CLASSIFICATION OF TRANSIENTS IN POWER SYSTEMS

By

Jin Chen

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

Thesis Advisor: W. Kinsner, Ph. D., P. Eng.

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

CLASSIFICATION OF TRANSIENTS IN POWER SYSTEMS

BY

JIN CHEN

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

of

MASTER OF SCIENCE

ABSTRACT

This thesis presents a method of feature extraction and classification of power system transients. The system developed in this thesis uses both the wavelet transform and multifractal analysis as tools to analyze power system transients for feature extraction, as well as probabilistic neural network that is used as a classifier for identifying various types of power transients associated with power system faults and switching.

This system has three major modules: (i) transients simulation, (ii) feature extraction of the transients, and (iii) features classification. A wavelet transform is used to generate multiresolution time-frequency features and multifractal analysis is conducted to generate a variance fractal dimension trajectory directly used as features of the power transient. The combination of these features characterizes the power transient in good and compact representations for neural network analysis. Finally, the modelled transient is classified using the multi-sigma PNN classifier that uses a separate sigma weight for each input variable.

Performance of the proposed classification system is evaluated based on simulated transients. The experimental results show that this identification system is precise and fast in discriminating the type of a transient event. More specifically, the system was trained with 250 out of 750 simulated pure transients belonging to five classes of disturbances. Testing the system with the remaining 500 transients yielded a correct classification rate around 99%. The average processing time for completing feature extraction and classification was about 2 second per transient with Pentium (R) III of 600 MHz. Experiments also show that the classification system is robust in a noisy environment.

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Dr. W. Kinsner for giving me his time and patience throughout the process of this thesis. Dr. Kinsner not only guided me doing this thesis, but also taught me the methodology of doing research, writing science reports, and presenting, which will benefit for my further career.

I would like to thank everyone in the Delta Research Group, past and present, particularly Luotao Sun, Hong Zhang, Kalen Brunham, Robert Barry, Michael Potter, Hakim El-Boustani, Martin Stadler, Sharjeel Siddiqui, Alexis Denis, Richard Dansereau, etc. for their help, support, friendship, and useful discussions with me for this thesis.

I would like to acknowledge the partial financial support provided by the Manitoba Hydro to fulfil the project.

I would also like to thank my parents and sister for taking care of my son when I cannot bring him with me. Finally, I dedicate the work to my son, Yujing Huang and my husband, Bin Huang.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

List of Abbreviations

BM	bin mean
BPN	backpropagation network
BRMS	bin root mean square
CWT	continuous wavelet transform
Daub4	Daubechies wavelet with four filter coefficients
DWT	discreet wavelet transform
FT	Fourier transform
MAP	maximum posterior probability
MSD	multiresolution signal decomposition
PDF	probability density function
PNN	probabilistic neural network
RMS	root mean square
SNR	signal to noise ratio
STFT	short-time Fourier transform
VFD	variance fractal dimension
VFDT	variance fractal dimension trajectory
WT	wavelet transform
WTMM	wavelet transform modulus maxima

LIST OF SYMBOLS

$ci[n]$	the smoothed signal at scale i
$di[n]$	the detailed signal at scale i
$\mathbf{d}_i$	conjugate gradient $\mathbf{d}_i$ of the error function at ith iteration
$D(\mathbf{x}, \mathbf{x}_r)^2$	scaled squared Euclidean distance between an unknown sample, $\mathbf{x}$, and a training case, $\mathbf{x}_r$
D_σ	variance dimension
γ	a factor for renewal of the conjugate gradient $\mathbf{d}_i$
$e_k(\mathbf{x})$	continuous error (assume that $\mathbf{x}$ belongs to class k)
E	embedding Euclidean dimension
Δt	time increment
$f_k(\mathbf{x})$	probability density function (PDF) for a class k
f_s	the sampling frequency
$\delta_k(\mathbf{x_r})$	a delta function that indicates whether or not a given case $\mathbf{x_r}$ in the training set belongs to a particular class k
$\mathbf{g}_i$	negative gradient $\mathbf{g}_i$ of the error function at ith iteration
$g_k(x)$	estimated PDF for a class k
$g[n]$	high-pass filter for wavelet transform
h_k	prior probability of a class k
$h[n]$	low-pass filter for wavelet transform
H	Hurst exponent
l_k	loss function associated with misclassifying a sample of the class k
m	embedding dimension
n_k	number of training cases for a given class k
N	total number of data points
N_i	the number of data points of the coefficients in ith bin
NS	the rectangular sliding window displacement for the local variance fractal dimension (VFD) calculation
N_w	the rectangular sliding window size for the local VFD calculation
p	number of features
P_c	classification rate of the PNN
λ	a length along the conjugate direction for an optimal renewal of the σ_i

