AUTOMATED TREND EXTRACTION OF SENSOR SIGNALS FOR PATTERN BASED DATA ANALYSIS

By

SRINIVAS S. GANTI

Bachelor of Engineering

Karnataka Regional Engineering College

Surathkal, India

1992

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1996

AUTOMATED TREND EXTRACTION OF SENSOR

SIGNALS FOR PATTERN BASED DATA

ANALYSIS

Thesis Approved:

[illegible]

Thesis Advisor

[illegible]

[illegible]

Thomas C. Collins

Dean of Graduate College

ACKNOWLEDGMENTS

I'd like to take this opportunity to thank the School of Chemical Engineering, Oklahoma State University for providing me with financial support.

I'd like to thank Dr. James R. Whiteley for being my advisor, and wish to acknowledge his constant support and guidance throughout my stay at OSU. Also, a large note of appreciation goes to Dr. Arland H. Johannes and Dr. Karen A. High for serving on my thesis committee.

I'd like to thank Bruce Colgate, Jack DeVeaux, Julia A. Rumsey, and Loy at the R&D center of the Phillips Petroleum Co., Bartlesville.

A personal note of gratitude goes to my friends in Surathkal, Visakhapatnam and Bombay, most of all Sarin and Raghu. Without their support, I would never have achieved all that I did. Another personal note of appreciation goes to my friends at OSU with whom I spent many a sleepless night at school, especially during the first two semesters. They made life at grad school real fun despite its trials and tribulations.

The deepest appreciation and thanks must go to my mother and brothers, Krishna and Jagan, for all that they had done for me. I'd like to thank my sister Bindu for her love and affection. A special note of thanks to Padma for bringing the sunshine to my life.

I wish to dedicate this work to the fond memory of my father.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY OF NOMENCLATURE

$\delta(t)$	A unit impulse response
α_k , β_k	Decomposition basis functions
$\varphi(t)$	Scaling function
ω	Sampling frequency
Ω_0	Fundamental frequency
τ_F	Time constant of digital filter
α, γ	Parameters that specify the fraction of the signal filtered
Δx	Maximum allowable change in filtered output in Noise spike filter
$f(t)$	Real time representation of a signal
g_k , h_k	Decomposition coefficients
$W(t)$	Wavelet function
c_k	Scaling function coefficients
d_k	Wavelet function coefficients
H, G	High and low pass filters respectively
a_j^i	j^{th} blurred decomposition coefficient at the i^{th} decomposition level
b_j^i	j^{th} detail decomposition coefficient at the i^{th} decomposition level
T_s	Sampling interval of the signal

T_{window}	Window of interest in time
N	Length of a signal
no_levels1	Parameter in code that represents the minimum levels of decomposition
min_levels	Minimum levels of decomposition
no_levels2	Parameters in code that represents the number of levels obtained from user input window size
act_levels	Final number of levels of decomposition of the signal
j	Parameter in code that represents cut-off frequency index
min_win	Parameter in code that represents the minimum window size
V	Current value of the variable
T	Current time
V_R	Most recent recorded value
T_R	Time corresponding to the most recent recording
V_L	Recorded value immediately prior to V_R
T_L	Time corresponding to recording of V_L
S	Slope defined as $(V_R-V_L)/(T_R-T_L)$
V_P	Predicted value
H	Recording limit
P	Adaptive parameter in the combination boxcar and backward slope algorithm

J	Moving window size in the moving average filter

Chapter 1

INTRODUCTION

Automated real-time process monitoring which provides "experienced-operator-level monitoring" is a major need in the chemical process industries. Work at Oklahoma State University is being performed to meet this need and is focusing on monitoring sensor patterns or trends, on digital computers, by using artificial neural networks [Whiteley, 1989; Whiteley , 1990a & 1991b]. The techniques under study are highly dependent on the pattern-based information content of sensor data. A trend plot for a typical plant sensor is shown in Figure 1.1. Pattern-based monitoring techniques require extraction of the fundamental trend and compact representation of the extracted trend in order to achieve the computational efficiency necessary for real-time application.

Sensor data is contaminated by noise which masks the fundamental trend followed by the signal. A compact representation of the fundamental trend can be obtained by smoothing the signal so as to remove only the noise, and then sampling the smoothed signal at an appropriate sampling frequency. Figure 1.1 shows a very noisy signal consisting of 1024 data points, and a smoothed representation of the fundamental trend over the time period T_{window}. By sampling the smoothed representation at an interval of T_s, the original signal consisting of 1024 data points can be compactly represented by $[(T_{window}/T_s) + 1]$ points (9 points as shown in Figure 1.1).

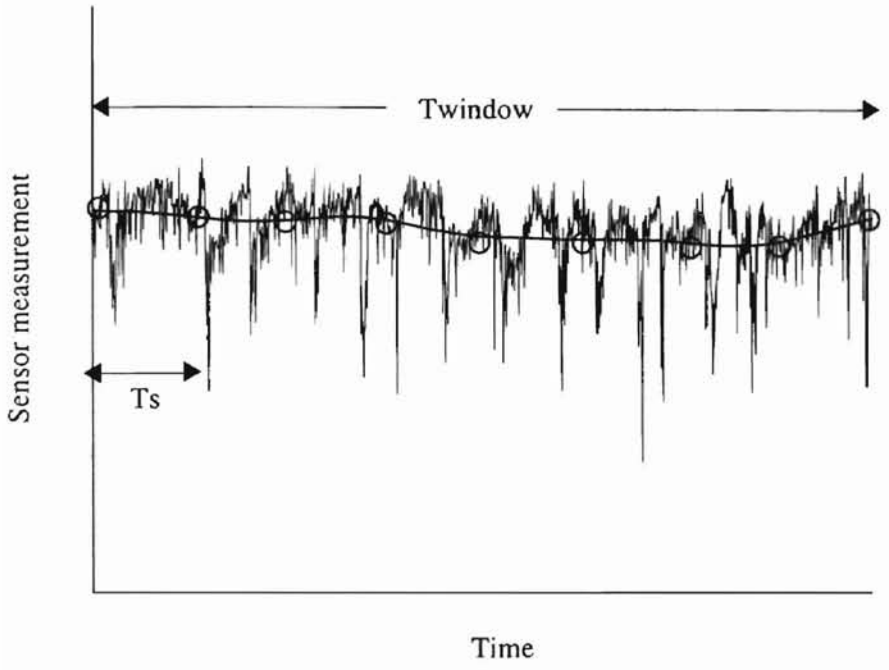
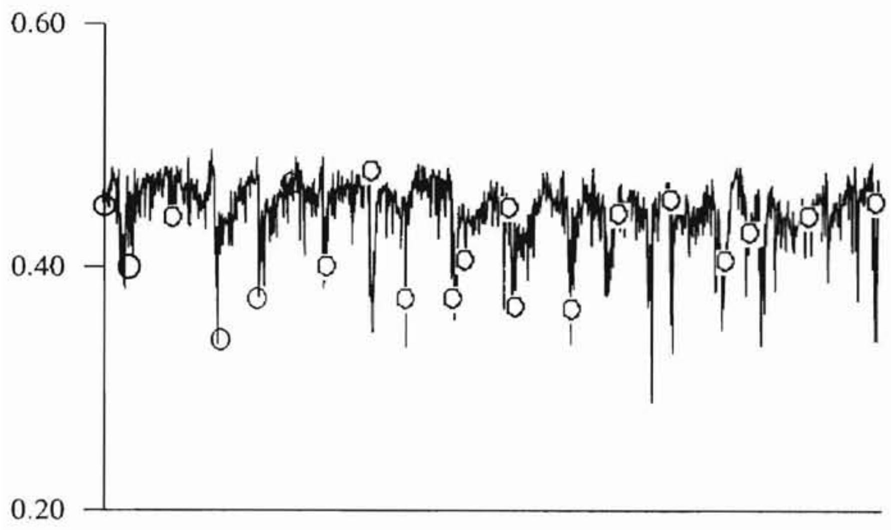


Figure 1.1. Typical sensor signal and its smoothed approximation which represents the fundamental trend.

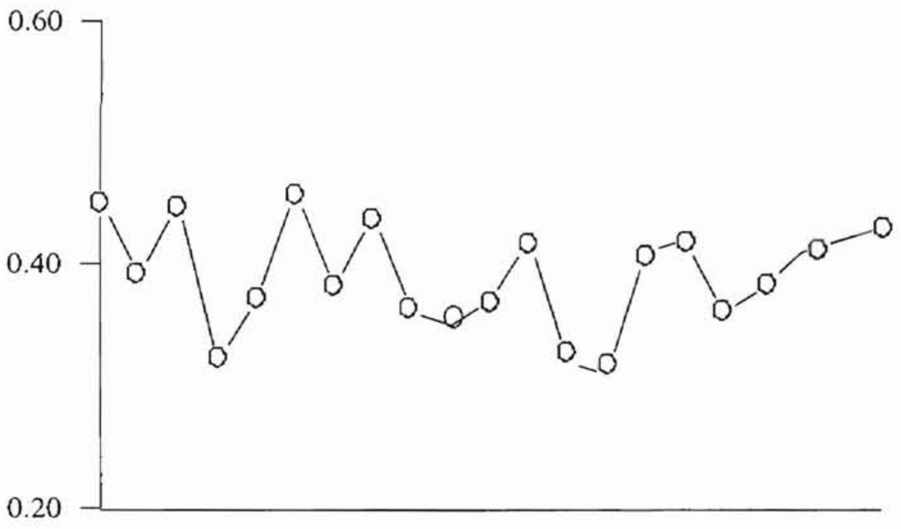
Sampling the sensor signal is the most obvious method for compact representation of sensor data. However, unless the signal is first smoothed, such a sampled version does not capture the true trend of a signal. This is illustrated in Figure 1.2. Various techniques can be used depending on the quality of trend resolution required, e.g., direct methods, digital filters, Fourier transform etc. However, as shown in the next chapter, these conventional methods typically fail to retain abrupt changes in the fundamental trend of the sensor signal. Consequently, these conventional techniques are unacceptable for the pattern-based methods being developed at Oklahoma State University.

Mr. Vinod Raghavan previously proposed the use of wavelet transforms for extracting the fundamental trends from noisy signals [Raghavan 1994]. Using wavelet transforms, various smoothed representations of a noisy sensor signal can be obtained by varying a parameter that represents the degree of smoothing. Figure 1.3 shows a noisy sensor signal and its smoothed representations using two different levels of wavelet smoothing. Different degrees of smoothing beyond those shown in Figure 1.3 are possible. The degree of smoothing is application dependent, and generally tied to the length of the window used for monitoring the process.

The features embedded in the patterns are used to trigger pattern recognition. This is achieved by comparing the patterns being monitored with those patterns that have been previously observed and classified. The degree of smoothing obviously affects these features. Previously, the degree of smoothing was determined manually [Raghavan, 1994]. This work is an extension of the work previously done by Raghavan. The purpose of this work is to automatically determine the desired degree of smoothing of a sensor signal using the characteristics of the signal and the length of the window used for

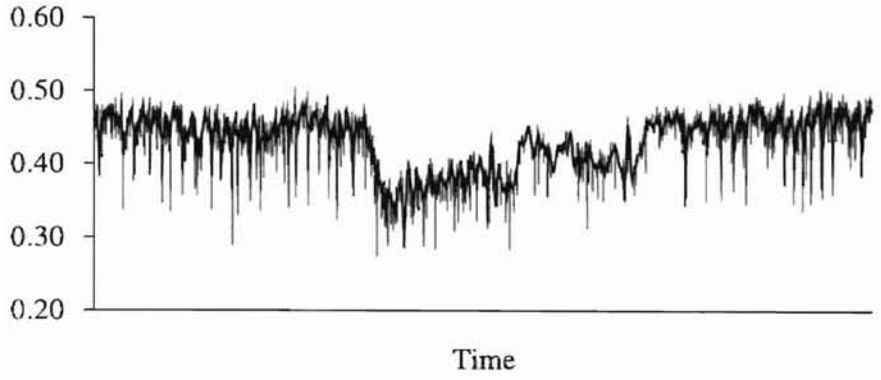


(a) Sampling without smoothing.

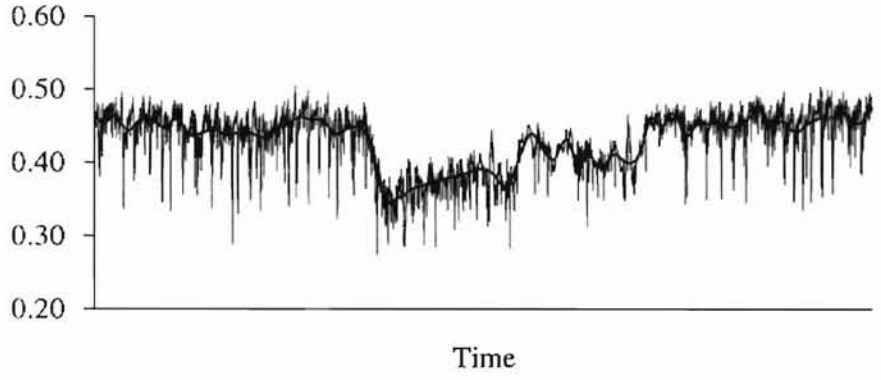


(b) Trend inferred from sampled points is incorrect.

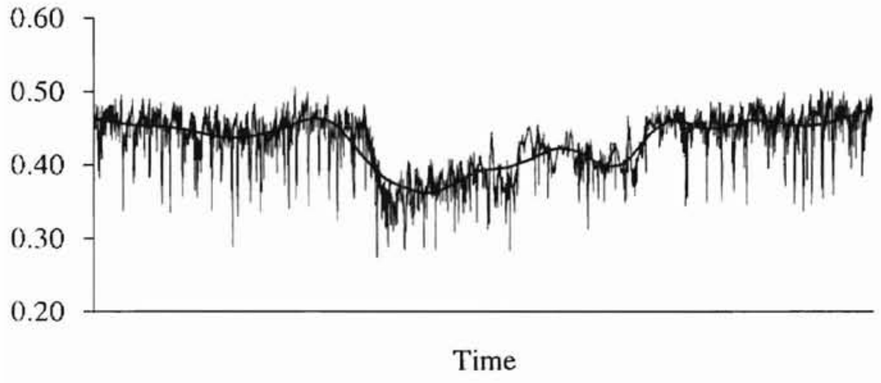
Figure 1.2. Sampled representation of the original signal without smoothing.



(a) Original signal and its smoothed approximation at the third level using sixth order Daubechies wavelets.



(b) Original signal and smoothed approximation at the seventh level using sixth order Daubechies wavelets.



(c) Original signal and smoothed approximation at the ninth level using sixth order Daubechies wavelets.

Figure 1.3. Noisy sensor signal and its smoothed approximations.

process monitoring. Our goal at Oklahoma State University is to develop a user-friendly system which allows an operator to construct his or her own pattern-based monitoring application for any plant problem, based on his or her expertise.

1.1 Thesis Outline

The organization of this thesis is as follows:

- Chapter 2 describes digital signal representation and sampling. This chapter describes the performance of some of the conventional signal processing methods, emphasizing their inability to extract the fundamental trends from noisy sensor signals. This chapter lays the groundwork for the next chapter by concluding with a brief comparison of essential features of the Fourier transform, Short-Time Fourier transform, and the wavelet transform.
- Chapter 3 discusses the general mathematical details of the discrete Fourier series and time-frequency relationships. This chapter introduces the concept of time-frequency localization and discusses the inability of the Fourier transform to capture both the time and frequency content of a signal simultaneously. This chapter introduces wavelets as the alternative for time-frequency localization and describes the general mathematical details of the wavelet transform with a brief mention of the Daubechies wavelet family. This chapter concludes by demonstrating wavelet smoothing of sensor signals, with the degree of smoothing decided manually by trial and error.
- Chapter 4 constitutes the main part of this work. This chapter discusses the need for automated trend extraction and presents a novel method to automatically determine

the desired degree of smoothing of a noisy sensor signal. This chapter describes the cumulative power spectrum which is used to determine the fundamental frequencies and eliminate noise.

- Chapter 5 presents the performance of the automated trend extraction algorithm with case studies, for a set of signals with different characteristics.
- Conclusions of this work are presented in Chapter 6 along with recommendations for future study.

Chapter 2

CONVENTIONAL SIGNAL PROCESSING TECHNIQUES

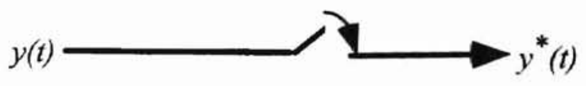
2.1 Digital signal representation and sampling

Sensor signals correspond to recorded measurements of process variables such as temperature, pressure, flow, etc. A discrete signal is " a function defined on only a discrete set of time values $t_0, t_1, t_2, \ldots\ldots$" [Seborg *et al*, 1989]. A digital signal is obtained from the analog (continuous) signal by analog to digital (A/D) converters. The continuous signal $y(t)$ is sampled at discrete points in time, $t_0, t_1, t_2, \ldots\ldots$, to obtain the sampled data, $y(t_k)$, $k = 0, 1, 2, 3, \ldots.$, or the digital signal. Figure 2.1.shows the sampling of an analog signal to form the digital signal. The sampling action of the sampler is repeated every   seconds. The action of the sampler is such that during the *kth* sampling interval, the sampler output $f^*(t)$ takes the value $f(t = t_k)$. Thus the complete sampled function $f^*(t)$ is a train of impulses whose values match that of the continuous function *only* at the sampling points and remain at zero elsewhere. A single sampler output at time t_k is most conveniently represented or modeled by the ideal delta (impulse) function as

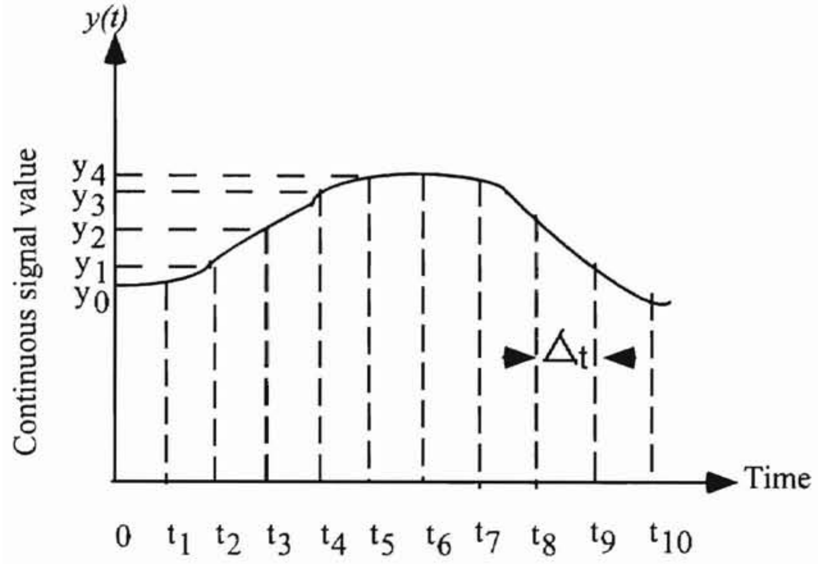
$$f^*(t_k) = \int_{t_k-\varepsilon}^{t_k+\varepsilon} f(t)\delta(t - t_k)dt \tag{1}$$

The property of the delta function for any arbitrary function is as shown below:

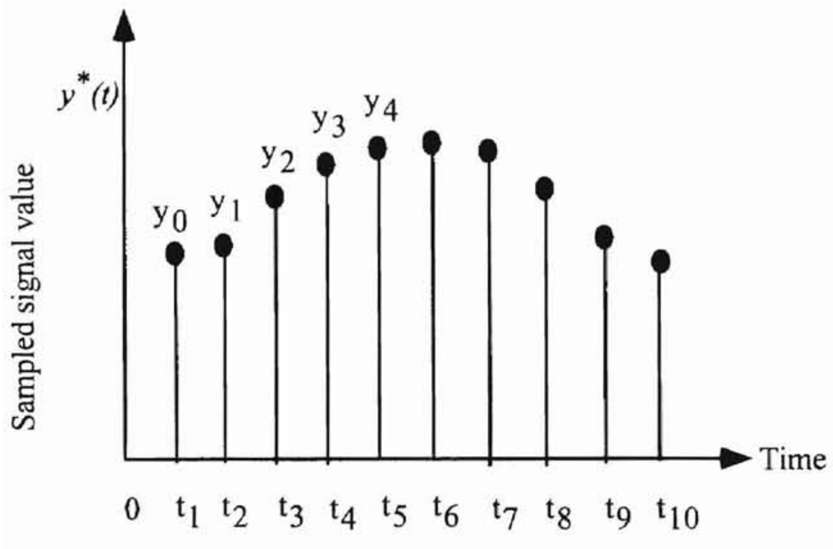
$$\int_{t-\varepsilon}^{t+\varepsilon} x(t)\delta(t-\theta)dt = \begin{cases} x(\theta); & t = \theta \\ 0; & t \neq \theta \end{cases} \tag{2}$$



(a) The sampler



(b) Continuous signal *y(t)* and samples.



(c) Sampled signal.

Figure 2.1. Peroidic sampling of an analog signal. [From Seborg (1989), page 531.]

The function $f^*(t)$ can be represented as:

$$f^*(t) = f^*(t_0) + f^*(t_1) + f^*(t_2) + \ldots\ldots\ldots + f^*(t_k) + \ldots \qquad (3)$$

Each entity indicated in the above sum remains at zero until t equals the argument in the parenthesis. When t equals the argument in the parenthesis, it instantaneously takes the indicated value and thereafter it returns to zero. The digital signal in Figure 2.1 is localized in time as it is expressed as a series of impulses in time.

One of the most important issues while sampling a continuous signal is choosing Δt, the sampling interval. Obviously, sampling too rapidly increases data storage requirements. On the other hand, by sampling too slowly, storage space may be saved but the sampled signal may not resemble the original signal. The optimum sampling time must lie between these two extremes. There are no hard and fast rules in choosing Δt. A judicious choice would be to chose a value small enough to assure that no significant information is lost in sampling and at the same time would not overload the data storage requirements. If the maximum frequency that needs to be captured, ω_{max} is known, then the *sampling frequency,* ω_s should be at least $2\omega_{max}$ or $2\pi/\Delta t$. This means that if an "event" in the signal occurs every t seconds, then to capture this "event", the signal needs to be sampled at least every $t/2$ seconds. For any sampling frequency ω_s, the largest frequency that can be captured is called the Nyquist frequency ω_N, and is given by

$$\omega_N = \omega_s/2 = \pi/\Delta t \qquad (4)$$

The choice of Δt depends very much on the application of interest.

2.2 Motivation for signal filtering

The analog representation of the measurement is usually not the true measurement because of errors introduced by the process, measurement device, and the signal transmission equipment. Process induced noise could be attributed to variations in mixing, turbulence, and non-uniform multiphase flows. Electrical noise is introduced because of improper shielding and grounding of cables. The error or noise introduced by the measurement device is specified by the vendor and is usually of the order of 0.25 - 1%. Further, the sampling techniques also introduce some error. The error introduced by all these sources is referred to as noise. Noise masks the actual trend of the digital signal. Therefore a certain degree of conditioning or filtering of the sampled signal is required to eliminate the noise and extract the actual trend before it could be used by a process monitoring technique. Various techniques are available to perform this conditioning or filtering. However, the effectiveness of a signal processing technique depends on the nature of the signals involved and the manner in which the processed signal will be used. The following section describes some of the conventional techniques for signal processing and demonstrates their ineffectiveness for pattern-based monitoring.

2.3 Conventional signal processing techniques

The main objective of the signal processing techniques described in this section is to eliminate high frequency noise, while retaining the fundamental trend of the original signal. Several conventional signal processing methods are described in detail in this section. Performance of these techniques are compared with that of the wavelet transform. The results presented indicate that the wavelet transform is a more effective trend extraction technique for pattern-based monitoring. Actual plant data is used for all performance evaluations.

2.3.1 The Direct methods

The boxcar, backward slope and the combination box car and backward slope methods [Hale and Sellars 1982] were used for data compression in the 1980s to save computer disk space. These techniques require two choices be made for each process variable to be included in the data archive: a recording limit and an algorithm to make the recording decision. The recording limit is most often selected to match the transducer's inherent accuracy, e.g., 1% of span or 0.5°C for thermocouples. For normalized sensor values this means that the recording limit is 0.01. These algorithms have been in use in the industry for several years. These methods record a particular value (of the sensor) only if the subsequent point fails a test determined by the algorithm, which varies for different methods. The test hinges on a pre-specified value of the recording limit.

2.3.1.1 Boxcar algorithm

This is a technique which uses straight line interpolation for data compression. In this algorithm the current value of a variable is compared to the last recorded value of the same variable. If the difference is greater than or equal to the recording limit of that variable, the previous input value processed is recorded, not the current value which triggered the recording. The recording limit is a hard coded value which is set by the user. The recorded value is initialized to the last "current value" on the previous processing cycle or the starting value of the variable being processed. The performance of this algorithm is obviously dependent on the value set for the recording limit. The principle of the box car algorithm is shown in Figure 2.2.

The Matlab driver program for running the boxcar algorithm is included in the Appendix. The driver program is called ***boxcar.m*** and the heart of the code is a function called ***car.m***.

Figure 2.3 illustrates the performance of the box car algorithm for various recording limits of 0.005, 0.0095, 0.015, 0.02. From these figures it is evident that for small values of the recording limit, e.g. 0.0095, though a substantial amount of the high frequency part of the signal is eliminated, a small but appreciable amount of high frequency is still retained. Moreover, the processed signal obtained is not smooth; it retains the shape of a spline. For a high value of the recording limit, above 0.015, the basic trend of the signal is distorted. In all four cases, it would not be possible to extract the true trend of the signal from a small number of samples of the boxcar approximation.

2.3.1.2 Backward slope algorithm

The backward slope algorithm uses the last two recorded values to predict the trend of the variable in the future. Figure 2.4 shows this algorithm. It is to be noted that any two recorded values need not necessarily be at consecutive time intervals. If the current value processed differs from the predicted value by a value greater than the recording limit, the

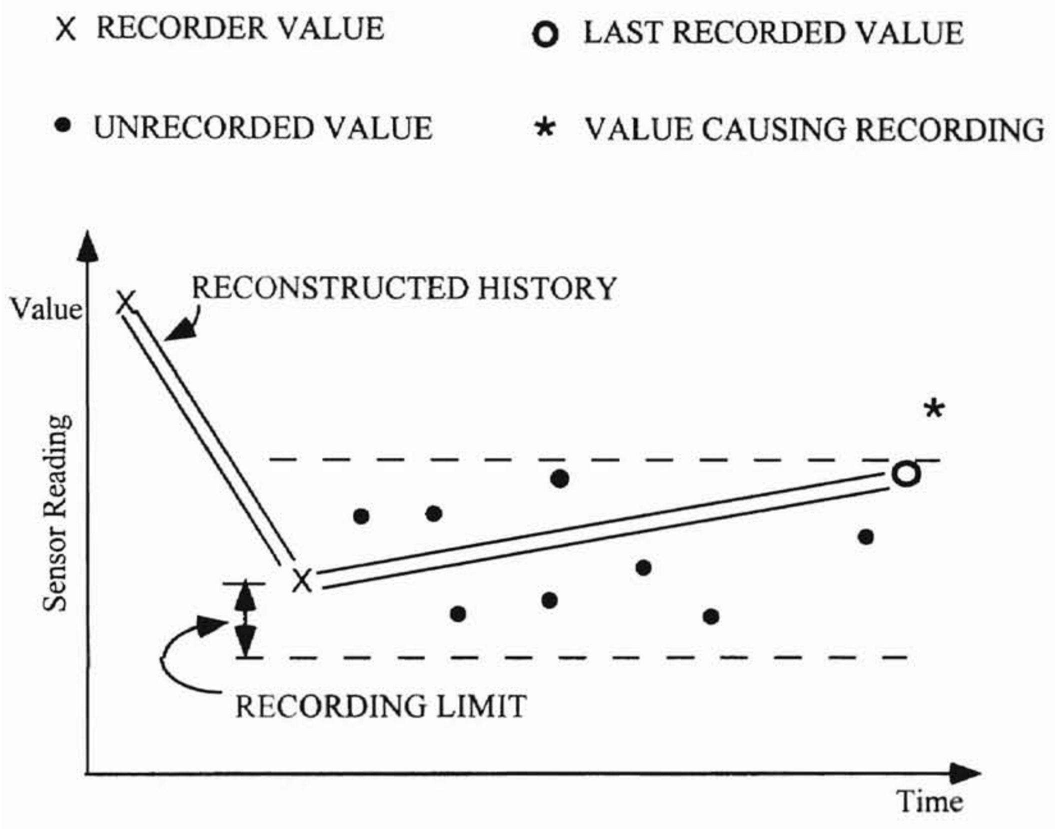
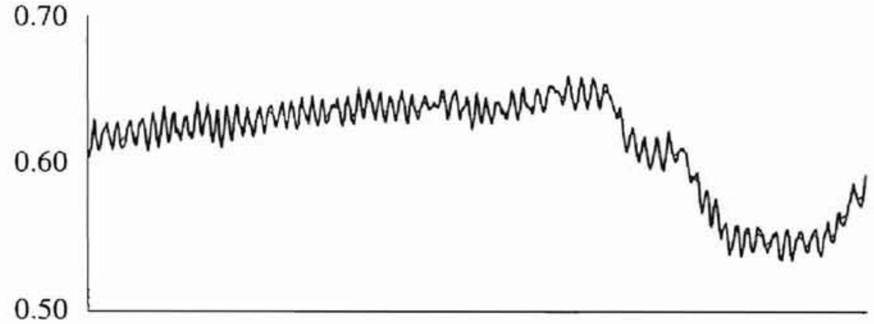
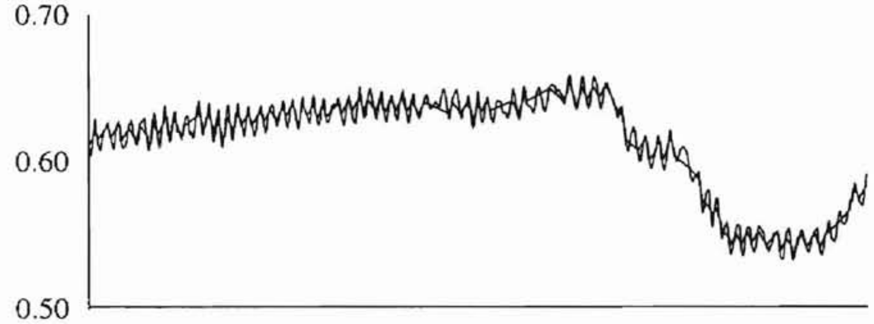


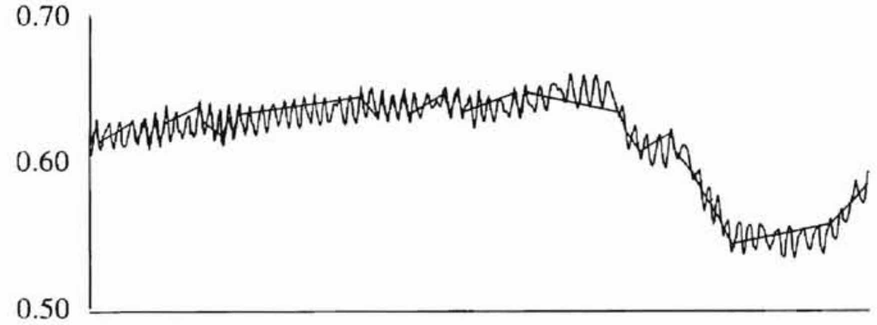
Figure 2.2. Boxcar algorithm. [From Hale (1981), page 38.]



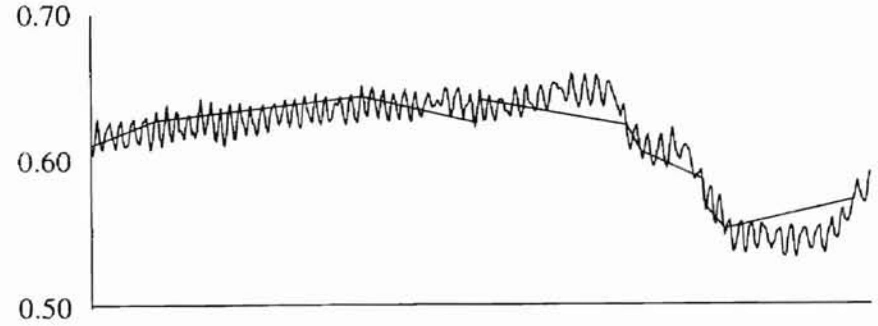
Recording limit 0.005.



Recording limit 0.0095.



Recording limit 0.015.



Recording limit 0.02.

Figure 2.3. Performance of Boxcar algorithm for a typical sensor signal.

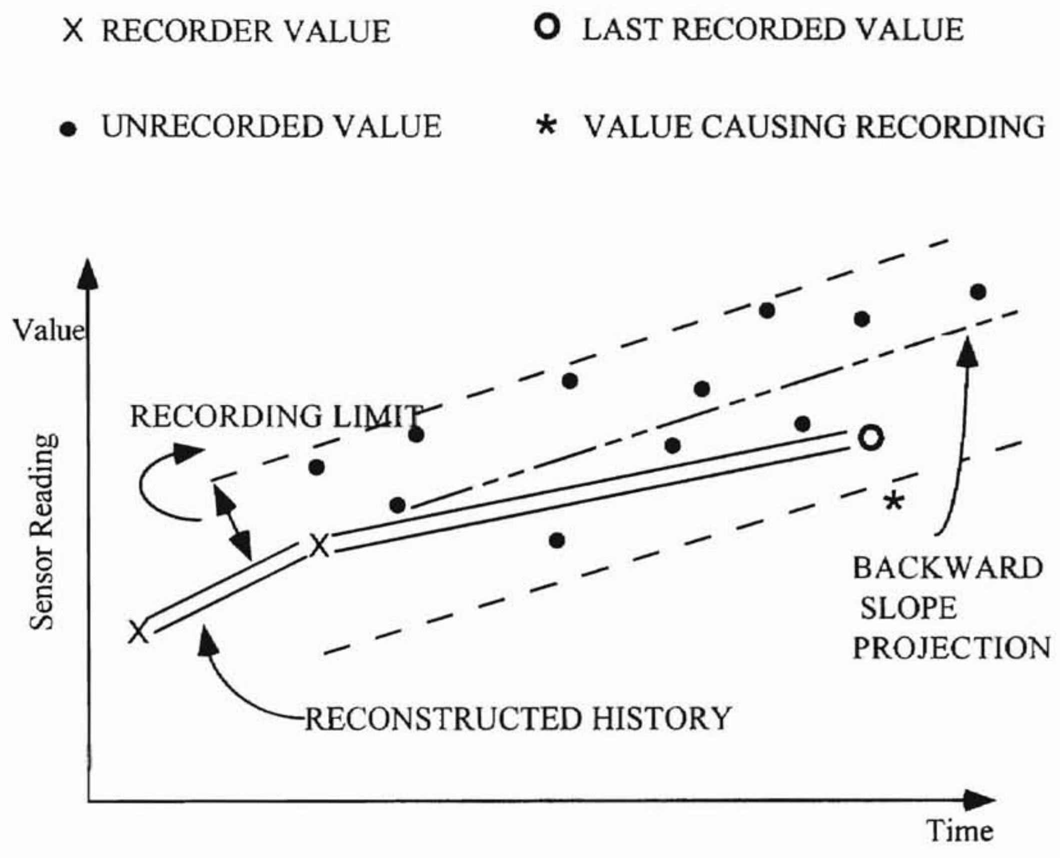


Figure 2.4. Backward slope algorithm. [From Hale (1981), page 39.]

previous value processed is recorded, not the current value responsible for triggering the recording. The first two recorded values are initialized to the first two input values in this case.

