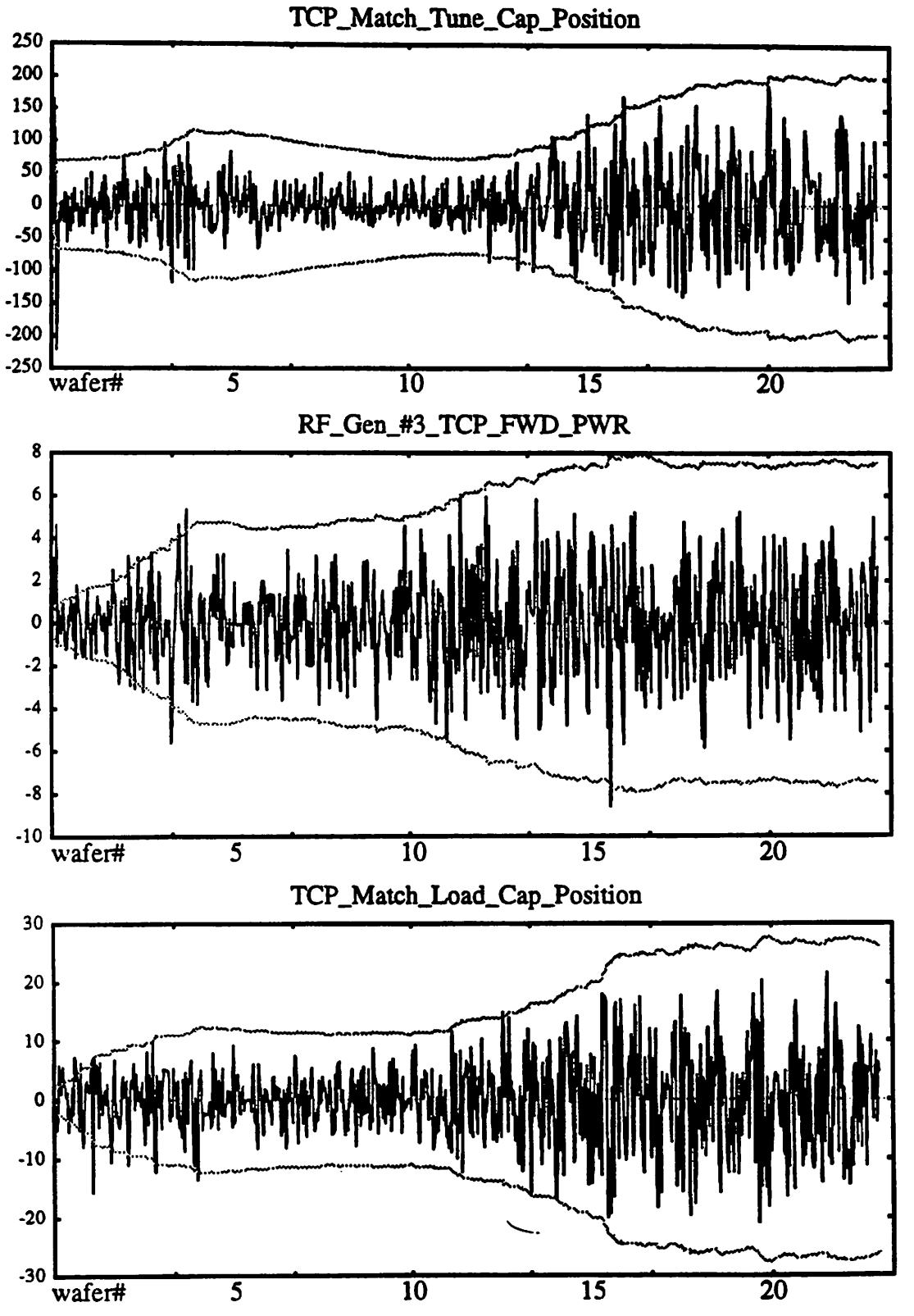

Figure 5.4.
Adaptive models: residuals and control limits.

Furthermore, the adaptive models can track changes in the auto-correlation structure of the signals, although this is not apparent from Figure 5.4. The tracking of the auto-correlation structure is better seen in Figure 5.5, where the values of the filter tap coefficients are plotted for the "tcpTune" signal. After the initial transients, the filter taps begin to settle down (wafers #5 to 11). However, after wafer #11 the taps begin to fluctuate again until around wafer #17, where they settle down to a new set of values.