$q_k(\mathbf{x})$	Bayesian confidence (assume that **x** belongs to class k)
s	slope of a log-log plot
$s(\mathbf{x})$	sum of the activation function
T_e	error threshold for the PNN training
T_s	sigma threshold for the PNN training
$u_k(\mathbf{x})$	activation function (assume that **x** belongs to class k)
$\|v_i\|$	RMS value of the ith bin in a detailed signal
$W(x)$	weight function (kernel)
$WT(a, b)$	continuous wavelet transform
$\mathbf{x}$	neural network input vector
$\mathbf{x_r}$	the rth sample from the class k in the training set
$x[n]$	discrete-time signal
$x(t)$	continuous temporal signal
$x(t_j)$	sampled temporal signal
X_k	log values for a log-log plot
$\mathbf{y}$	neural network output vector
Y_k	log values for a log-log plot
σ	scaling parameter for the single-sigma PNN
$\sigma_1, \ldots, \sigma_p$	scaling parameters for the multi-sigma PNN
σ^2	variance of a sampled temporal signal's amplitude
$\Phi(t)$	scaling function
$\Psi(t)$	mother wavelet or primary wavelet function

CHAPTER I
INTRODUCTION

1.1 Problem Definition and Motivation

Electromagnetic transients in power systems result from various disturbances on transmission lines, such as capacitor switching, lightning strike, different types of faults, overloads, and other events. Recently, concern over such transients has been increasing since poor electric power quality causes many problems for affected loads such as misoperation, short life time, instabilities, and even break down. In order to improve power quality, the sources and causes of such disturbances must be known before correct and quick actions can be taken. Therefore, electromagnetic transient analysis in power systems plays an important role in the design and operation of the power systems.

Digital transient recorders are able to record various transients and provide some rudimentary assistance information about them. However, it is difficult to analyze transients on-site due to a vast amount of data recorded by transient recorders. Moreover, different types of transients need different responses and different types of faults require taking different measures. So transient classification and identification are very important. Current transient recorders lack classification capability. Therefore, it is imperative to develop intelligent recorders which have the ability to classify different types of transients, especially distinguish faults from other switching events because switching operations may cause transients seemingly like fault induced transients. In order to develop a powerful approach for transient data capture and analysis, one of the important steps is to model

such transients, so that it is more efficient for computer to extract features of power transients and recognize them automatically [MoKi97]. Some studies have been conducted on classification and identification of transients in power systems [MaAB99], [MoKi98].

This thesis focuses mainly on **transients generation, feature extraction of transients analysis using wavelet and multifractal approach**, and **neural network classification**. Figure 1.1. shows a block diagram of the proposed classification system scheme. A short description of each function of the blocks is described next.

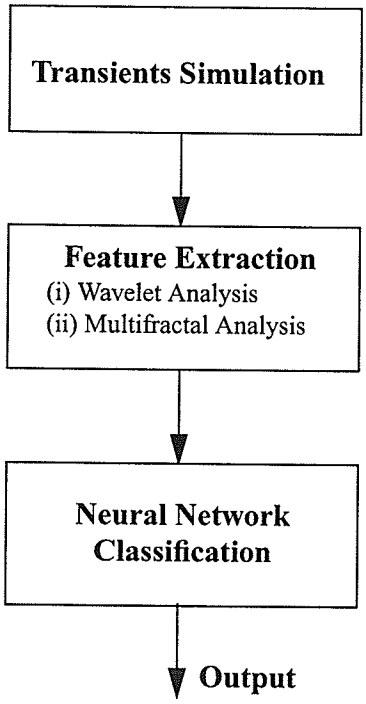

Fig. 1.1. The structure of a power transient classification system.

1.1.1 Transients Simulation Module

Transients simulation module is to generate simulated electromagnetic transient disturbances for test and verification of the proposed scheme for this classification system.