The notation to describe this algorithm mathematically is as shown below:

V	*current value of the variable*
T	*current time*
V_R	*most recent recorded value*
T_R	*time corresponding to the most recent recording*
V_L	*recorded value immediately prior to* V_R
T_L	*time corresponding to recording of* V_L
S	*slope defined as* $(V_R-V_L)/(T_R-T_L)$
V_P	*predicted value defined as*

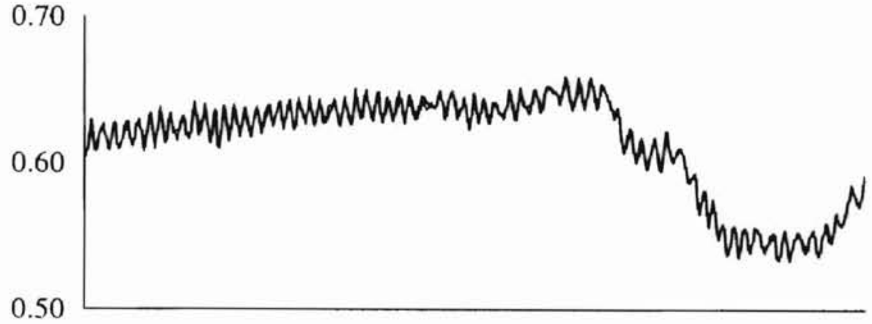
$$V_P = V_R - (V_R - V_L / T_R - T_L)(T - T_R)$$

H	*recording limit*

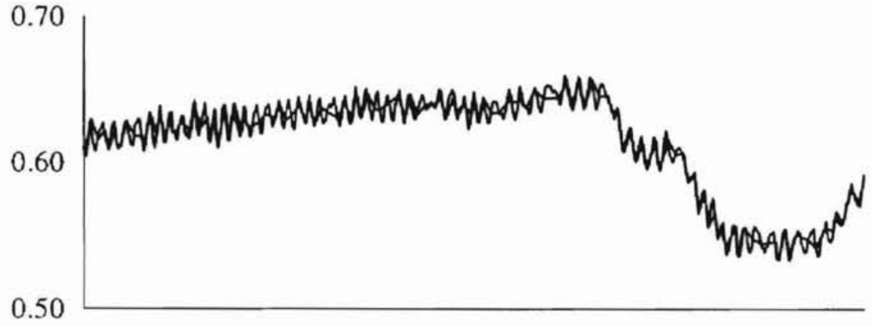
If the current value of the variable differs from the above projected value by an amount greater than or equal to H, then the previous value processed is recorded. The test condition is given by

$$\left|V - V_P\right| \geq H \qquad (5)$$

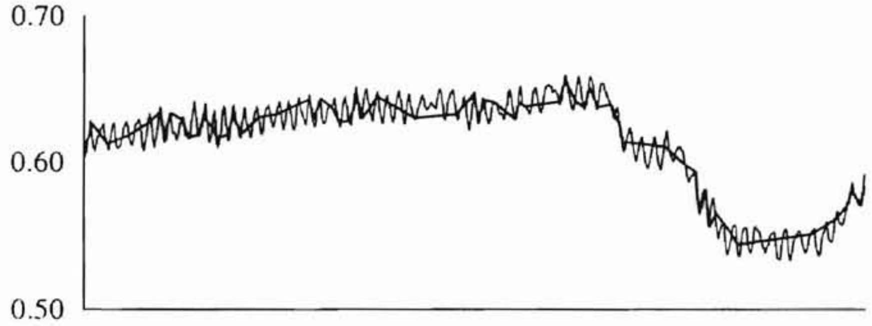
The performance of the backward slope algorithm for various recording limits is shown in Figure 2.5. The performance of the backward slope algorithm is almost the same as that of the boxcar algorithm. An appreciable amount of the high frequency content of the sensor signal is still retained for small values of the recording limit like



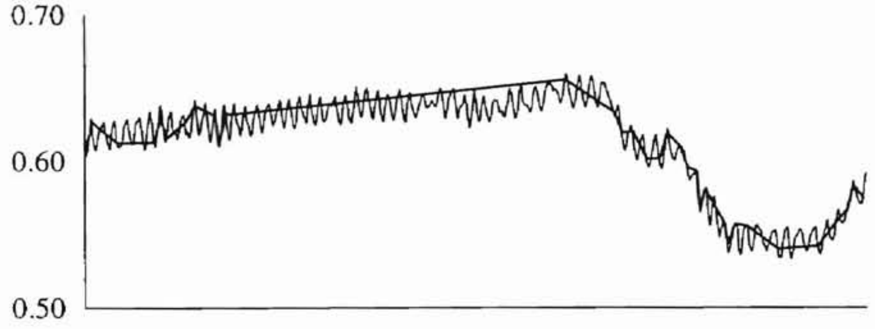
Recording limit 0.005.



Recording limit 0.0095.



Recording limit 0.0095.



Recording limit 0.02.

Figure 2.5. Performance of Backward Slope algorithm for a typical sensor signal.

0.005 and 0.0095. For high values of 0.015, 0.02 and above the fundamental trend of the processed signal tends to get distorted. The processed signal resembles a higher order spline and is not smooth.

The Matlab code, ***backslope.m*** for running the backward slope algorithm is included in the Appendix. The heart of the code is a function called ***slope.m***.

The backward slope method is much more sensitive to noise. For such cases, the boxcar algorithm remains the best choice. The best algorithm to be used depends on the variable being studied and could even vary at different points in time. The obvious choice would be to make the computer decide dynamically on the type of algorithm for each variable at any point in time. This resulted in the combination boxcar and backward slope algorithm.

2.3.1.3 Combination boxcar and backward slope algorithm

This algorithm combines the two previous algorithms by using an adaptive parameter, P. P is initialized to zero and remains unchanged as long as both the boxcar and backward slope test conditions are satisfied. If the backward slope test is not satisfied but the boxcar test is, P is set to one, and the algorithm reverts to the boxcar method until a recording is made. After this recording, the combination algorithm is reinitialized by setting P to zero. If the boxcar test is not satisfied but the backward slope test is, P is set to two, and the algorithm reverts to the backward slope method until a recording is made. After this recording, the combination algorithm is again reinitialized by setting P to zero. If both test conditions are not satisfied, P retains the value zero and recording is not performed. The principle of the combination algorithm is illustrated in Figure 2.6.

The Matlab code for the combination boxcar and backward slope algorithm, ***comb.m*** is included in the Appendix.

Figure 2.7 shows the performance of the combination algorithm for various recording limits. The performance of the combination algorithm for the signal under consideration is almost the same as that of the boxcar algorithm or the backward slope algorithm. An appreciable amount of the high frequency content of the sensor signal is still retained for small values of the recording limit, e.g., 0.005 and 0.0095. For larger recording limits such as 0.02 and above the fundamental trend of the processed signal tends to get distorted. The processed signal resembles a high order spline and is not smooth.

The direct methods described previously are not suitable for pattern-based trend extraction for two reasons:

(1) Approximations are not smooth but piecewise continuous. A sparsely sampled pattern would not be representative of true trend.

(2) They require specification of a hard-coded limit which complicates rapid application development and lacks flexibility to accommodate changes in the level of noise which can occur.

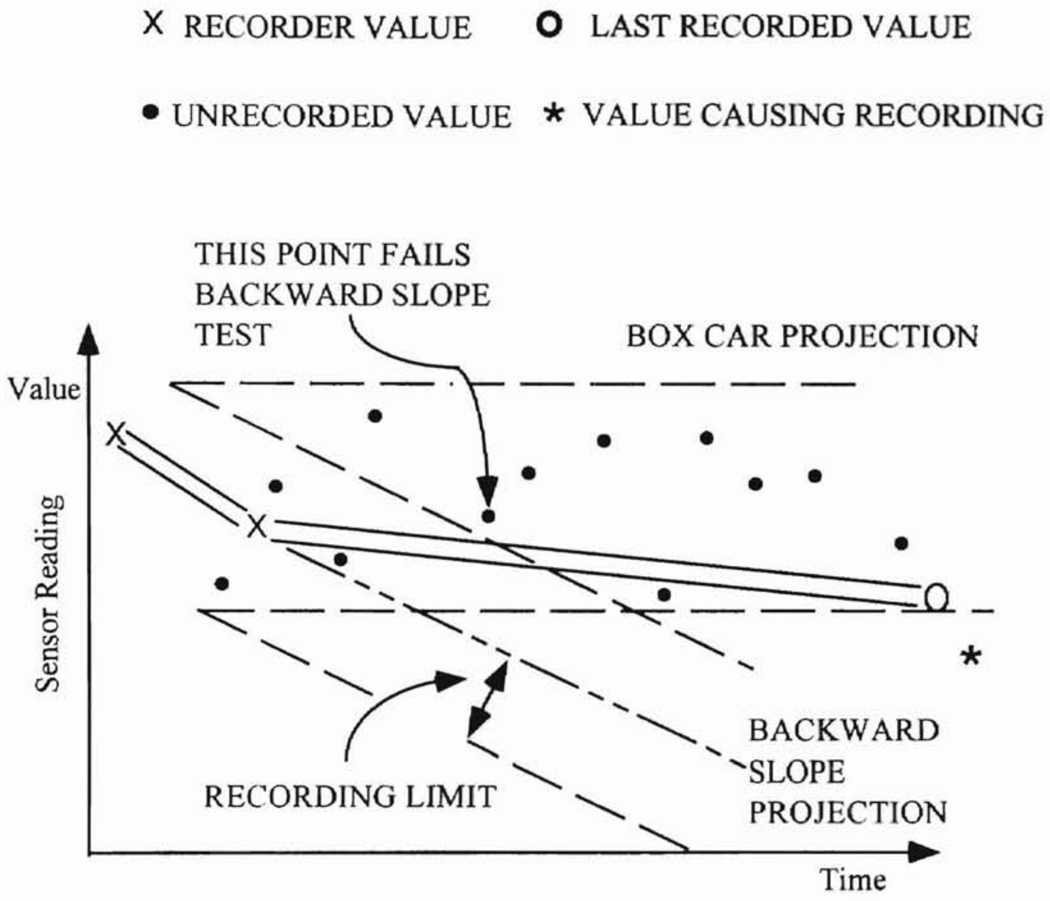
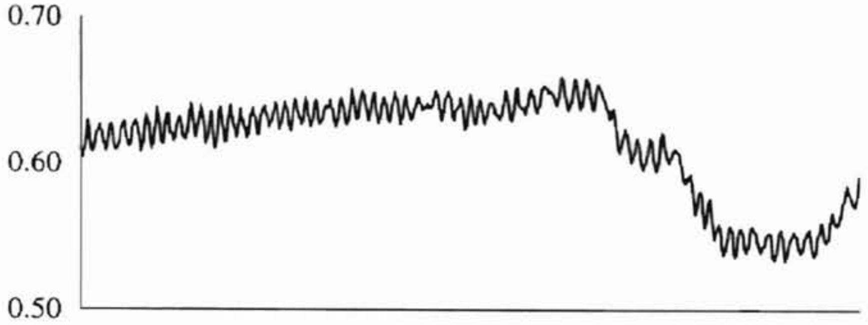
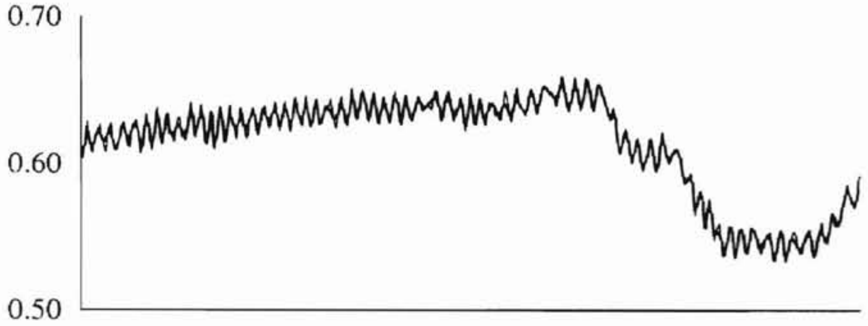


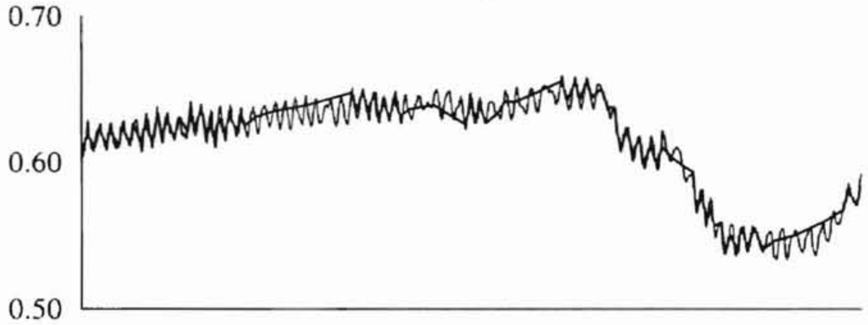
Figure 2.6. Combination Backward slope and Boxcar algorithm.
[From Hale (1981), page 40.]



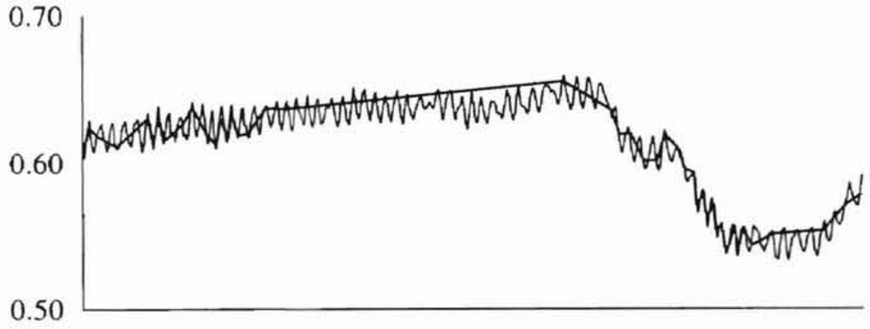
Recording limit 0.005.



Recording limit 0.009.



Recording limit 0.015.



Recording limit 0.02.

Figure 2.7. Performance of Combination backward slope and box car algorithm.

2.3.2 Digital filters

Digital filters represent an alternate approach to the data compression methods described previously. A digital filter can be visualized as a device that removes specific frequency components from a signal. Several popular digital filters designed to eliminate high frequency effects are considered in this section [Seborg, 1989]. A more comprehensive treatment of digital filtering and signal processing is available in [Oppenheim, 1975].

2.3.2.1 Single exponential filter

This filter is a digital version of the analog exponential filter which is often used to damp out high-frequency oscillations due to electrical noise; hence it is called a low-pass filter. The operation of this filter is described by a first order differential equation,

$$\tau_F \frac{dy(t)}{dt} + y(t) = x(t) \tag{6}$$

where x is the raw sensor value which is the input to the filter, y is the filtered value, and τ_F is the time constant of the filter.

For the digital version of the exponential filter, samples of the sensor signal are denoted as x_n, x_{n-1},...and the corresponding filtered values as y_n, y_{n-1}, where n refers to the current sampling instant. Δt is defined as the sampling interval i.e., the duration between any two sampling instants. The derivative in (6) can be approximated by the backward difference formula

$$\frac{dy}{dt} \cong \frac{y_n - y_{n-1}}{\Delta t} \tag{7}$$

Substituting (7) in (6) and replacing *y(t)* and *x(t)* by y_n and x_n, respectively, and on rearranging the following equation is obtained

$$y_n = \frac{\Delta t}{\tau_F + \Delta t} x_n + \frac{\tau_F}{\tau_F + \Delta t} y_{n-1} \tag{8}$$

Defining

$$\alpha \cong \frac{1}{\tau_F / \Delta t + 1} \tag{9}$$

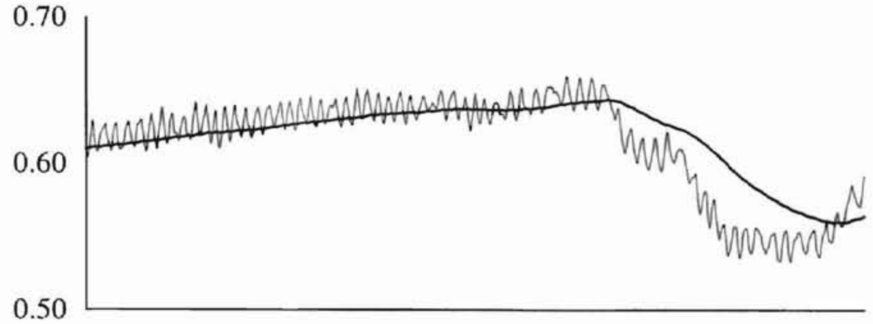
where $0 < \alpha \leq 1$, equation (9) becomes

$$y_n = (\alpha) x_n + (1 - \alpha) y_{n-1} \tag{10}$$

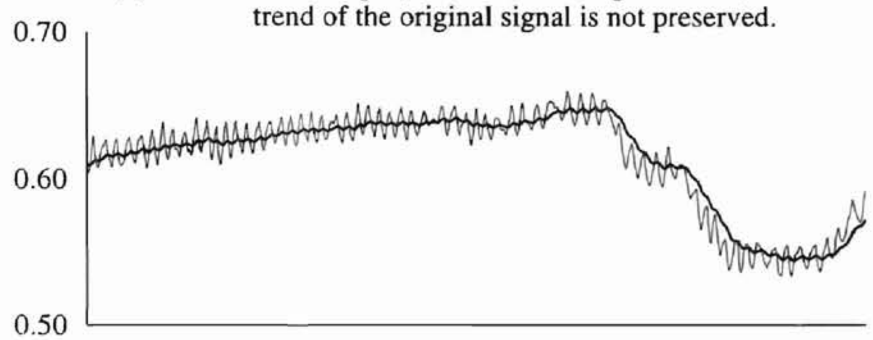
Equation (10) indicates that the filtered value of the raw signal is a weighted sum of the current measurement x_n and the filtered value at the previous sampling instant, y_{n-1}. The limiting cases for α are

$\alpha = 1$:	No filtering (the filtered output is the raw sensor value) as $\tau_F \rightarrow 0$
$\alpha \rightarrow 0$:	The measurement is ignored as $\tau_F \rightarrow \infty$

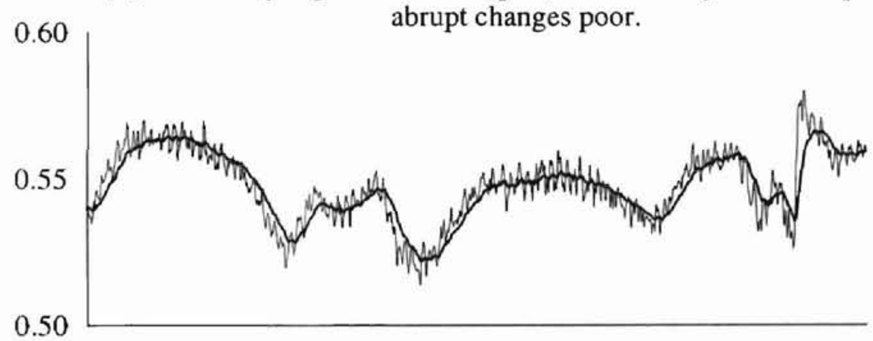
The action of this filter is performed by the Matlab code ***singexpfil.m***, a copy of which is included in the Appendix. Figures 2.8a, 2.8b, 2.8c and 2.8d show two signals with different characteristics, and their filtered signals for different values of α. Depending on the value of α used, the filtered signal may be smooth at the expense of preserving the fundamental trend of the original signal, or it may retain noise. For low values of α the smoothing obtained may be desirable, but the fundamental trend of the smoothed signal is entirely different from that of the original signal. The "events" or trends in the smoothed signal are found to occur much later than their occurrence in the original signal because of the averaging effect. This performance is undesirable. For relatively high values of α,



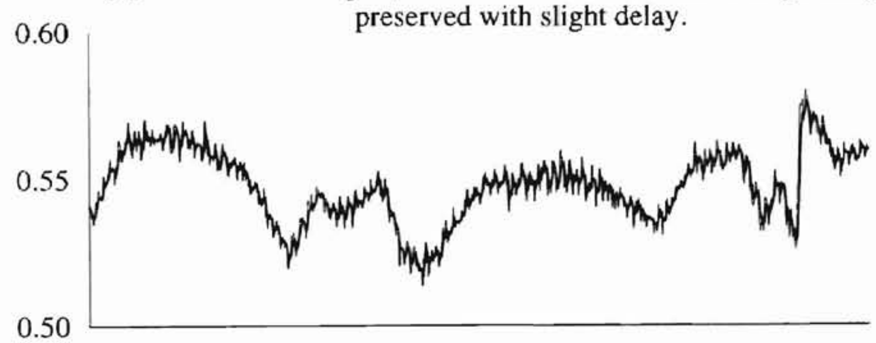
(a) Low value of alpha, 0.01. Smoothing desirable, fundamental trend of the original signal is not preserved.



(b) Relatively higher value of alpha, 0.05. Ability to catch up with abrupt changes poor.



(c) Low value of alpha, 0.01. Fundamental trend of original signal preserved with slight delay.



(d) Higher value of recording limit, 0.05. High frequency noise is retained.

Figure 2.8. Performance of the Single Exponential filter for typical sensor signals.

the trend of the smoothed signal is closer to the trend. However, the smoothed signal still retains some high frequency noise and is not smooth enough. Further, the capability of the algorithm to record abrupt changes in the smoothed signal is limited. The changes in the trend in the original signal are recorded at a much later time in the smoothed signal because of the averaging effect of α. The of performance of this filter is not acceptable for real-time pattern-based process monitoring.

2.3.2.2 Double exponential filter

The double exponential filter is an improvement over the single exponential filter for removing high frequency noise. This is a second order filter which is essentially equivalent to two first order filters in series where the second filter treats the output signal from the exponential filter in (10). The second filter can be expressed as

$$y'_n = (\gamma)y_n + (1-\gamma)y'_{n-1} \tag{11}$$

$$y'_n = (\gamma)(\alpha)x_n + \gamma(1-\alpha)y_{n-1} + (1-\gamma)y'_{n-1} \tag{12}$$

Writing the filter equation (11) for the previous sampling instant yields

$$y'_{n-1} = (\gamma)y_{n-1} + (1-\gamma)y'_{n-2} \tag{13}$$

Solving for y_{n-1} from (13), substituting in (12), rearranging, and selecting $\gamma = \alpha$ gives

$$y'_n = \alpha^2 x_n + 2(1-\alpha)y'_{n-1} - (1-\alpha)^2 y'_{n-2} \tag{14}$$

The advantage of the double exponential filter over the single exponential filter is that it provides better filtering of high frequency noise, especially if $\gamma = \alpha$.

The Matlab code for this filter is ***dubexpfil.m*** and is included in the Appendix.

Figures 2.9a, 2.9b, 2.9c and 2.9d show two signals with different characteristics and their filtered signals, for different values of α. Depending on the value of α used, the filtered signal may be smooth at the expense of preserving the fundamental trend of the original signal, or it may retain noise. For low values of α the smoothing obtained may be desirable, but the initial response of the filter is in the opposite direction and the fundamental trend of the filtered signal is totally distorted. This could be explained by equation (14). The processed value at any instant of time is dependent on the weighted sum of the value of the original signal at that time and the processed values at the previous two instants of time. This inverse response performance could lead to a wrong diagnosis of the plant situation if the filtered signal were used for process monitoring purposes. For a relatively higher value of α, the filtered signal retains some high frequency noise and is unacceptable for pattern-based process monitoring purposes.

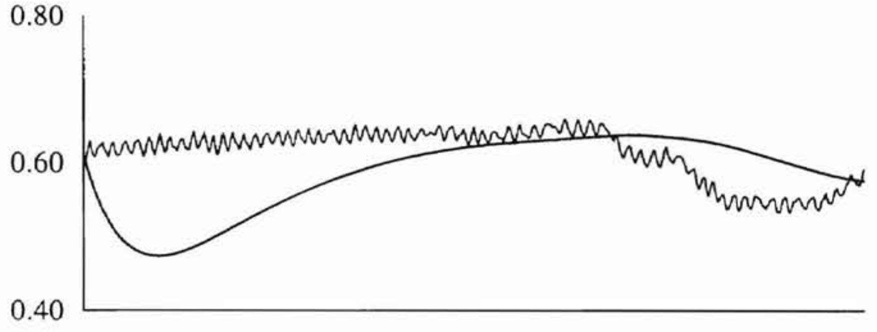
2.3.2.3 Moving average filter

This filter averages a specified number of past data points, by giving equal weight to each data point. The moving-average filter is usually less effective than the exponential filter , which gives more weight to the most recent data.

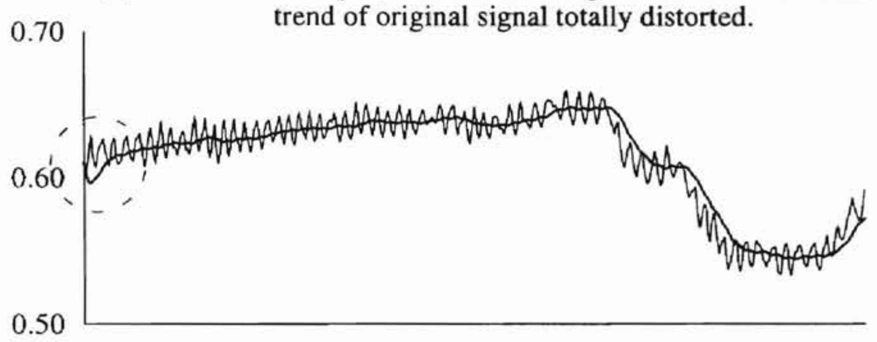
The moving-average filter can be expressed as

$$y_n = \frac{1}{J}\sum_{i=n-J+1}^{n} x_i \tag{15}$$

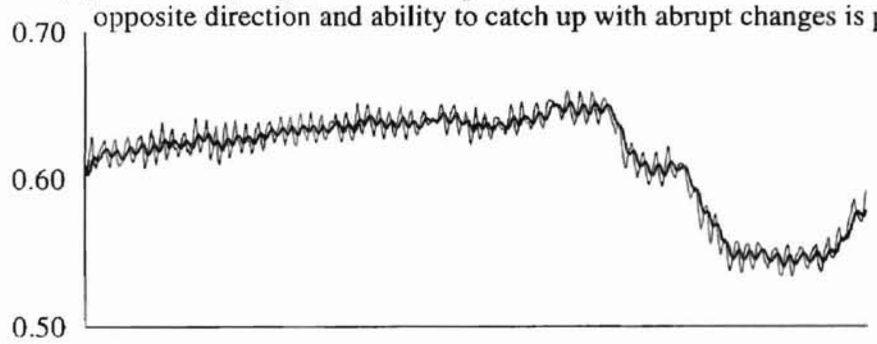
where J is the number of past data points that are being averaged. The previous filtered value y_{n-1} can be expressed as



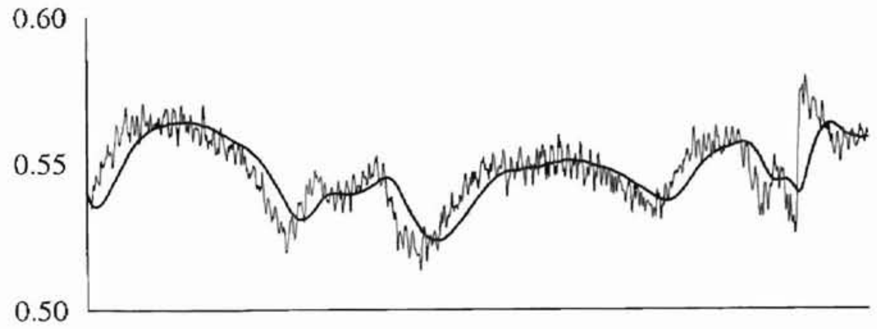
(a) Low value of alpha, 0.01. Smoothing desirable but fundamental trend of original signal totally distorted.



(b) Relatiively higher value of alpha, 0.1. Initial trend of filtered signal in opposite direction and ability to catch up with abrupt changes is poor.



(c) Higher value of alpha, 0.25. High frequency noise retained.



(d) Low value of alpha, 0.01. Fundamental trend of filtered signal preserved but delayed in time.

Figure 2.9. Performance of the Double Exponential filter for various values of alpha.

$$y_{n-1} = \frac{1}{J} \sum_{i=n-J}^{n-1} x_i \qquad (16)$$

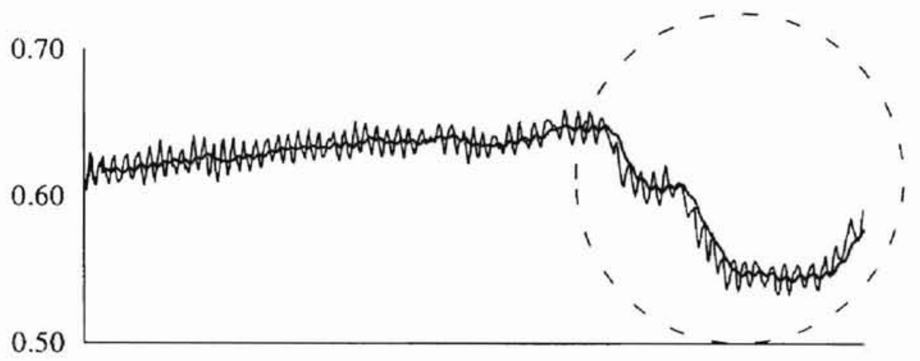
Subtracting (16) from (15) gives the recursive form of the moving-average filter:

$$y_n = y_{n-1} + \frac{1}{J}(x_n - x_{n-J}) \qquad (17)$$

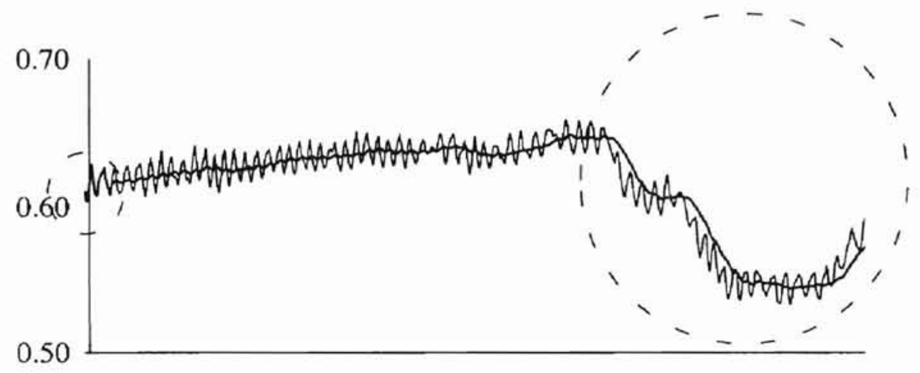
The Matlab code for this filter is ***mov_ave_fil.m***, a copy of which is included in the Appendix. The performance of this filter is shown for various values of J in Figure 2.10. The first J data points of the original signal are initialized as filtered values. For low values of J, the high frequency noise of the original signal is retained. The response of this filter is too slow to accurately capture abrupt changes in the trend of the original signal. These regions are indicated by the dotted circle in Figure 2.10. For higher values of J, the fundamental trend gets distorted. Figures 2.10a, 2.10b and 2.10c show two signals with different characteristics, and their filtered signals for different values of J. Depending on the value of J used, the filtered signal may be smooth at the expense of preserving the fundamental trend of the original signal, or it may retain noise. Further, the first J values of the filtered signal are the same as that of the original signal. The above reasons make the filtered signals obtained using this filter unacceptable for pattern-based monitoring purposes.

2.3.2.4 Rate of change (Noise spike) filter

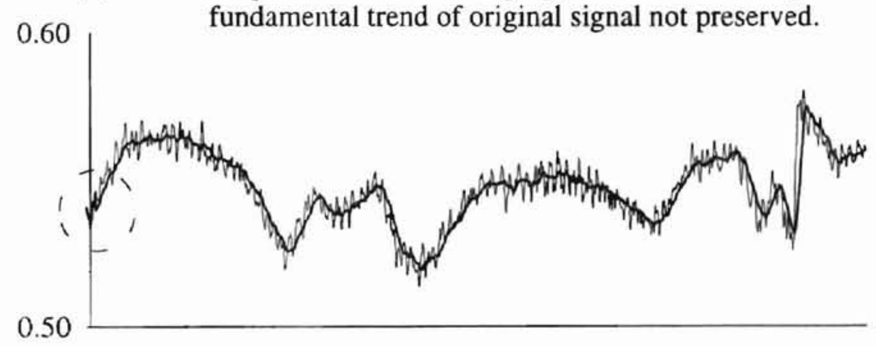
If a noisy measurement changes suddenly by a large amount and then returns to the original value at the next sampling instant, or close to it, a noise spike is said to have occurred. This filter is designed to eliminate such noise spikes. These filters are used to



(a) Past data points used for averaging, J = 25. Filtered signal not very smooth.



(b) Past data points used for averaging, J = 35. Filtered signal smooth, fundamental trend of original signal not preserved.



(c) Past data points used for averaging, J = 100. High frequency noise retained.

Figure 2.10. Performance of the Moving Average filter.

limit how much the filtered output is permitted to change from one sampling instant to the next. If *Δx* is the maximum allowable change, the noise spike filter can be written as

$$y_n = \begin{cases} x_n & \text{if} \quad \left| x_n - y_{n-1} \right| \le \Delta x \\ y_{n-1} - \Delta x & \text{if} \quad y_{n-1} - x_n \rangle \Delta x \\ y_{n-1} + \Delta x & \text{if} \quad y_{n-1} - x_n \langle -\Delta x \end{cases} \qquad (18)$$

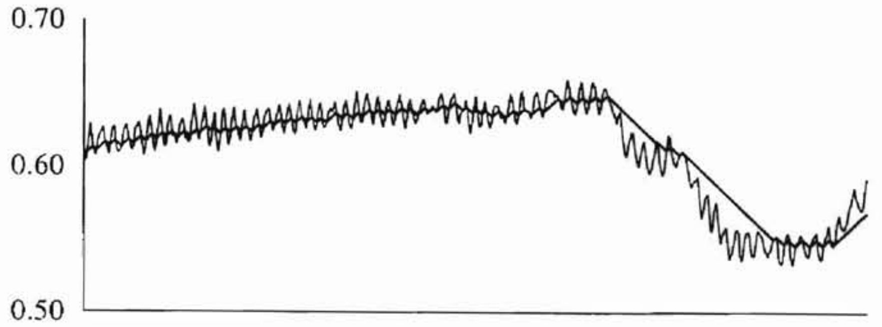
If a large change in the measurement occurs, the filter replaces the measurement by the previous filter output plus (or minus) the maximum allowable change.

The Matlab code for this filter is ***rate_of_ch_fil.m,*** a copy of which is included in the Appendix.

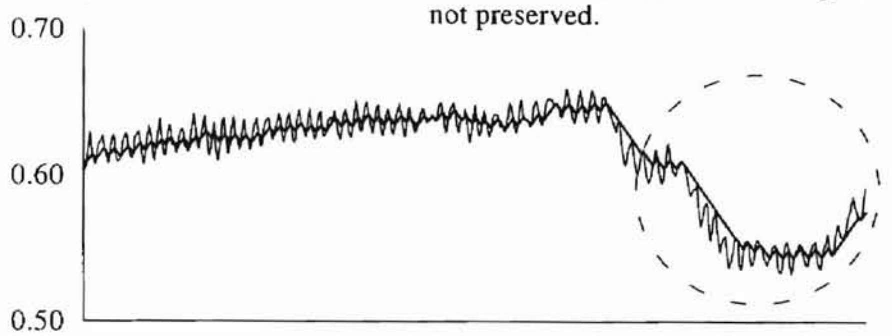
The performance of this filter for two signals with different characteristics is shown in Figure 2.11. The first signal and its overlapped filtered output are shown in Figures 2.11a and 2.11b for Δx equal to 0.0005 and 0.00075 respectively. The second signal and its overlapped filtered output are shown in Figures 2.11c and 2.11d for Δx equal to 0.00009 and 0.002 respectively. Depending on the value of Δx used, the filtered signal may be smooth at the expense of preserving the fundamental trend of the original signal, or it may retain noise. A smoothed signal preserving the fundamental trend of the original signal is not obtained using this filter. Consequently, this filter is not acceptable for smoothing signals for pattern-based data analysis.

2.4 Performance using wavelets

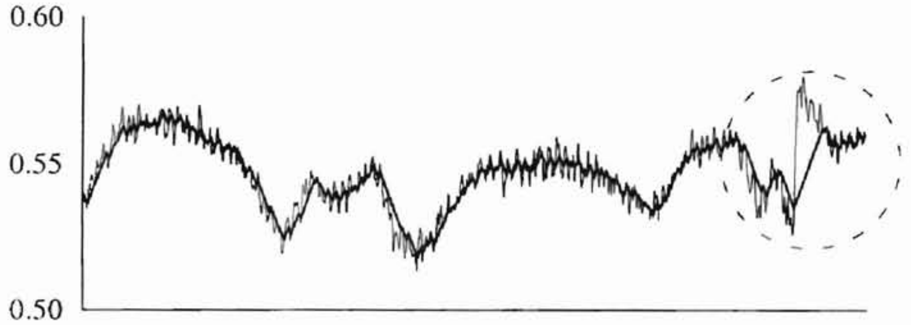
The basic problem with conventional methods described previously is the inability to reject high frequency noise while simultaneously retaining the ability to capture abrupt



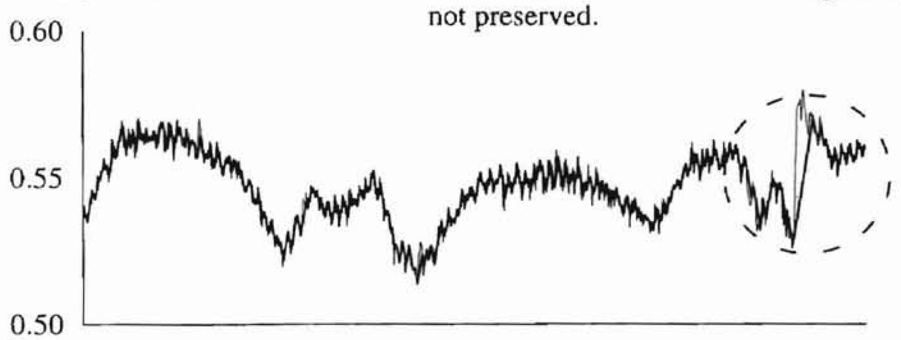
(a) Low value of Delta x = 0.0005. Fundamental trend of original signal not preserved.



(b) Delta x = 0.00075, ability to catch up with abrupt changes poor.



(c) Low value of Delta x = 0.00009. Fundamental trend of original signal not preserved.



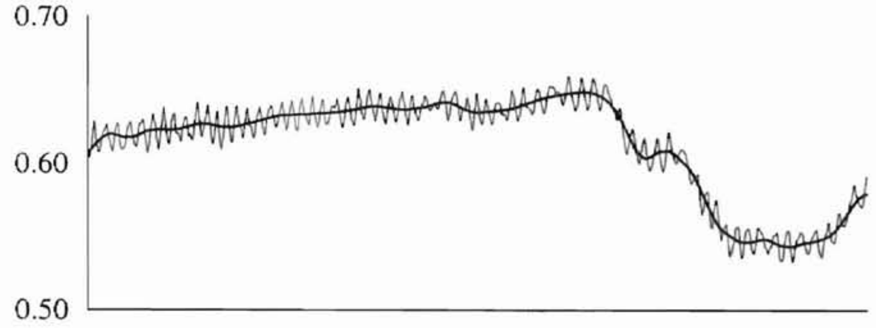
(d) Delta x = 0.002, high frequency noise retained.

Figure 2.11. Performance of the Rate of Change filter.