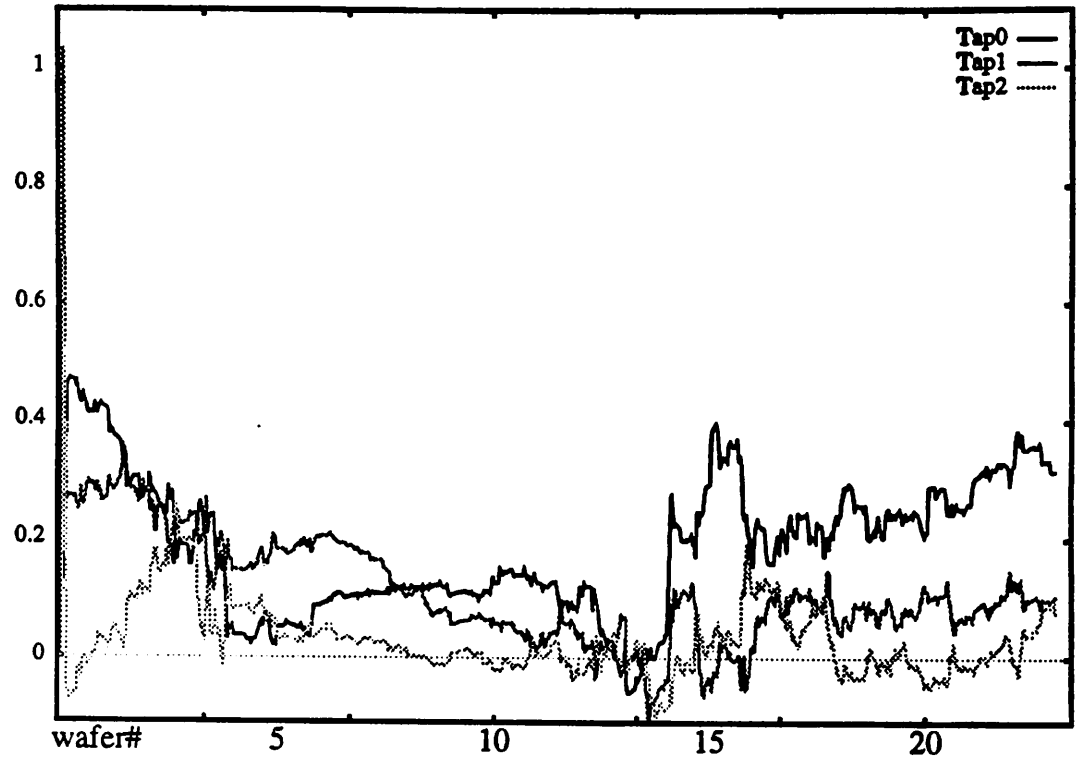

Figure 5.5.
Tap coefficients for "TCP_Match_Tune_Cap_Position".

Using the methodology of Section 5.2, the normalized T^2 values were calculated with $\lambda = 0.99$ and a group size of $n = 10$; these are plotted in Figure 5.6. After the transients in the first few wafers, no other alarms occur. In contrast, the T^2 values for the residuals of the static models (not shown) would have many alarms throughout the second half of the wafer lot.

5.4.3. Discussion

One might argue that the fact that no alarms are exhibited in Figure 5.6 is undesirable. After all, the structure of the sensor signals changed dramatically; what if there really

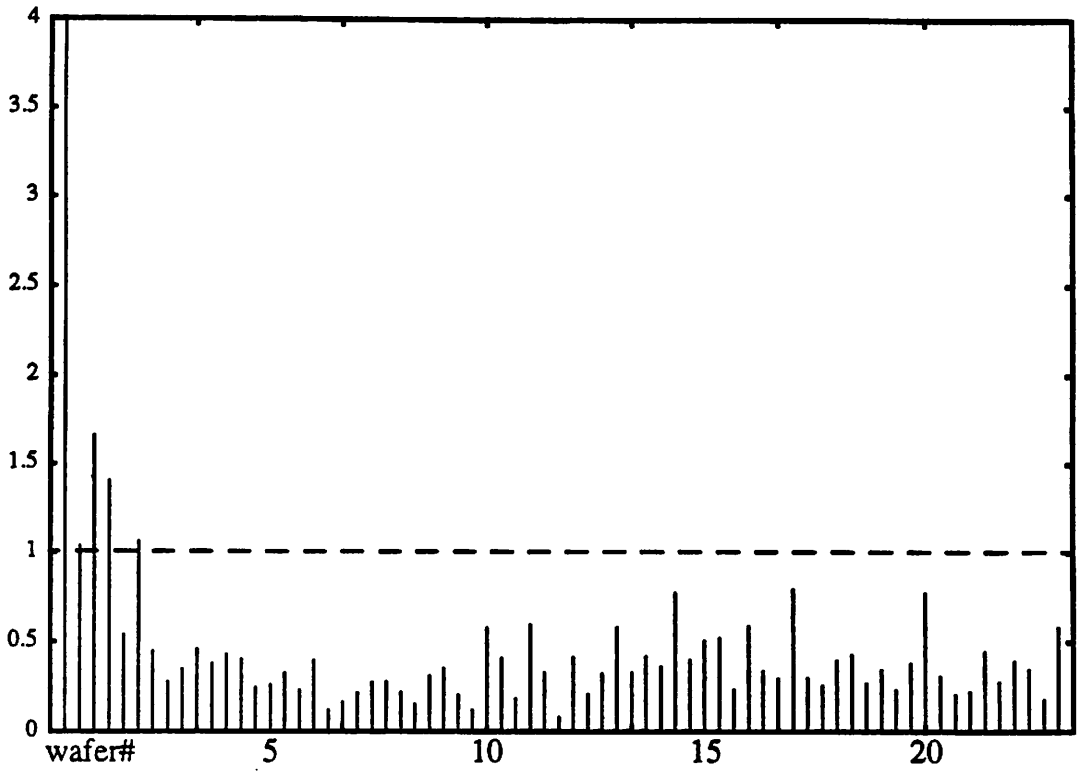

Figure 5.6.
Normalized T^2 (adaptively estimated).

was a problem with the equipment or process? The important point here is that the adaptive methodology can be tuned to any desired sensitivity. For example, any of the forgetting factors for the adaptive algorithm, for the control limit estimation, or for the T^2 estimation could be increased so that alarms are triggered more easily. Note that although no alarms are seen in Figure 5.6, the individual control charts of Figure 5.4 do show some alarms; one might wish to monitor signals individually, instead of using the multivariate T^2 statistic.

The other important point is that regardless of whether or not alarm situations are correctly identified by the control chart, the adaptive methodology does not require one to re-model the process, whereas static models will *always* require a new baseline experiment once the statistics of the sensor signals change.

Chapter 6 Software

6.1. Introduction

This chapter describes a software system known as **RTSPC**, which stands for Real-Time Statistical Process Control. **RTSPC** is the platform used to implement all the techniques presented in this paper. As the product of numerous years of work, **RTSPC** has undergone many changes by many people and is now in its third generation of development. In this chapter, only the current version of the program will be discussed; for an overview of past work, see [18][29]. See Appendix B for information on how to obtain the **RTSPC** software.

As its name suggests, the purpose of **RTSPC** is to implement real-time statistical process control on a computer. The software system consists of a user interface, several software modules, including those for numerical analysis and input/output functions, and a database. In Section 6.2 a brief overview of **RTSPC** is presented in order to give the reader a general impression of the software, and also to familiarize the reader with its basic operation (details can be found in the manuals included with the software distribution). Section 6.3 outlines some of the essential features of **RTSPC**, especially those that are in contrast to previous generations of the software. In Section 6.4 the organization of the software modules is described, as well as some of the programming details.