Five kinds of typical voltage disturbances including switching transient and four types of fault transients have been taken into consideration.

1.1.2 Feature Extraction Module

The purpose of feature extraction is to identify specific signatures of the disturbances in power systems. For instance, a capacitor-switching and a short circuit event result in different disturbed voltage and should have different features. Generally, real-time transient modelling and classification are very challenging because most of transient signals are aperiodic, nonstationary, and short-term duration. To extract features from transient signals, the *discrete wavelet transform* (DWT) and the *variance fractal dimension trajectory* (VFDT) approach are employed in this thesis.

Since the mid-1980s, the *wavelet transform* (WT) has been received considerable interest in many other areas such as acoustics, image compression, communications, seismics. We propose to use it as a fast and effective means of analyzing voltage and current waveforms generated during power system disturbances. Like the familiar Fourier transform, wavelet transform decomposes a signal into its frequency components. Unlike the Fourier transform, the WT provides a non-uniform division of the frequency domain; i.e., a greater time resolution at high frequencies and a smaller resolution at low frequencies, thus allowing good localization in both time and frequency domains. The ability of wavelet to focus on short time intervals for high-frequency components improves the analysis of signals with localized impulses and oscillations, particularly in the presence of a fundamental and low-order harmonics. This ability to tailor the frequency resolution can greatly facilitate the detection of signal features which may be useful in characterizing the source

of the transient. Since the WT is well suited to transient signals, we apply it to extract time-frequency features of power transients from the wavelet time-scale domain in our thesis. For computer implementation, the DWT is used. In wavelet domain, if the all coefficients of wavelet components are taken as features, the computational overheard is large and is not suitable for near real-time speed requirement for classification using neural network. In this thesis, we select the first five detailed signals of wavelet decomposition to study power transients and extract features by calculating the values of *root mean square* (RMS) of wavelet coefficients, as described in Section 5.3.1.

Since fractal analysis can measure the complexity of an object or a signal, it provides a new approach to characterize a nonstationary signal [Cant83], [Kins94a], [Kins95]. The most important advantage of this technique is that it can achieve input data normalization automatically because of its theoretical lower and upper bounds for fractal dimension values regardless of the actual amplitude of the original signal. For example, the variance dimension for a temporal signal, in particular, has the lower bound of 1.0 and the upper bound of 2.0. Fractal analysis of transients is usually conducted in terms of fractal dimensions. We use *variance fractal dimension* (VFD) to characterize a time series. The VFD can be used to observe nonstationarity in the signal, thus leading to a *variance fractal dimension trajectory* (VFDT). An important advantage of the VFDT approach is that it can be formulated either for a batch or real-time computation, while a spectral approach permits the batch processing only. Another advantage of the VFDT is that it does not require a window in the Fourier sense, and therefore does not introduce the window artifact [Kins94b]. In addition, high data compression ratio can be achieved using the VFDT approach.

It has been shown that the classification performance of power transients is quite good when we use combination features extracted from the DWT and the VFDT approach.

1.1.3 Neural Network Classification Module

After the features of a power transient are extracted, the next step is to identify the transient. Such features are input to a classifier for classification. The classifier is designed using neural network technique that has been developed fast during last ten years.

Neural networks have the capability to learn complex mappings, linear or nonlinear, from the input space to output space while remaining robust in a noise environment. Therefore, they are often used as classifiers. In this thesis, we use a *probabilistic neural network* (PNN) for classification. The PNN combines some of the best attributes of statistical pattern recognition and feed-forward neural networks. It is the neural network implementation of kernel discriminant analysis and was introduced into the neural network literature by Donald Specht [Spec90a]. The PNN features very fast training times and produces outputs with Bayes posterior probabilities. Therefore, we choose it as our classifier for classification of power transients. Since the PNN processes p input variables (i.e., p features), we use the multi-sigma PNN classifier that uses a separate sigma weight for each input variable. It has also been proved that the multi-sigma PNN can achieve better classification performance than the one-sigma-fits-all PNN.

1.2 Thesis Statement and Objectives

The purpose of this thesis is to develop an intelligent recorder that has the ability to classify various types of power transients automatically based on effective and efficient wavelet and multifractal modelling, as well as neural network classifier. In order to achieve this research goal, the following objectives are required:

a) A technique for robust feature extraction which focuses on a time-frequency description and complexity measure for the transients; and

b) An efficient neural network classifier for classification of the transients based on the extracted features.