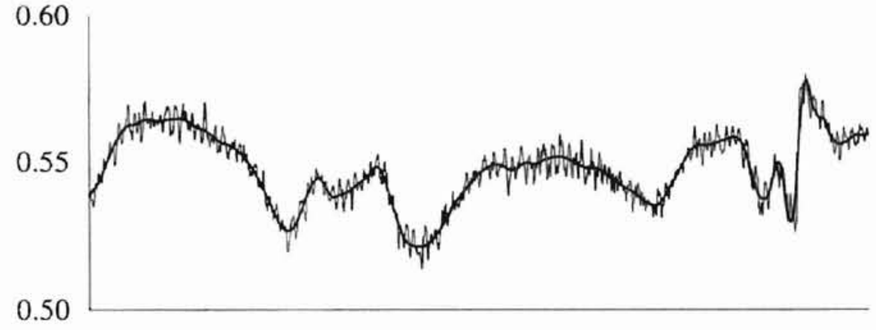
changes, which are genuine. Wavelets can achieve this. For demonstration purposes Figure 2.12 shows the same signals considered in the above techniques smoothed using wavelets. The technical discussion on wavelets is presented in the following chapter. The intent here is to simply show the reader that a superior technique exists and to offer visual proof.

Summary of this chapter

This chapter presented an overview of the conventional methods of signal processing and explains why they are not suitable for accurate pattern-based process monitoring. Moreover, the conventional techniques involve the use of parameters that need to be hard-coded by the user. Further experimentation is needed to find the optimum values of these parameters which give as close to the desired performance as possible. Unfortunately, even when optimum values of these parameters are selected, the conventional techniques do not perform as well as wavelet smoothing. Signal processing using wavelets is highly desirable because of the high quality of resolution obtained. Moreover, this method does not involve hard coded filter constants or recording limits. The degree of smoothing can be adjusted by simply altering the number of levels of decomposition of the signal. The next chapter describes the mechanics of wavelet smoothing, with a brief introduction to the Fourier transform.



Original signal and smoothed approximation obtained using sixth order Daubechies wavelets at the fifth level of decomposition.



Original signal and its smoothed approximation obtained using sixth order Daubechies wavelets at the seventh level of decomposition.

Figure 2.12. Performance of sixth order Daubechies wavelets.

Chapter 3

THE FOURIER AND WAVELET TRANSFORMS

This chapter describes the fundamentals associated with two different signal processing techniques which we propose to use at OSU for automated trend extraction. The first technique employs a Fourier or frequency-domain decomposition of a time-domain signal. The material covered serves two purposes. The first is to describe the analytic approach used to characterize a time-domain signal. The second is to provide an introduction to the primary signal processing technique that we use for automated trend extraction, wavelet signal decomposition. This chapter lays the foundation for the automated trend extraction procedure proposed in the following chapter.

3.1 Digital Signal analysis and processing

The study of a signal as a function of time is called time analysis. Such an analysis can provide information about the source and medium of propagation. However, if other properties of the signal are to be studied, it would be better to study the signal using a different representation. One of the most powerful representations is the spectral representation, which is generated using a frequency-domain analysis. For stationary signals, i.e., signals whose properties are statistically invariant over time, the most common method to perform a frequency analysis is the Fourier transform. The Fourier transform coefficients give a clear picture of the frequency content of the signal. This method works well when the signal is fundamentally composed of a few periodic components. However, if the signal is non-stationary, i.e., exhibits abrupt changes due to unexpected transient events, a Fourier decomposition yields a wide range of frequency components as analytically required to reproduce abrupt changes of the signal in time. Many of the frequency components are not representative of the process which is generating the signal but are simply analytical artifacts required to approximate sharp transitions. Therefore an analysis of non-stationary signals requires more than the Fourier transform.

Two techniques commonly employed for analysis of non-stationary signals include wavelets decomposition and the Short-Time Fourier transform. The latter analyzes a signal in (data) windows of fixed length, using the Fourier transform, to determine the dominant frequencies prevalent in the signal. A recently developed technique is the wavelet transform [Rioul, 1991]. The Short-Time Fourier transform uses a single analysis window whereas the wavelet transform uses short windows (contraction) at high frequencies and long windows (dilation) at low frequencies. The contractions and dilations are called scalings. The notion of scale in wavelet transform could be envisaged as an alternative to frequency in Short-Time Fourier transform. This effectively means that a signal is mapped into a time-scale plane in wavelet transform as compared to the time-frequency plane in the Short-Time Fourier transform. The most attractive feature of the wavelet transform is that a signal can be analyzed at various scales and resolutions. Thus, an explicit representation of the temporal features of a signal in a joint time-frequency plane can be obtained.

3.1.1 Signal Decomposition

Decomposition of a given signal into a weighted sum of basis functions is a convenient way of solving many engineering problems. Fourier and wavelet decompositions both utilize this approach. When a decomposed signal is input to a linear, time invariant system, then superposition theory can be applied and the output obtained by adding up the weighted responses to each of the basis functions. Generally, a signal can be decomposed into the sum of a pair of basis functions $\alpha(t)$ and $\beta(t)$.

$$f(t) = \sum_k g_k \alpha_k(t) + \sum_k h_k \beta_k(t) \tag{19}$$

Here, k is the index defining the length of the signal. Various basis functions could be adopted depending on the nature of the problem at hand. In equation (19), g_k and h_k are the decomposition coefficients associated with the basis functions $\alpha(t)$ and $\beta(t)$ respectively.

3.2 The Fourier transform

When the basis functions of equation (19) are represented by the traditional cosine and sine functions, the Fourier series representation is obtained. The Fourier transform, when applied to any signal, decomposes the signal into sine and cosine coefficients of various magnitude for each of the frequencies covered. The original signal can be perfectly reconstructed by using all the sine and cosine coefficients in equation (19).

In this section, the Fourier series expansion for continuous-time periodic signals is reviewed, followed by a discussion of the Fourier series expansion for discrete-time signals. Next, the definition and properties of the discrete Fourier transform (DFT) using the fast Fourier transform, an efficient method for computing the discrete Fourier transform, are discussed. Linear filtering and spectral analysis are addressed by interpreting DFT results in terms of frequency content of discrete-time signals and by using the DFT to perform linear convolution, which is the basic filtering operation.

3.2.1 Continuous-time Fourier series

A periodic, continuous time signal *f(t)* with period *T* can be expressed as a weighted sum of a countable number of complex exponential continuous time functions in the following way [Ludeman, 1986].

$$f(t) = \sum_{k=-\infty}^{k=\infty} F_k e^{jkt\Omega_0} \qquad \text{for all t} \qquad (20)$$

The expansion given by equation (20) is called the exponential Fourier series representation of a continuous-time periodic signal *f*(t). Here, the complex term $e^{jkt\Omega_0}$ implicitly represents the sine and cosine basis functions by virtue of the definition

$$e^{jkt\Omega_0} = \cos(k\Omega_0 t) + j\sin(k\Omega_0 t) \qquad (20a)$$

where *j* is the complex variable with the property $j^2 = -1$. In equation (20), F_k are the coefficients of the expansion, and Ω_0, the fundamental frequency. They are determined by the following equations:

$$F_k = \frac{1}{T}\int_0^T f(t) e^{-jk\Omega_0 t} dt \qquad (21)$$

$$\Omega_0 = 2\pi/T \qquad (21a)$$

A convenient trigonometric representation of equation (21) in terms of a weighted sum of sine and cosine basis functions is

$$f(t) = a_0 + \sum_{k=1}^{\infty} (a_k \cos k\Omega_0 t + b_k \sin k\Omega_0 t) \tag{22}$$

where the constants a_0, a_k, and b_k can be determined as

$$a_0 = \frac{1}{T}\int_0^T f(t)dt \tag{23a}$$

$$a_k = \frac{2}{T}\int_0^T f(t)\cos k\Omega_0 t\, dt \quad k = 1,2,\ldots\ldots\ldots\ldots\infty \tag{23b}$$

$$b_k = \frac{2}{T}\int_0^T f(t)\sin k\Omega_0 t\, dt \quad k = 1,2,\ldots\ldots\ldots\ldots\infty \tag{23c}$$

The constants a_k, and b_k, of the trigonometric form, and F_k, of the complex exponential form are related as follows:

$$a_0 = F_0 \tag{24a}$$

$$a_k = F_k + F_{-k} \tag{24b}$$

$$b_k = (F_{-k} - F_k)/\mathrm{j}, \qquad k=1, 2, \ldots\ldots\ldots \tag{24c}$$

3.2.2 Discrete-time Fourier series

A real, periodic discrete-time signal $x(n)$ of period N can be expressed as a weighted sum of complex exponential sequences. By virtue of the fact that sinusoidal sequences are unique only for digital frequencies from 0 to 2π, the expansion contains only a finite number of complex exponentials as follows:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{jk\omega_0 n} \tag{25}$$

where the coefficients of the expansion $X(k)$ and the fundamental digital frequency ω_0 are given by

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-jk\omega_0 n} \qquad \textit{for all } k \tag{26a}$$

$$\omega_0 = 2\pi/N \tag{26b}$$

Equations (25) and (26a) are called the discrete Fourier transform series (DFS). The expression given in (25) is referred to as the exponential form of the Fourier series for a periodic discrete-time signal. Equation (26) is sometimes written in the following equivalent form:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kN} \tag{26c}$$

where $W_N = e^{-j2\pi/N}$ (26d)

An alternative form of the discrete Fourier series, for a periodic discrete-time signal in terms of the sine and cosine basis functions can be written as shown below. This is called the trigonometric form of the Fourier series and is represented by two equations , for odd and even N.

Even N

$$x(n) = A(0) + \sum_{k=1}^{N/2-1} A(k)\cos(k\frac{2\pi}{N}n) + \sum_{k=1}^{N/2-1} B(k)\sin(k\frac{2\pi}{N}n) + A(\frac{N}{2})\cos n\pi \tag{27}$$

The last term contains $\cos n\pi$, which is just $(-1)^n$, and is associated with the highest frequency possible.

Odd N

$$x(n) = A(0) + \sum_{k=1}^{(N-1)/2} A(k)\cos(k\frac{2\pi}{N}n) + \sum_{k=1}^{(N-1)/2} B(k)\sin(k\frac{2\pi}{N}n) \tag{28}$$

The constants *A(0), A(k), and B(k)* for even *N* can be shown to be

$$A(0) = \frac{1}{N}\sum_{n=0}^{N-1} x(n) \tag{29a}$$

$$A(k) = \frac{2}{N}\sum_{n=0}^{N-1} x(n)\cos(k\frac{2\pi}{N}n), \qquad k=1,2,......,(N/2)-1 \tag{29b}$$

$$B(k) = \frac{2}{N}\sum_{n=0}^{N-1} x(n)\sin(k\frac{2\pi}{N}n), \qquad k=1,2, \quad ,(N/2)-1 \tag{29c}$$

$$A(N/2) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\cos n\pi \tag{29d}$$

If *N* is odd, equations (29a -c) apply but *A(N/2)* =0.

The coefficient *A(0)*, is the most dominant Fourier coefficient and represents the average value of the original signal. The coefficient *A(0)* can also be considered to be the cosine coefficient at frequency index zero (i.e., at *k = 0*). Note that *B(0)*, the sine coefficient at *k = 0* is zero and hence is not considered.

The relationships between the *A(k)*, *B(k)* of the trigonometric form, and the *X(k)* of the discrete Fourier coefficients for a real *x(n)* with an even *N* are given by

$$A(0) = X(0)/N \tag{30a}$$

$$A(k) = [X(k) + X(N - k)]/N, \qquad k = 1, 2, ..., (N/2)-1 \tag{30b}$$

$$B(k) = j[X(k) - X(N - k)]/N, \qquad k = 1, 2, ..., (N/2)-1 \tag{30c}$$

$$A(N/2) = X(N/2)/N \tag{30d}$$

Again, if N is odd, equations (30a - c) apply and $A(N/2) = 0$. The following example illustrates the complex exponentials and the trigonometric forms of the discrete Fourier representation. From Figure 3.1, *x(n)* is *[1 2 3 4]*

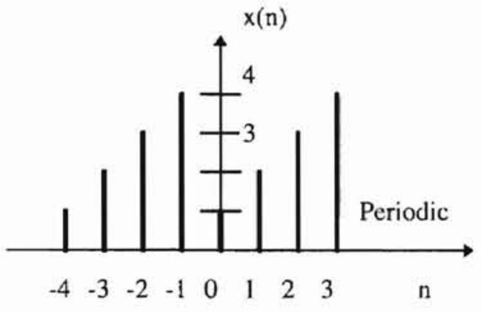


Figure 3.1. Example problem

The exponential form of the Fourier series is specified once the *X(k)* are calculated from equation (26a). Before carrying out these calculations using equation (26a), the powers of W_N must be determined for $N = 4$. Here, $N = 4$ because the period associated with this signal is *4*. W_4 is given by

$$W_4 = e^{-j(2\pi/4)} = \cos(\pi/2) - j\sin(\pi/2) = -j$$

The other powers of W_4 are easily seen to be

$$W_4^2 = W_4^1 \cdot W_4^1 = (-j)(-j) = -1$$
$$W_4^3 = W_4^2 \cdot W_4^1 = (-1)(-j) = j$$
$$W_4^4 = W_4^3 \cdot W_4^1 = (j)(-j) = 1$$

Using these powers of W_4, the *X(k)* for $k = 0, 1, 2, 3$ are calcluated as follows:

$$X(0) = \sum_{n=0}^{4-1} x(n)W_4^{0\cdot n} = x(0) + x(1) + x(2) + x(3)$$
$$= 1+2+3+4 = 10$$

$$X(1) = \sum_{n=0}^{4-1} x(n)W_4^{1\cdot n} = x(0)W_4^0 + x(1)W_4^1 + x(2)W_4^2 + x(3)W_4^3$$
$$= 1(1) + 2(-j) + 3(-1) + 4(j) = -2 + 2j$$

$$X(2) = \sum_{n=0}^{4-1} x(n)W_4^{2\cdot n} = x(0)W_4^0 + x(1)W_4^2 + x(2)W_4^4 + x(3)W_4^6$$
$$= 1(1) + 2(-1) + 3(1) + 4(-1) = -2$$

$$X(3) = \sum_{n=0}^{4-1} x(n)W_4^{3\cdot n} = x(0)W_4^0 + x(1)W_4^3 + x(2)W_4^6 + x(3)W_4^9$$
$$= 1(1) + 2(j) + 3(-1) + 4(-j) = -2 - 2j$$

Now, the trigonometric coefficients are calculated from the exponential coefficients as shown below, using equation (30). As $N = 4$, the number of cosine coefficients would be $(N/2 + 1) = 3$ and the number of sine coefficients would be $(N/2 - 1) = 1$.

$$A(0) = X(0)/N = \frac{10}{4} = \frac{5}{2}$$

$$A(1) = [X(1) + X(4-1)]/4 = [(-2+2j) + (-2-2j)]/4 = -1$$

$$B(1) = j[X(1) - X(4-1)]/4 = j[(-2+2j) - (-2-2j)]/4 = -1$$

$$A(2) = X(\frac{4}{2}) = X(2)/4 = -1/2$$

Now, *x(n)* can be determined using these trigonometric coefficients as shown, using equation. (20) for even *N*.

$$x(0) = A(0) + \sum_{k=1}^{(4/2)-1} A(k)\cos(k\frac{2\pi}{4}0) + \sum_{k=1}^{(4/2)-1} B(k)\sin(k\frac{2\pi}{4}0) + A(4/2)\cos(0\pi)$$

$$= A(0) + A(1)\cos(0\pi) + B(1)\sin(0\pi)$$

$$=(5/2) + (-1)1 + 0 + (-1/2)$$

$$=1$$

Similarly,

$$x(1) = A(0) + \sum_{k=1}^{(4/2)-1} A(k)\cos(k\frac{2\pi}{4}1) + \sum_{k=1}^{(4/2)-1} B(k)\sin(k\frac{2\pi}{4}1) + A(4/2)\cos(1\pi)$$

$$= A(0) + A(1)\cos(\pi/2) + B(1)\sin(\pi/2) + A(2)\cos(1\pi)$$

$$= (5/2) + (-1)0 + (-1)1 + (-1/2)(-1)$$

$$= 2$$

$$x(2) = A(0) + \sum_{k=1}^{(4/2)-1} A(k)\cos(k\frac{2\pi}{4}2) + \sum_{k=1}^{(4/2)-1} B(k)\sin(k\frac{2\pi}{4}2) + A(4/2)\cos(2\pi)$$

$$= A(0) + A(1)\cos\pi + B(1)\sin\pi + A(2)\cos 2\pi$$

$$= (5/2) + (-1)(-1) + (-1)0 + (-1/2)1$$

$$= 3$$

$$x(3) = A(0) + \sum_{k=1}^{(4/2)-1} A(k)\cos(k\frac{2\pi}{4}3) + \sum_{k=1}^{(4/2)-1} B(k)\sin(k\frac{2\pi}{4}3) + A(4/2)\cos(3\pi)$$

$= A(0) + A(1)\cos(3\pi/2) + B(1)\sin(3\pi/2) + A(2)\cos(3\pi)$

$= (5/2) + (-1)0 + (-1)(-1) + (-1/2)(-1)$

$= 4$

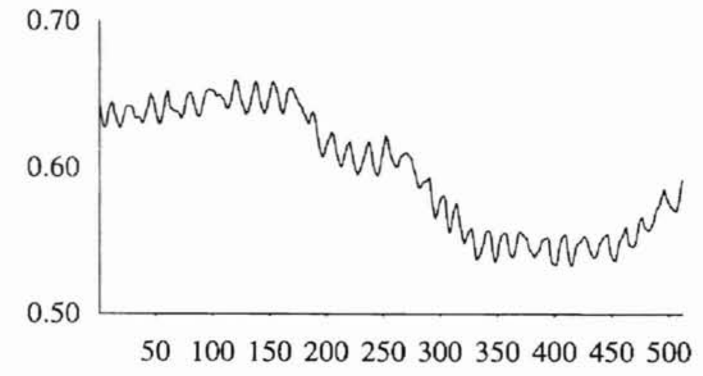
From the above calculations it is apparent that the *X(k)*'s can be written in matrix form as follows:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^0 & W_4^2 \\ W_4^0 & W_4^3 & W_4^2 & W_4^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \tag{31}$$

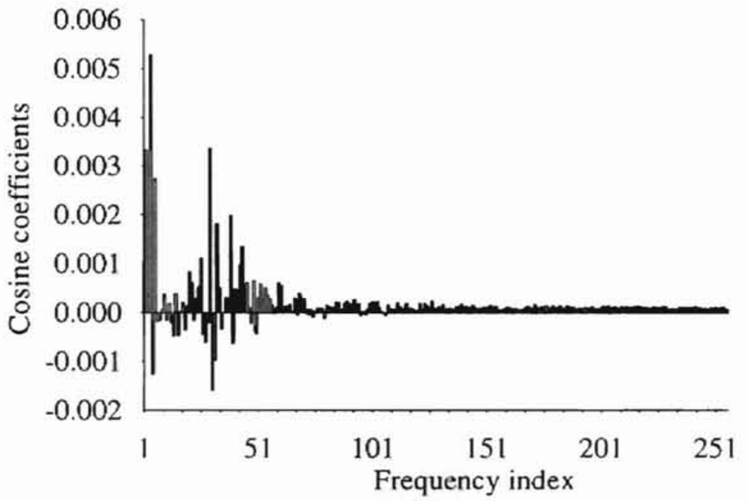
The entries in the coefficient matrix have been reduced in powers of W_4 by using the fact that $W_4^4 = 1$. Fast ways of making the calculations above can be shown to be equivalent to decomposing the square matrix into products of relatively sparse matrices. This idea is exploited further in the fast Fourier transform.

The Fourier transform is an excellent tool to determine the frequency content of a signal. Figure 3.2 shows a raw sensor signal and its Fourier decomposed sine and cosine coefficients. Figure 3.2a shows the sensor signal in the time domain i.e., the magnitude of the parameter measured at various instants of time. Figures 3.2b and 3.2c show the same signal transformed to the frequency domain by the Fourier transform. The cosine and sine coefficients obtained by performing the Fourier transform operation on the time domain representation of the sensor signal are a measure of the contribution from the different frequencies which make up the original signal.

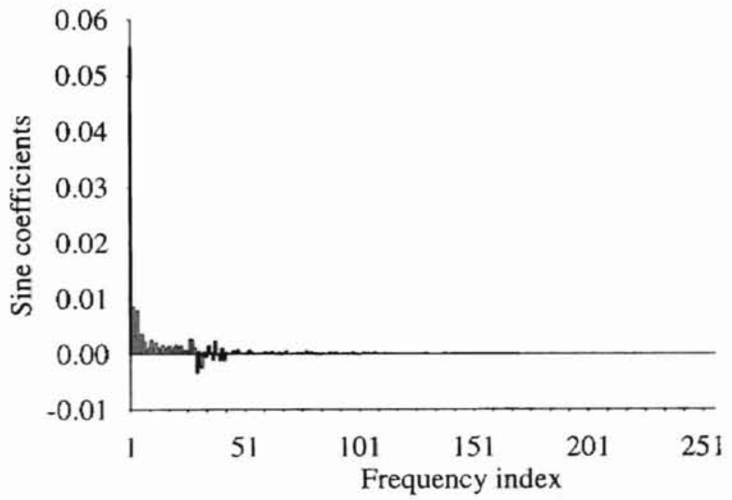
The magnitude of the average value of the signal (cosine coefficient at frequency index zero), *A(0)*, is 0.5977. The sine coefficient, *B(0)*, at frequency index zero is zero. The cosine and sine coefficients have not been drawn to the same scale so that the reader



(a). Original signal.



(b) Cosine coefficients against frequency index. Most dominant Fourier coefficient, A(0) = 0.5977, is not shown in plot.



(c) Sine coefficients against frequency index.

Figure 3.2. Original signal decomposed into cosine and sine components.

can appreciate the difference in their magnitude. From Figures 3.2b and 3.2c, it can be observed that the first few coefficients are relatively large in magnitude. These coefficients capture the amplitude and frequency of the main body of the signal. The other relatively small coefficients represent the amplitude and frequency of sensor noise introduced in the signal. Thus, the Fourier transform gives a complete picture of the frequency content of the signal.

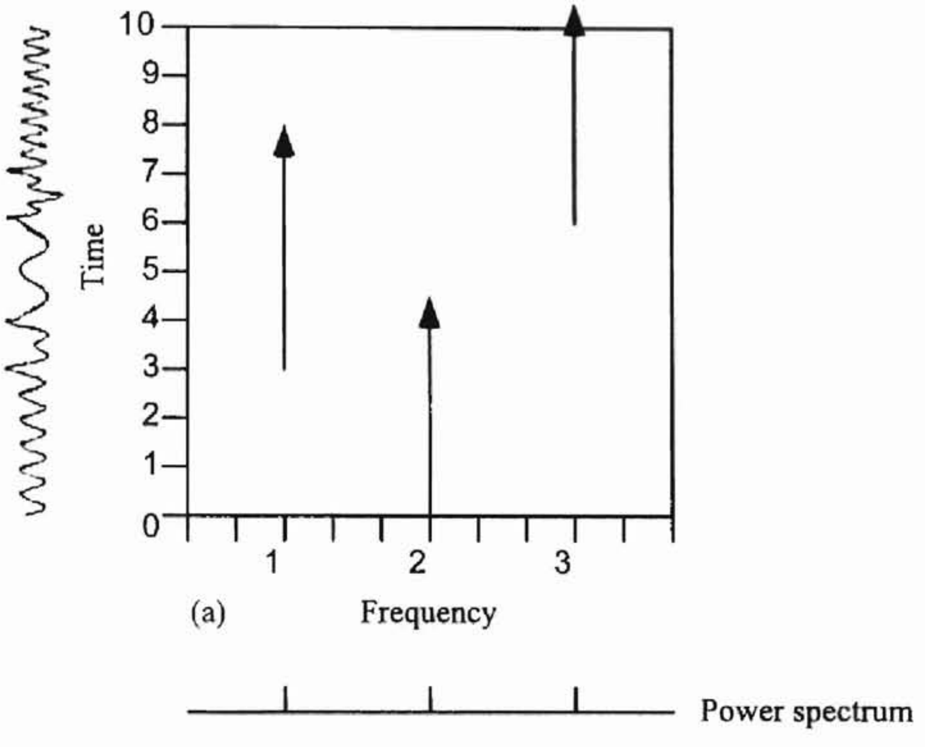
The Fourier transform is an excellent tool that could be used to alternate between the time and frequency domains of a signal representation. However, it fails to provide information about the time and frequency domains of a signal simultaneously i.e., the Fourier transform gives information of the different frequencies that existed for the total duration of the signal but not the frequencies which exist at a particular time. Simultaneous information on the time and frequency domains of a sensor signal representation is known as *time-frequency localization* and is explained in the following section.

3.3 Time-frequency localization

The need for a simultaneous time-frequency analysis is discussed below by considering a progression of examples. If a simple sine wave that lasts forever is considered, then to describe it completely it would suffice to say that it is a sine wave of fixed magnitude for a particular frequency for all time. The time-frequency description would show only one particular frequency at all times. Now, if the sum of three sine waves that last forever, with different frequencies are considered, the time-frequency spectrum would show three different frequencies present for all times. Thus time has played a passive role since whatever is occurring, occurs for all time, and the standard amplitude-time spectrum provides a satisfactory description [Cohen, 1995].

Figure 3.3a illustrates the case where the signal is formed by concatenating three different sinusoidal components. To fully describe such a situation, the frequencies for each time ought to be given. The time-frequency plot for this illustration (top left) shows three different frequencies and their duration. The power spectrum shown just below the time-frequency plot shows the frequencies 1, 2, and 3 but does not tell when they existed. Figure 3.3b is an extension of the previous illustration. This time-frequency plot shows that there are times when all the three frequencies coexist (time index 4 to 6.5), only two of the frequencies exist (time index 2 to 4 and 6.5 to 8.5), and only one of the three frequencies exists (time index 0 to2 and 8.5 to 10.5). This information cannot be extracted from the power spectrum only. Figures 3.3c and 3.3d illustrate this idea further. Note that the power spectrum is roughly the same in all four parts of Figure 3.3 and indicates the different frequencies existing but does not give any idea when the different frequency components were present.

A technique that gives a good time-frequency description is needed. The Fourier transform is only capable of identifying the frequencies present over the total duration of the signal and not the frequencies that exist at a particular time. This is where the beauty of the wavelet transform comes into play. Wavelet transforms provide for good-time frequency localization. The next section describes briefly the mathematical basis of the wavelet transform, construction of simple wavelets, multi-resolution analysis, and signal decomposition and reconstruction.



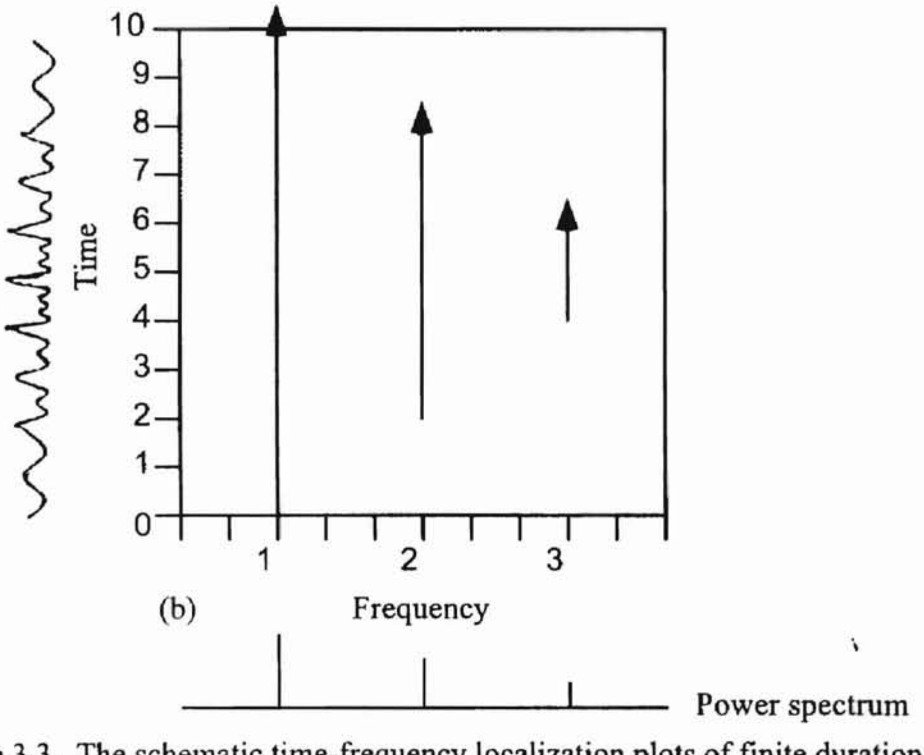
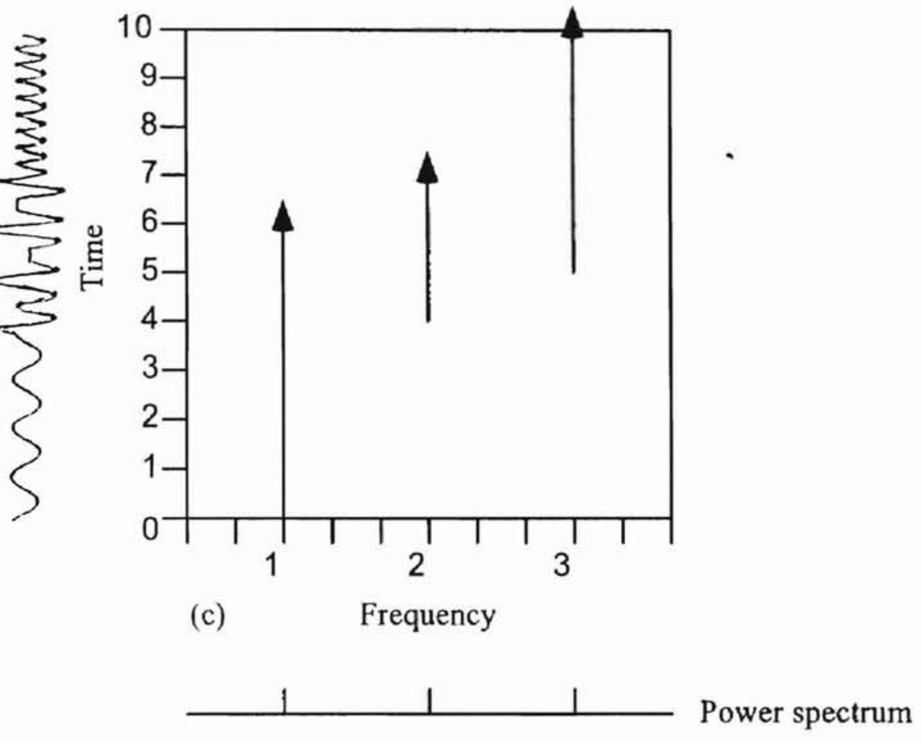


Figure 3.3. The schematic time-frequency localization plots of finite duration sine waves. From the power spectrum, it is not possible to estimate the duration of waves. A time-frequency plot clearly shows the frequencies existing for each time.



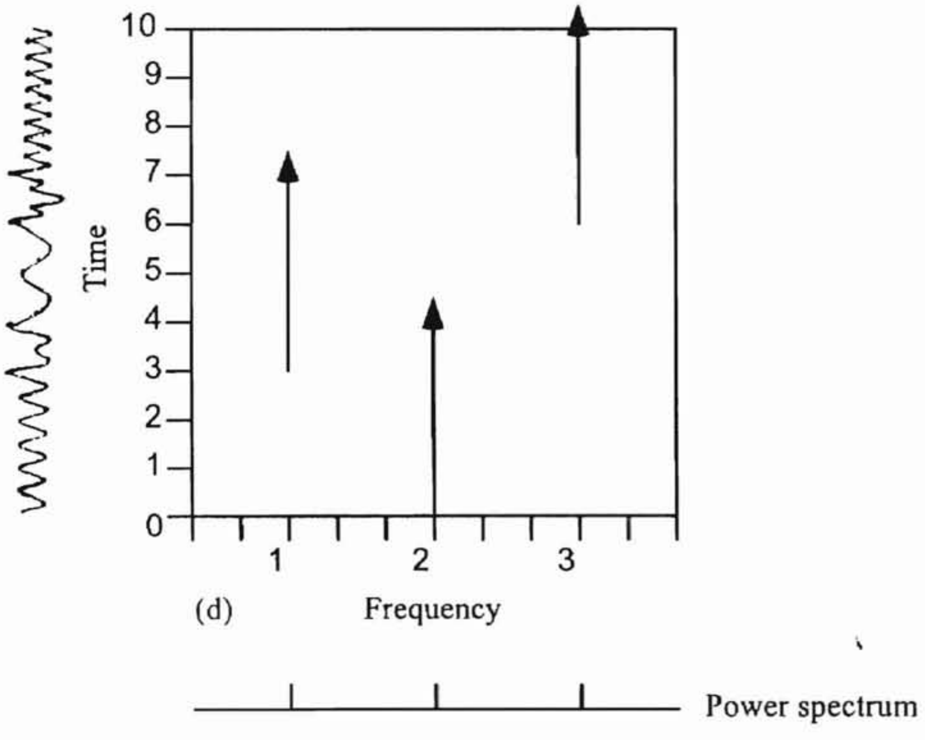


Figure 3.3 (continued)

3.4 The wavelet transform

The wavelet transform is analogous to the Fourier transform. In the Fourier signal decomposition, the basis functions are the sine and cosine functions. As indicated previously, a discrete signal *f(t)* can be represented as follows:

$$f(t) = \sum_k g_k \cos(tk) + \sum_k h_k \sin(tk) \tag{32}$$

In a similar fashion, the Wavelet series representation has two basis functions, the scaling function and the wavelet function.

$$f(t) = \sum_k g_k \alpha_k(t) + \sum_k h_k \beta_k(t) \tag{33}$$

Here, α and β are the *scaling* and *wavelet* basis functions, k is the index which defines the length of the signal, and g_k and h_k are the decomposition coefficients. An important difference between the basis functions of the Fourier transform and the wavelet transform is that the scaling and wavelet basis functions are complex functions which are derived and do not occur naturally. Furthermore, the wavelet basis function is derived from the scaling function. An introduction to the discrete wavelet transform is given below in which the scaling and wavelet functions are discussed in more detail.

3.4.1 Discrete wavelet transforms

The discrete wavelet transform, like the fast Fourier transform, is a fast, linear operation that operates on a data vector whose length is an integer power of two, transforming it into a numerically different vector of the same length. The wavelet transform, like the fast Fourier transform is invertible and, in fact, orthogonal [Press, 1992]. The inverse transform, when applied as a matrix, is simply the transpose of the transform matrix

[Strang, 1993, 1989]. Unlike the Fourier transform which is uniquely defined by sine and cosine basis functions, there is not one single unique set of basis functions for the wavelet transform; in fact, there an infinite number of possible basis functions. Each of these scaling and wavelet basis functions has a unique representation over a definite interval and vanish outside this interval. In wavelet mathematics, different scaling and wavelet functions are obtained by specifying the family and order of the wavelet. A brief explanation of the scaling and wavelet functions is given below.

3.4.1.1 Scaling functions and wavelet functions

Scaling functions and wavelet functions are represented by special types of equations called *dilation* equations. The general form of a dilation equation is as follows:

$$f(x) = \sum l_k f(2x - k) \tag{34}$$

Here, l_k is a vector of dilation function coefficients. The factor two provides for the dilation, i.e., expansion or contraction whereas the factor k provides for translation. Thus, frequency and time localization can both be achieved. The general equation of a scaling function is given by

$$\phi(x) = \sum c_k \phi(2^j x - k) \tag{35}$$

Here, c_k is the scaling function coefficient. The factor j provides for dilation and k provides for translation. A very important point to note here is that the scaling function is expressed as a sum of dilations and translations of the scaling function itself.

The wavelet equation is derived from the scaling function by taking "differences" as shown

$$W(x) = \sum_k d_k \phi(2x - k) \tag{36}$$

Here, the wavelet coefficients are given by d_k. Wavelets are defined as functions of the scaling functions because the decomposition basis functions have to be interdependent.

There are many different wavelet families; a discussion of which is beyond the purview of this work. The interested reader may refer to [Kaiser, 1994; Walter, 1994; Motard, 1994]. A very brief introduction to the Daubechies family of wavelets is given in the following section. For more information, the interested reader may refer to [Daubechies 1992, 1993].

3.4.2 The Daubechies Family

Daubechies wavelets are a distinct family of wavelets. These wavelets are orthogonal and are compactly supported. Support is defined as the span of the scaling or wavelet function over which the function has a non-zero value. They are constructed in the frequency domain where it is easy to handle the dilation and translation parameters. Certain constraints are imposed on the construction of the scaling and wavelet functions depending on the properties of the wavelet desired. These constraints are ultimately reflected in the respective sets of coefficients, c_k and d_k. For the Daubechies family, the constraints are orthonormality and orthogonality as explained below. Note that the following is a summary overview of the material presented in [Daubechies 1992, 1993].

A function $\phi(t)$ is said to be orthogonal if the following relation is satisfied:

$$\int_a^b \phi_j(t)\phi_k(t)dt = \begin{cases} 0 & j \neq k \\ c & j = k \end{cases} \tag{37}$$

where c is a constant. The function is said to be orthonormal when c takes the value of unity.