6.2. Overview

RTSPC has three basic operations:

1. Build baseline time series models from a selected set of baseline wafer data
2. Monitor the residuals and T^2 statistics of arriving wafer data (using a selected baseline model)

3. View the residuals and T^2 statistics of pre-recorded wafer data (using a selected baseline model)

The difference between the second and third items is that in the former case ("monitor" mode) the program analyzes signals as they are received from an equipment that is currently in operation, whereas in the latter case, analysis is done on wafer data that were collected sometime in the past.

Each of the three basic operations is described in the next three sub-sections. Figure 6.1 shows the main window of **RTSPC**'s user interface, which allows the user to perform any of the operations: model building is selected by pressing the "Build Model" button (Section 6.2.1), equipment monitoring is selected by pressing the "Start Monitor" button (Section 6.2.2), and statistical analysis is selected by pressing the "View Wafers" button (Section 6.2.3). Note that some operations require configuration procedures before they can be used; for example, the "Load Model" button must be selected to load a baseline model before the user is permitted to select the "Start Monitor" or "View Wafers" buttons.

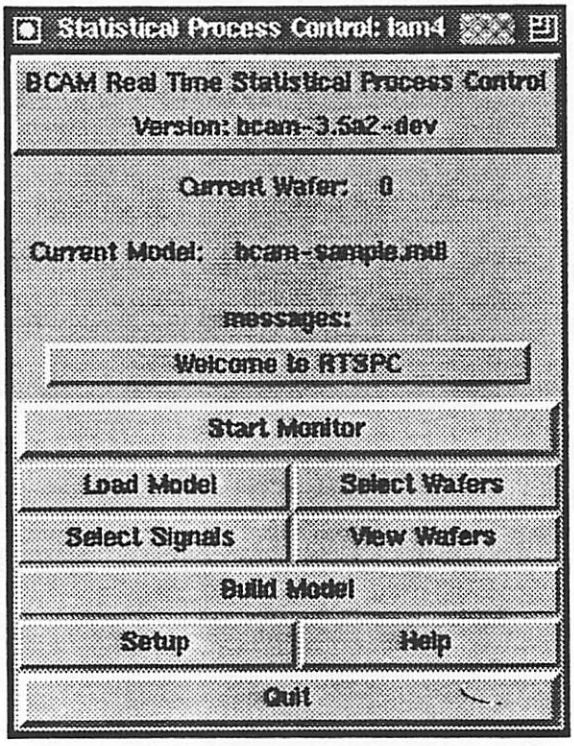

Figure 6.1.
The main window of **RTSPC**.

6.2.1. Model Building

Pressing the "Build Model" button from the main window brings up the model building window (not shown). Figure 6.2 depicts a diagram of the model building procedure. First the sensor data to be modeled is chosen. Usually the data for each wafer are stored in a separate file. The software allows the user to select a list of data files.

The next step, as shown in the diagram, is to perform any pre-processing or formatting necessary. For example, if models are to be built for a certain subset of signals (Section 2.3.2), then these should be filtered out from the rest of the data. Similarly, if models are to be built for a certain step of the process (Section 2.3.3), then only the data from that step should be retained. The pre-processing and formatting functions are implemented with *data filters*. A data filter is simply a short program or script that reads in data, applies some sort of filtering function, and then writes out the transformed data. The basic data filtering operations are the selection of signals, the selection of steps, and the averaging of signals. However, more sophisticated data filters can also applied, for instance taking a logarithm transformation of some data.

After wafer data has been selected and pre-processed, then **RTSPC**'s automatic model generation module (Section 2.2.4) proceeds to create time series models and a covariance matrix of the residuals. Models built from "baseline" wafer data are known as baseline models. The user will load these models into **RTSPC** before doing SPC or other statistical analysis. Expert users can more precisely control the model building process by specifying algorithm parameters, like the maximum model order, the maximum differencing order, or the significance threshold of model variables.

6.2.2. Equipment Monitoring

Pressing the "Start Monitor" button runs **RTSPC** in monitoring mode. Figure 6.3 depicts a diagram of the SPC (monitoring) procedure. One of two types of models can be selected: static or adaptive. To use static models, the program requires baseline models for all signals to be monitored and the covariance matrix of the residuals. If adaptive models are selected, no time series models need to be selected; the covariance matrix will be estimated from the prediction errors of the adaptive algorithm.

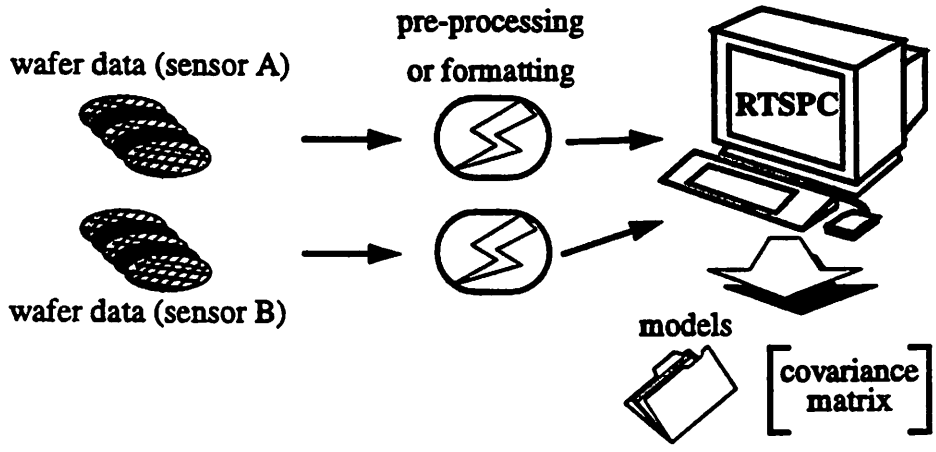

Figure 6.2.
Diagram of the model building procedure.