1.3 Thesis Organization

This thesis is organized from theoretical background to implementation of algorithms. Chapter 1 states importance of classification of power transients, introduces main methods to be used for this project, and outlines motivation and objectives of this thesis.

Chapter 2 provides the basis of wavelet analysis, with emphasis on discrete Daubechies wavelet, one of the most popular orthonormal wavelets. The implementation for such transform is introduced via *multiresolution signal decomposition* (MSD) technique. Finally, wavelet analysis on power transient signals is examined in order to highlight the performance of the method.

Chapter 3 begins with the concepts of fractal and fractal dimensions. Then, detail of the VFDT technique is introduced for multifractal analysis on nonstationary signals.

The experimental results of calculating the VFDT for power transients are shown, along with the discussion of the parameter setting in computing the VFDT for feature extraction purpose.

Chapter 4 gives an overview of the underlying scheme for PNN classification. The structure of the PNN is given and the PNN processing is described, particularly the Bayes classification strategy and Parzen's method for density estimation. This chapter also discusses the PNN training.

Chapter 5 is concerned with the experimental design of power transients identification system and the software implementation. Such system consists of three modules: transients simulation, feature extraction, and neural network classification. The specific functions to be implemented in each module are described.

In Chapter 6, experimental work is accordingly conducted to test more data sets with and without noises for realization and verification of the proposed approach. First, this chapter begins with an overview of the signals studied in this thesis. Then, it deals with the approaches used for feature extraction. After that, it focuses on the transient classification procedures. Experimental results are presented and discussed.

Chapter 7 provides conclusions and contributions made by this thesis based on the experimental results. Recommendations for further research are also given.

CHAPTER II
WAVELET ANALYSIS ON POWER TRANSIENTS

2.1 Introduction

Real-time transient classification is very challenging because transient signals are usually aperiodic, nonstationary, and short-term duration. To extract features from transient signals, many methods such as the *short-time Fourier transform* (STFT), the wavelet transform (WT), and the multifractal analysis are proposed. The STFT technique has limitation with transients analysis. The WT is a better approach than the STFT by allowing localization in both frequency and time.

We discuss the WT beginning with comparing it to Fourier analysis method. The *Fourier transform* (FT) has been proved extremely valuable for periodic, time-invariant, or stationary phenomena. The basis functions used in Fourier transform are sine and cosine functions that are precisely located in frequency, but exist for the all time. The frequency information of a signal revealed by the FT is an average over the entire time duration of the signal. Thus, if there is a local transient happened at a certain time interval, it may contribute to the FT, but its location on the time axis will be lost. The STFT overcomes the time location problem by using a window function whose position is translated in time axis by τ. If the width of the window covers a time span T_w, the frequency spacing between harmonics is $1/T_w$. The use of a short window means that $1/T_w$ is large, thus precise frequency is impossible. Therefore, the STFT cannot provide multiple resolution in time and frequency simultaneously, which is a drawback when analyzing transients

containing both high and low frequency components. It also suffers from the artifact of the choice of a proper window size. Therefore, a new time-frequency analysis method is needed to analyze transient signals. The WT provides a good time-frequency description and overcomes the limitations of the Fourier-based methods by employing analyzing functions (i.e., wavelets) that are localized both in time and frequency domain. The wavelet analysis transforms the power transient signal into different time-frequency scales. More specifically, it decomposes the transients into a series of wavelet components, each of which corresponds to a time-domain signal that covers a specific octave frequency band containing more detail information. Such wavelet components are useful to detect, locate, and classify the sources of transients [JBWCR99], [KaMI00]. Additionally, wavelets can be used to reconstruct power signals with a relatively small number of components. Even by using only a small percent of the wavelet coefficients, the reconstruction of such signals is quite good [PiBh96], [SaPG97]. It means that compact representation of the signals can be obtained in wavelet domain, which benefits fast speed classification. In recent years, a few studies which investigate feasibility of using WT for power transients analysis have been reported [ADDT98], [HuHH99], [LiMo99], [Moki97], [PaSa98], [PiBh96], [RCMG96], [SPGH96]. It has been showed that the wavelet analysis has ability to characterize nonstationary signals such as the transients in power systems.