For the Daubechies family, the following two conditions are imposed. The scaling function, must be orthonormal.

$$\int \phi(x) = 1 \tag{38}$$

The wavelet functions must satisfy the condition

$$\int W(x)dx = 0 \tag{39}$$

Integrating equation (35) and applying the orthonormality condition, equation (38)

$$\int \phi(x)dx = \int \sum c_k \phi(2x-k)dx = 1 \tag{40}$$

$$\Rightarrow \sum c_k = 2 \tag{41}$$

Similarly, integrating equation (36) with the condition described by equation (39)

$$\int W(x)dx = \int \sum d_k W(2x-k)dx = 0 \tag{42}$$

$$\Rightarrow \sum d_k = 0 \tag{43}$$

The conditions for a wavelet to be both orthogonal and orthonormal are that the sum of it's scaling function coefficients should be two and the sum of its wavelet function coefficients should be zero. When the scaling function and the wavelet function are quadrature mirror filters of each other [Akansu, 1993; Cohen, 1992a; Daubechies, 1988] the relationship between their coefficients is given by

$$d_k = (-1)^k c_{1-k} \tag{44}$$

This implies that the wavelet coefficients are obtained directly from the scaling function coefficients by reversing the order and changing the sign on every alternate coefficient. Thus the tedious process of calculating them separately is avoided.

A brief description of the construction of first order Daubechies wavelets is given in the following section. A very detailed treatment on this topic is available in [Akansu, 1993; Chui, 1992a; Daubechies, 1992; Haykin, 1991; Strang, 1989].

3.4.2.1 Construction of first order Daubechies wavelet

The first step in the construction of wavelets involves the computation of the scaling function and wavelet function coefficients. Once these coefficients are computed, the actual scaling and wavelet functions are then determined. The following describes the least complicated of the Daubechies family of wavelets. The first order wavelet, commonly referred to as the Haar wavelet, is constructed below.

The scaling function is a simple box function given by

$$\phi(x) = \begin{cases} 1 & 0 \le x \le 1 \\ 0 & otherwise \end{cases} \tag{45}$$

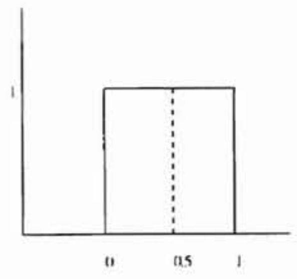


Figure 3.4a. The box function

The scaling function then becomes

$$\phi(x) = c_0\phi(2x) + c_1\phi(2x-1) \tag{46}$$

This is where the beauty of orthogonality comes into play. The translates of the dilation equation are orthogonal. $\phi(2x)$ and $\phi(2x-1)$ are orthogonal. $\phi(2x)$ exists as long as 2x lies between zero and one i.e., x lies between zero and 0.5. Similarly, $\phi(2x-1)$ exists as long as x lies between 0.5 and 1. Substituting the orthogonality property in the above equation, the values of both c_0 and c_1 are obtained as 1. This shows that the box function is the sum of two half sized boxes of uniform dilation and one of the boxes is translated by a unit value.

The wavelet equation is given by

$$W(x) = \sum_k d_k \phi(2x-k)$$

As k has two values 0 and 1, $W(x)$ reduces to

$$W(x) = d_0\,\phi(2x) - d_1\,\phi(2x-1) \tag{47}$$

Equations (35) and (39) are both satisfied when $d_0 = 1$ and $d_1 = -1$. So,

$$W(x) = \phi(2x) - \phi(2x-1) \tag{48}$$

The wavelet function takes the following values

$$W(x) = \begin{cases} 1, & 0 \le x \le 1/2 \\ -1, & 1/2 \le x \le 1 \\ 0, & otherwise \end{cases}$$

The wavelet function is shown below.

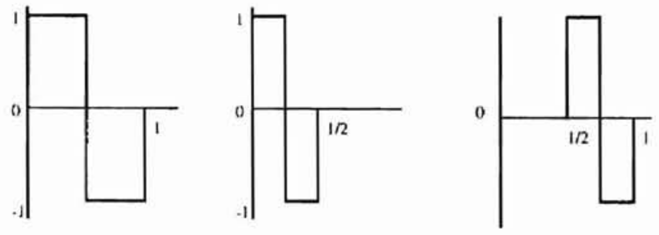


Figure 3.4b. The Haar wavelet and its dilated and translated versions

3.5 Signal filtering using scaling and wavelet functions

Mathematically, signal filtering is performed by the convolution operation. Convolution is the multiplication of the coefficients of polynomials. If two polynomials with their coefficients in two different vectors *a* and *b* are considered, then the coefficients of the polynomial obtained by convolving these two polynomials is represented by a vector *c*, the length of which is given by

$$length(\ a\ ) + length(\ b\ ) - 1 \tag{49}$$

The *k-th* element of the convolution product of *c* is given by

$$c_k = \sum_j a(j)b(k+1-j) \tag{50}$$

The sum is over all the values of *j*, which is from 1 to *(length(a) + length(b) -1)*. When both vectors *a* and *b* are of the same length, *n*

$$c(1) = a(1)b(1)$$

$$c(2) = a(1)b(2) + a(2)b(1)$$

$$c(3) = a(1)b(3) + a(2)b(2) + a(3)b(1)$$

......

$c(k) = a(1)b(k) + a(2)b(k-1) + \dots\dots\dots + a(n)b(1)$

......

$c(2n-2) = a(n-1)b(n) + a(n)\, b(n-1)$

$c(2n-1) = a(n)b(n)$

Wavelet decomposition is performed by convolving the signal with the scaling and wavelet function coefficients. The scaling and wavelet function coefficients are also known as filter coefficients. The resulting signal after convolution is dyadically sampled i.e., every alternate sample is taken into consideration.

If *f* is the signal, *H* and *G* are the low pass and high pass filters respectively, then the signal is decomposed as follows

$$\textit{Blurred signal} = H*f \tag{51}$$

$$\textit{Detail signal} = G*f \tag{52}$$

A high pass filter retains the low frequency part and allows the high frequency part of the signal to pass through. Similarly, the low pass filter retains only the high frequency part of the signal and allows low frequency part of the signal to pass through. Here, *H* is the filter containing the scaling function coefficients and *G* is the filter containing the wavelet function coefficients. The blurred signal contains the low frequency part of the original signal and the detail signal retains the high frequency part of the signal. The notation "*" represents the convolution operation followed by downsampling in which every alternate value is retained. The information content of the blurred and the detail signals is mutually exclusive, and the original signal at any level of decomposition can be obtained by the combination of the blurred and detail signals at that level. This results from the principle of orthogonality.

The original signal is reconstructed by reversing the decomposition operation.

$$H^{T\wedge}(\textit{Blurred signal}) + G^{T\wedge}(\textit{Detail signal}) = \textit{original signal} \qquad (53)$$

H^T and G^T are the transpose of H and G. The notation "^" represents the upsampling operation and convolution with the signal. Upsampling is doubling the length of the signal by inserting zeros between each value. Figure 3.5 illustrates the basic representation of the decomposition and reconstruction of a signal.

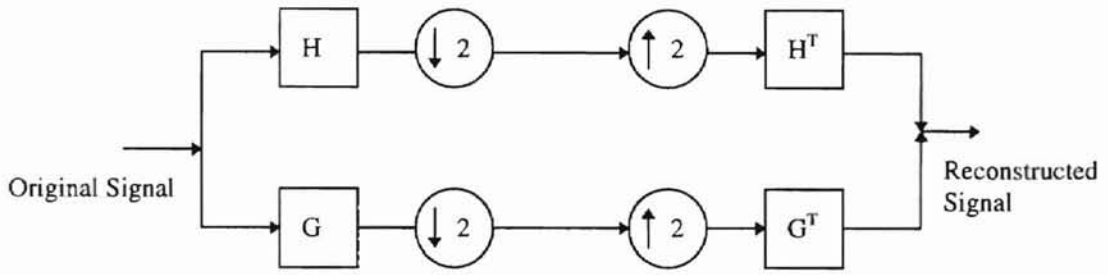


Figure 3.5. Basic decomposition and reconstruction representation

Consider a signal *f* having *n* samples at its original resolution which is represented in the time domain as the vector a^0. At the original resolution, the elements of the vector a^0 (the digitally sampled signal) are the values of the signal itself. At the first level of decomposition, the decomposition coefficients are given by

$$a_j^1 = \frac{1}{2}\sum_k a_k^0 c_{2j-k} \qquad j = 1,\ldots\ldots n/2;\ k = 1,\ldots\ldots n \qquad (54)$$

$$b_j^1 = \frac{1}{2}\sum_k a_k^0 d_{2j-k} \qquad j = 1,\ldots\ldots n/2;\ k = 1,\ldots\ldots n \tag{55}$$

Here, c_k and d_k are the scaling and wavelet function coefficients respectively. The constant ½ is the normalization constant (this takes into account that the signal is divided equally into blurred and detail signals). At any level of signal decomposition the values of a and b are computed by recursion from their values at the previous level. It is to be noted that a signal can only be decomposed $n/2$ levels because at the $n/2^{th}$ level, the blurred and detail signal would contain only one coefficient each.

The decomposition procedure is illustrated below using the Haar wavelet and a short signal so as to give the reader a good feel of this subject. The Haar scaling coefficients $[c_0\ c_1]$ are [1 1] and the Haar wavelet coefficients $[d_0\ d_1]$ are [1 -1]. The signal is given by [1 2 3 4]. The decomposition coefficients are given by

$$a_1^1 = \frac{1}{2}\left[a_1^0 c_1 + a_2^0 c_0\right] = \frac{1}{2}[1\times 1 + 1\times 2] = 3/2$$

$$a_2^1 = \frac{1}{2}\left[a_3^0 c_1 + a_4^0 c_0\right] = \frac{1}{2}[1\times 3 + 1\times 4] = 7/2$$

$$b_1^1 = \frac{1}{2}\left[a_1^0 d_1 + a_2^0 d_0\right] = \frac{1}{2}[1\times -1 + 2\times 1] = 1/2$$

$$b_2^1 = \frac{1}{2}\left[a_3^0 d_1 + a_4^0 d_0\right] = \frac{1}{2}[3\times -1 + 4\times 1] = 1/2$$

The blurred signal at the first level is a^1 = [3/2 7/2] and the detail signal at the first level is b^1 = [1/2 1/2]. Reconstruction is performed as follows:

$$a_k^0 = \sum_j a_j^1 c_{2j-k} + \sum_j b_j^1 d_{2j-k} \qquad j = 1,\ldots\ldots,n/2. \tag{56}$$

For this example,

$$a_1^0 = [1\times 3/2 + 0\times 7/2] + [-1\times 1/2 + 0\times -1] = 1$$

$$a_2^0 = [1\times 3/2 + 0\times 7/2] + [1\times 1/2 + 0\times 1/2] = 2$$

$$a_3^0 = [0\times 3/2 + 1\times 7/2] + [0\times -1 + 1/2\times -1] = 3$$

$$a_4^0 = [0\times 3/2 + 1\times 7/2] + [0\times -1 + 1\times 1/2] = 4$$

This recreates the original signal $a^0 = [1\ 2\ 3\ 4]$.

The above example illustrates perfect reconstruction. For smoothing purposes, perfect reconstruction is not used. Instead, the high frequency part of the signal is removed. If the high frequency part is to be removed, the detail signal coefficients are made zero and the signal reconstructed from the blurred signal coefficients and the zeroed detail coefficients. In the above example, the level of decomposition is one. However, a signal could be decomposed to more levels than one and viewed at each level of decomposition (resolution). The process can be repeated until a single blurred non-zero coefficient is obtained.

For smoothing purposes, the key is determining how many levels of decomposition should be performed. The most straightforward approach is as follows. First the signal is decomposed one level, setting the detail signal coefficients to zero (this eliminates the high frequency part), and reconstructing the signal and checking the degree of smoothing. If the reconstructed signal still contains some high frequency noise, the blurred signal at the first level is decomposed again. The detail coefficients at the second level are again made zero and the signal reconstructed from the blurred and detail signals at the second level. This process is repeated until the desired degree of smoothing is achieved. This process of viewing the signal at multiple resolutions is called multi-resolution analysis. A brief description of multi-resolution analysis is given below.

3.6 Multi-resolution analysis

As explained above, the process of decomposing a signal to various levels is known as multi-resolution analysis. The detail and blurred signal coefficients at any level are computed by recursion from the results at the previous level. At any level, the blurred or the detail signals are considered to be an averaged version of the signal (blurred or detail) at the previous level. The frequency of the signals is twice that of the frequency at the previous level (also known as a scale twice as large). If the blurred signal at any level is considered the averaged version of the blurred signal at the previous level, then the detail signal is obtained from the difference between the signal at the previous level and the blurred signal generated at the current level, i.e., the information present in the original signal but filtered out in the averaged version. In other words, the blurred signal at any level j is obtained by the combination of the blurred and detail signals at the next lower level $j+1$ and the detail signal at level j from the difference between the blurred signals at level j and level $j-1$.

The basic idea behind multi-resolution analysis can be summarized by Figure 3.6.

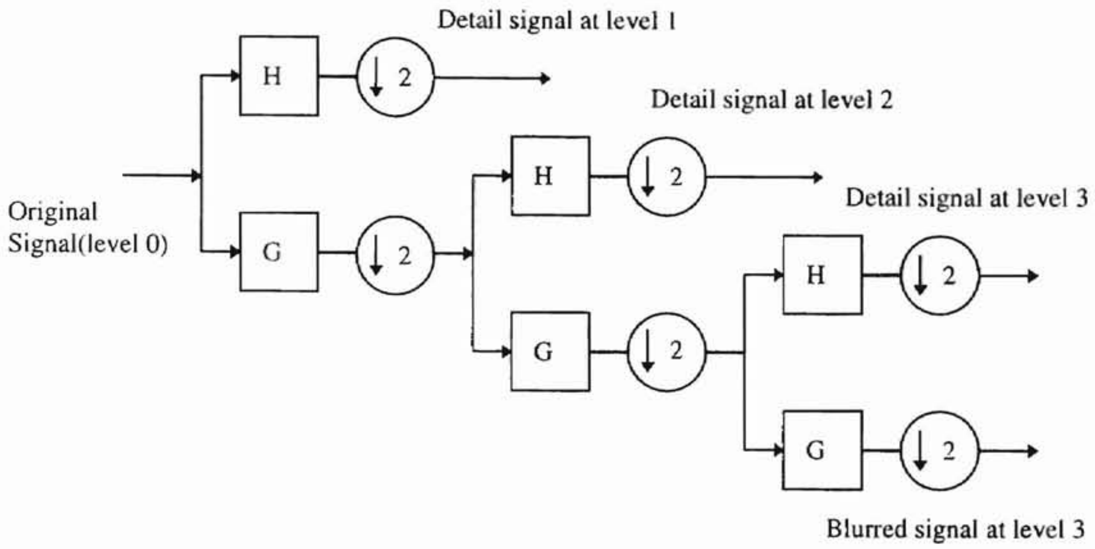
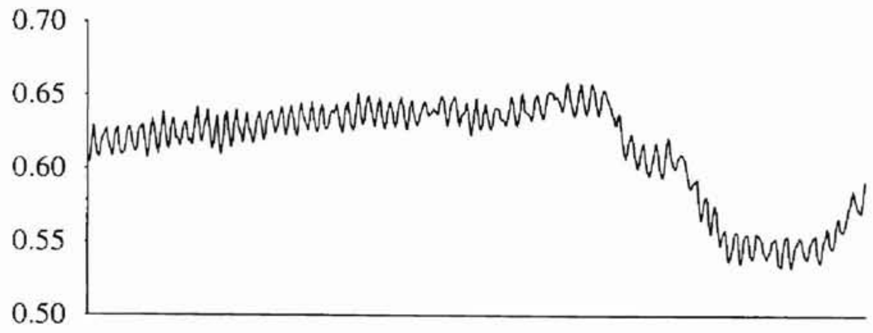


Figure 3.6. Basic representation of the multi-resolution algorithm

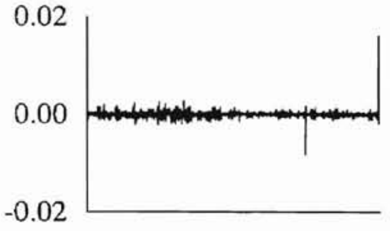
Figure 3.6 shows a signal decomposed three levels. It is evident that the signal at any level is the sum of the detail and blurred signals at the previous or next higher level.

Application of multi-resolution analysis to an actual plant signal is illustarted in Figure 3.7. The original signal is decomposed into blurred (low frequency) and detail (high frequency) signals at the first level. It can be seen that the low frequency or blurred signal basically follows the fundamental trend of the original signal with magnitude almost the same as that of the original signal. However, the high frequency part of the original signal does not follow any particular trend relative to the trend of the original signal and is of magnitude substantially lesser than that of the original signal. Subsequent results after two, three, four, five and seven levels of decomposition are presented in Figure 3.7.

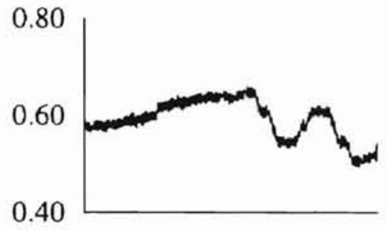
The reconstructed signal with the detail coefficient zeroed is superimposed on the original signal is presented in Figure 3.8. From Figure 3.8, it can be seen that the



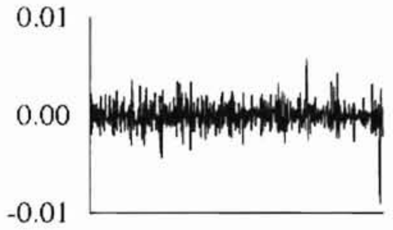
Original signal.



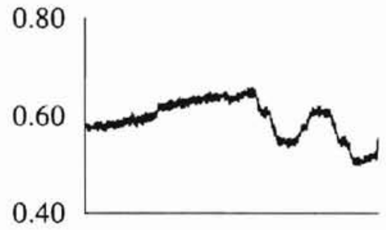
Detail coefficients at first level.



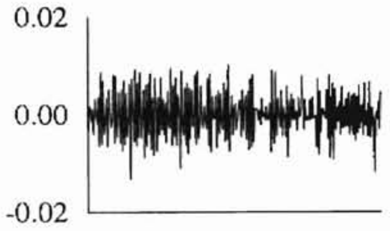
Blurred coefficients at first level.



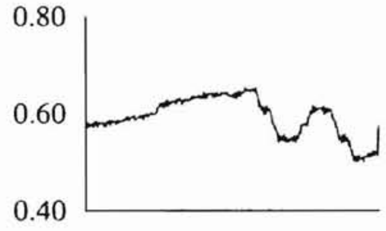
Detail coefficients at second level.



Blurred coefficients at second level.

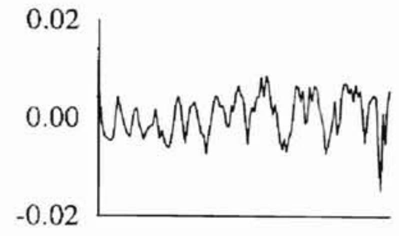


Detail coefficients at third level.

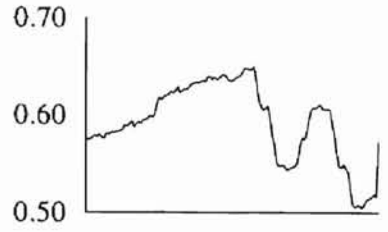


Blurred coefficients at third level.

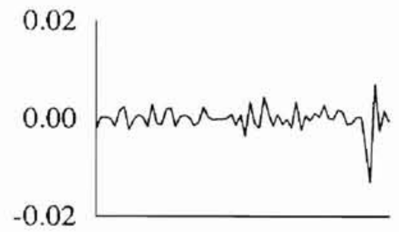
Figure 3.7. Detail and blurred coefficients for the first four levels of decomposition using sixth order Daubechies wavelets.



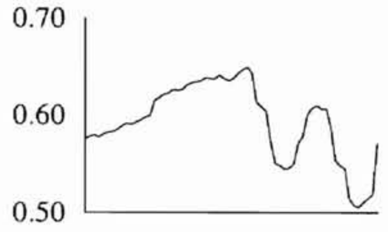
Detail coefficients at fourh level.



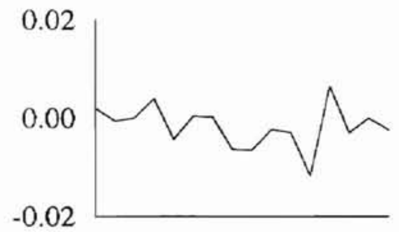
Blurred coefficients at fourth level.



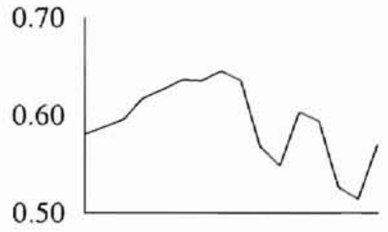
Detail coefficients at fifth level.



Blurred coefficients at fifth level.

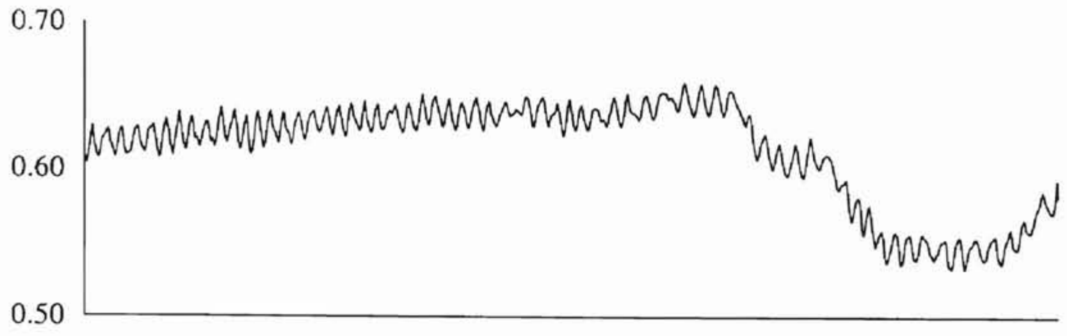


Detail coefficients at seventh level.

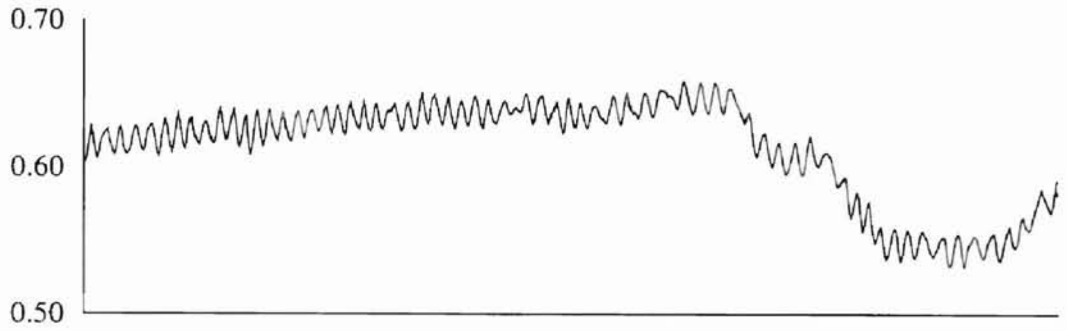


Blurred coefficients at seventh level.

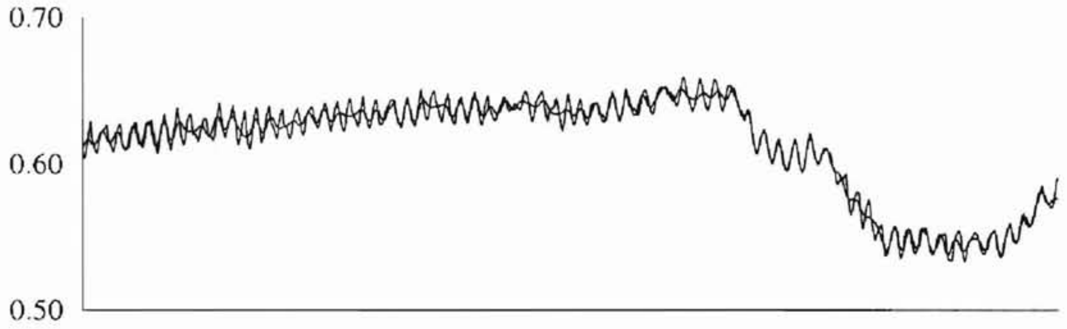
Figure 3.7. (contd.) Detail and blurred coefficients for fourth, fifth, and seventh levels of decomposition using sixth order Daubechies wavelets.



First level of decomposition.

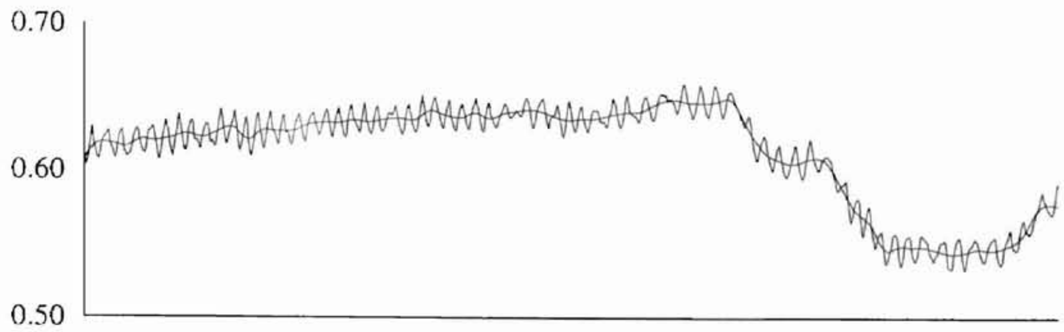


Second level of decomposition.

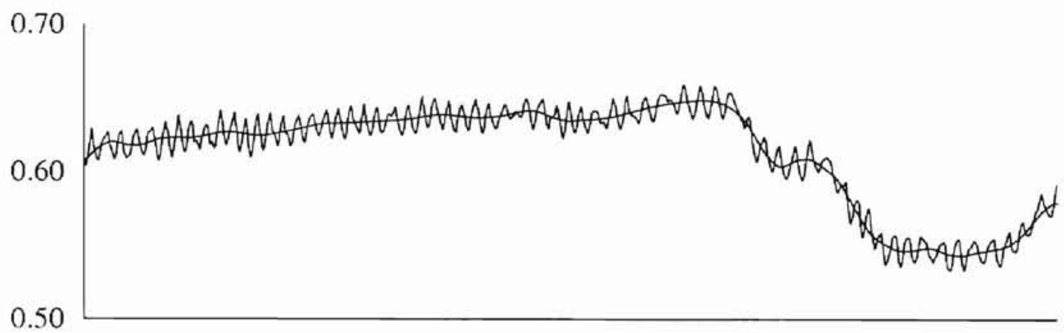


Third level of decomposition.

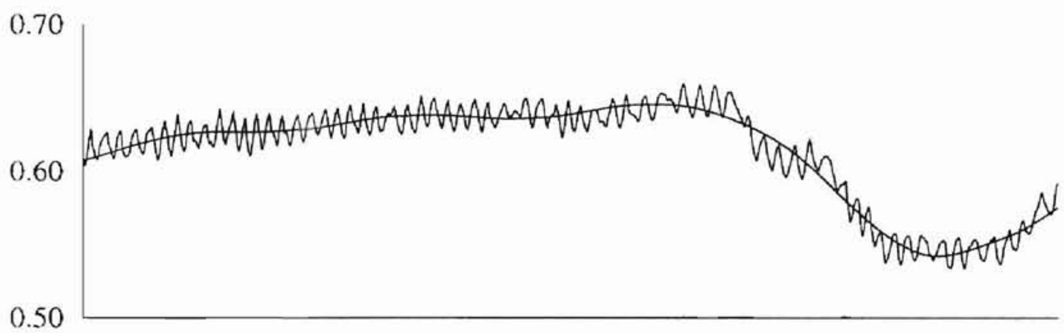
Figure 3.8. Smoothed signal obtained (with detail coefficients set to zero) using sixth order Daubechies wavelets, superimposed on original signal for various levels of decomposition.



Fourth level of decomposition.



Fifth level of decomposition.



Seventh level of decomposition.

Figure 3.8. (contd.) Smoothed signal obtained (with detail coefficients set to zero) using sixth order Daubechies wavelets, superimposed on original signal for various levels of decomposition.

smoothed signal obtained from reconstruction at the fourth level of decomposition is an excellent representation of the raw sensor signal. Further, it can be seen that as the level of decomposition increases beyond the optimum level of decomposition, the capacity of the smoothed signal to retain the fundamental trend of the original raw signal is reduced. The smoothed signals reconstructed from beyond the fifth level of decomposition prove this beyond doubt.

Multi - resolution analysis [Cohen, 1992b; Daubechies, 1991; Mallat, 1989a; Mallat, 1989b; Mallat, 1989c] is a very powerful tool for trend extraction and pattern recognition. Accurate representation of the fundamental trend of the original sensor signal can be obtained from decomposition of the signal followed by reconstruction without the detail coefficients. However, increasing the number of levels of decomposition also increases the degree of smoothing; so an optimum level must be selected so that a smoothed representation is obtained without sacrificing the fundamental trend of the original signal. The optimum level of decomposition was previously determined by trial and error, i.e., by experimenting the signal with various levels of decomposition and finding the level of decomposition which best smoothes the raw sensor signal.

The optimum level of decomposition has previously been determined by experimentation and depends on the pattern recognition problem being addressed. This can be avoided and the optimum level or the level very near to the optimum level of decomposition can be determined automatically. The next chapter presents our proposed method for automated trend extraction.

Chapter 4

AUTOMATED TREND EXTRACTION

4.1 Need for automated trend extraction

In Chapter 3 the desired level of smoothing of signals is achieved empirically by trial and error. Experimentally determining the optimum level of smoothing of a raw sensor signal is not suitable for real-time applications.

An automated approach is needed if we expect to provide a stand alone system which allows operators to create their own process monitoring applications. Furthermore, different signals may require different degrees of smoothing depending on the characteristics (e.g., noise content) of individual sensors. Ideally, we would like to have a system which analyzes the characteristics of the monitoring application as well as the sensor signals and recommends the appropriate degree of smoothing for each signal. The system would graphically illustrate the recommended degree of smoothing and allow the user (operator) to either accept the recommendation or specify more or less smoothing.

There is no a priori correct level of smoothing, as the desired degree of smoothing is application dependent. We have empirically determined that the degree of smoothing for pattern-based analysis is usually determined by the length of the data window used for observing and monitoring a process. Typically, it suffices to provide sufficient smoothing so that approximately four blurred coefficients are used over the span of the window length after decomposition and before reconstruction. The reconstructed signal can then be sampled as desired to obtain a compact representation, as described in Chapter 1. In other words, the "correct" number of levels of decomposition normally depends on the number of blurred coefficients after decomposition that are fixed over the window length used for process monitoring.

4.2 Determination of the degree of smoothing from the user input window length

The smoothing obtained for a typical sensor signal fixing four blurred coefficients after decomposition, over a 45 minute window of observation is shown in Figure 4.1. The number of levels of decomposition of the raw signal is back calculated from the window length used for process monitoring, the sampling period of the raw data, and the desired number of blurred coefficients over the window of observation, after decomposition.

The window length for process monitoring is problem specific and left to the discretion of the operator (user). Nevertheless, a minimum window length required for monitoring purposes can be helpful in providing insight when specifying the actual window length. Determination of the minimum window length for process monitoring is described later in the chapter. The following discussion describes how the recommended degree of smoothing is determined from the window length.

The user input window length, *winx* (units of time) is first converted to the associated number of data points with the aid of the signal sampling period, T_s and this value is checked to see if it is less than half the length of the original signal, *N*. If the user input window length is less than *N/2*, then *winx* is used to calculate the number of levels of wavelet decomposition, otherwise the user is prompted to enter a window of

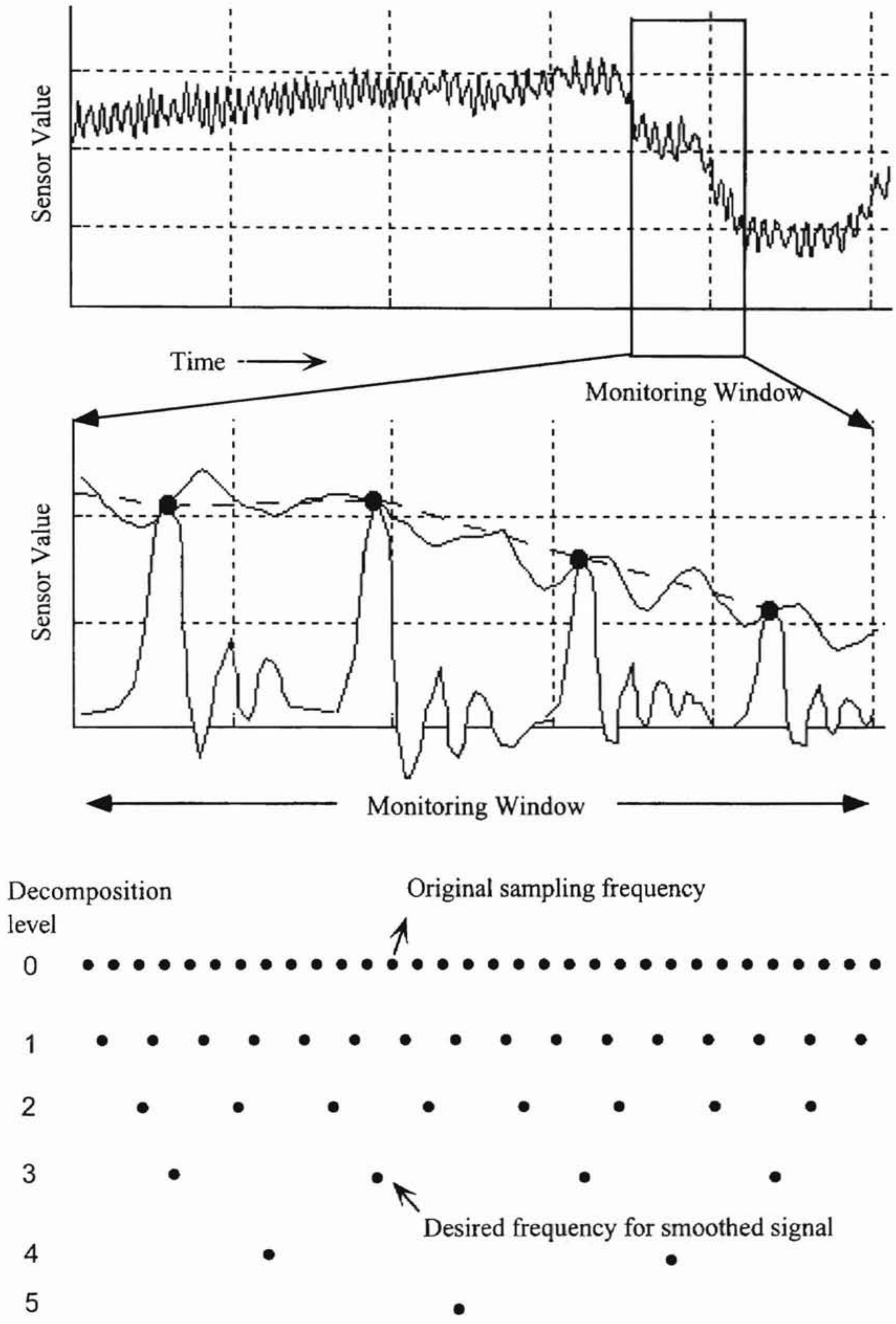


Figure 4.1 Span of scaling functions in the process monitoring window and the desired frequency level for the smoothed signal.

length less than $N*T_s/2$ units of time. Figure 4.1 illustrates how the desired number of levels of decomposition are determined so that there are approximately four points spanning the window length. Analytically, the number of levels of decomposition, *act_levels* can be calculated according to the following formula,

$$act\_levels = log2(winx/4*T_s) \tag{57}$$

It is to be noted here that, for a specific problem, *winx* is a constant and *act_levels* depends only on the sampling frequency. The number of levels of wavelet decomposition obtained, *act_levels*, gives the best estimate of the optimum degree of smoothing. Nevertheless, the user must verify this result and be given the option of specifying more or less smoothing if deemed necessary.

The algorithm is built around the Matlab smoothing module first proposed by Mr. V. Raghavan [Raghavan, 1994]. This smoothing module uses sixth order Daubechies wavelets. The following discussion documents the Matlab m-files which are utilized.

The scaling and wavelet function coefficients are determined by the function ***daub.m***. All smoothing techniques involve convolution which tend to distort the trend of the smoothed signal towards the ends. Distortion of the smoothed signal towards the end is unacceptable for pattern-based process monitoring purposes. To avoid trend distortion, the original signal is padded on either side by half its length by the function ***net.m*** [Raghavan, 1994]. The padded signal is then decomposed to *act_levels* number of levels by the function ***fwt.m*** and reconstructed with the blurred coefficients at this level. All detail coefficients are set to zero. Signal reconstruction is then performed by the function ***ifwt.m***. The extensions to the signal are then removed to obtain the smoothed representation of the original signal. The user is then prompted if the smoothed representation is satisfactory, too smooth, or still needs to be smoothed. Depending on the input of the user, appropriate action is taken.

4.3 Determination of the minimum degree of smoothing from raw signal characteristics

The number of levels of decomposition obtained from the user input window length generally produces the desired level of smoothing. However, in special cases where the signal contains a steady, low frequency oscillation produced from control system interactions, more smoothing may be required. In this situation, it is necessary to identify the dominant low frequency component and ensure that the signal is decomposed beyond this level.

The frequency spectrum of the original signal is useful for identifying the dominant low frequency component. In this work, the term "cut-off" frequency is introduced to differentiate the first few low frequencies of large magnitude that are of interest, and other frequencies of smaller magnitude that are not of interest. For our work, we assume that the cut-off frequency is the dominant frequency and the two terms can be used interchangeably.

In this work, the term frequency index is often used. This is just another way of representing frequency [Ludemann, 1992]. Referring to equation (29b) in Chapter 3, we note that the frequency index k corresponds to the frequency "$k2\pi /N$" in a Fourier decomposition.