Once the program is ready, it will idle, waiting for wafer data to arrive. If a wafer-wafer prediction model is available, it will also expect to receive recipe data for each wafer. Data filters for both the wafer data and the recipes can be specified. Each time **RTSPC** is notified that new data has arrived, it processes the data with the data filters and sends the transformed data to its SPC module. The SPC module applies the static or adaptive models to the new data and plots the signal residuals and T^2 statistics on a graphical display.

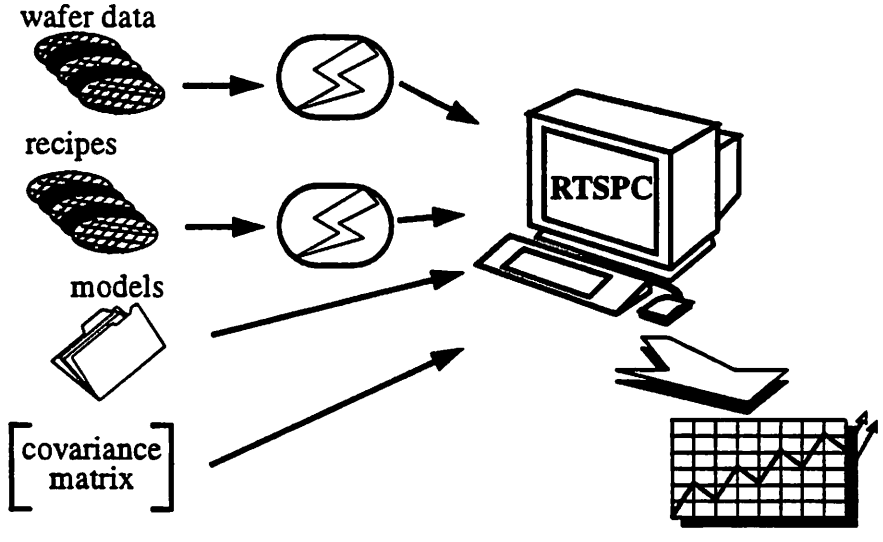

Figure 6.3.
Diagram of the SPC procedure.

6.2.3. Statistical Analysis

As mentioned earlier, the statistical analysis mode (started by pressing the "View Wafers" button) in **RTSPC** is essentially the same as the equipment monitoring mode. The only difference is that wafer data is selected from existing disk files, rather than having the program wait for new data arriving from an equipment in operation. This mode is useful for analyzing historical data, or for the evaluation and testing of models. It can also be used for off-line (i.e., not real-time) SPC or for diagnosis of past equipment faults.

6.3. Features

This section outlines important features in the current version of **RTSPC** and how they differ from those in previous versions. Of course the actual software is continually being modified, so for the most up-to-date information one must read the notes accompanying the current software release (see Appendix B).

A fundamental shortcoming in older versions of **RTSPC** was that they only implemented univariate time series (ARIMA) models. The new version has been extended to admit predictive wafer-wafer models (Chapter 3) and adaptive multivariate time series models (Chapter 4). Future releases of the software may permit even more types of models.

Another shortcoming in the previous generation of **RTSPC** was that the types of wafer data that could be accepted for analysis was quite limited. In particular, the program was hard-coded to read data formatted in a particular way: SECS-II data via LamStation software. This constraint turned into a serious obstacle when new sensors became available. In order for **RTSPC** to analyze data from the new sensors, the data had to be made to resemble LamStation data.

The introduction of data filters in the current version of **RTSPC** eliminates reliance on a particular wafer data format. In addition, the use of data filters offers a whole range of new possibilities. Data of arbitrary formats can be analyzed and any type of pre-processing can be applied. An example of a commonly used pre-processing filter is a data filter that performs some sort of normalization to a signal—ratioing the signal to a given reference signal, perhaps.

Data filters can also be used to average real-time signals into wafer-wafer signals, and to average wafer-wafer signals into lot-lot signals. This enables **RTSPC** to read in real-time data, but do analysis at the wafer-wafer and lot-lot levels. Previously one would have to pre-process the real-time data outside of **RTSPC** before importing them into the program, but now everything can be automated by applying the appropriate data filters. Basic data filters, such as the one for generating wafer-wafer and lot-lot signals, are included in the software distribution.

Other improvements to **RTSPC** include more robust input/output operations (better handling of error conditions) and a more compact representation of models. Also, future developers will benefit from the more consistent interactions between auxiliary functions and the user interface, the more meaningful command-line options, and the improved handling of dynamic memory.

6.4. Organization

This section describes the different software modules that make up **RTSPC** and how they relate to one another. The overall architecture is simple; see Figure 6.4 for the organization of the software modules. The user interface is written in Tcl/Tk (Tcl version 7.4, Tk version 4.0) [24], an interpreted programming language with good graphical user interface support; the user interface is the main module, known as **rtspc**. The event-driven **rtspc** module services user requests by either calling the proper auxiliary and library routines, or by executing the proper processes.

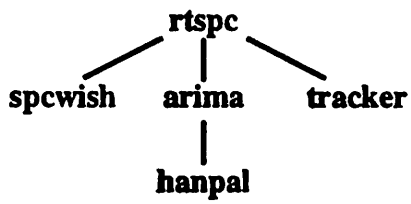

Figure 6.4.
Organization of software modules in **RTSPC**.

Auxiliary functions for equipment monitoring and statistical analysis are found in a module called **spcwish**; these functions are written in C++ and C [30][16] (and compiled with the public-domain Gnu compiler version 2.7.0). A matrix package is included for C++

matrix class support [31], and subroutines from a well-known numerical package are used for solving linear algebraic systems and inverting matrices [27].

The model building module **arima** is a separate process started by **rtspc** for identifying time series models and estimating their parameters. A second process called **hanpal** is a non-linear optimizer run by **arima** to estimate the moving average parameters of a time series model (see Section 2.2.4). The matrix package and numerical package used in **spcwish** are also shared with **arima** and **hanpal**.