This chapter includes four sections. Firstly, we discuss why the wavelet time-frequency analysis can be applied to nonstationary power transient signals. Then brief theoretical description of wavelet transform is provided, with emphasis on Daubechies wavelet, one of the most popular orthonormal wavelets. The implementation for discrete wavelet transform is introduced via *multiresolution signal decomposition* (MSD) tech-

nique. Finally, wavelet analysis on power transient signals is examined in order to highlight the performance of the method.

2.2 Wavelet Transform

2.2.1 Theory of Wavelet Analysis

The wavelet transform is the mathematics associated with building a model for nonstationary signals, using a set of components that look like small waves, called wavelets. Like the Fourier analysis, wavelet analysis deals with expansion of functions in terms of a set of basis functions. Unlike Fourier-based analysis that expands functions in terms of trigonometric polynomials, wavelet analysis expands functions in terms of wavelets.

The wavelet transform of a signal consists of computing coefficients that are inner products of the signal and a family of wavelets. A *continuous wavelet transform* (CWT) is defined as [Daub92]

$$WT(a,b) = \frac{1}{\sqrt{a}}\int_{-\infty}^{\infty} x(t)\Psi\left(\frac{t-b}{a}\right)dt \tag{2.1}$$

where $x(t)$ is a continuous signal to be analyzed, $WT(a,b)$ is its CWT, and a and b are the scale and translation parameters, respectively. The factor $1/\sqrt{a}$ is used to ensure energy preservation. The analyzing function $\Psi(t)$ is not limited to a complex exponential. It can be of any function that satisfies conditions: being oscillatory, quickly decaying to zero at both sides, and having at least one vanishing moment. $\Psi(t)$ is said to have d vanishing moments, if and only if, for all positive integer $q < d$, it satisfies

$$\int_{-\infty}^{\infty} t^q \Psi(t) dt = 0 \tag{2.2}$$

These restrictions ensure that the integral in (2.1) is finite and gives the name wavelet or small wave to the transform, with $\Psi(t)$ referred to as the "mother wavelet" and its dilations and translations simply as "daughter wavelets" or "wavelets". Wavelets are of finite duration, whereas in the Fourier transform the sine and cosine waves are periodic and of infinite duration. This short time duration of the wavelets helps in achieving the localization in both time and frequency. Like the discrete Fourier transform, there is a discrete wavelet transform (DWT) for the sake of computer implementation. Using similar notation, the corresponding DWT is defined as

$$DWT[m, n] = \frac{1}{\sqrt{a_0^m}} \sum_{k=-\infty}^{\infty} x[k] \Psi\left(\frac{k - n b_0 a_0^m}{a_0^m} \right) \tag{2.3}$$

where $x[n]$ is a discrete-time signal to be transformed and $\Psi(n)$ is the mother wavelet. The mother wavelet may be dilated and translated discretely by selecting $a = a_0^m$ and $b = n b_0 a_0^m$, where a_0 and b_0 are fixed constants with $a_0 > 1$, $b_0 > 0$, $m, n \in Z$, and Z is the set of positive integers. A simple choice of a and b are $a_0 = 2$ and $b_0 = 1$. In this case, the wavelet transform is called a dyadic-orthonormal wavelet transform. Due to the orthonormal properties, there is no information redundancy among the decomposed signals. Moreover, with this orthonormal basis, there exists an elegant algorithm, known as the multiresolution signal decomposition (MSD), to decompose a signal into two parts, which represent smoothed and detailed signals of the original signal. The detail for implementation is presented in the following subsection.

In practice, (2.3) is usually truncated at finite limits. The scaling gives the DWT logarithmic frequency coverage in contrast to the uniform frequency coverage of the Fourier-based transforms. The magnitude of the wavelet coefficients provides the information on how close the scaled and translated wavelet is to the original signal.

2.2.2 Implementation of Discrete Wavelet Transform

The DWT can be easily and quickly implemented by means of a suitable algorithm, based on the consideration that the mother wavelet and its scaled versions can be thought of, in the frequency domain, as band-pass filters with constant relative bandwidth (ratio between frequency bandwidth and center frequency). Each filter extracts a proper frequency sub-band of the signal. As a consequence, a multiresolution signal decomposition and reconstruction can be carried out.