4.3.1 Dominant frequency identification

This section describes how we quantify the cut-off frequency and use this information to determine the minimum amount of decomposition required to smooth through low level, steady oscillations.

As described in Chapter 3, the discrete Fourier transform (DFT) can be applied to split a signal to obtain the frequencies associated with the signal in terms of cosine and sine coefficients of various amplitudes and frequencies. We utilize the properties of the discrete Fourier transform to determine the cut-off frequency index.

Figure 3.2 in the previous chapter shows the cosine and sine coefficients plotted against their respective frequency indices for a representative time domain signal. By visual inspection of Figure 3.2, it can be seen that the first few cosine coefficients are of relatively large magnitude. The cosine coefficients around the frequency index forty and above are of smaller magnitude and can be categorized as noise for purposes of determining the cut-off frequency. The maximum sine coefficient occurs at frequency index one. Sine coefficients from two to four are comparable to the maximum value but coefficients above frequency index four can be ignored. This is an illustrative example where the sine and cosine coefficients do not drop off at the same frequency index.

The question that arises now is the basis on which the cut-off frequency index is to be determined, i.e., to select the sine or the cosine cut-off frequency index. Also, there may be a few coefficients well within the cut-off frequency that are of very small magnitude. This questions the validity of the cut-off frequency index itself. An alternative approach, the cumulative power spectrum, gives a fairly good idea of the cut-off frequency index. The next subsection addresses this topic.

4.3.2 The Cumulative power spectrum

The power spectra of a signal is determined from the output of a Fourier decomposition. The value of the power of a signal (consisting of real values) at any frequency is determined by taking the sum of the squares of the real and imaginary parts of the Fourier coefficient obtained at that frequency and dividing it by the length of the signal. The cumulative power is determined by adding the values of the power at each succeeding frequency. This is demonstrated by the following example.

The Fourier coefficients of the signal shown in Figure 3.1 of the previous chapter, $x(n) = [1\ 2\ 3\ 4]$, are given by

$$X(0) = 10,\ X(1) = -2+2j,\ X(2) = -2,\ X(3) = -2-2j$$

The power at each of these points is determined as shown:

$$P(0) = (10)^2/4 = 25$$

$$P(1) = [(-2)^2 + (2)^2]/4 = (4 + 4)/4 = 2$$

$$P(2) = (-2)^2/4 = 4/4 = 1$$

$$P(3) = [(-2)^2 + (-2)^2]/4 = (4 + 4)/4 = 2$$

The cumulative power at each of these points is determined as shown below:

$$P(0) = 25$$

$$P(1) = P(0) + P(1) = 25 + 2 = 27$$

$$P(2) = P(0) + P(1) + P(2) = 25 + 2 + 1 = 28$$

$$P(3) = P(0) + P(1) + P(2) + P(3) = 25 + 2 + 1 + 2 = 30$$

The cumulative power spectrum for the above signal is *[25 27 28 30]*.

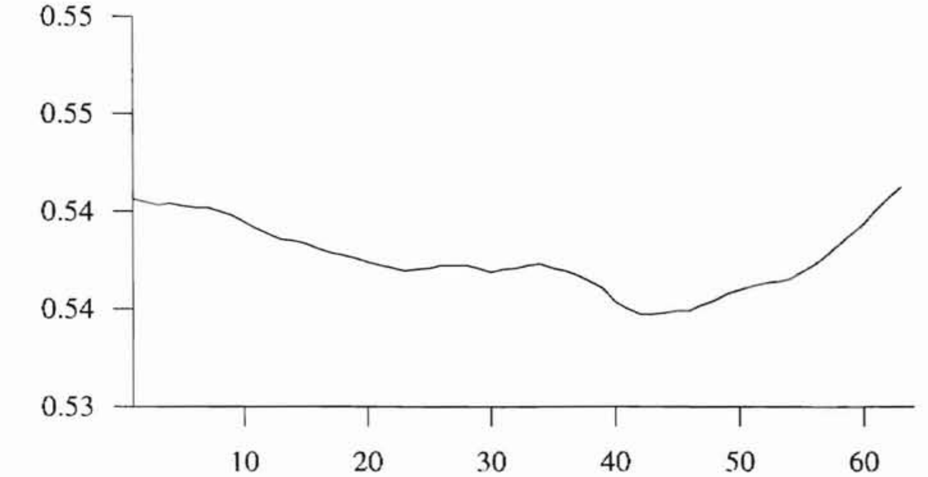
In a similar manner, the cumulative power spectrum of any signal can be determined. However, the cumulative power spectrum obtained by this method is not very smooth but is characterized by small changes in magnitude at some points. A smoother estimate of the power could be obtained by using Welch's averaged periodogram method. The signal is divided into overlapping sections, each of which is detrended and windowed by a Hanning window [Oppenheim, 1975]. The adjacent records of the power are averaged so as to obtain more reliable spectral estimates. All the values thus obtained are in a confidence interval of 95%. The cumulative power spectrum obtained by this method, and the source signal are shown in Figure 4.2.

As shown in Figure 4.2, the cumulative power spectrum increases steadily and then flattens. The reason for the initial steep increase is that the first few values of the power are relatively large due to the large magnitude of the first few low frequency cosine and sine coefficients. Subsequent coefficients are of relatively smaller magnitude and contribute less power. This effect is reflected by the flattening of the cumulative power spectrum after the first few low frequency coefficients. For our purposes, we define the frequency index where the cumulative power spectrum "bends" as the cut-off or dominant frequency index. The cut-off frequency index is a key parameter because it is used to determine two important parameters,

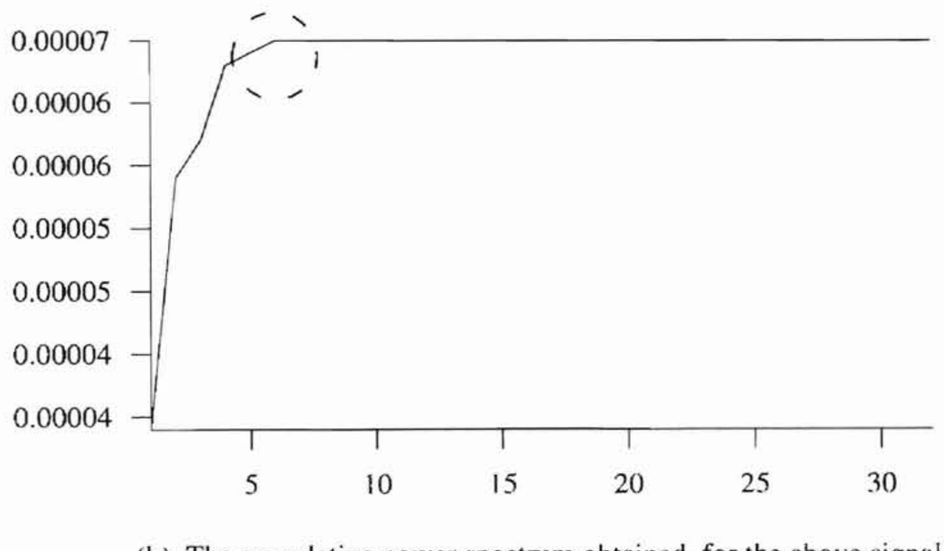
(1) the minimum number of levels of decomposition of the signal so as to retain the dominant frequencies and

(2) the minimum window length required for effective process monitoring, which can be used as a basis for selecting the actual window length for process monitoring.

The cut-off frequency is determined by the examination of the second derivative of the CPS. The interested reader may refer to the documentation in the function ***detect.m***, a copy of which is included in the Appendix.



(a) Signal.



(b) The cumulative power spectrum obtained for the above signal.

Figure 4.2. Source signal and its cumulative power spectrum.

4.3.3 Determination of the minimum number of levels of signal decomposition from the cut-off frequency index

Figure 4.3 lays the foundation for understanding how the cut-off frequency index determines the minimum number of levels of wavelet signal decomposition.

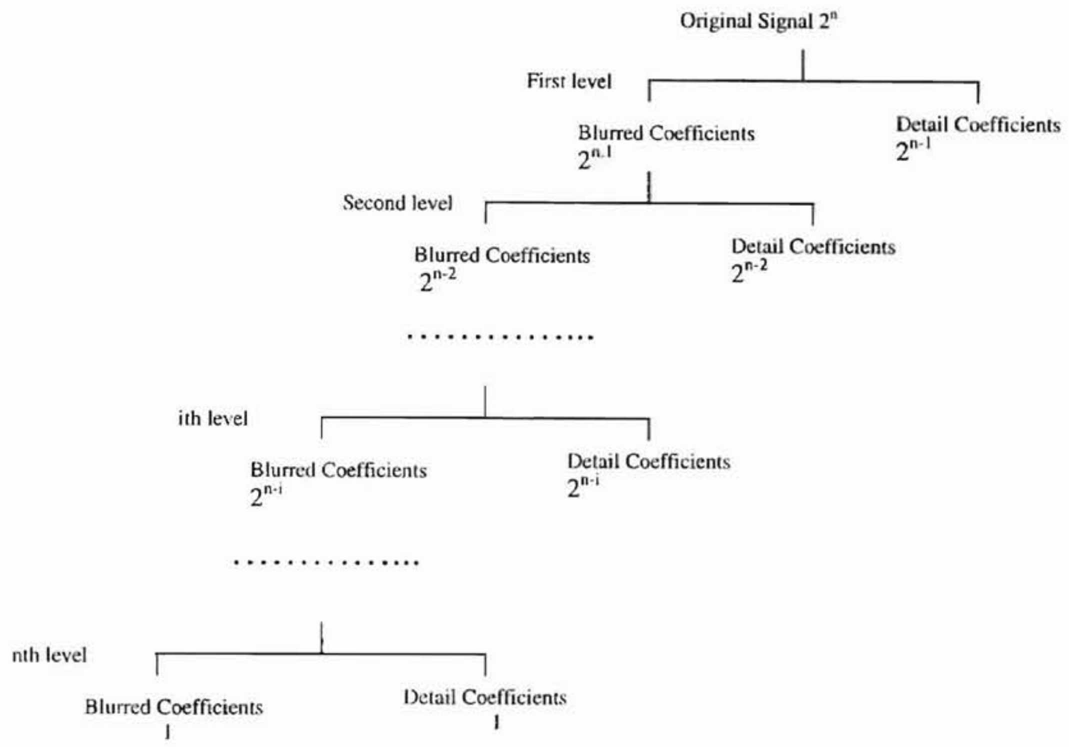

Original Signal 2^n
First level
Blurred Coefficients 2^{n-1}
Detail Coefficients 2^{n-1}
Second level
Blurred Coefficients 2^{n-2}
Detail Coefficients 2^{n-2}
ith level
Blurred Coefficients 2^{n-i}
Detail Coefficients 2^{n-i}
nth level
Blurred Coefficients 1
Detail Coefficients 1

Figure 4.3. Multi resolution tree analysis for a signal of length $N = 2^n$

Consider a signal of some finite length N, where N is equal to 2^n. After performing the fast Fourier transform on this signal, the number of cosine and sine coefficients obtained are $(N/2)+1$ and $(N/2)-1$ respectively. Now if the original signal is decomposed fully to n levels of decomposition using the wavelet transform, one blurred and a total of $N-1$ detail coefficients would be obtained. The blurred and detail coefficients at the nth level approximate the original signal with scaling and wavelet functions which can be characterized by the lowest non-zero Fourier frequency. From the definitions in Chapter 3, the lowest non-zero Fourier frequency corresponds to $(1(2\pi/N))$ and is associated with the a_1 coefficient. Therefore, a frequency index of 1 (equal to 2^0) corresponds to a wavelet signal decomposition of n levels.

If the wavelet signal decomposition is performed n-1 levels, two blurred and a total of N-2 detail coefficients are obtained. The coefficients at the $(n-1)$ level of decomposition are associated with scaling and wavelet functions which span half the original signal and can be thought of as having the same frequency as the a_2 and b_2 Fourier coefficients with frequency equal to $(2(2\pi/N))$. In other words, a Fourier frequency index of 2 (equal to 2^1) corresponds to a wavelet signal decomposition of n-1 levels.

Likewise, if the wavelet signal decomposition is performed n-2 levels, four blurred and a total of $(N-4)$ detail coefficients would be obtained. The coefficients at level $(n-2)$ can be visualized as having the same frequencies as the a_4 and b_4 Fourier coefficients corresponding to frequency equal to $(4(2\pi/N))$. In other words, a cut-off frequency of 4 (equal to 2^2) corresponds to a wavelet signal decomposition of n-2 levels.

Similarly, for a wavelet signal decomposition of n-i levels, 2^i blurred and a total of $(N-2i)$ detail coefficients would be obtained. The coefficients at the $(n-i)$ level can be visualized as having the same frequencies as the Fourier coefficients a_{2^i} and b_{2^i}. In other words, a Fourier frequency index of 2^i corresponds to a wavelet signal decomposition of n-i levels. The following table gives the relation between the cut-off frequency index and the corresponding levels of wavelet decomposition.

Table 4.1.

The number of levels of wavelet decomposition for various cut-off indices

Frequency index	Corresponding levels of decomposition
$2^0=1$	n
$2^1=2$	$n-1$
$2^2=4$	$n-2$
$2^3=8.$	$n-3.$
.	.
2^i	$n-i$
.	.
.	.
$2^{n-1}=N/2$	0

From Table 4.1, we deduce that the minimum number of levels of decomposition, *min_levels*, can be calculated from the dominant frequency index by the following formula:

$$min\_levels = log_2(cut\text{-}off\ frequency\ index) \tag{58}$$

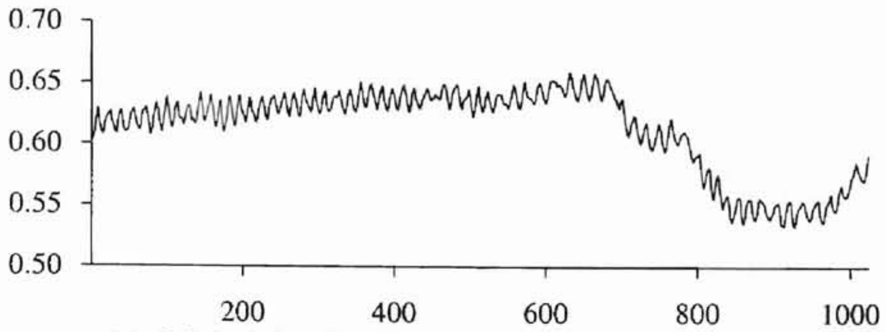
From the above discussion, it is evident that for the number of levels of decomposition to be an integer value, the cut-off frequency index needs to be a power of two. However, the cut-off frequency obtained in practice may or may not be a power of two. When not an integer, the calculated minimum number of levels of wavelet decomposition is incremented to the next integer. In doing so, a few of the low frequencies are discarded.

The number of levels of decomposition thus obtained *min_levels,* gives the *minimum* number of levels of decomposition which retain all the dominant frequencies. The smoothed signal thus obtained generally requires additional smoothing as discussed previously.

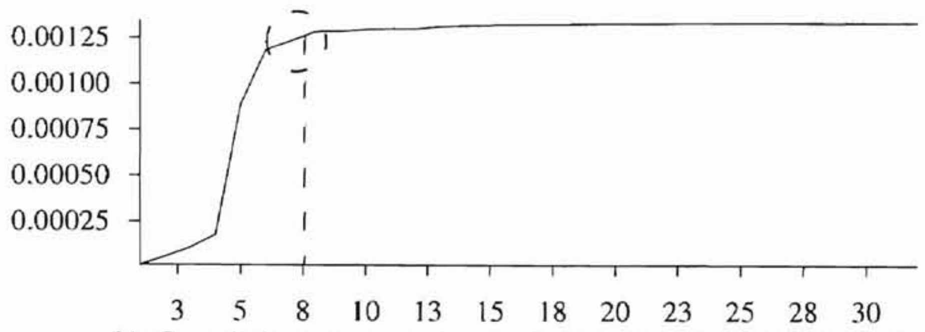
The concept of a dominant frequency is further illustrated in Figure 4.4. A signal with a strong periodic component is shown in Figure 4.4a. The cumulative power spectrum obtained using the first 64 samples of the signal is presented in Figure 4.4b. The cut-off frequency index is determined to be 8. The minimum number of levels of decomposition is therefore 3. The original signal smoothed 3 levels using sixth order Daubechies wavelets is shown in Figure 4.4c. From Figure 4.4c, it can be confirmed that the smoothed signal still retains the periodic swings in the original signal. Figure 4.4d verifies that decomposing the signal one level beyond this level yields a smoothed approximation without the dominant low frequency oscillation.

4.3.4 Determination of the minimum window length required for process monitoring

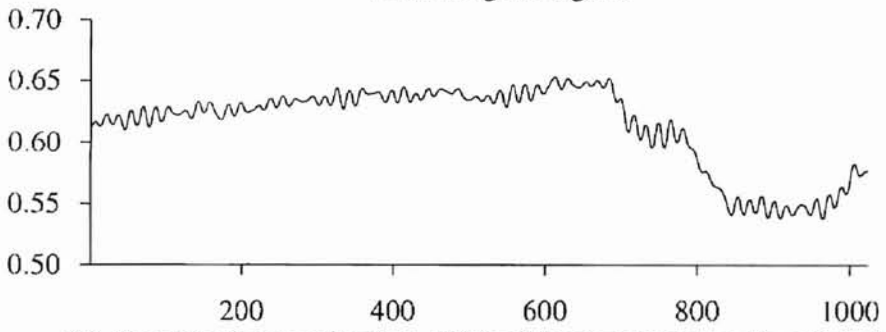
A minimum suggested window length for pattern-based data analysis can be determined from the cut-off frequency index by an empirical algorithm as described below. We recommend that the minimum window length should be long enough to capture four cycles of all the contain four cycles of all the frequencies below the cut-off, inclusive of the cut-off.



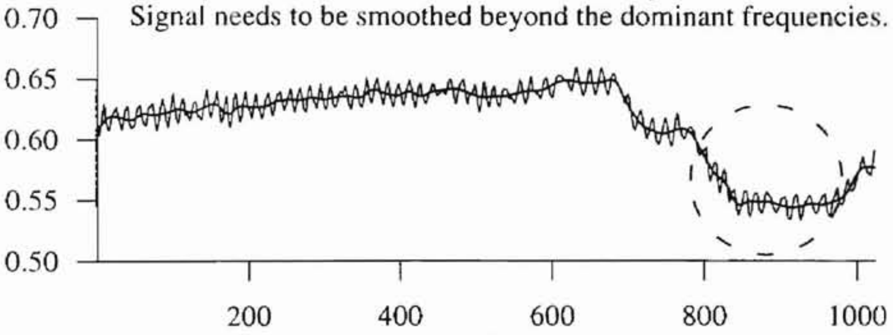
(a) Original signal characterized with highly periodic swings.



(b) Cumulative power spectrum obtained from the first 64 data points of the original signal.



(c) Smoothed approximation obtained by decomposing the original signal three levels retains the dominant frequencies as shown. Signal needs to be smoothed beyond the dominant frequencies.



(d) Smoothed approximation of the original signal obtained by decomposing the original signal four levels using sixth order Daubechies wavelets, and the original signal.

Figure 4.4: Representation of the dominant frequencies inherent in a signal and the need to smooth beyond these frequencies.

This is an empirical recommendation and is intended to ensure that a sufficient portion of the signal is used to determine the fundamental trend. The cut-off frequency index, *freq_index*, is first converted to its value in radians per unit time, using the sampling interval of the data, T_s and the length (number of data points) of the signal considered for fast Fourier transform computation, *N1*, by the relation

$$W = (freq\_index)*(2\pi)/(N1*T_s) \tag{59}$$

The value of *W* in cycles per unit time is given by

$$W = (freq\_index)/N1*T_s \tag{60}$$

Then, the length of the minimum window in units of time is given by

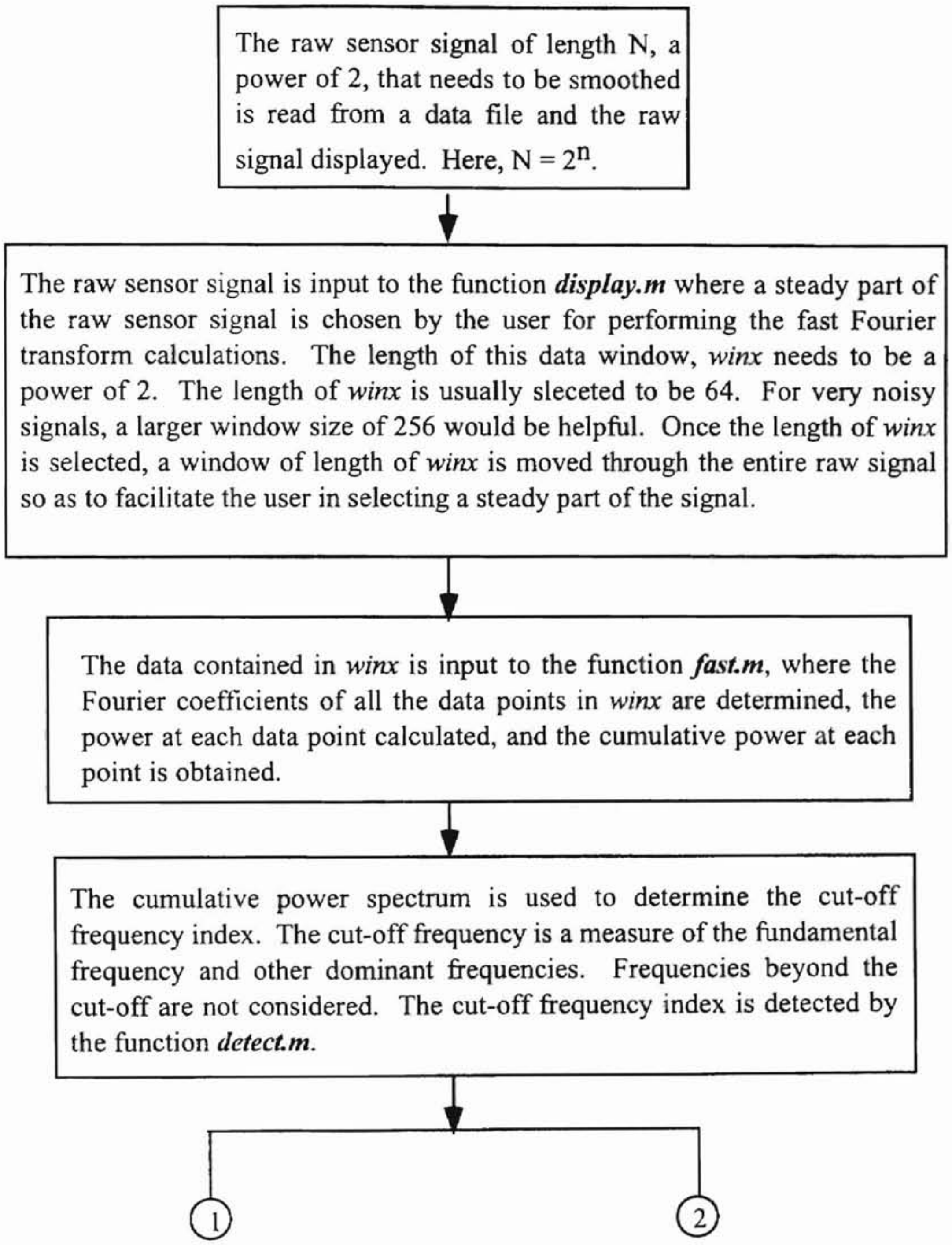
$$min\_win = 4*(N1*T_s)/(freq\_index) \tag{61}$$

This value of the minimum window size in time is displayed and the user prompted to input the length of window of observation in time.

4.4 Automated Trend Extraction algorithm

Our proposed automated trend extraction (ATE) algorithm is described in this section. A flow sheet that shows the various functions involved in the code, and the key variables is included. The algorithm incorporates all the concepts described in this chapter.

FLOW OF CONTROL IN THE AUTOMATED TREND EXTRACTION ALGORITHM

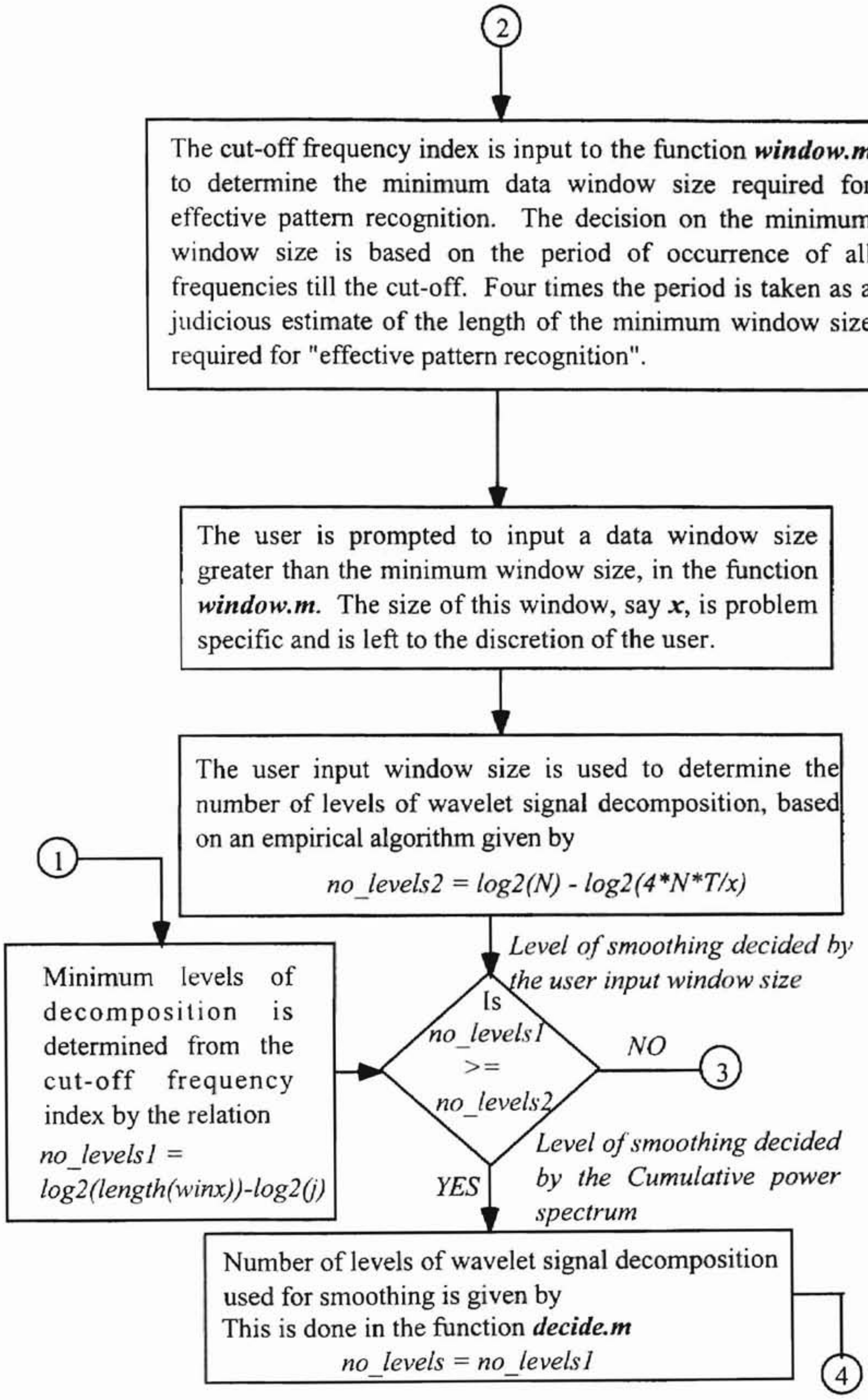


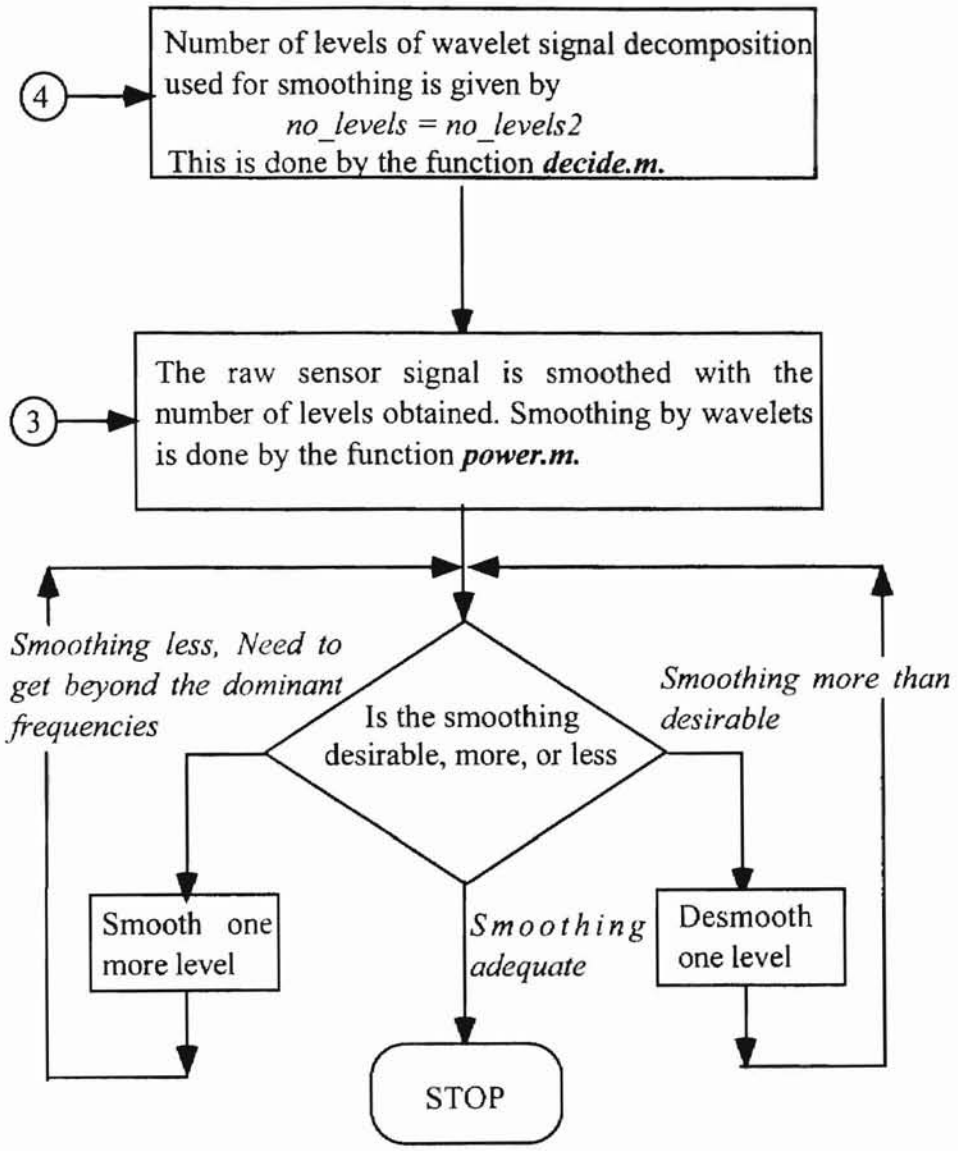
The raw sensor signal of length N, a power of 2, that needs to be smoothed is read from a data file and the raw signal displayed. Here, $N = 2^n$.

The raw sensor signal is input to the function ***display.m*** where a steady part of the raw sensor signal is chosen by the user for performing the fast Fourier transform calculations. The length of this data window, *winx* needs to be a power of 2. The length of *winx* is usually sleceted to be 64. For very noisy signals, a larger window size of 256 would be helpful. Once the length of *winx* is selected, a window of length of *winx* is moved through the entire raw signal so as to facilitate the user in selecting a steady part of the signal.

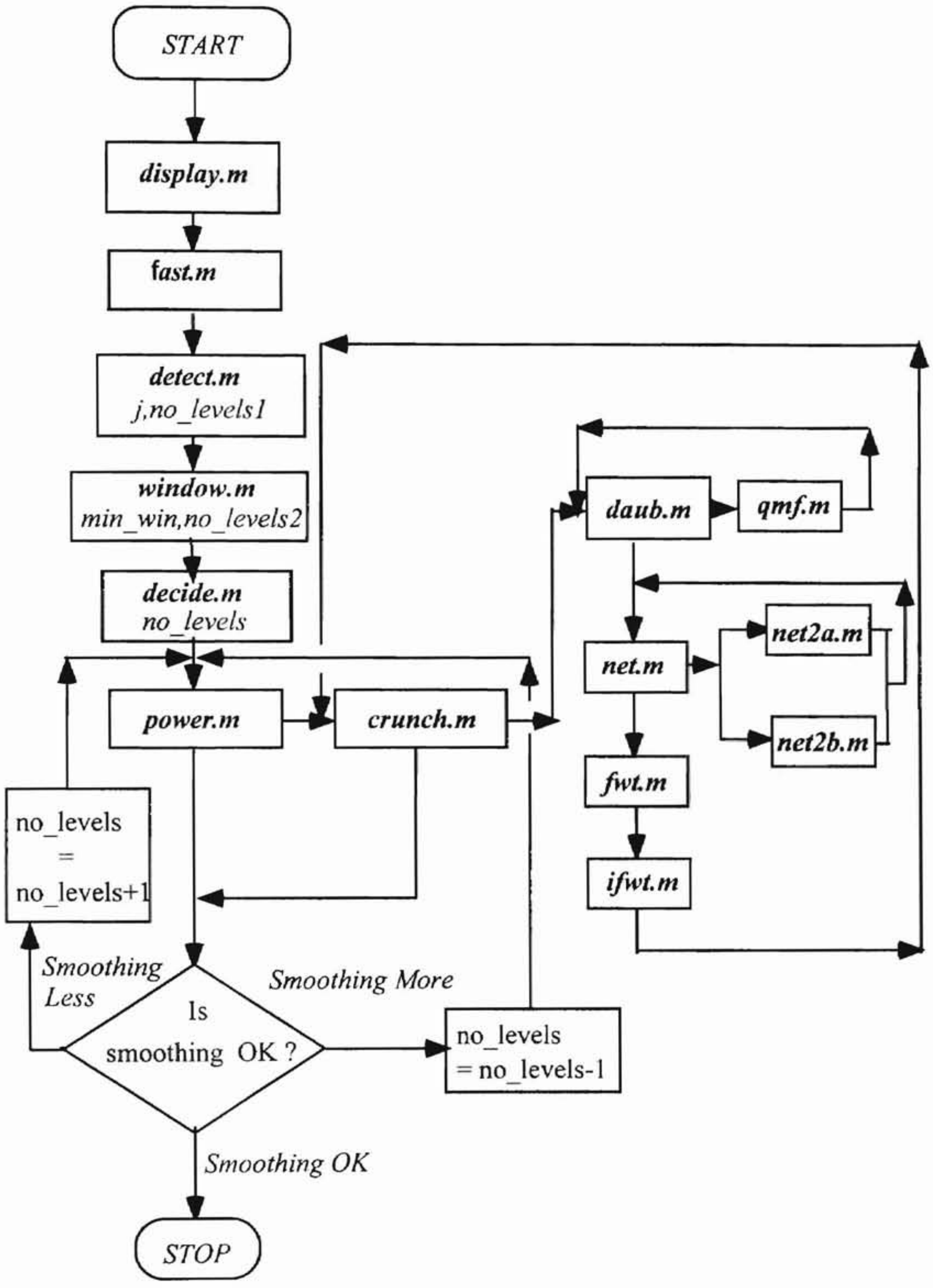
The data contained in *winx* is input to the function ***fast.m***, where the Fourier coefficients of all the data points in *winx* are determined, the power at each data point calculated, and the cumulative power at each point is obtained.

The cumulative power spectrum is used to determine the cut-off frequency index. The cut-off frequency is a measure of the fundamental frequency and other dominant frequencies. Frequencies beyond the cut-off are not considered. The cut-off frequency index is detected by the function ***detect.m***.


2
The cut-off frequency index is input to the function ***window.m*** to determine the minimum data window size required for effective pattern recognition. The decision on the minimum window size is based on the period of occurrence of all frequencies till the cut-off. Four times the period is taken as a judicious estimate of the length of the minimum window size required for "effective pattern recognition".
The user is prompted to input a data window size greater than the minimum window size, in the function ***window.m***. The size of this window, say ***x***, is problem specific and is left to the discretion of the user.
The user input window size is used to determine the number of levels of wavelet signal decomposition, based on an empirical algorithm given by *no_levels2 = log2(N) - log2(4*N*T/x)*
1
Minimum levels of decomposition is determined from the cut-off frequency index by the relation *no_levels1 = log2(length(winx))-log2(j)*
Level of smoothing decided by the user input window size
Is *no_levels1* >= *no_levels2*
NO
3
Level of smoothing decided by the Cumulative power spectrum
YES
Number of levels of wavelet signal decomposition used for smoothing is given by This is done in the function ***decide.m*** *no_levels = no_levels1*
4


4
Number of levels of wavelet signal decomposition used for smoothing is given by
no_levels = no_levels2
This is done by the function ***decide.m.***
3
The raw sensor signal is smoothed with the number of levels obtained. Smoothing by wavelets is done by the function ***power.m.***
Smoothing less, Need to get beyond the dominant frequencies
Is the smoothing desirable, more, or less
Smoothing more than desirable
Smooth one more level
Smoothing adequate
Desmooth one level
STOP

AUTOMATED TREND EXTRACTION CODE



Chapter Summary

This chapter described the bases for an Automated Trend Extraction algorithm. The algorithm utilizes information concerning the monitoring application (user specified window size) and the characteristics of the signal to be smoothed to generate a recommendation for the appropriate degree of wavelet smoothing. Application of the ATE algorithm is illustrated in the next chapter.