Another process called by **rtspc** is **tracker**, the C++ program discussed in Section 4.5.2 (and listed in Appendix C). This module reads in signal data and applies adaptive filtering. The residuals (prediction errors) produced by the module are returned to **rtspc** for plotting and calculating T^2 statistics.

Most of the data filters mentioned in Section 6.2 and Section 6.3 are written in perl [35], a language ideal for text manipulations. However, any language can be used, for instance csh [1], sh, or Tcl. All of the above have the advantage of being simple to use, modify, and read. On the other hand, if speed is one's highest priority, then the data filters can be written in C++ or C.

As a final note, the **RTSPC** system also contains a few other peripheral modules. A program **retdata** exists for notifying **rtspc** when new data is available from a particular machine. A program **rtspcdemo** exists for running **RTSPC** in a demonstration. Finally, the **RTSPC** system manages and maintains a database for the storage of sensor data and previously built models.

Chapter 7 Conclusion

7.1. Summary

The main goal of this thesis is to present modeling techniques that make real-time SPC more effective. The thesis also explains how the techniques can be implemented and exhibits their use on actual sensor data. The first technique, predictive modeling, models the effect of abrupt changes, like changes in the machine's wafer-wafer input settings. The second technique, adaptive modeling, tracks statistical variations and slow drifts, for example caused by natural aging of an equipment. But most importantly, the adaptive models eliminate the need for a baseline training experiment—probably the primary impediment to practical implementation of real-time SPC.

However, in order to apply the techniques effectively, a comprehension of their strengths and weaknesses are required. The preceding chapters attempted to give the reader the necessary information regarding what the models are capable of and what they are not. Many examples were offered to demonstrate the ideas and suggest possible applications. In addition, a full description of a publicly available software package that implements all the techniques was included.

7.2. Future Extensions

The techniques presented here for modeling signals obtained from semiconductor manufacturing processes are general in the sense that they can be applied to many other types of processes or environments. The focus of this thesis was on plasma etching equipment, but other equipment for which real-time data are available could benefit as well, for instance, chemical vapor deposition furnaces. Moreover, extending the modeling techniques to other types of sensors, such as spectroscopy, and learning how to integrate mul-

tiple sensors into a combined model would enable one to take full advantage of the abundant real-time data that is available.

Predictive and adaptive modeling techniques can also be used as simple building blocks for large models. For example, projects like equipment diagnosis or closed-loop control will require complex systems composed of many models working together. Understanding the interaction between the models and the overall behavior of the system can become quite difficult, but necessary if the project is to succeed.

References

[1] Gail Anderson, Paul Anderson, *The UNIX™ C Shell Field Guide*, Englewood Cliffs, NJ: Prentice-Hall, 1986.

[2] Howard Anton, *Elementary Linear Algebra*, 5th ed., NY: John Wiley & Sons, 1987.

[3] Richard A. Becker, John M. Chambers, Allan R. Wilks, *The New S Language: A Programming Environment for Data Analysis and Graphics*, Pacific Grove, CA: Wadsworth & Brooks/Cole Advanced Books & Software, 1988.

[4] George E. P. Box, William G. Hunter, J. Stuart Hunter, *Statistics for Experimenters*, NY: John Wiley & Sons, 1978.

[5] George E. P. Box, G. M. Jenkins, G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, 3rd ed., Englewood Cliffs, N.J.: Prentice Hall, 1994.

[6] John M. Chambers, Trevor J. Hastie, eds., *Statistical Models in S*, NY: Chapman & Hall, 1993.

[7] Peter M. Clarkson, *Optimal and Adaptive Signal Processing*, Boca Raton, Florida: CRC Press, 1993.

[8] P. L. Feintuch, "An Adaptive Recursive LMS Filter," *Proceedings of the IEEE*, Nov 1976, pp. 1622-4.

[9] Hai-Fang Guo, "Real Time Statistical Process Control for Plasma Etching," M.S. thesis, University of California, Berkeley, Memorandum No. UCB/ERL M91/61, 2 Jul 1991.

[10] David M. Haaland, Edward V. Thomas, "Partial Least-Squares Methods for Spectral Analyses," *Analytical Chemistry*, Vol. 60, No. 11, 1 Jun 1988, pp. 1193-.

[11] Simon Haykin, *Adaptive Filter Theory*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1991.

[12] Simon Haykin, *Introduction to Adaptive Filters*, London: Macmillan Publishing Company, 1984.

[13] C. Richard Johnson, Jr., Michael G. Larimore, "Comments on and Additions to 'An Adaptive Recursive LMS Filter'," *Proceedings of the IEEE*, Sep 1977, pp. 1399-1402.

[14] Steven M. Kay, *Modern Spectral Estimation*, Englewood Cliffs, N.J.: Prentice Hall, 1988.

[15] Steven M. Kay, S. L. Marple, Jr., "Spectrum Analysis--A Modern Perspective," *Proceedings of the IEEE*, Nov 1981.

[16] Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1988.

[17] Sherry F. Lee, "Semiconductor Equipment Analysis and Wafer State Prediction System Using Real-Time Data," Ph.D. thesis, University of California, Berkeley, Memorandum No. UCB/ERL M94/104, 15 Dec 1994.

[18] Sherry F. Lee, Eric D. Boskin, Hao C. Liu, Eddie H. Wen, Costas J. Spanos, "RTSPC: A Software Utility for Real-Time SPC and Tool Data Analysis," *IEEE Transactions on Semiconductor Manufacturing*, Vol. 8, No. 1, Feb 1995, pp. 17-25.

[19] Sherry F. Lee, Costas J. Spanos, "Prediction of Wafer State After Plasma Processing Using Real-Time Tool Data," *IEEE Transactions on Semiconductor Manufacturing*, Vol. 8, No. 3, Aug 1995, pp. 252-261.

[20] Hao-Cheng Liu, "Automatic Time-Series Model Generation for Real-Time Statistical Process Control," M.S. thesis, University of California, Berkeley, Memorandum No. UCB/ERL M93/45, 8 Jun 1993.

[21] Dennis M. Manos, Daniel L. Flamm, eds., Plasma Etching: An Introduction, Boston: Academic Press, 1989.

[22] H. Martens, T. Naes, *Multivariate Calibration*, Wiley, 1989.

[23] Douglas C. Montgomery, *Introduction to Statistical Quality Control*, 2nd. ed., NY: John Wiley & Sons, 1985.

[24] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[25] Alan Pankratz, *Forecasting with Univariate Box-Jenkins Models: Concepts and Cases*, NY: John Wiley & Sons, 1983.

[26] Robert S. Pindyck, Daniel L. Rubinfeld, *Econometric Models and Economic Forecasts*, 2nd ed., NY: McGraw-Hill Publishing Company, 1981.

[27] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recipes in C*, 2nd ed., Cambridge University Press, 1992.

[28] John J. Shynk, "Adaptive IIR Filtering," *IEEE ASSP Magazine*, April 1989, pp. 4-21.

[29] Costas J. Spanos, Hai-Fang Guo, Alan Miller, Joanne Levine-Parrill, "Real-Time Statistical Process Control Using Tool Data," *IEEE Transactions on Semiconductor Manufacturing*, Vol. 5, No. 4, Nov 1992, pp. 308-18.

[30] Bjarne Stroustrup, *The C++ Programming Language*, 2nd ed., Addison-Wesley, 1991.

[31] E. Robert Tisdale, *C++ Matrix Class*, ftp://ftp.cs.ucla.edu/pub/Matrix.tar.Z, March 31, 1994.

[32] John R. Treichler, C. Richard Johnson, Jr., Michael G. Larimore, *Theory and Design of Adaptive Filters*, NY: John Wiley & Sons, 1987.

[33] *Using Framemaker®*, San Jose, CA: Frame Technology Corporation, 1995.

[34] Walter Vandaele, *Applied Time Series and Box-Jenkins Models*, NY: Academic Press, 1983.

[35] Larry Wall, Randal L. Schwartz, *Programming perl*, Sebastopol, CA: O'Reilly & Associates, 1991.

[36] Bernard Widrow, John M. McCool, "Comments on 'An Adaptive Recursive LMS Filter'," *Proceedings of the IEEE*, Sep 1977, pp. 1402-4.

[37] Bernard Widrow, Samuel D. Stearns, *Adaptive Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1985.

Appendix A List of Symbols

a_t	forecast errors of a time series
B	backshift (delay) operator
$d, d(t)$	time series data for desired response
D	integration order of an ARIMA model
$e, e(t)$	prediction (estimation) error
$\bar{\mathbf{e}}$	vector of signal residuals, averaged over a specified group size
$F_{\alpha, M, N-M}$	F-distribution value with M and $N-M$ degrees of freedom and Type I error α
h_k	filter tap
$\mathbf{h}, \mathbf{h}(n)$	filter tap vector
$\hat{\mathbf{h}}, \hat{\mathbf{h}}(n)$	estimate of filter tap vector
$\mathbf{h}^*$	optimum filter tap vector
$\mathbf{H}$	matrix whose each column is the regression coefficients for a single variable
$\mathbf{I}$	identity matrix
k	iteration index
L	number of modeled response signals
m	mean value of a wide-sense stationary process
M	number of explanatory variables minus one; model order
$MSE, MSE(n)$	mean square estimation error
MSE^*	optimum (minimum) value of MSE
n	group size for calculating T^2 statistic; iteration index

N	number of baseline observation; number of observations of a variable
$p(k)$	cross-correlation function between desired signal and input signal
$\mathbf{p}, \mathbf{p}(n)$	cross-correlation vector between input vector $\mathbf{x}(n)$ and desired response $d(t)$
$\hat{\mathbf{p}}, \hat{\mathbf{p}}(n)$	estimated cross-correlation vector
P	auto-regressive order of an ARIMA model
$\mathbf{P}, \mathbf{P}(n)$	inverse of the estimated correlation matrix
$\mathbf{q}_k$	unit-length eigenvector of $\mathbf{R}_x$
Q	moving average order of an ARIMA model
$\mathbf{Q}$	unitary matrix consisting of columns of $\mathbf{q}_k$
$r_x(k), r_x(t_1, t_2)$	autocorrelation function for signal $x(t)$
R^2	statistic indicating the percentage of total variance that is explained by a model
$\mathbf{R}_x$	correlation matrix
$\hat{\mathbf{R}}_x, \hat{\mathbf{R}}_x(n)$	estimated correlation matrix
$\hat{\mathbf{S}}$	estimated residual covariance matrix, used to calculate T^2 statistic
$\tilde{\mathbf{S}}$	an intermediate matrix used in the recursive calculation of $\hat{\mathbf{S}}$
t	time index
T^2	statistic used for multivariate statistical process control
UCL	upper control limit
$v_k, v_k(n)$	element of the transformed filter tap vector $\mathbf{v}$
$\mathbf{v}, \mathbf{v}(n)$	transformed filter tap vector
w_t	differenced time series data
$x_i, x_{i,t}$	explanatory stationary processes
$x(t)$	input time series data
$\mathbf{x}, \mathbf{x}(n)$	vector of predictor data; vector of current and previous values of $x(t)$
$\bar{\mathbf{x}}$	vector of averaged samples for each signal, used to calculate T^2 statistic
$\bar{\bar{\mathbf{x}}}$	vector of baseline signal averages, used to calculate T^2 statistic