Chapter 5

CASE STUDIES

5.1 Introduction

To evaluate the performance of the ATE algorithm, a set of signals with different characteristics and noise content were considered. Plant data from temperature, pressure and flow sensors was used. In every case, a window length of 45 minutes was selected for process monitoring. Table 5.1 summarises the results obtained by the automated trend extraction algorithm for all case studies. The following section presents an analysis of all test cases considered.

Table 5.1

Summary of the results obtained by the ATE code

Case	Figure	Signal length	T_s in mins.	*freq. index*	*min_levels*	Window length in mins.		*act_levels*
						minimum	actual	
1	5.1	1024	1	8	3	32	45	4
2	5.2	1024	1	11	2.54	24	45	4
3	5.3	1024	1	9	2.83	29	45	4
3a	5.4	512	1	8	3	32	45	4
4	5.5	8192	1/6	5	3.67	9	45	7
4a	5.6	2048	2/3	11	2.54	16	45	5
4b	5.7	4096	1/6	6	3.42	8	45	7
4c	5.8	2048	1/6	6	3.68	9	45	7
5	5.9	8192	1/6	15	2.09	3	45	7
5a	5.10	4096	1/6	6	3.42	8	45	7
5b	5.11	1024	1/6	6	3.42	9	45	7

5.2 Individual Case Studies

5.2.1 Case 1-Figure 5.1: The signal considered in this case is characterized by periodic swings of high frequency. The minimum number of levels of wavelet signal decomposition obtained from the cut-off frequency index, *min_levels,* is 3. The number of levels of wavelet signal decomposition determined from the user input process monitoring window length, *act_levels,* is 3.49. As the number of levels of decomposition needs to be an integer, the actual number of levels of decomposition, *act_levels* is incremented to the next integer value 4. The smoothed signal representation obtained by smoothing four levels using sixth order Daubechies wavelets is shown in Figure 5.1d.

5.2.2 Case 2-Figure 5.2: A signal with different characteristics is considered in this case. Although this signal also appears to contain a highly periodic component, the amplitude is much smaller and the frequency much higher than Case 1. The minimum number of levels of wavelet signal decomposition obtained from the cut-off frequency index, *min_levels,* is 2.54. The number of levels of wavelet signal decomposition determined from the user input window size, *act_levels,* is 3.49, and is incremented to 4. The smoothed signal is shown in Figure 5.2d. The smoothed signal approximation clearly captures the true trend of the signal.

5.2.3 Case 3-Figure 5.3: In this case, a signal characterized by moderately large, abrupt changes in its fundamental trend is considered. The number of levels of wavelet signal decomposition obtained from the cut-off frequency index, *min_levels,* is 2.83. The

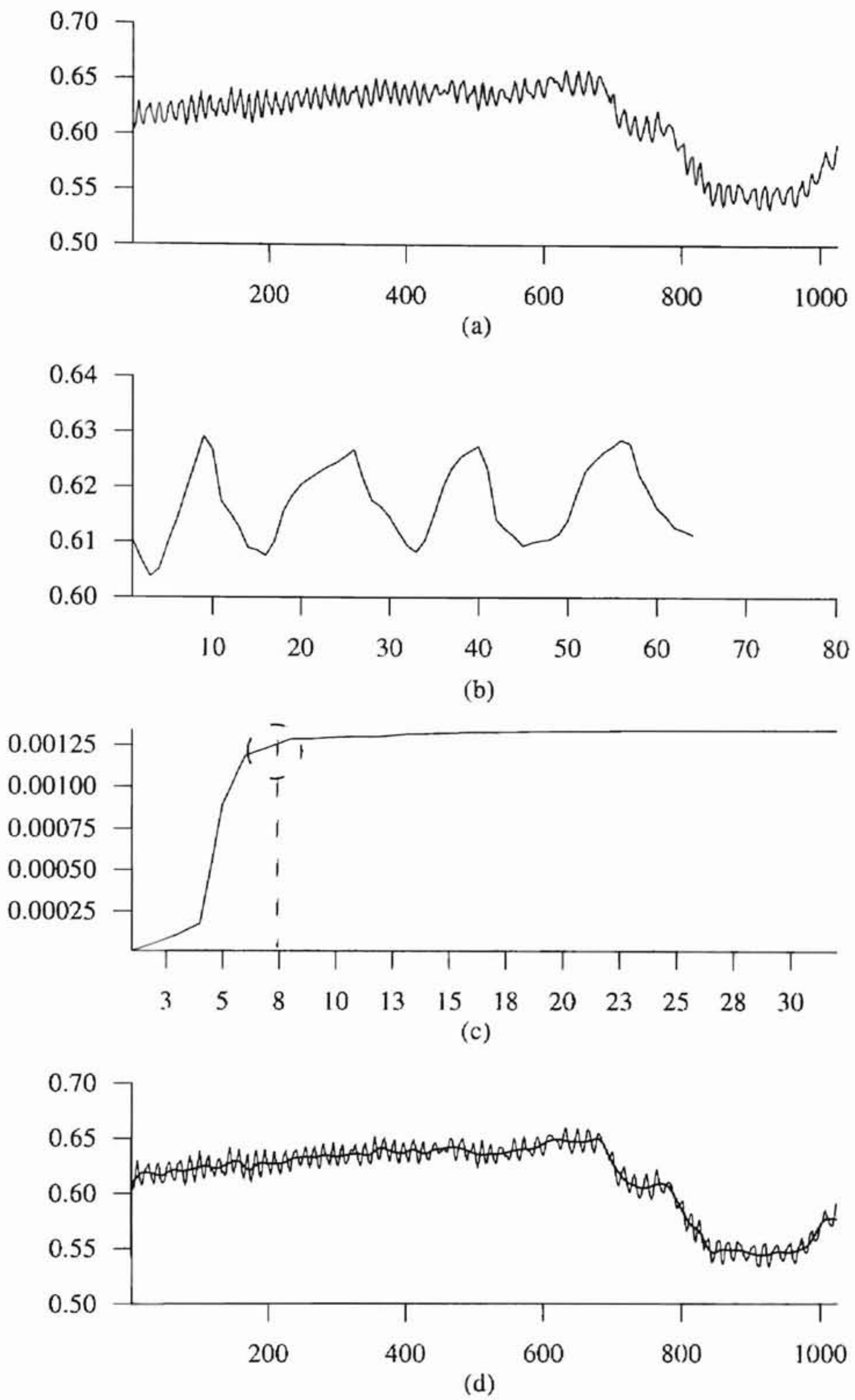


Figure 5.1. (a) original signal with periodic swings (b) First 64 data points of the original signal for obtaining the CPS (c) Cumulative power spectrum (d) Smoothed representation of the original signal using sixth order Daubechies and decomposing four levels.

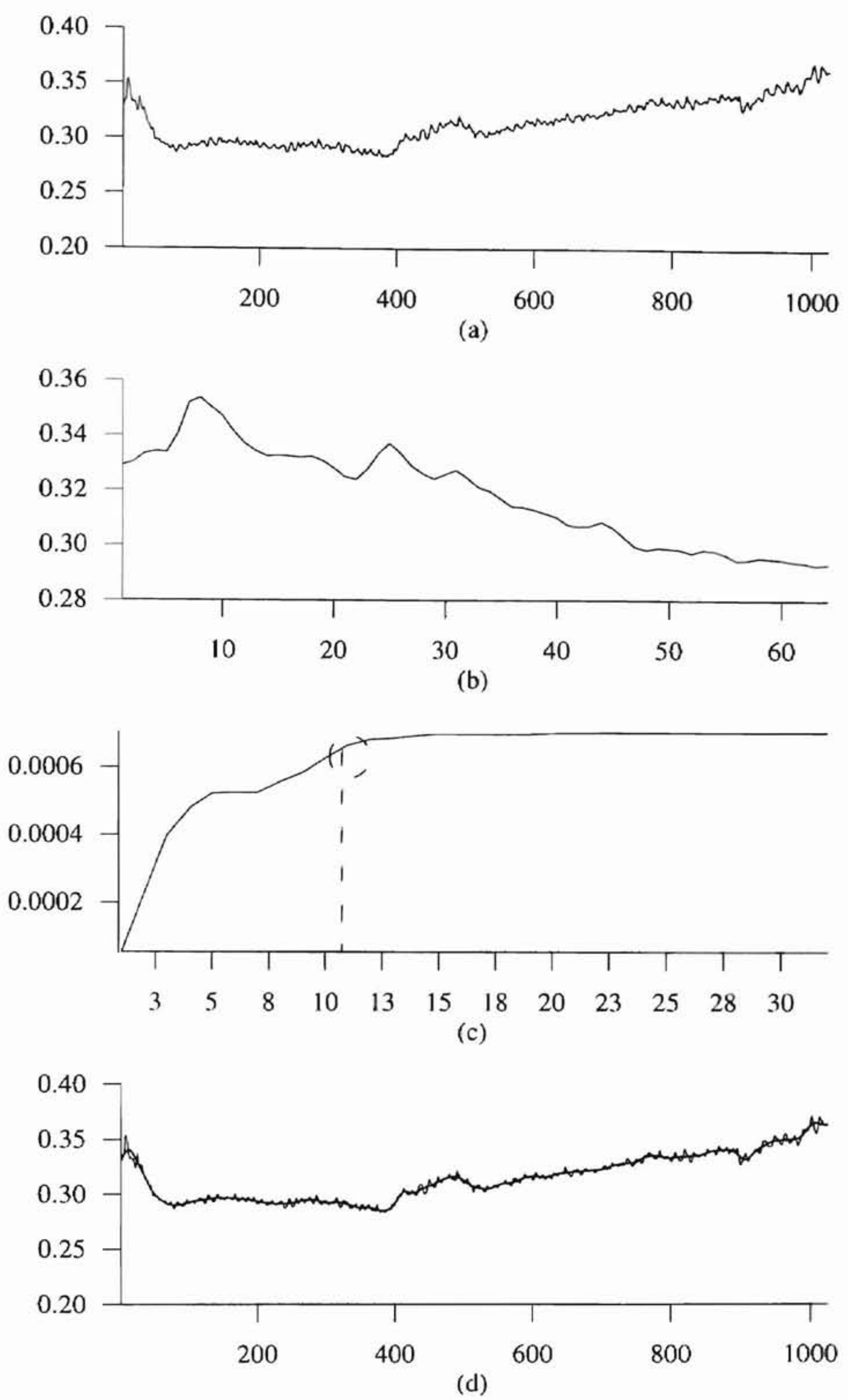
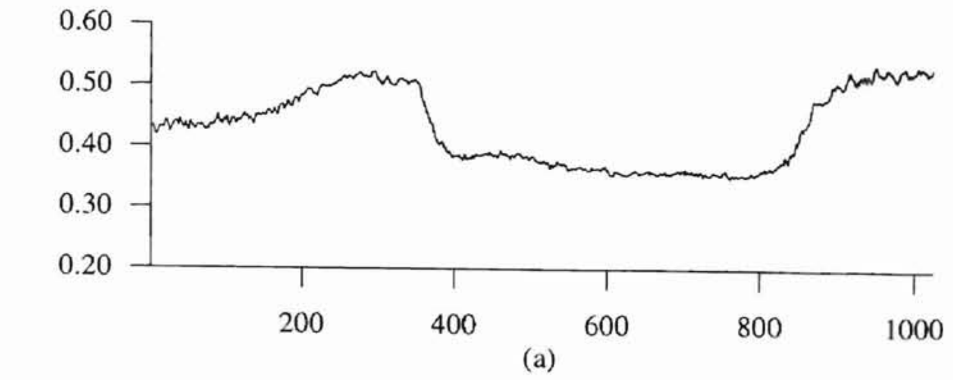
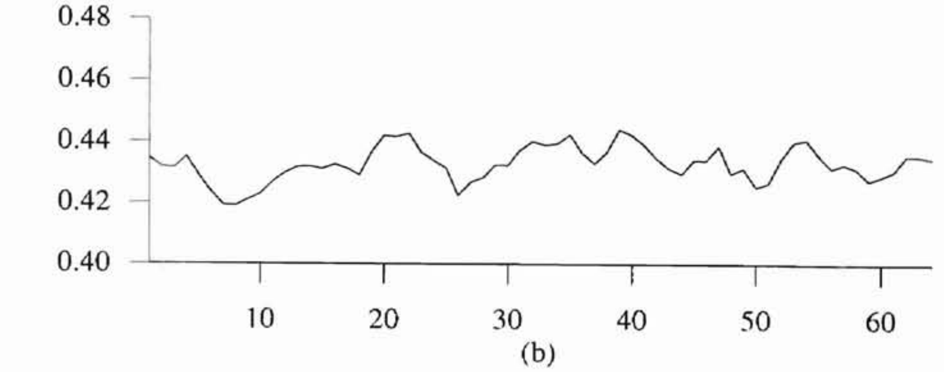
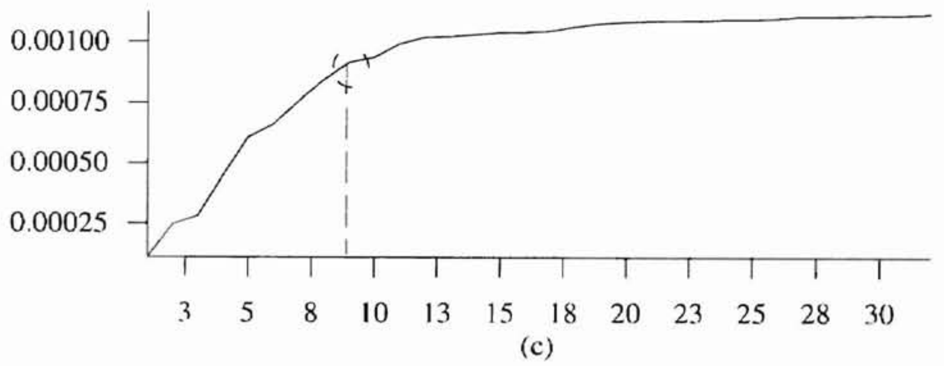


Figure 5.2. (a) original signal (b) First 64 data points of the original signal for obtaining the cumulative power spectrum (c) cumulative power spectrum (d) smoothed representation of the original signal obtained by smoothing four levels, using sixth order Daubechies.







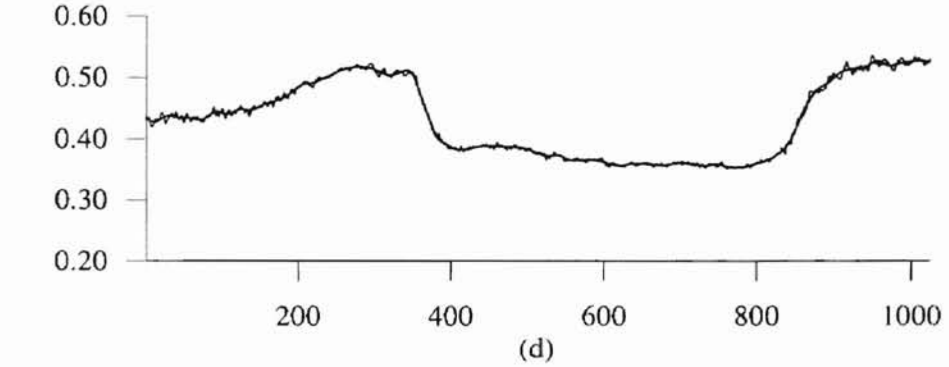


Figure 5.3. (a) original signal (b) First 64 data points of the original signal for for obtaining the cumulative power spectrum (c) cumulative power spectrum (d) Smoothed representation of the original signal, smoothed four levels using sixth order Daubechies wavelets.

number of levels of wavelet signal decomposition determined from the user input window size, *act_levels,* is 3.49, and is incremented to 4. The smoothed signal representation obtained by smoothing four levels using sixth order Daubechies wavelets is shown in Figure 5.3d. Excellent performance is again noted.

5.2.3a Case3a-Figure 5.4: The second half of the signal in Figure 5.3a is considered separately to verify that the smoothing algorithm is independent of signal length. The algorithm is expected to provide the same degree of smoothing as in Figure 5.3d as the noise content in both these signals is the same. The calculated minimum number of levels of wavelet signal decomposition obtained from the cut-off frequency index, *min_levels,* is 3.0. The number of levels of wavelet signal decomposition determined from the user input window size, *act_levels* is 3.49, and is incremented to 4. The smoothed signal representation is shown in Figure 5.4d. To compare the smoothing obtained in Figure 5.4d with that of Figure 5.3d, the second half of the original and smoothed signals in 5.3d is shown in Figure 5.4e. From figures 5.4d and 5.4 e, it is observed that the degree of smoothing obtained is the same in both the cases, and the smoothed patterns are identical.

5.2.4 Case 4-Figure 5.5: The signal considered in this case is sampled at a rate (T_s = 0.167 min.) which is six times higher than the sampling rate of the signals considered to this point (T_s = 1 min.). As a consequence, the high frequency content of the signal is observed to be very prominent. Further, the signal is also characterized by sharp changes

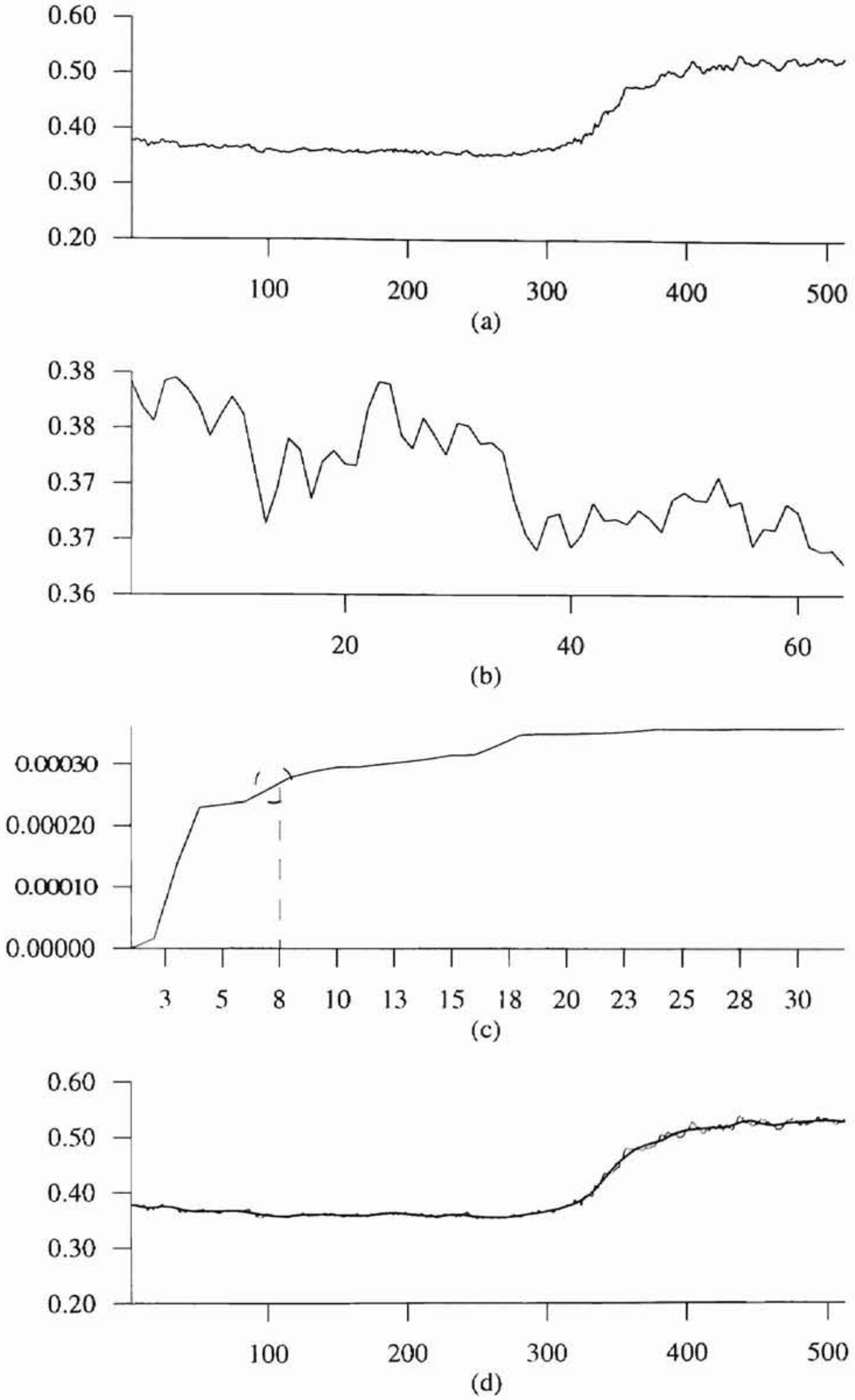

0.60
0.50
0.40
0.30
0.20
100
200
300
400
500
(a)
0.38
0.38
0.37
0.37
0.36
20
40
60
(b)
0.00030
0.00020
0.00010
0.00000
3
5
8
10
13
15
18
20
23
25
28
30
(c)
0.60
0.50
0.40
0.30
0.20
100
200
300
400
500
(d)

Figure 5.4

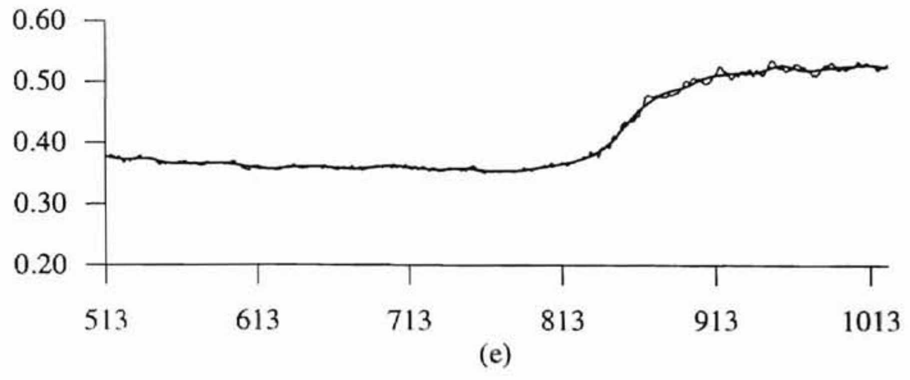


Figure 5.4. (a) original signal, last 512 samples of the signal in Figure 5.3 (a) (b) First 64 data points of tthe signal in (a) for obtaining the cumulative power spectrum (c) cumulative power spectrum (d) smoothed representation of the signal in (a), smoothed four levels, using sixth order Daubechies wavelets (e) last 512 samples of the original and smoothed signal in Figure 5.3(d).

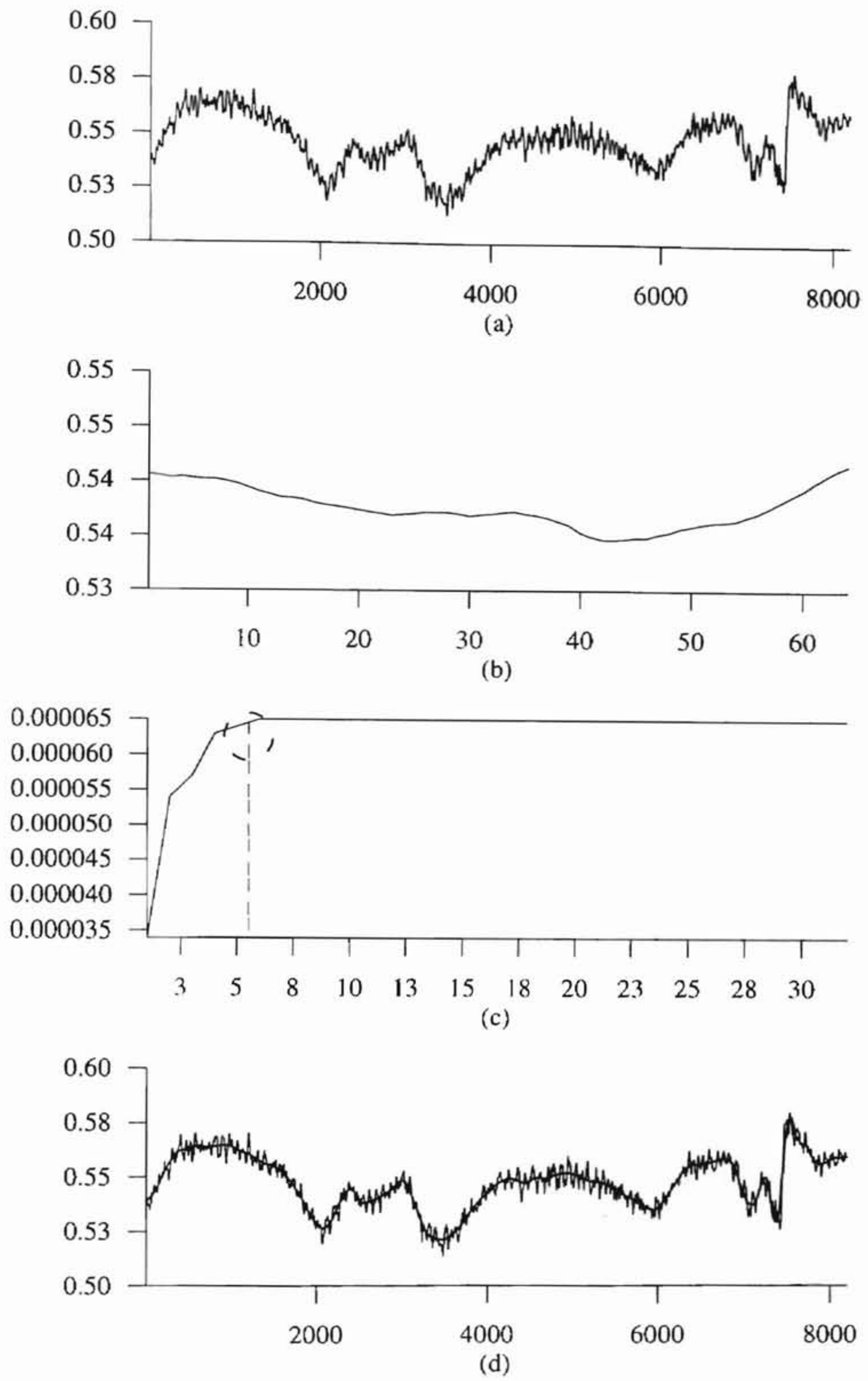


Figure 5.5. (a) original signal (b) First 64 data points of the original signal taken for obtaining the cumulative power spectrum (c) cumulative power spectrum (d) Smoothed representation of the original signal, smoothed seven levels using sixth order Daubechies wavelets.

in its fundamental trend. As the sampling interval is much smaller compared to the previous cases, the number of levels of decomposition in this case should be relatively high. The number of levels of wavelet signal decomposition obtained from the cut-off frequency index, *min_levels,* is 3.67. The number of levels of wavelet signal decomposition determined from the user input window size, *act_levels,* is 6.08, and is incremented to 7. The smoothed signal representation obtained by smoothing seven levels using sixth order Daubechies wavelets is shown in Figure 5.5d. Excellent performance is again noted.

5.2.4a Case 4a-Figure 5.6: For this case study, the previous signal was sampled at a rate four times slower (T_s = 0.667 min. as compared to 0.167 min.). The purpose of this case study is to demonstrate that smoothing performance is dependent on the sampling rate. As compared to the signal in Figure 5.5a, the high frequency content of signal in Figure 5.6a is reduced due to the longer sampling interval (0.667 min. against 0.167 min.). Consequently, the number of levels of decomposition based on the process monitoring window should be two less than in Case 4.

The minimum number of levels of decomposition obtained from the cut-off frequency index is 2.54. The number of levels of decomposition obtained from the user input window length is 4.08 as expected. Figure 5.5d shows the original and smoothed signals. The degree of smoothing is the same as produced in Case 4. This case study demonstrates that the automated trend extraction code is adaptive and achieves a degree of smoothing that is dependent only on the length of the pattern recognition window or

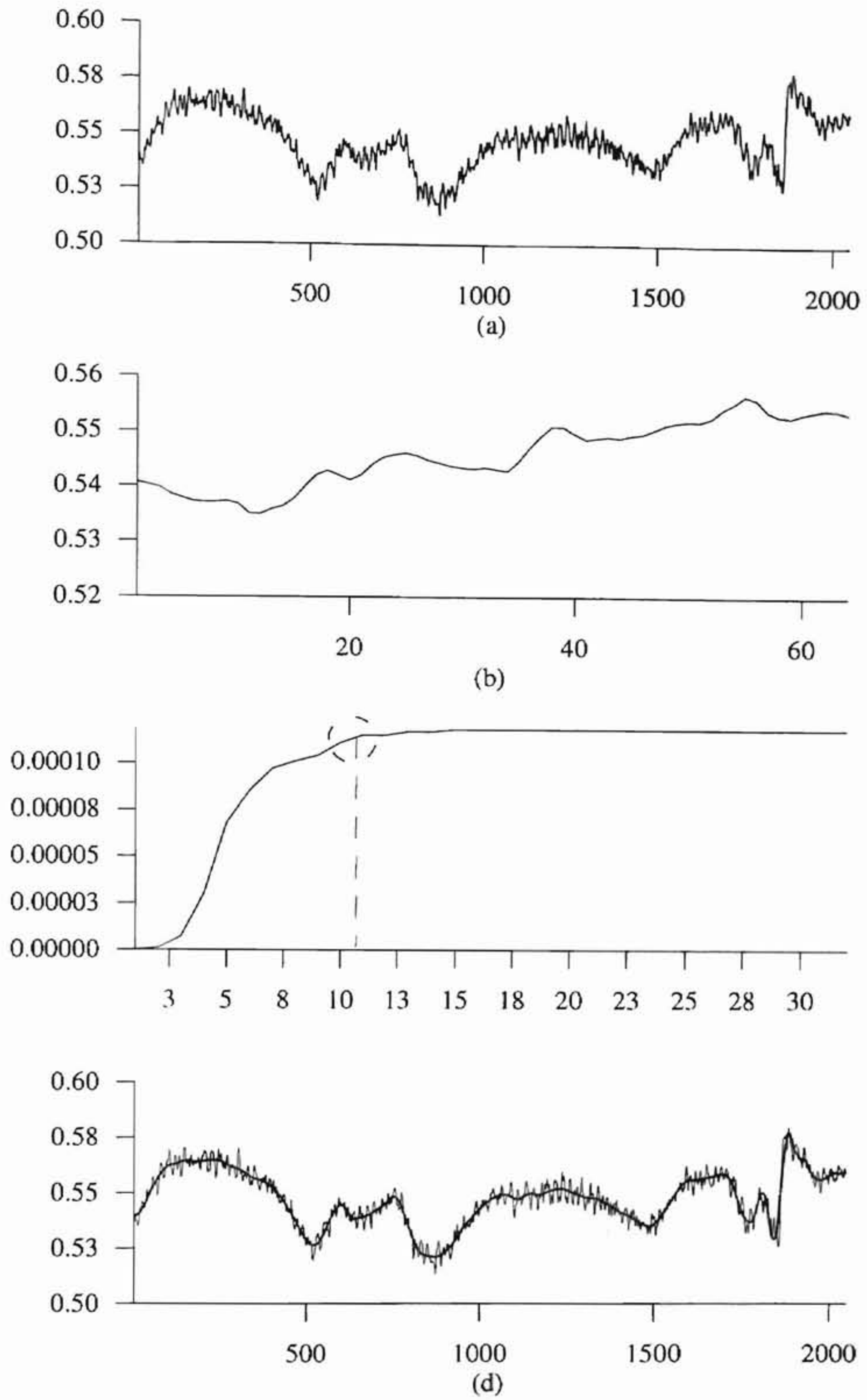


Figure 5.6. (a) original signal, signal sampled every 40 seconds (b) First 64 data points of the signal in (a) for obtaining the cumulative power spectrum (c) cumulative power spectrum (d) Smoothed representation of the original signal, smoothed five levels using sixth order Daubechies wavelets.

the frequency content of the signal, irrespective of the length of the signal.

5.2.4b Case 4b-Figure 5.7: For this case study, the second half of the signal in Figure 5.5a is considered. The purpose of this case is to again demonstrate that smoothing performance is independent of the length of the signal. The noise content of this signal is the same as that in the signal shown in Figure 5.5a; therefore the same degree of smoothing is anticipated. The number of levels of decomposition obtained from the cut-off frequency index, *min_levels,* is 3.42. The number of levels of decomposition from the process monitoring window size, *act_levels,* is 6.08, and is incremented to 7 levels. The signal in Figure 5.7a smoothed seven levels using sixth order Daubechies wavelets is shown in Figure 5.7d. To compare the smoothing obtained in Figure 5.7d with that of Figure 5.5d, the second half of the original and smoothed signals in 5.5d are shown in Figure 5.7e. From figures 5.7d and 5.7e, it is observed that the degree of smoothing obtained is the same in both the cases, and the smoothed patterns are identical.

5.2.4c Case 4c-Figure 5.8: A signal consisting of the last 2048 samples of the signal in Figure 5.5a is considered to demonstrate again that the smoothing algorithm is independent of the length of the signal. The number of levels of decomposition obtained from the cut-off frequency index, *min_levels,* is 3.68. The number of levels of decomposition obtained from the user input process monitoring window size is 6.08, and is incremented to 7. The signal in Figure 5a smoothed seven levels using sixth order Daubechies wavelets is shown in Figure 5.8d. The last 2048 samples of the original and smoothed signals in Figure 5.5d are shown in Figure 5.8e for comparing the degree of

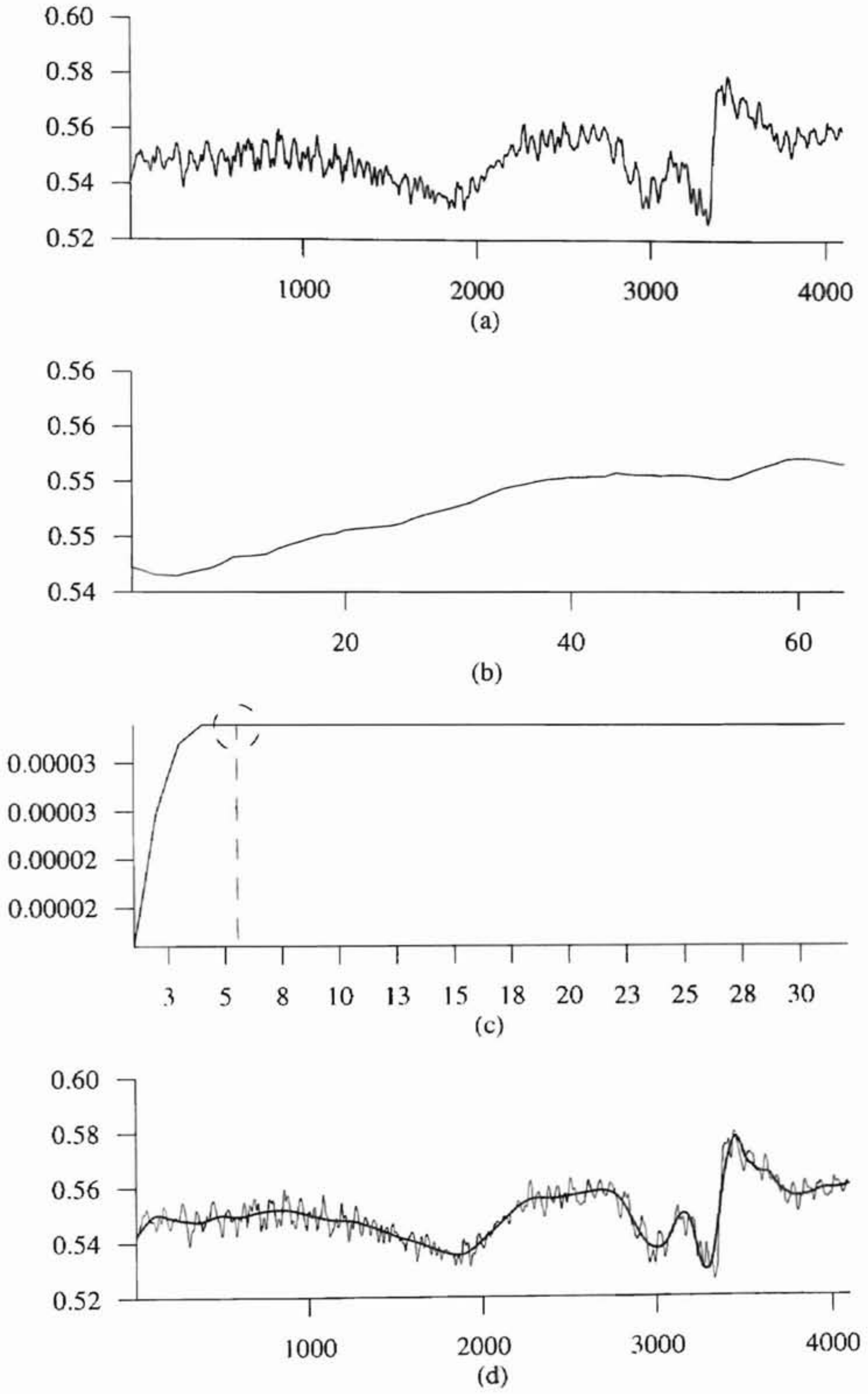

0.60
0.58
0.56
0.54
0.52
1000
2000
3000
4000
(a)
0.56
0.56
0.55
0.55
0.54
20
40
60
(b)
0.00003
0.00003
0.00002
0.00002
3
5
8
10
13
15
18
20
23
25
28
30
(c)
0.60
0.58
0.56
0.54
0.52
1000
2000
3000
4000
(d)

Figure 5.7

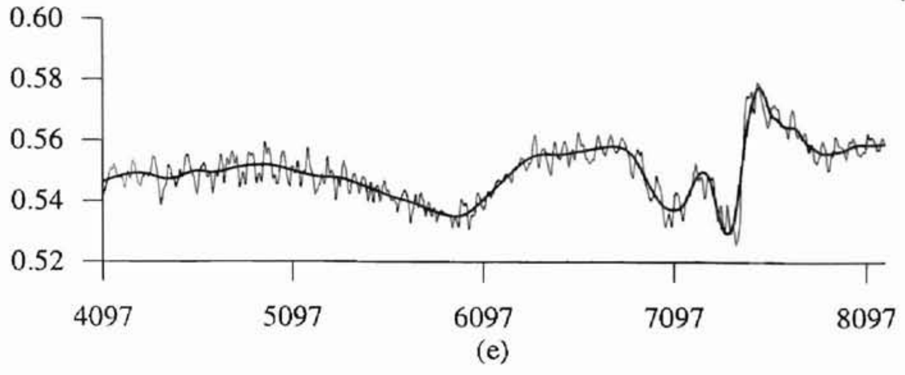


Figure 5.7. (a) original signal, 4096 samples, second half of the signal in Figure 5.5(a) (b) First 64 data points of the signal in (a) for obtaining the cumulative power spectrum (c) cumulative power spectrum (d) Smoothed representation of the signal in (a), smoothed seven levels using sixth order Daubechies wavelets (e) second half of the original and smoothed signal in Figure 5.5(d).