	data matrix whose each column contains the observations for one variable
y_t	stationary process to be modeled
$y, y(t)$	output time series data
$\mathbf{y}$	vector of observed responses for estimating a regression model
$\hat{\mathbf{y}}$	vector of predicted responses
z_t	original time series data
α	Type I error (probability of false alarm)
β_i	regression coefficient
$\hat{\beta}$	vector of estimated regression coefficients
δ	small positive constant for initializing the RLS algorithm or the estimation of the error covariance matrix $\hat{\mathbf{S}}$
$\delta_i(B)$	moving average part of an ARIMAX model
θ_k	moving average coefficient of an ARIMA model
$\theta(B)$	moving average part of the transfer function of an ARIMA model
λ	exponential weighting factor for the RLS algorithm or for the estimation of the error covariance matrix $\hat{\mathbf{S}}$
λ_k	eigenvalue corresponding to eigenvector $\mathbf{q}_k$
λ_{min}	minimum eigenvalue
λ_{max}	maximum eigenvalue
Λ	diagonal matrix of eigenvalues
ϕ_k	auto-regressive coefficient of an ARIMA model
$\phi(B)$	auto-regressive part of the transfer function of an ARIMA model
σ_d^2	variance of desired response signal $d(t)$
σ_x^2	variance of input signal $x(t)$
τ_k	time constant for the convergence of the k-th mode of a steepest descent algorithm
$\tau_{\mathbf{h}}$	overall time constant for the convergence of the filter tap vector $\mathbf{h}$
μ	step size for LMS algorithm
$\mu(n)$	variable step size for RLS algorithm
$\tilde{\mu}$	step size for normalized LMS algorithm

	auto-regressive part of an ARIMAX model
$\chi^2_{\alpha, M}$	Chi-squared distribution with M degrees of freedom and Type I error α

Appendix B How to Get the RTSPC Software

The **RTSPC** software is a part of the Berkeley Computer-Aided Manufacturing (BCAM) software distribution. More information, including manuals and ordering instructions, can be found on the World Wide Web at

http://bcam.eecs.berkeley.edu

For questions or information on how to obtain **RTSPC**, contact:

Industrial Liaison Program
205 Cory Hall #1770
University of California at Berkeley
Berkeley, CA 94720-1770

Telephone: (510) 643-6687
Fax: (510) 643-6694
email: software@eecs.berkeley.edu
anonymous ftp: ilpsoft.eecs.berkeley.edu

Appendix C Software Code for "tracker"

```
/*-------------------------------------------------------------------*/
/* tracker */
/* by Herb Huang 1996 */

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <unistd.h> //for getopt()
/* C++ Matrix Class by E. Robert Tisdale, 03-31-94. (ftp.cs.ucla.edu) */
#include "Matrix/double.Matrix.h"

const char* Means_filename="waferMeans.dat";

const int cInitOrder=3;
const int cInitFbOrder=0;
const double cInitLambda=1, cInitMu=0.001;
const int cInitNumSignals=1;
/* InitP should be large compared to 100/(variance of data)
   [Haykin 1991, p.484] */
double InitP=1e2;

long DataSize=3000; //initial data size
const double ZERO=0;

const int RLSmode=1; // set RLS mode
int Multivariate=0; // defaults to univariate mode

const int BUFF_SIZE=1024;
const char TOKEN_DELIM[]=", \t\n\r\f";
inline int STREQ(const char* a, const char* b) { return (!strcmp(a,b)); }

/*-------------------------------------------------------------------*/
/*
   order = number of feedforward taps (coefficients)
   fbOrder = number of feedback taps
   U = data vector
   H = filter taps (each row is the tap vector for one signal)
   K = gain vector
   e = prediction error
   P = filter-error correlation matrix (except for scaling)
   lambdainv = inverse of RLS forgetting factor
   mu = LMS step size
   iter = iteration number
 */
int main(int argc, char* argv[])
{
  long ReadData(doubleMatrix& Data, long numData, int numSignals);
```

```
  char filenameBuff[BUFF_SIZE];

  int order=cInitOrder;
  int fbOrder=cInitFbOrder;
  double lambda=cInitLambda;
  double lambdainv=(1./cInitLambda);
  double mu=cInitMu;
  int numSignals=cInitNumSignals;
  long numData=-1;

  extern char* optarg;
  int c;
  while ((c = getopt(argc, argv, "o:b:l:s:n:p:mh")) != -1) {
    switch(c) {
    case 'h':
    case '?':
    default:
      cerr << "Usage: " << argv[0] << " < data.txt > e.out" << endl;
      if (c == 'h')  exit(0);
      exit(1);
      break;

    case 'o':
      order = atoi(optarg);
      break;
    case 'b':
      fbOrder = atoi(optarg);
      break;
    case 'l':
      lambda = atof(optarg);
      lambdainv = 1./lambda;
      break;
    case 's':
      numSignals = atoi(optarg);
      break;
    case 'n':
      numData = atol(optarg);
      break;
    case 'm':
      Multivariate = 1;
      break;
    case 'p':
      InitP = atof(optarg);
      break;
    }
  }

/*-------------------------------------------------------------------*/
  int numTaps=(Multivariate ? (order+fbOrder)*numSignals : (order+fbOrder));

  DataSize = (numData > 0 ? numData : DataSize);
  doubleMatrix Data(numSignals, (int)DataSize);
  doubleMatrix H(numSignals, numTaps);
  const int pdim=(Multivariate ? numTaps : numSignals*numTaps);
  doubleMatrix P(pdim, numTaps);
  doubleMatrix PP(numTaps, numTaps); // a temporary variable

  FILE* meansFile=0;
  if (*Means_filename) {
```

```
    sprintf(filenameBuff, "%s", Means_filename);
    meansFile=fopen(filenameBuff, "r");
    if (!meansFile)  {
      cerr << "Warning: can't open wafer-mean file: " << Means_filename <<endl;
    }
  }

  const int udim=(Multivariate ? 1 : numSignals);
  doubleMatrix U(udim, numTaps),  K(udim, numTaps);
  doubleMatrix PU(1, numTaps); // a temporary variable

  doubleMatrix y(1, numSignals);
  doubleMatrix e((fbOrder ? fbOrder : 1), numSignals);
  doubleMatrix yhat((fbOrder ? fbOrder : 1), numSignals);
  doubleMatrix mean(1, numSignals);

/*-------------------------------------------------------------------*/
  /* "soft-constrained initialization" */

  /* INITIALIZE H = 0 */
  H = ZERO;

  /* INITIALIZE P = diag(InitP) */
  P = ZERO;
  if (Multivariate) {
    for (int i=0; i<numTaps; i++)  P[i][i] = InitP;
  } else {
    for (int i=0; i<numSignals; i++) {
      for (int j=0; j<numTaps; j++) {
       P[i*numTaps + j][j] = InitP;
      }
    }
  }

/*-------------------------------------------------------------------*/
  /* loop over each wafer */
  while (1) {


  yhat = ZERO;
  e = ZERO;
  Data = ZERO;

/**
   ReadData(): read from stdin into Data.
   read 'numData' lines.
   return the number of lines read, or
   return -1 if EOF is reached or an error occurs.
**/
  numData = ReadData(Data, numData, numSignals);
  if (numData <= 0 )  break; // all done

  {
    /* CENTER data */
    mean = ZERO;
    if (meansFile) {
      /* read a line from the "waferMeans.dat" file into mean[0][] */
    }
    else {
```

```
      mean = Data.sum() / (double)numData;
    )

    for (int i=0; i<numSignals; i++) {
      double m=mean[0][i];
      for (int j=0; j<numData; j++) {
       Data[i][j] -= m;
      )
    )
  )

/*-------------------------------------------------------------------------*/
  /* START iteration LOOP */
  for (int iter=0; iter<numData; iter++) {
    /* fill y */
    for (int sigNum=0; sigNum<numSignals; sigNum++) {
      y[0][sigNum] = Data[sigNum][iter];
    )

    {
      /* Fill U (with pre-windowed data) */
      U = ZERO;
      if (Multivariate) {
       int cnt=0;
       /* Feedforward */
       for (int j=0; j<order; j++) {
         for (int i=0; i<numSignals; i++) {
           if (j <= iter-1)  U[0][cnt] = Data[i][iter-1-j];
           ++cnt;
         )
       )
       /* Feedback */
       for (int j=0; j<fbOrder; j++) {
         if (j > iter-1)  break;
         for (int i=0; i<numSignals; i++) {
           U[0][cnt] = yhat[j][i];
           ++cnt;
         )
       )
      )
      else {
       /* Feedforward */
       for (int j=0; j<order; j++) {
         if (j > iter-1)  break;
         for (int i=0; i<numSignals; i++) {
           U[i][j] = Data[i][iter-1-j];
         )
       )
       /* Feedback */
       for (int j=0; j<fbOrder; j++) {
         if (j > iter-1) break;
         for (int i=0; i<numSignals; i++) {
           U[i][order + j] = yhat[j][i];
         )
       )
      )
    )

    /* shift yhat; shift e */
```

```
    for (int j=fbOrder-1; j>=0; j--) {
      for (int sigNum=0; sigNum<numSignals; sigNum++) {
       if (j != 0) {
         yhat[j][sigNum] = yhat[j-1][sigNum];
            e[j][sigNum] =    e[j-1][sigNum];
       }
      }
    }

    /* compute (a priori) prediction error, e */
    if (Multivariate) {
      yhat.s(0) = U % H; //yhat.s(i) is row i of yhat
    } else {
      yhat.s(0) = (U * H).sum(); //yhat.s(i) is row i of yhat
    }
    e.s(0) = y - yhat.s(0);

/*-------------------------------------------------------------------*/
    /* compute gain vector, K[] */
    if (RLSmode) {
      if (Multivariate) {
       /* Using (U % P.t()) here is more numerically robust than (U % P).
          [Haykin 1991, p.485, 695] */
       PU = (U % P.t());
       double denom = lambda + (double)(U % PU);
       K = (U % P) / denom;

       /* update filter-error correlation matrix */
       P = lambdainv * (P - (K & PU));
      } else {
       /* This is univariate mode, so calculate K and P individually for
          each signal. */
       for (int i=0; i<numSignals; i++) {
         /* PP points to numTaps rows beginning with row i*numTaps */
         PP = P.s(i*numTaps,numTaps);

         /* PP.t() not PP !!! */
         PU = (U.s(i) % PP.t()); // U.s(i) is row i of U
         double denom = lambda + (double)(U.s(i) % PU);
         K.s(i) = (U.s(i) % PP) / denom;

         /* update filter-error correlation matrix */
         PP = lambdainv * (PP - (K.s(i) & PU));

         P.s(i*numTaps,numTaps) = PP;
       }
      }
    }
    else { /* LMS (tracking mode) */
      K = mu * U;

#ifdef NOTYET
       /* or use Normalized LMS [Haykin 1991, p.356]:
         a is a damping constant
       */
       if (Multivariate) {
        K = mu * U / ((U * U).sum() + a);
       } else {
        K = mu * U / (((U * U).sum()).t() + a);
```

```
      )
#endif NOTYET
    )

    /* update filter */
    if (Multivariate) {
      H +=  e.s(0) & K;
    } else {
      H += (e.s(0)).t() * K;
    }

    /* Stability checks (only for univariate mode) */
    if (fbOrder == 1 && !Multivariate) {
      int col=order;
      for (int j=0; j<numSignals; j++) {
       double a1=H[j][col];
       if (fabs(a1) >= 1) {
         cerr << "Warning: a1 outside of stability region: "
           "H[" << j << "][" << col << "] = " << H[j][col] << endl;
       }
      }
    }
    if (fbOrder == 2 && !Multivariate) {
      int col=order;
      for (int j=0; j<numSignals; j++) {
       double a1=H[j][col];
       double a2=H[j][col+1];
       if (a2 <= -1 ||
           a2 >=  a1 + 1 ||
           a2 >= -a1 + 1) {
         cerr << "Warning: (a1,a2) outside of stability region: "
           "H[" << j << "][" << col   << "] = " << H[j][col] << ", "
           "H[" << j << "][" << col+1 << "] = " << H[j][col+1] << endl;
       }
      }
    }
  }

  } //while (1) {

  return 0;
}
/*--------------------------------------------------------------------*/
```