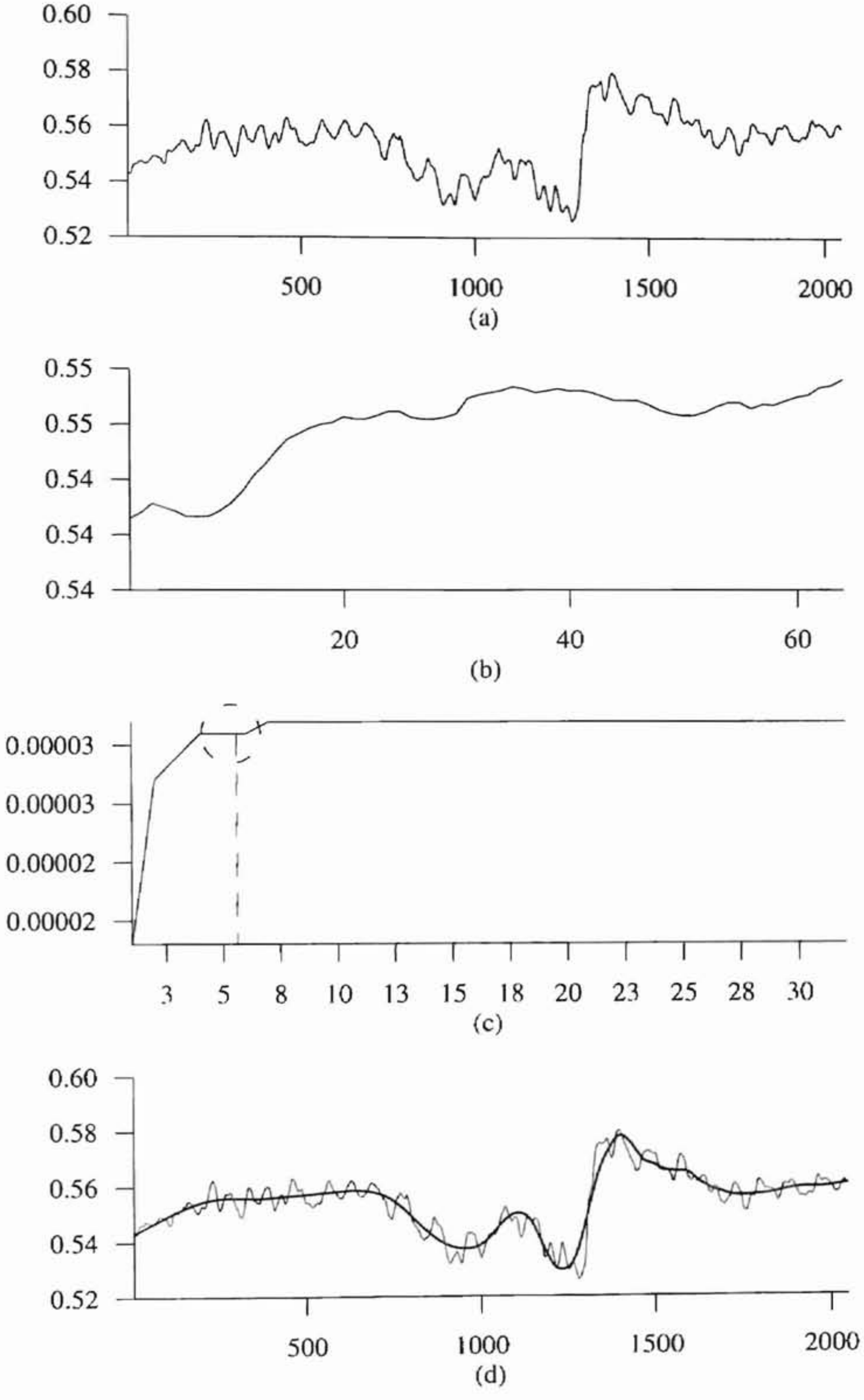

0.60
0.58
0.56
0.54
0.52
500
1000
1500
2000
(a)
0.55
0.55
0.54
0.54
0.54
20
40
60
(b)
0.00003
0.00003
0.00002
0.00002
3
5
8
10
13
15
18
20
23
25
28
30
(c)
0.60
0.58
0.56
0.54
0.52
500
1000
1500
2000
(d)

Figure 5.8

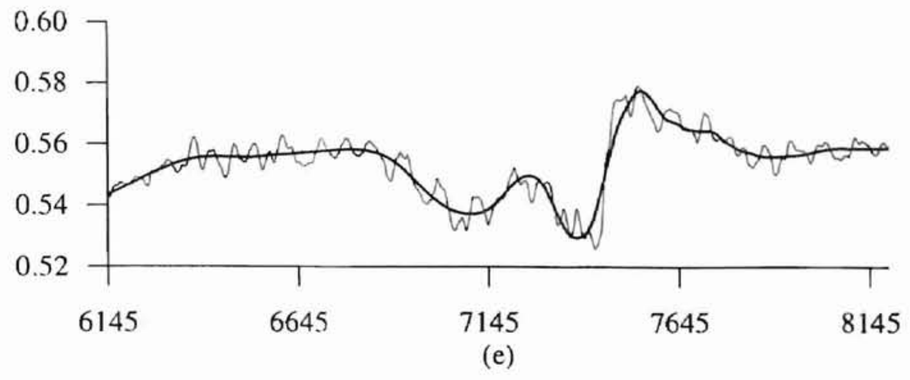


Figure 5.8. (a) original signal, last 2048 samples of signal in Figure 5.5 (a) (b) First 64 samples of signal in (a) for obtaining the cumulative power spectrum (c) cumulative power spectrum (d) smoothed representation of the signal in (a), smoothed seven levels using sixth order Daubechies wavelets (e) last 2048 samples of the original and smoothed signal in Figure 5.5(d).

smoothing obtained in Figure 5.8d. From both these figures it can be observed that the degree of smoothing obtained is the same in both cases, and the smoothed patterns are identical in nature. The above two case studies demonstrate that the degree of smoothing obtained is the same for signals of the same frequency content, even though they may be of different lengths, when the process monitoring window length employed is the same.

5.3 Discussion

In all the test cases, the smoothing obtained is as desired and the smoothed representation preserves the fundamental trend of the original signal. All the above cases show clearly that the degree of smoothing obtained is dependent on the pattern recognition window length and is independent of the length of the original signal, for a fixed sampling rate. We note, however, that the length of the signal should be sufficiently large so as to provide more available levels of decomposition than required to smooth the signal.

The accuracy of the cut-off frequency index is critical in determining the minimum level of smoothing, as the actual window length used for process monitoring is indirectly dependent on the value of the cut-off frequency obtained. In all the above cases, the cumulative power spectrum obtained is well defined and the cut-off frequency index obtained is accurate. However, if the cumulative power spectrum is not well defined, determination of the cut-off frequency is difficult and the degree of smoothing obtained would depend entirely dependent on the discretion of the user in selecting the window length for process monitoring. The following case presents such an example.

5.3.1 Case 5: The signal considered in this case is characterized by a large amount of high frequency content distributed unevenly, as can be seen from Figure 5.9a. The cumulative power spectrum obtained is shown in Figure 5.9c. It can be observed that the cumulative power spectrum does not increase steadily and level off. It increases in small steps. Consequently, a single striking bend is not observed as in all the previous cases. In such cases where the cumulative power spectrum continues to increase steadily without leveling off, determining the cut-off frequency index accurately becomes extremely difficult. In this case, the cut-off frequency index is determined to be 15. The number of levels of wavelet signal decomposition obtained from the cut-off frequency index, *min_levels,* is 2.09.

The accuracy of the minimum window length required for process monitoring depends on how accurately the cut-off frequency index is determined. In such cases, the minimum window size required may be lesser or greater than the actual window size the user expects to input based on his experience. If the minimum window size required is less than the window size anticipated by the user, and if the user input window size is judiciously selected, appropriate smoothing is expected. However, if the minimum window size required is greater than the window size anticipated by the user, the user will have to compromise and input a larger window size than really desired. This may bring about greater smoothing than desired (refer equation 57).

In this specific case, a user input window size of 45 minutes is greater than the minimum obtained from the ATE code, so the degree of smoothing obtained is as desired. The number of levels of wavelet signal decomposition determined from the user input window size, *act_levels,* is 6.08, which is incremented to 7. The smoothed signal

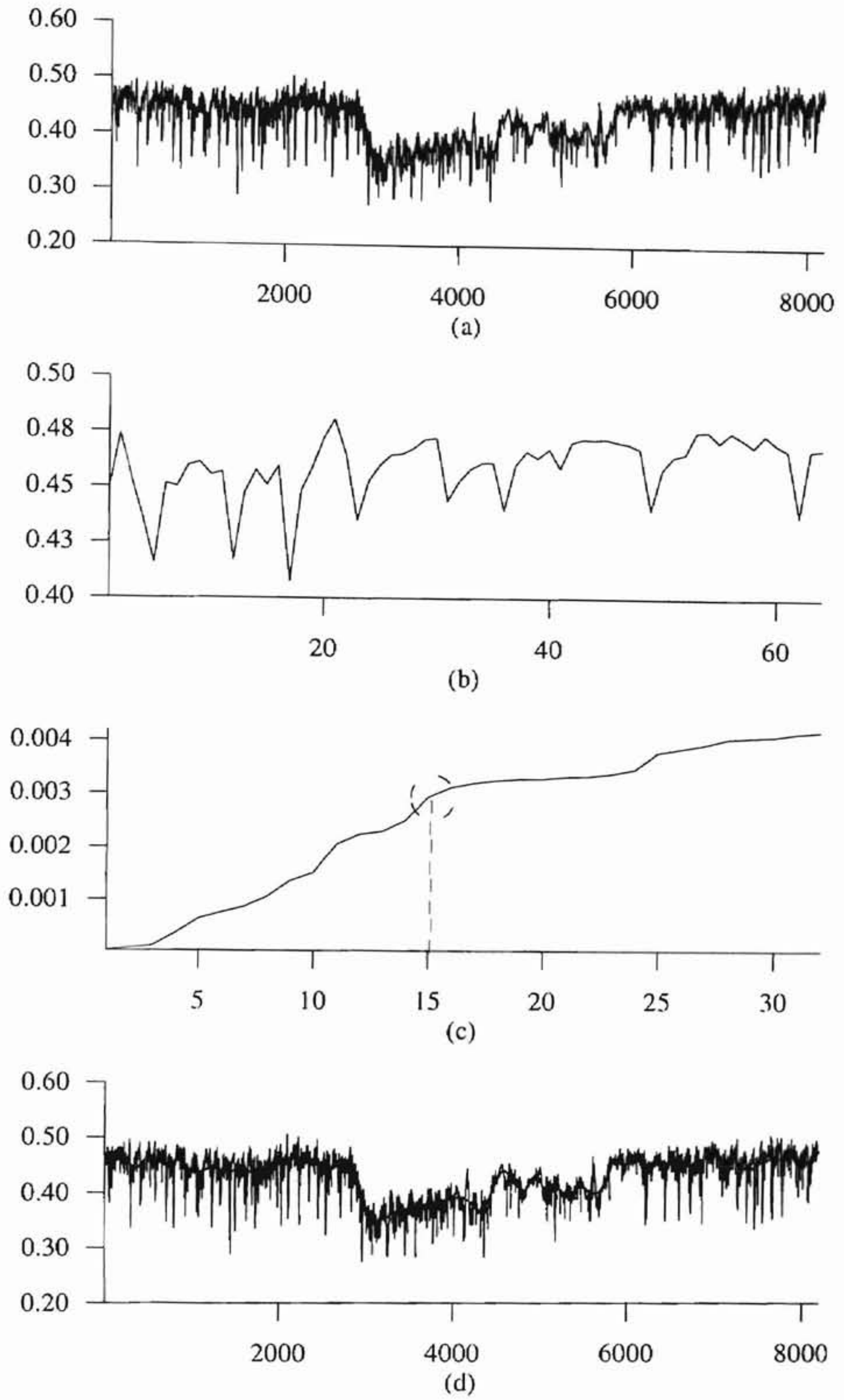


Figure 5.9. (a) original signal (b) First 64 data points of the original signal taken for obtaining the cumulative power spectrum (c) cumulative power specctrum (d) Smoothed representation of the original signal, smoothed seven levels using sixth order Daubechies wavelets.

representation obtained by smoothing seven levels using sixth order Daubechies wavelets is shown in Figure 5.9d. From this figure, it can be observed that the degree of smoothing is as desired. However, the desirable degree of smoothing obtained is entirely dependent on the process monitoring window size as the cut-off frequency index obtained is not accurate.

The presence of large quantity of noise distributed unevenly does not yield a well behaved cumulative power spectrum. In such cases, using a larger window size for the fast Fourier transform may yield a cumulative power spectrum that increases steeply initially and then levels off.

Figure 5.10a shows the first 128 data points of the signal in Figure 5.9a. The associated cumulative power spectrum is shown in Figure 5.10b. Figure 5.10c and 5.10d show the first 256 data points of the original signal in Figure 5.9a, and the resulting cumulative power spectrum obtained. In both cases, the cumulative power spectrum obtained is not as desired. However, when a data window of the first 512 points of the signal in Figure 5.9a is used for obtaining the cumulative power spectrum, the cumulative power spectrum obtained is defined much better as seen in Figure 5.10f. The cut-off frequency index obtained is 15.

The second set of 512 data points of the original signal is considered and the resulting cumulative power spectrum is shown in Figure 5.10h. The cut-off frequency obtained from this cumulative power spectrum is 19. All these results show that for a signal with non-uniform noise, selecting a relatively larger window size for performing the fast Fourier transform results in a well behaved cumulative power spectrum.

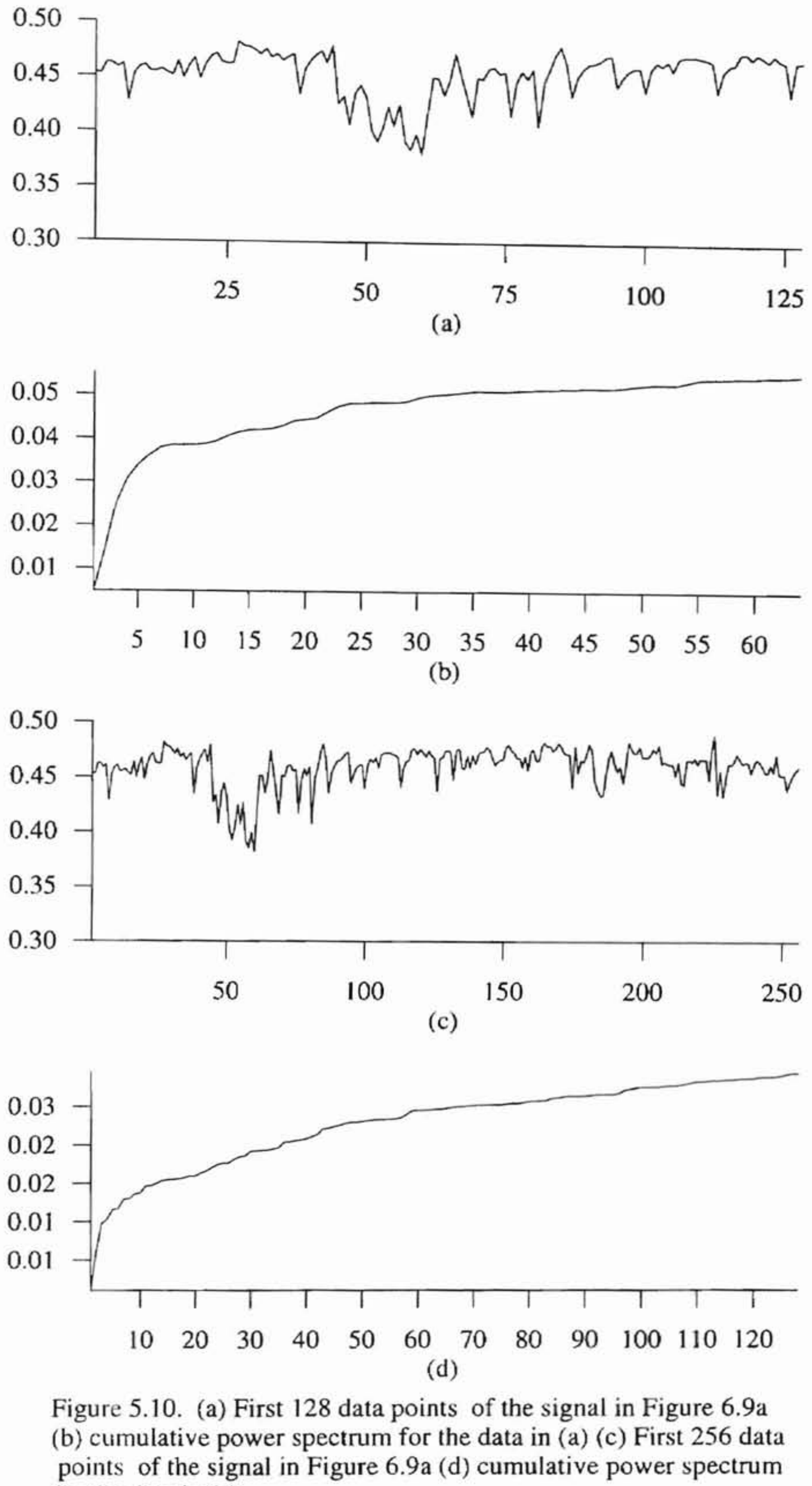


Figure 5.10. (a) First 128 data points of the signal in Figure 6.9a (b) cumulative power spectrum for the data in (a) (c) First 256 data points of the signal in Figure 6.9a (d) cumulative power spectrum for the data in (c).

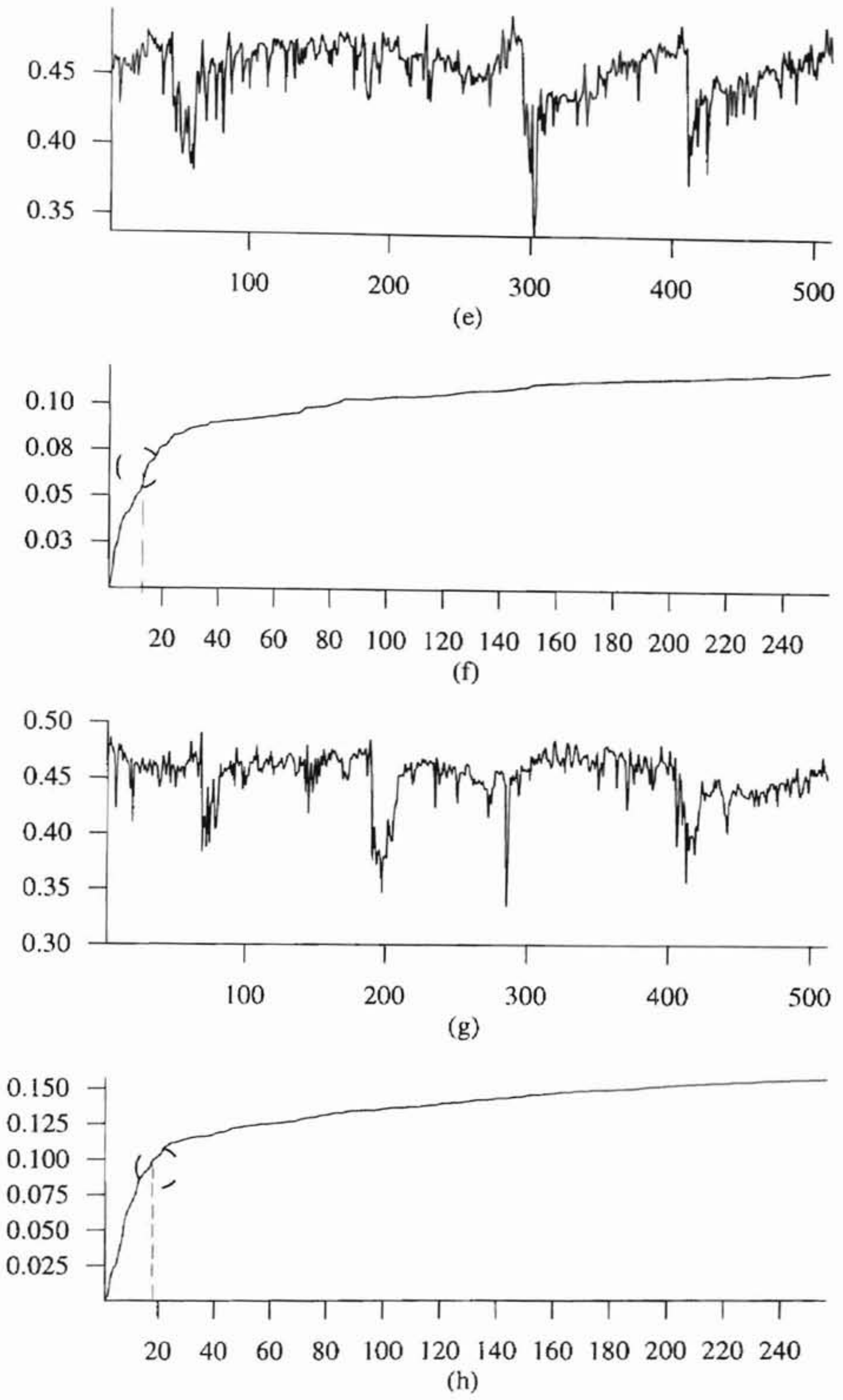


Figure 5.10. (contd.) (e) First 512 data points of signal in Figure 6.9a. (f) cumulative power spectrum for data in (e) (g) Second set of 512 data points of the signal in Figure 6.9a (h) cumulative power spectrum for data in (g).

5.4 Chapter Summary

Various signals with different characteristics have been studied to evaluate the performance of the automated trend extraction algorithm. The important observations are listed below:

(1) The smoothed representation obtained is normally determined by the user input window length for process monitoring. The number of levels of decomposition obtained from the cut-off frequency, *min_levels*, only gives the minimum number of levels to be decomposed so as to retain the dominant frequencies present in the original signal. To obtain a smoothed representation of the original signal, smoothing should be performed beyond the dominant frequencies.

(2) The window length for process monitoring is determined by the user based on experience with the problem at hand. The user input window length determines the number of levels of decomposition beyond the dominant frequencies and results in the desired degree of smoothing. The minimum window length required for process monitoring is directly dependent on how accurately the cut-off frequency index is determined.

(3) Accurate determination of the cut-off frequency index depends on the behavior of the cumulative power spectrum. If the cumulative power spectrum is not well defined, the cut-off frequency index determined may not be accurate. In such a situation, the smoothed representation obtained depends totally on the discretion of the user in selecting the window length for process monitoring.

(4) If the cumulative power spectrum is well defined, the required degree of smoothing is the same for signals with the same frequency content, irrespective of the length of the signal, for the same process monitoring window length.

Chapter 6

CONCLUSIONS

Wavelets provide a much better alternative for trend extraction than conventional methods of signal processing such as the direct methods, digital filters, and the Fourier transform. Wavelets possess the ability to extract essential trends from process signals thus helping to provide compact representation which is imperative for efficient real-time pattern-based monitoring and control.

This work presented an automated approach to obtain the desired degree of smoothing as required for real-time pattern-based monitoring applications. The properties of the wavelet and fast Fourier transforms are exploited to achieve this purpose. Our automated trend extraction system can be used as a standalone system which helps operators create their own process monitoring applications.

The automatic trend extraction algorithm automatically recommends an appropriate degree of smoothing by utilizing information concerning the monitoring application (user specified window size) and the characteristics of the signal to be smoothed. The minimum degree of smoothing is determined from the characteristics of the signal and is dependent on how accurately the cut-off frequency index is determined. The cut-off frequency is determined from the cumulative power spectrum, which is determined by performing the fast Fourier transform operation on a selected part of the original signal. In some cases, when the cumulative power spectrum is not be well defined, the minimum window size determined would be inaccurate may even be greater than the actual window size anticipated by the user. In such cases, the user will have to compromise and input a larger window size resulting in less smoothing than desired.

Performance of the automated trend extraction algorithm is independent of signal length for a fixed sampling rate. Consequently, the algorithm is suitable for widespread application without constraint.

Recommendations and future work

The approach adopted for most of the work done is basically empirical in nature. This work needs to be consolidated with a theoretical background. Most of the relations and parameters used in this work are based purely on experience and knowledge of sensor signal behavior. This empirical work needs to be supported by a more generalized mathematical basis. Following are some of the recommendations

- The relations used to convert the cut-off frequency index and the user input window size to the number of levels of wavelet signal decomposition are empirical. These relations need to be given a strong theoretical basis.

- This work uses window lengths of 64, 128, 256 and 512. For most signals with uniformly distributed noise, a window length of 64 for fast Fourier transform computation suffices. However, for signals with high frequency content distributed non-uniformly, a large window length for Fourier decomposition in order to obtain a well behaved cumulative power spectrum. A robust relation between the characteristics of the signal and the length of the data window chosen for fast Fourier transform computation needs to be developed.

- In this work, sixth order Daubechies wavelets have been used exclusively. The wavelet order could be adaptively modified with the level of wavelet signal decomposition, to potentially provide better decomposition. Higher order wavelets could be used at the lower levels of decomposition and vice versa. This technique would minimize distortions due to convolution.

- In this work, wavelets belonging to the Daubechies family were used. Other wavelets families should be studied and used for this purpose. A generalized technique for determining the most appropriate wavelet family for a particular signal would be desirable.

BIBLIOGRAPHY

(Edited by) Benedetto J.J., and Frazier, W.F., Wavelets Mathematics and Applications, CRC Press, Boca Raton (1994).

Bogert, B.P. Informal comments on the uses of Power Spectrum Analysis, *IEEE Transactions on Audio and Electroacoustics,* Vol. AU-15, No.2, 70-73, June 1967.

Brigham, E.O. *Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, New Jersey (1974).

Chui, C.K. *An introduction to wavelets.* Academic Press, Inc., College Station, Texas (1992b).

Chui, C.K. Wavelets: *A Tutorial in Theory and Applications.* Academic Press, Inc., College Station, Texas (1992b).

Cohen A., and R.D. Robert, *Wavelets and Multiscale Signal Processing*, Chapman & Hall (1995).

Cohen, L. *Time - frequency analysis* Prentice Hall, Englewood Cliffs, New Jersey (1995).

Daubechies, I. The wavelet transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theory.* 36, 961-1005 (1990).

Daubechies, I. The wavelet transform: a method for time-frequency localization. Advances in Spectrum Analysis and Array Processing. Prentice Hall, Englewood Cliffs, New Jersey (1991).

Daubechies, I. and J.C. Lagarias. Two - scale difference equations I exiistence and global regularity of solutions. *SIAM Journal of Mathematical Analysis* 2, 1388-1410 (1992a).

Daubechies, I. *Ten Lectures on wavelets.* Society for Industrial and Applied Mathematics, Philadelphia (1992b).

Daubechies, I. Orthonormal bases of compactly supported wavelets. *SIAM Journal of Mathematical Analysis* 24, 499-519 (1993).

Hale J.C., and Sellars H.L., Historic Data Recording for Process computers, *Chemical Engineering Progress, Nov. 1981*, 38-43.

Hamming, R.W. *Digital Filters*, Prentice-Hall, Englewood Cliffs, New Jersey (1977).

Kaiser, G.F. *A friendly guide to wavelets*. Birkhauser, Boston (1994).

Kudic, A. and N. Thornhill, Data Compression for process Monitoring, Preprints *ISPE* (1995).

Ludemann, L.C., *Fundamentals of Digital Signal Processing*, Harper & Row, New York (1992).

Meyer, Y., *Wavelets and Operators*, Cambridge University Press, (1992).

Meyer, Y. (Translated and revised by Robert D. Ryan), *Wavelets Algorithms and Applications*, SIAM (1994).

Morrison, N., *Introduction to Fourier Analysis*, John Wiley & Sons, New York (1994).

Motard, R.L. and B. Joseph. Wavelet applications in Chemical Engineering. Kluwer Academic Publishers, Norwell, MA (1994).

Oppenheim, A.V. and R.W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewoood Cliffs, New Jersey (1975).

Press, W. H., S.A. Teukolsky, W.T. Vetterling and B.P. Flannery. *Numerical Recipes in C The art of Scientific Computing*. Cambridge Universiy Press, (1992).

Programs for Digital Signal Processing, IEEE Press, John Wiley & Sons, New Tyork (1979).

Rabiner, L.R. and B. Gold, *Theory and Applications of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey (1975).

Raghavan, Vinod. Wavelet representation of sensor signals for monitoring and control. M.S. thesis, Oklahoma State University (1994).

Ramamohan Rao K. and D.F. Elliott, *Fast Fourier Transform - Algorithms, Analyses, Applications*, Academic Press, New York (1982).

Rioul O., and V. martin, Wavelets and Signal Processing, IEEE Signal Processing magazine, 14-38, October 1991.

Seborg, D.E., T.F. Edgar., and D.A. Mellichamp, *Process Dynamics and Control*, John Wiley and Sons (1989).

Strang, G. Wavelets and dilation equations: A brief introduction. *Society for Industrial and Applied Mathematics* 31, 614-627 (1989).

Strang, G. Wavelet transforms versus Fourier transforms. *Bulletin of the American Mathematical Society* 28, 288-305 (1993).

Walter, G.G., *Wavelets and other Orthogonal Systems with Applications*, CRC Press, Boca Raton, (1994).

Welch, P.D., The use of Fast Fourier Transform for the Estimation of Power Spectra: A method based on time averaging over short, modified periodograms, *IEEE Transactions on Audio and Electroacoustics*, Vol. AU-15, No.2, 74-76, June 1967.

Whiteley, J.R. Application of adaptive networks to trend analysis of process sensor data. M.S. thesis, The Ohio State University (1989).

Whiteley, J.R. and J.F. Davis. Application of neural networks to qualitative interpretation of process sensor data. *Presented at AIChE 1990 Spring. Mtg,* Orlando, FL, (1990a).

Whiteley, J.R. and J.F. Davis. Back-propagation neural networks for qualitative interpretation of process sensor data. *Presented at AIChE 1990 Annl. Mtg,* Chicago, (1990b).

Whiteley, J.R. and J.F. Davis. Knowledge-based interpretation of sensor patterns. *Accepted for publication in Computers chem. Engng* (1991a).

Whiteley, J.R. and J.F. Davis. Qualitative interpretation of process sensor data: Characterization of the problem and analysis of potential implementation methods. Technical report, Department of Chemical Engineering and Laboratory for AI research, The Ohio State University (1991b).

APPENDIXES

% boxcar.m

```
% Written by Ganti S. Srinivas on February 15, 1996
% This is a code for data compression by the Box Car algorithm
% The original authors are J.C. Hale and H.L. Sellars

% This algorithm reads the input vector and the recording limit from
% the user and passes the input vector and the recording limit to the
% function car.m where recording takes place.

% For the entire length of the input vector, each value is compared with the
% previously recorded value . If the difference is more than a prespecified
% limit, the previous input value processed is recorded, not the input value
% which triggered the recording.

% The first recorded value is initialized to the first input value.


clear

fprintf('BOXCAR ALGORITHM IS BEINGRUN\n');

%freak=1;

%while(freak==1)

 file_namea=input('Enter the input file you want  : ','s');
 [fida]=fopen(file_namea,'r');

 file_namec=input('Enter the name of the file to be generated ; This file contains the
smoothed values of the Sensors :','s');

 H=input('Enter the recording limit parameter ');

 fprintf('\n\n');


 X=fscanf(fida,' %f ',inf);

 [y,t,N]=car(X,H); % Record the appropriate input values and return them

 M=length(y);

 DCR=(1-(M/N));
```

```
fprintf(['DATA COMPRESSION OBTAINED IS ',num2str(DCR)]);
fprintf('\n\n');

plot(X);
hold on
plot(t,y,'g');
xlabel('Time index')
ylabel('Normalized value')

fprintf('Click the mouse on the graph for the recording limit to be displayed');


gtext(['Recording limit is ',num2str(H)]);
fprintf('Hit any key to continue after the recording limit is displayed');
hold off

pause

title('Boxcar version of the original signal');
%print -Pps407en


plot(t,y,'g');
xlabel('Time index')
ylabel('Normalized value')
title('Boxcar version of the original signal');
fprintf('Click the mouse on the graph for the recording limit to be displayed');
gtext(['Recording limit is ',num2str(H)]);
fprintf('Hit any key to continue after the recording limit is displayed');
%print -Pps407en



fopen(file_namec,'w');
for i=1:M
fprintf(file_namec, '%d %f\n', t(i), y(i));
end
fclose('all');


%freak=input('Enter 1 if you want to try the whole loop again, else 0 : ');

%end
```

% car.m

```
% Written by Ganti S. Srinivas on February 15, 1996
% This is a code for data compression by the Box Car algorithm
% This function takes input vector and the recording limit from
% the boxcar algorithm and returns the recorded vector and the
% times at wich recording took place

function [rec_vec,t,N] = slope(inp_vec,H)

 N=length(inp_vec);

 j=1;

% For the entire length of the input vector, each value is compared with the
% previously recorded value . If the difference is more than a prespecified
% limit, the previous input value processed is recorded, not the input value
% which triggered the recording.

% The first recorded value is initialized to the first input value.

 for i=1:N

        if(i>1)

                if(abs(inp_vec(i)-rec_vec(j-1))>=H)

                        rec_vec(j)=inp_vec(i-1); % previous input value recorded
                        t(j)=i;            % recording time
                        j=j+1;

                else

                end     % end of second if-loop

        else
                rec_vec(j)=inp_vec(i); % initialization for first rec. value
                t(j)=i;
                j=j+1;

        end     % end of first if-loop


 end    % end of for-loop
```

% backslope.m

```
% Written by Ganti S. Srinivas on February 25, 1996
% This is a code for data compression by the Backward Slope algorithm

% This algorithm takes input vector and the recording limit from
% the user and passes them to the function slope.m where the processing takes
% place. The recorded values and times are then returned to this algorithm
% The heart of this algorithm is the function slope.m. For more details please
% refer to it.

clear

fprintf('BACKWARD SLOPE ALGORITHM IS BEINGRUN\n');

%freak=1;

%while(freak==1)

 file_namea=input('Enter the input file you want  : ','s');
 [fida]=fopen(file_namea,'r');

 file_namec=input('Enter the name of the file to be generated ; This file contains the
smoothed values of the Sensors :','s');

 H=input('Enter the recording limit parameter ');
 fprintf('\n\n');


 X=fscanf(fida,' %f ',inf);

[y,t,N] = slope(X,H); % input vector and recording limit being passed and
                      % the recorded values and times being returned

 M=length(y);
 DCR=(1-(M/N));

 fprintf(['DATA COMPRESSION OBTAINED IS ',num2str(DCR)]);
 fprintf('\n\n');

 plot(X);
 hold on
 plot(t,y,'g');
 xlabel('Time index')
 ylabel('Normalized value')
```

```
fprintf('Click the mouse on the graph for the recording limit to be displayed');

gtext(['Recording limit is ',num2str(H)]);
fprintf('Hit any key to continue after the recording limit is displayed');
hold off

pause

title('Backslope version of the original signal');
%print -Pps407en


plot(t,y,'g');
xlabel('Time index')
ylabel('Normalized value')
title('Backslope version of the original signal');
fprintf('Click the mouse on the graph for the recording limit to be displayed');
gtext(['Recording limit is ',num2str(H)]);
fprintf('Hit any key to continue after the recording limit is displayed');
%print -Pps407en


fopen(file_namec,'w');
for i=1:M
fprintf(file_namec, '%d %f\n',t(i), y(i));
end
fclose('all');


%freak=input('Enter 1 if you want to try the whole loop again, else 0 : ');

%end
```

% slope.m

```
% Written by Ganti S. Srinivas on February 25, 1996
% This is a code for data compression by the Backward Slope algorithm

% This function takes input vector and the recording limit from
% the backward slope algorithm and returns the recorded vector and the
% times at wich recording took place

% In this case, the decision to record is based on a projection defined
% by the last recorded value and the one immediately prior to that
% The input value at this point in time is then predicted using the last
% recorded value and the slope ( or projection ). This predicted input value
% is then compared to the ACTUAL input value really obtained. If the
% difference is greater than or equal to the prespecified recording
% limit, the previous input value processed is recorded, not the input value
% which triggered the recording.

% In this case the first two recorded values are initialized to the first two
% input values.


function [rec_val,t,N] = slope(inp_vec,H)

% file_namea=input('Enter the input file you want  : ','s');
% [fida]=fopen(file_namea);
% inp_vec=fscanf(fida,' %f ',inf);
% H=0.01;

 N=length(inp_vec);

 j=1;

 for i=1:N

        if(i>2)
                % Calculationtion of slope or projection

                S(i)=((rec_val(j-1)-rec_val(j-2))/(t(j-1)-t(j-2)));

                % Checking the test condition

                k(i)=abs(inp_vec(i)-(rec_val(j-1)-S(i)*(i-t(j-1))));

                if(k(i) >= H)
                rec_val(j)=inp_vec(i-1); % Recording of previous input
                                         % value done
                t(j) = i;          % Recording of time
                j=j+1;

            else    % for second if-loop


            end     % end of second if-loop

        else    % for first if-loop

            rec_val(j)=inp_vec(i);          % Initialization of recorded values
            t(j)=i;
            j=j+1;

        end     % end of first if-loop


    end     % end of for-loop
```

comb.m

```
% Written by Ganti S. Srinivas on February 27, 1996
% This is a code for data compression by the combination of Backward Slope
% and Box Car algorithm

% The original authors are J.C. Hale and H.L. Sellars

% This method combines the abovementioned two methods by using an adaptive
% parameter P. P is initialized to 1 and remains at 1 as long as both the
% boxcar and backward slope test conditions are satisfied. If the backward
% slope test condition is not satisfied, but the Boxcar test condition is, then
% P is set to 2, and the method reverts to the Boxcar until a recording is made.% Once a
recording is made, the algorithm is reinitialized by setting P to 1.
% If the Boxcar test condition is not satisfied, but the Boxcar test condition
% is, then P is set to 3, and the method reverts to the Backward Slope until a
% recording is made. Once a recording is made, the algorithm is reinitialized
% by setting P to 1. If both the test conditions are not satisfied P retains
% the value 1 and recording is not done.

% This algorithm captures the advantages of both the techniques into a single
% algorithm which dynamically selects the technique to be applied to the next
% data point. This algorithm works better than either the Boxcar or Backward
% Slope methods, but requires more computation.


clear

fprintf('COMBINATION OF BOXCAR AND BACKWARD SLOPE ALGORITHMS
BEING RUN\n');

% freak=1;

% while(freak==1)

file_namea=input('Enter the input file you want  : ','s');
[fida]=fopen(file_namea);

file_namec=input('Enter the name of the file to be generated ; This file contains the
smoothed values of the Sensors :','s');

H=input('Enter the recording limit parameter');
```

```
X=fscanf(fida,' %f 'inf);
N=length(X);
P=1;
j=1;

for i=1:N

        if(i>2)          % first if-loop


                a(i)=abs(X(i)-y(j-1));
                S(i)=((y(j-1)-y(j-2))/(t(j-1)-t(j-2)));
                b(i)=abs(X(i)-(y(j-1)-S(i)*(i-t(j-1))));

                if(P==1)        % second if-loop

                        % if both boxcar and backslope tests are passed

                        if(a(i) >= H & b(i) >= H)       % third if-loop
        fprintf('Both Boxcar and Backward slope test conditions are satisfied, recording is
done \n');
                                y(j)=X(i-1);
                                t(j)=i;
                                j=j+1;
                                P=1;

                                elseif(a(i) < H & b(i) >= H )  % if boxcar
                                                               % test fails
        fprintf('Boxcar test failed, recording done using Backward slope algorithm \n');

                                P=3;

                                elseif(b(i) < H & a(i) >= H )  % if backslope
                                                               % test fails
        fprintf('Backward slope test failed, recording done using Boxcar algorithm \n');


                                P=2;

                                elseif(a(i) < H & b(i) < H) % if both boxcar and
                                                            % backslope tests fail
        fprintf('Both Boxcar and Backward slope test conditions are not satisfied,
recording is not done \n');

                                P=1;
        else

        end     % end of third if-loop

else

end             % end of second if-loop


        % if the backslope test fails, use the boxcar algo.

        if(P==2)        % fourth if-loop

                if(a(i)>=H & j~=j+1)

                        y(j)=X(i-1);
                        t(j)=i;
                        j=j+1;

                        else
                        P=1;

                end

        else
        end             % end of fourth if-loop


        % if the boxcar test fails, use the backslope algo.

        if(P==3)        % fifth if-loop

                if(b(i)>=H & j~=j+1)

                        y(j)=X(i-1);
           t(j)=i;
           j=j+1;

           else
           P=1;

        end
```

```
                        else
                end             % end of fifth if-loop


        else            % for first if-loop

                y(j)=X(i);
                t(j)=i;
                j=j+1;

        end             % end of first if-loop

end             % end of for-loop



M=length(y);
DCR=(1-(M/N));
fprintf(['DATA COMPRESSION OBTAINED IS ',num2str(DCR)]);
fprintf('\n\n');


plot(X);
hold on
plot(t,y,'g');
xlabel('Time index')
ylabel('Normalized value')

fprintf('Click the mouse on the graph for the recording limit to be displayed');

gtext(['Recording limit is ',num2str(H)]);
fprintf('Hit any key to continue after the recording limit is displayed');
hold off

pause

title('Performance of combination of boxcar and backward slope algorithm');
%print -Pps407en


plot(t,y,'g');
xlabel('Time index')
```

```
ylabel('Normalized value')
title('Performance of combination of boxcar and backward slope algorithm');
fprintf('Click the mouse on the graph for the recording limit to be displayed');
gtext(['Recording limit is ',num2str(H)]);
fprintf('Hit any key to continue after the recording limit is displayed');
%print -Pps407en


fopen(file_namec,'w');
for i=1:M
fprintf(file_namec, '%d %f\n',t(i), y(i));
end
fclose('all');


%freak=input('Enter 1 if you want to try the whole loop again, else 0 : ');

%end
```

% singexpfil.m

```
% This code is written for the single exponential filter
% Written by Ganti S. Srinivas on March 25, 1996


clear


 file_namea=input('Enter the input file you want  : ','s');
 [fida]=fopen(file_namea,'r');

 X=fscanf(fida,' %f ',inf);
 fclose(fida);


 N=length(X);
 alpha = input('Enter the value of alpha, 0<alpha<1');

 for i=1:N

        if(i>1)
        Y(i) = alpha*X(i) + (1-alpha)*Y(i-1);
        else
        Y(i)=X(i);
        end

        Z(i)=X(i)-Y(i);
 end

 plot(X,'y')
 hold on
 plot(Y,'r')
 hold off

  file_nameb=input('Enter the name of the output file ','s');
  [fidb]=fopen(file_nameb,'w');


 for i=1:N
 fprintf(fidb, '%f %f\n',X(i),Y(i));
 end

 fclose(fidb);
```

% dubexpfil.m

```
% This is the code for the double exponential filter
% Written by Ganti S. Srinivas on March 25, 1996


clear


 file_namea=input('Enter the input file you want  : ','s');
 [fida]=fopen(file_namea,'r');

 file_nameb=input('Enter the name of the output file ','s');


 X=fscanf(fida,' %f 'inf);
 N=length(X);
 alpha = input('Enter the value of alpha, 0<alpha<1');

 for i=1:N

        if(i>2)
        Y(i) = (alpha^2)*X(i) + 2*(1-alpha)*Y(i-1) - ((1-alpha)^2)*Y(i-2);
        else
        Y(i)=X(i);
        end

        Z(i)=X(i)-Y(i);
 end

 plot(X,'y');
 hold on
 plot(Y,'r');
 hold off
 title('Double Exponential filter');
 xlabel(['Alpha = ',num2str(alpha)]);
 print -Pps407en

 [fidb]=fopen(file_nameb,'w');

 for i=1:N
 fprintf(fidb, '%f\n',Y(i));
 end
 fclose('all');
```

% mov_ave_fil.m

```
% This code is written for the moving average filter
% Written by Ganti S. Srinivas on March 25, 1996

clear

 file_namea=input('Enter the input file you want : ','s');
 [fida]=fopen(file_namea,'r');
 X=fscanf(fida,' %f ',inf);
 fclose(fida);

 N=length(X);
 J=input('Enter the number of pastdatapoints to be averaged');

  for i=1:N

        if (i>=J)
        sum = 0;

        for j=(i-J+1):i
        sum = sum + X(j);
        end

        Y(i) = sum/J;
        Z(i)=X(i)-Y(i);

        else
        Y(i)=X(i);

        end
  end

 plot(X,'y')
 hold on
 plot(Y,'r')
 hold off

 file_nameb=input('Enter the name of the output file ','s');
 [fidb]= fopen(file_nameb,'w');
 for i=1:N
 fprintf(fidb, '%f\n',Y(i)):
 end
 fclose(fidb);
```

% rate_of_ch_fil.m

```
% This code is written for the rate of change filter
% Written by Ganti S. Srinivas on March 25, 1996


clear


file_namea=input('Enter the input file you want  : ','s');
[fida]=fopen(file_namea,'r');

X=fscanf(fida,' %f 'inf);
fclose(fida);

N=length(X);
delx = input('Enter the value of the limiting parameter');

for i=1:N

        if(i>1)

                if(abs(X(i)-Y(i-1)) <= delx)
                        Y(i)=X(i);
                %fprintf('abs. difference <= delx\n');

                elseif(Y(i-1)-X(i) > delx)
                        Y(i)=Y(i-1)-delx;
                %fprintf('Y(i-1) is > X(i) by delx\n');

                elseif(Y(i-1)-X(i) < (-delx))
                        Y(i)=Y(i-1)+delx;
                %fprintf('Y(i-1) is < X(i) by -delx\n');

                else

                end

        else

        Y(i)=X(i);

        end
        Z(i)=X(i)-Y(i);
 end

plot(X,'y')
hold on
plot(Y,'r')
hold off

 file_nameb=input('Enter the name of the output file ','s');
 [fidb]=fopen(file_nameb,'w');


for i=1:N
fprintf(fidb, '%f\n',Y(i));
end

fclose(fidb);
```

% auto_level.m

```
% Written by Ganti S. Srinivas on June 5, 1996

%**********************************************************************
% This is a code for automated trend extraction. This code takes a raw sensor %
% signal as input, processes this signal and outputs the smoothed signal      %
% directly. The user is then prompted if he's satisfied with the smoothing
%%*********************************************************************

clear
format long

file_namea=input('Enter the input file you want  : ','s');
[fida]=fopen(file_namea,'r');          % Opens the above file for reading

X=fscanf(fida,' %f 'inf);              % Reading the above file to a vector
fclose(fida);                          % Closes the file after reading
clf;
plot(X,'c');
title('Original sensor signal');
fprintf('\nHit any key to continue\n');
pause

N = length(X);

flag = 1;
while (flag == 1)
%**********************************************************************

% This function displays the window of choice for performing the FFT

               [winx,index] = display(X,N);

file_nameb = input('Enter the file name forwinx : ','s');
[fidb] = fopen(file_nameb,'w');
for i=1:length(winx)
fprintf(fidb,'%f\n',winx());
end
fclose(fidb);
%**********************************************************************
 figure(1)
 set(gcf,'Nextplot','add');
 clf;
 plot(index,winx,'c');
```

```
title('Selected window from original signal for determining the CPS');
grid
% print -Pps407en
%*****************************************************************

% This function performs the FFT calculates the cumulative power spectrum and
% the cumulative power spectrum (CPS).

                [P,N1,x] = fast(winx);

%*****************************************************************
figure(2)
set(gcf,'Nextplot','add');
clf;
plot(P,'g');
grid
% hold on
% plot(x,'r');
title('Cumulative power spectrum (CPS)');
% hold off

%  print -Pps407en

file_namec = input('Enter the file name for CPS : ','s');
[fidc] = fopen(file_namec,'w');
for i=1:length(P)
fprintf(fidc,'%f\n',P(i));
end
fclose(fidc);

%*****************************************************************

% This function determines the slope and the rate of change of slope for all
% the data points in the CPS.

                [freq,no_levels1,delP] = detect(P,N1,N);
                        if (freq == [])
                                error('Null cut-off frequency index');
                                break;
                        end

%*****************************************************************
[val,ind] = sort(delP);
figure(3)
set(gcf,'Nextplot','add');
```

```
clf;
plot(delP,'r');
grid;
title('Rate of change of slope');

%*********************************************************************

% This is a function to determine the minimum window size required for
% "effective" pattern based data analysis. Then it prompts the user to input
% a window size greater than the minimum. Based on this user-input window size
% the no. of levels of decomposition are determined. The no. of levels thus
% obtained are compared to that obtained from detect.m

              [no_levels2, min_win] = window(N, N1, freq);

%*********************************************************************

% This part of the code compares the no. of levels obtained from detect.m and
% window.m and increments them by 1

        [no_levels,levels1,levels2] = decide(no_levels1,no_levels2);

%*********************************************************************

% This part of the code uses the no. of levels obtained from the above
% calculations to smooth the raw signal.


              [f,f1,no_levels] = power(X,no_levels, min_win);


%*********************************************************************

 file_named = input('Enter the name of the file for f and f1  : ','s');
 [fidd]=fopen(file_named,'w');          % Opens the above file for reading

 for i=1:length(f1)
 fprintf(fidc,'%f %f\n', f(i), f1(i));
 end

 fclose(fidd);

flag = input('Enter 1 if you want to continue with this file, else enter 0');
end
```

% display.m

% This code was written by Ganti Srinivas, dated July 10, 1996

```
%***********************************************************************
% This part of the code runs the FFT and displays the cumulative power spectrum.
% For running the FFT a data window size of 256 is used. The window is selected
% over a steady part of the signal. This window is chosen at the discretion of
% the user.
%***********************************************************************


function [winx,x] = display(X,N)

W = input('Enter the no. of data points (should be a power of 2) for the FFT');
flag1 = 1;

while (flag1 == 1)
        j=0;
        flag1
        for i=1:(N/W)
                i
                j
                 x = (W*j+1):(W*i);
                 plot(x,X((W*j+1):(W*i)),'c');

                 fprintf('\nEnter 1 if this plot is steady and is satisfactory for FFT
computation, else enter 0\n');

                 flag = input('');

                 while (flag ~=0 & flag ~=1)

                         fprintf('\nEnter 1 if this plot is steady and is satisfactory for FFT
computation, else enter 0\n');

                         flag = input('')

                 end

                         if (flag == 1)
                                 winx = X((W*j+1):(W*i));
                                 flag1 = 0;
                                 break;
                         else
```

```
                        flag1 = 1;
                    end

            j=j+1;
            fprintf('Hit any key to view the next data window');
            pause;


    end

end
```

% fast.m

```
% This code was written by Ganti srinivas on June 5, 1996

%**********************************************************************
% This code computes the fast Fourier transform and the cumulative power     %
% spectrum. Then it calls another function regress.m which regresses data in  %
% the power spectrum. This regressed data is then used to determine the       %
% cut-off frequency.                                                          %
%**********************************************************************


        function[P,N1,x] = fast(winx)



        N1 = length(winx);
        P=spectrum(winx,N1)
%       Y=(1/N1)*fft(winx);
%       P=Y.*conj(Y)

        for i=1:(N1/2)-1
        P(i+1)=P(i)+P(i+1);    % Cumulative power spectrum

% We consider only the first half of P because the other half is symmetric
% and a mirror image of the first half

        end
        P=P(1:N1/2)

%**********************************************************************
% This part of the code runs the regression on the data from the FFT window   %


%                 [x] = regress(P,N1);


%**********************************************************************
```

detect.m

```
% This code was written by Ganti Srinivas, dated June 5, 1996

%*******************************************************************
% This part of the code decides automatically where the cut-off frequency is.
% All the frequencies below this cut-off frequency index are considered to be
% frequencies inherent in the signal. The frequencies above the cut-off freq
% -uency are considered to be noise. By a pre determined relation, the cut-ff
% frequency index is used directly calculate the optimum number of levels for
% smoothing.
%*******************************************************************


function [j,no_levels,delP] = detect(P,N,n)

                        % N is the FFT data window size
                        % n is the length of the original signal
                        % The CPS length is N/2

fprintf('The CPS length is %f\n', N);
fprintf('The signal length is %f\n', n);

        %***************FFT and CPS determination****************%

% The CPS of length N is symmetrical about N/2. So, only the first half of the
% CPS is considered to determine the cut-off frequency index

        for i=2:(N/2)-1

                delP(i)=P(i+1)-2*P(i)+P(i-1); % Rate of change of slope

        end % end of for loop

        N1 = length(delP) % This is the length of the second derivative vector

% For signals of any length and any size of the FFT data window, consider only
% the first half of the CPS for detecting the bending point. The other half of
% the CPS is ignored because it is assumed that the CPS bends much before half
% it's length i.e., before N/4. Moreover, a bending point beyond half the
% length would yield only two levels of smoothing which is almost equivalent to
% the original signal. Thus, we always obtain more than two levels of smoothing.

                delP = delP(1:((N/4)-1))
```

```
delP
[y1,i1] = sort(delP);
i1

%*******************************************************************

% If the index of the minimum most value of delP is just one index before the
% last value of delP, select the values of delP between 1 and (i1(1)-1) to find
% the bending point of the CPS.


            if (((i1(1)+1) == ((N/4)-1)) | (i1(1) == ((N/4)-1)))

        fprintf('\nThe x index of the most minimum value occurs just before half the
length of the CPS\n');
        fprintf('\nThis means that the CPS does not bend before half itslength; it goes
on increasing steadily\n');

            delP1(1:(i1(1)-1)) = delP(1:(i1(1)-1));

            else
% Else, consider the values of delP between i1(1) and N/2. Note that the CPS is
% expected to bend much before half its length. Also,it has been observed that
% it bends much after the most minimum value of delP. So, the points in between
% the x index of the minimum  most value of delP and N/2 are considered to
% determine the bending point of the CPS more accuartely.

                delP1((i1(1)+1):((N/4)-1)) = delP((i1(1)+1):((N/4)-1));

            end

        end     % End of outer if loop

%*******************************************************************

[y2,i2] = sort(delP1);
i2


% This part of the code determines how many indices after the index of the
% minimum most value the code needs to skip, to find the bending point of the
% CPS accuaretly.
```

```
        if (N==32)      % N is the length of the FFT data window
                buff = 0;
        elseif (N==64)
                buff = 2;
        elseif (N==128)
                buff = 3;
        elseif (N==256)
                buff = 4;
        elseif (N==512)
           buff = 5;
        else
                error('The length of the FFT data window should be >= 32 and <= 512');
                break;
        end


                %********************************%


% If the x index of the minimum most value of i2 > the x index of the
% minimum most value of delP, then i2(1) is chosen as the bending point of the
% CPS.

        if (i2(1) > (i1(1)+buff))
                j = i2(1)


% Else, for any length of the original signal and any length of the FFT data
% window, if the x index of the most minimum value of delP1 occurs close to the
% N/2th index, the x index of the second most minimum value of delP1 is taken
% as the bending point of the CPS. This is a very special case.

        elseif (i2(1) == max(i2))
                j = i2(2);

% Else, for any length of the original signal and any length of the FFT data
% window, if the maximum value of the "x index of the most minimum value of
% delP1" is less than the x index of the most minimum value of delP, i.e.,i1(1)
% the x index of the second most minimum value of delP1 is taken
% as the bending point of the CPS. This is again a possibility if the x index
% of the most minimum value of the second derivative lies very close to N/2.
% This is a very special case.

        elseif (max(i2) < i1(1))
                j = i2(1);
```

```
% Else, the x index of the  minimum value of the second derivative "buff"
% indices after the most minimum value is determined as the bending point of
% the CPS.


        else
                index = 1;

                while (i2(index) <= (i1(1)+buff))

                        j = i2(index+1)
                        index = index +1;

                end     % End of while loop


        end     % End of if loop


j

                no_levels = (log2(N)-log2(j));
```

% window.m

```
% Code written by Ganti Srinivas; July 9, 1996

% This is a function to determine the number of levels to go down while smooth-
% -ing, based on the user provided information on the window size. Here, it is
% taken that 4 cycles of the cut-off frequency obtained should be considered
% in obtaining the minimum window size for "effective" pattern based data
% analysis . The number 4 is empirical and was suggested by Dr. Whiteley, based
% on the experience of observing the ARDS data.

% This code assumes that the sampling interval of the raw data,"T" is 1 min.
% If the sampling interval is not 1min. then:
%       winx = (length of window in min. )/(sampling interval in min.);


        function [no_levels2, min_win1] = window(N, N1, freq)


% To determine the minimum window size, the maximum allowable i.e., the cut-off
% frequency is used. This frequency is used to compute the cut-off ANALOG
% frequency in cycles per minute. The relation between the digital frequency
% index "k", digital frequency "w", and analog frequency "W" is given by the
% following relation

%       w = k(2*pi)/T radians;            W = k(2*pi)/(N1*T) radians per min.
%                                         W = (k/N1*T) cycles/min

% where N1 is the total number of data points under consideration in FFT
% 2*pi radians is 1 cycle. One cycle occurs every (N1*T/k) minutes. 4 cycles
% occur in 4*(N1*T/k) minutes.

%***************************** Code begins
***********************************%

        T=input('Enter the sampling interval of data in minutes');
        min_win = ceil(4*(N1*T/freq)); % This is the size in minutes

        fprintf('\n\nThe minimum window size is %f minutes\n',min_win);

        fprintf('\n Please keep in mind the sampling frequency when the minimum
window size is displayed\n');

        min_win1 = round (min_win/T); % This is the size in data points.
                                      % This needs to be converted because the
% size of this window is used for plotting the smoothed and original signal
% values

 fprintf('\nThe window size should be greater than the minimum window size\n');

        %****************Error checking section*************%


        flag = 1;

        while (flag == 1)

        winx = input('Enter the length of the window in minutes');

         if (winx < min_win )
           fprintf('Selected window size is smaller than the minimum required');
           flag = 1;
         elseif (winx == [])
           fprintf('Null window size selected; select again');
           flag = 1;
         else
           fprintf('\nSelected window size is greater than minimum required\n');
           flag = 2;
         end    % end of if loop

        end     % end of while loop

                      %*******************************%

% This part of the code checks to see if the user input window is greater than
% 1/4 of the original signal. We do not observe more than 1/4 length of the
% original signal

         while ((winx/T) > N/2)

                fprintf('\nSelected window is greater than one half of original signal\n');
                fprintf('\nSELECT SMALLER WINDOW\n');
                winx = input('Enter the length of the window in minutes');

         end


                      %*******************************%

% This part of the code makes use of the user input window size to determine
% empirically the total number of levels to go down. It is seen how many
% windows of user-input size fit into the original signal of length N. The
% number obtained is then multiplied by a factor 4. This number was suggested
% by Dr. Whiteley and is based on experience.

% There need to be 4 points over a span of winx, after going down certain
% number of levels "n", without reconstruction. Corresponding to a signal of
% length N, the number of points  after going down the same number of levels,
% without reconstruction is x=(N/length(winx))*4. Then number of levels gone
% down is given by    (log2(N)-log2(x))


% Length of user input window is given by   winx    as the window size
% given by the user is already in minutes

        x = (N/(winx/T))*4;

        no_levels2 = (log2(N)-log2(x));

                        %********THE END*********%
```

```
                          % decide.m

% This part of the code compares the no. of levels obtained from detect.m and
% window.m and increments them as shown below

function [no_levels,no_levels1,no_levels2] = decide(no_levels1,no_levels2)


%       if ((no_levels1-floor(no_levels1) <= 0.2))
%               no_levels1 = floor(no_levels1);

% This value is floored because it is closer to this power of 2

%       else
                no_levels1 = ceil(no_levels1);

% This value is ceiled because it is closer to this power of 2

%       end

%       if ((no_levels2-floor(no_levels2) <= 0.2))
%               no_levels2 = floor(no_levels2);
% This value is floored because it is closer to this power of 2

%       else
                no_levels2 = ceil(no_levels2);
% This value is ceiled because it is closer to this power of 2

%       end

% NOTE: The 1 added below accounts for padding the signal on either side by
% half its length.

 if (no_levels1 == no_levels2)
      no_levels = (no_levels1) ;

 elseif (no_levels1 > no_levels2)
      no_levels = (no_levels1) ;

 elseif (no_levels1 < no_levels2)
      no_levels = (no_levels2) ;
 else
        error('Error in comparing no_levels1 and no_levels2');
      break;
 end
```

```
%       no_levels3 = no_levels3 +1; % The 1 added accounts for an extra length
                                    % of the original signal formed by padding

% if (no_levels <= no_levels3)
%       no_levels = no_levels;

% else
%       fprintf('\nThe user input window length chosen is small\n');
%       fprintf('\n                     OR\n');
%fprintf('\nThe cut-off frequency obtained from the CPS may not be accurate\n');%
pause;
%       no_levels = no_levels3;

% end
```

% power.m

```
% This function uses the function crunch to show the user the level of smooth-
% -ing. It also determines the final value of the number of levels to go down
% that the user finds satisfactory.

        function [f,f1,no_levels] = power(X,no_levels,min_win)

 min_win = round(min_win);

%*********************************************************************

% This part of the code smooths the raw sensor signal using wavelets

              [f,f1] = crunch(X,no_levels,min_win);

%*********************************************************************

 fprintf('\n\nIf the plot is not smooth enough, please enter 1\n\n');
 fprintf('If the plot is too smooth, please enter 2\n\n');
 fprintf('If the plot is fine, please enter 0\n\n');
 flag = input('');

      while(flag==1|flag==2|flag==0)
          if(flag==1)
                no_levels = no_levels+1;
               [f,f1] = crunch(X,no_levels,min_win);

          elseif(flag==2)
                no_levels = no_levels-1;
               [f,f1] = crunch(X,no_levels,min_win);

          elseif(flag==0)
                break;
          else
          end

          fprintf('If the plot is not smooth enough, please enter 1\n\n');
          fprintf('If the plot is too smooth, please enter 2\n\n');
          fprintf('If the plot is fine, please enter 0\n\n');
          flag = input('');
         end
```

% **crunch.m**

```
% This function was modified from fun.m by Ganti Srinivas on June 5, 1996

        function[f,f1] = crunch(f,J,min_win)


[h,g]=daub(6);
N=length(f);            %Check for the signal length
check=log(N)/log(2);   %Check to see if it is a Power of 2.
if(floor(check)~=check),
error('LENGTH OF THE SIGNAL IS NOT A POWER OF 2');
end

f=net(f);          % This function extends the length of the vector f by half it's
                   % length on either side.

%*********************************************************************

[c,count,proxy]=fwt(h,g,f,J,3);          %Calculate the Detail and Blurred Coffs..

        proxy=zeros(length(proxy),1);

 [f1]=ifwt(h,g,c,count,proxy); %Reconstruct the Signal from the Detail and
                               %Blurred Coffs from FWT.M

 %Calculate the error of Reconstruction

        f1=f1(N/2+1:1.5*N);
        f=f(N/2+1:1.5*N);
%       f1=f1(N/4+1:3*N/4);
%       f=f(N/4+1:3*N/4);
%       err=f(:)-f1(:);err=sum(err.^2)/N;err=sqrt(err)

 %Plot the original and Reconstructed Signal for Comparison

      figure(4);
      set(gcf,'Nextplot','add');
         clf;
      x = 1:N;
      subplot(2,1,1);
      plot(x,f(:),'y');
      grid;
      v=axis;
```

```
    hold on
    xlabel('Original signal versus Smoothed signal');
    title(['No. of levels down is ',int2str(J)]);
    plot(x,f1(:),'r');
%     print -Pps407en

    subplot(2,1,2);
    plot(x,f1(:),'g');
    grid;
    axis([v(1) v(2) v(3) v(4)]);
    xlabel('Smoothed signal');
%     print -Pps407en
    hold off

    figure(5);
    set(gcf,'Nextplot','add');
       clf;
    y=1:min_win;
    plot(y,f((N/2):((N/2)+min_win-1)),'w');
    hold on
    plot(y,f1((N/2):((N/2)+min_win-1)),'m');
    hold off
    xlabel('Smoothed signal overlapped on the original');
    title('Pattern data analysis on window of minimum size');
    hold off
```

% daub.m

```
        function [h,g]=daub(n)
%This code generates the filter coefficients for the Daubechies
%Family of Orthonormal group of Wavelets.
%The input to this subroutine is the order of the Wavelet and the
%output is the filter coefficients for both the Low pass filter and
%the Band pass filter.
%
%Ref:1. Orthonormal Bases of Compactly Supported Wavelets by
%     Ingrid Daubechies in Communications on Pure and Applied
%     Mathematics ,(41) 1988  pp 909-996
%   2. Introduction to Wavelets  by Charles K. Chui, pp 229-234
%
%      The references made here are from the second article.
%
%      VINOD K. RAGHAVAN
%      SCHOOL OF CHEMICAL ENGINEERING
%      OKLAHOMA STATE UNIVERSITY
%      STILLWATER, OK 74078
%
%      FUNCTION [H,G]=DAUB(N)
%      Program Coded  19 June 1993
%      Modified      20 June 1993

if nargin<1,error('Please define the order of theWavelet.');end

if n>1,

zcoff=[-1/4,1/2,-1/4];            %Generate the polynomial
z=zcoff;
pk=1;
q=1;
for j=1:n-1,
pk=pk*(n+j-1)/j;
q=[0,q,0]+z*pk;
z=conv(zcoff,z);
end

 q=q*2;                          %P_A(z)=(1/2)*summation(a_|z|*z^(N=k) )

r=roots(q);                      %find the roots of the Polynomial

r=r(find((abs(r)<1)));           %Pick the roots within the Unit Disk
[t1,t2]=size(r);
```

```
s=[1 -r(1)]/sqrt(r(1));           %Find S(z) using Theorem 7.17 p232-233
for i=2:t1,
s=(conv(s,[1 -r(i)]))/sqrt(r(i));
end
s=s/sqrt(2);

f=[1 1 ]/2;                       %P(z)=[((1+z)/2)^N]*S(z)
for i=1:n,
s=conv(s,f);
end

p=real(s);                        %Only the real parts are considered
h=sqrt(2)*p/sum(p);               %Calculate the low pass coefficients
else,
h=[1 1]/sqrt(2);
end

g=qmf(h);                         %Calculate the Band pass coefficients
clc;
```

% qmf.m

```
%this subroutine calculates the  Band filter coefficients.
%the input is the Low pass filter coffs.The subroutine returns the
%Band pass filters that are the Quadrature Mirror Filters of the low pass
%filters.The filter length is the same as that of the low pass filter.
%
%Working Version made on March 10 1993
%
% Vinod K. Raghavan
% School of Chemical Engineering
% Oklahoma State University.
% Stillwater,OK 74078

function g=qmf(h)
echo
i=1:length(h);
g(i)=((-1).^(i-1)).*h(length(h)-i+1);
```

% net.m

```
function  y= net(x)

% this signal extension method is based on Dr.Whiteley's suggestions
% on 10/5/94
% modified 16/5/94
% modified again 12./21./94
% Vinod Raghavan  5/10/94

x=x(:);x=x';
N1=length(x);
% this part has been added after discussion with Dr.Whiteley
% on 5/16/94
%n1=input('Enter the number of points to calculate the mean:');
% this part was added on June 2, 1994
% diff means are being used for LHS and RHS

        K=ceil(1.25*N1/100);
                n1=net2a(x,K);
                n2=net2b(x,K);
%extend the signal

%y=[2*mean(x(1:n2))-x(N1/2+n2:-1:n2+1),x,2*mean(x(N1:-1:N1-n1+1))-x(N1-n1:-
1:N1/2-n1+1)];
% the modification below was made on 27th June 1994
mean1=mean(x(1:n2));mean2=mean(x(N1:-1:N1-n1+1));
y=[2*mean1-x(N1/2:-1:1),x,2*mean2-x(N1:-1:N1/2+1)];

% end of modification

i1=find(y>1);i2=find(y<0);
% replace all the values >1 to 1
if length(i1)>0,
        for i=1:length(i1),
        y(i1(i))=1;
        end
end
% replace all the values <0 to 0
if length(i2)>0
        for i=1:length(i2),
        y(i2(i))=0;
        end
end
```

% net2a.m

```
function n = net2a(x,K)

% programmed  on 2 June 1994

i=[ceil(0.4*K):K];
y=x(length(x):-1:length(x)-K+1);
mean_tmp=[];
for j=1:length(i),
%calc the mean deviation
%mean_d=sum(abs(y-mean(y)))/length(y);
mean_d=std(y(1:i(j)));
mean_tmp=[mean_tmp mean_d];
end
n=i(find(mean_tmp==min(mean_tmp)));
```

% net2b.m

```
function n = net2b(x,K)

% programmed  on 2 June 1994

i=[ceil(0.4*K):K];

mean_tmp=[];
for j=1:length(i),
y=x(1:i(j));
%calc the mean deviation
%mean_d=sum(abs(y-mean(y)))/length(y);
mean_d=std(y);
mean_tmp=[mean_tmp mean_d];
end
n=i(find(mean_tmp==min(mean_tmp)));
```

% fwt.m

```
%this subroutine is based on Stephane Mallat's article in IEEE
% Transactions on Pattern Recognition and Machine Intelligence
%This subroutine calculates the Blurred and Detail Coffs for the Signal
% at each level.
%
%       FUNCTION  [F,B,COUNT]=FWT(H,G,F,J)
%
%       Working Version created on May 14,1993
%       Last modified on        Dec 13,1993
%
%       VINOD K. RAGHAVAN
%       School of Chemical Engineering
%       Oklahoma State University
%       Stillwater, OK 74078
%
%Ref:

        function [f,count,proxy]=detblur(h,g,f,J,sign_menu)
clc;
N=length(f);N11=N/2; %check for length  of signal

N1=max(length(h),length(g));        %check for length of  filter

h=h(N1:-1:1);g=g(N1:-1:1);  %Flip the filter coffs to get their Conjugates

 b=zeros(1,N11);           %Initialize the Matrix b to contain detail coffs
 Ax1=[];Ax2=[];            %Initialize the Matrix to contain the coords

        count=0;
        for j=1:J,        %Start the decomposition...
        count=count+1;
%the signal has to be symmetric about n=0 and n=N...
        f1=f(N:-1:1);
        f1(N+N1)=0;
        for i=1:N1,
        f1(i+N)=f1(i);
        end
        f1=f1(length(f1):-1:length(f1)-N1+1);
        f(N+N1)=0;
        for i=1:N1,
        f(i+N)=f(i);
        end
        f=[f1 f];
```

```
            d=filter(g,1,f)/sqrt(2);  %Calculate the Detail Coffs
            f=filter(h,1,f)/sqrt(2);  %Calculate the Blurred Coffs

%take only one out of every two samples(down sampling)....
d=d(N1*2:2:N+2*N1-2);
f=f(N1*2:2:N+2*N1-2);


%save the vector
proxy=[d(:);proxy];
%Continue the grind if the user wants to go down further.
N=N/2;
end
             %Next Iteration..Halve the signal length
```

% ifwt.m

```
%this subroutine is based on Stephane Mallat's article in IEEE
% Transactions on Pattern Recognition and Machine Intelligence
%This subroutine takes in the Values from the FWT and Reconstructs the
% Signal at each Level.
%       FUNCTION [F]  =IFWT(H,G,F,B,count)
%
%Working Version created on May 14,1993
%Last modified on 7 July 29  1993
%
%       VINOD  K. RAGHAVAN
%       School of Chemical Engineering
%       Oklahoma State University
%       Stillwater, OK 74078

        function[f]=ifwt(h,g,f,count,proxy)

 N=length(f);N11=N; %check for length  of signal
 f=f(:);
 N1=max(length(h),length(g));     %check for length of  filter

        for j=1:count,         %Start the  Reconstruction..

 d=proxy(1:length(f))';
 proxy=proxy(length(f)+1:length(proxy));
 %
 % upsample the signal by inserting a zero inbetweem every sample..
        d=[d;zeros(1,length(d))];d=d(:);
        f=[f';zeros(1,length(f))]f=f(:);
 %
 % Extend the length of the samples....
        %periodize the signal........
        f1=f(2*N:-1:1);d1=d(2*N:-1:1);
      f1(2*N+N1)=0;d1(2*N+N1)=0;
      for i=1:N1,
      f1(i+2*N)=f1(i);
         d1(i+2*N)=d1(i);
      end
      f1=f1(length(f1):-1:length(f1)-N1+1);
         d1=d1(length(d1):-1:length(d1)-N1+1);
         f(2*N+N1)=0;d(2*N+N1)=0;
      for i=1:N1,
      f(i+2*N)=f(i);
         d(i+2*N)=d(i);
```

```
    end
    f=[f1(:);f(:)]*sqrt(2);
       d=[ d1(:);d(:)]*sqrt(2);

%convolve with the filters to obtain the original signal again..

        fb= filter(h,1,f);
        fb=fb(N1+1:N1+2*N);
        fd=filter(g,1,d);
        fd=fd(N1+1:N1+2*N);
        f=fd+fb;            %Keep the Coffs in our region of interest
N=N*2;                              %Next Iteration .. Double the signal length
end
```

VITA

Srinivas S. Ganti

Candidate for the Degree of

Master of Science

Thesis: AUTOMATED TREND EXTRACTION OF SENSOR SIGNALS FOR PATTERN BASED DATA ANALYSIS

Major Field: Chemical Engineering

Biographical:

Personal Data: Born in Visakhapatnam, Andhra Pradesh, India, May 02, 1971, the son of Kalyani and Murty Ganti.

Education: Graduated from Mrs. A.V.N. College, Visakhapatnam, India, in June 1988; received Bachelor of Engineering Degree in Chemical Engineering from Karnataka Regional Engineering College, Surathkal, Karnataka, India, in June 1992; completed requirements for the Master of Science Degree in Chemical Engineering, at Oklahoma State University in December 1996.

Experience: Project Engineer, Indian Organic Chemicals Ltd., Bombay, India, July 1992 to July 1994; Teaching Assistant, School of Chemical Engineering, Oklahoma State University, August 1994 to December 1994; Research Assistant, School of Chemical Engineering, Oklahoma State University, January 1995 to August 1996.