The algorithm can be applied in a forward and a backward way. The forward algorithm serves to compute the DWT coefficients, i.e., to perform the analysis of the signal while the backward version computes the inverse transform, i.e., to perform the synthesis of the original signal from the DWT coefficients. Here, we concern the forward algorithm for feature extraction purpose of the signal from wavelet domain. The forward algorithm uses two digital filters, low pass and high pass, which, combined in a tree structure, constitute a filter bank able to decompose the signal into low and high frequency components. These filters are combined with down-sampling operations, that is, steps that throw away every other sample at each operation, halving the data each time. The MSD term holds in all the procedures to obtain low-pass smoothed signals and high-pass detailed signals from an input signal. The MSD technique is described mathematically in the following.

Let $x[n]$ be an original discrete-time signal. From the MSD technique, the decomposed components at scale 1 are the smoothed signal, $c1[n]$, and the detailed signal, $d1[n]$, in the form of wavelet transform coefficient [Daub92]

$$c1[n] = \sum_k h[k-2n]x[k] \tag{2.4}$$

$$d1[n] = \sum_k g[k-2n]x[k] \tag{2.5}$$

where $h[n]$ and $g[n]$ are the associated filter coefficients that decompose $x[n]$ into $c1[n]$ and $d1[n]$, respectively. We will further discuss the meanings of the $h[n]$ and $g[n]$ in this section. The next higher scale decomposition is now based on the signal $c1[n]$. The decomposed signals at scale 2 are given by

$$c2[n] = \sum_k h[k-2n]c1[k] \tag{2.6}$$

$$d2[n] = \sum_k g[k-2n]c1[k] \tag{2.7}$$

Higher scale decompositions are performed in a similar manner. Notice that at each decomposition level, the number of sample points of the decomposed signals is half of that of the signal in the previous stage. For instance, if $x[n]$ has N sample points for the entire observation time, then signals $c1[n]$ and $d1[n]$ have $N/2$ sampling points for the same observation period.

The filters $h[n]$ and $g[n]$ determine the wavelets used to analyze the signal $x[n]$. The $h[n]$ and $g[n]$ have the same finite length L, where L is related to the number of van-

ishing moments in $\Psi(t)$, defined in (2.2). If one chooses the filters $h[n]$ and $g[n]$ with four filter coefficients as in (2.8) and (2.9), then one will use the so-called Daubechies wavelet with four filter coefficients (or Daub4 for the sake of brevity). Other choices of filter coefficients are possible such as 6, 8, 10, etc., coefficients, and the analyzing wavelets are called Daub6, Daub8, and Daub10, respectively [SPGH96].

Filters $h[n]$ and $g[n]$ form a family of scaling function $\Phi(t)$ and primary wavelet $\Psi(t)$ functions. These functions are defined recursively, as linear combination of scaled and shifted version of $\Phi(t)$ that is defined by (2.8) and (2.9).

$$\Phi(t) = \sqrt{2}\sum_{k=0}^{L-1} h[k]\Phi(2t-k) \tag{2.8}$$

$$\Psi(t) = \sqrt{2}\sum_{k=0}^{L-1} g[k]\Phi(2t-k) \tag{2.9}$$

The scaling function $\Phi(t)$ satisfies,

$$\int_{-\infty}^{\infty} \Phi(t)dt = 1 \tag{2.10}$$

In the wavelet representation of signals, $h[n]$ behaves as low-pass filter and $g[n]$ behaves as a high-pass filter to the input signal. The coefficients of the $h[n]$ are the reverse of the $g[n]$ with every other coefficient negated, thus these two filters are called *quadrature mirror filters*. The signal $x[n]$ is filtered by $h[n]$ and $g[n]$. The resulting signal from $h[n]$ is $c1[n]$, a smoothed signal of the original signal $x[n]$, because filter $h[n]$

has a low-pass frequency response. The resulting signal from $g[n]$ is $d1[n]$, a detailed signal of the original signal $x[n]$, because filter $g[n]$ has a high-pass frequency response. Signal $d1[n]$ is called the wavelet transform coefficient at scale one. Similarly, signal $d2[n]$ is called the wavelet transform coefficient at scale two. Figure 2.1 illustrates implementation of the DWT through a filter bank configuration.

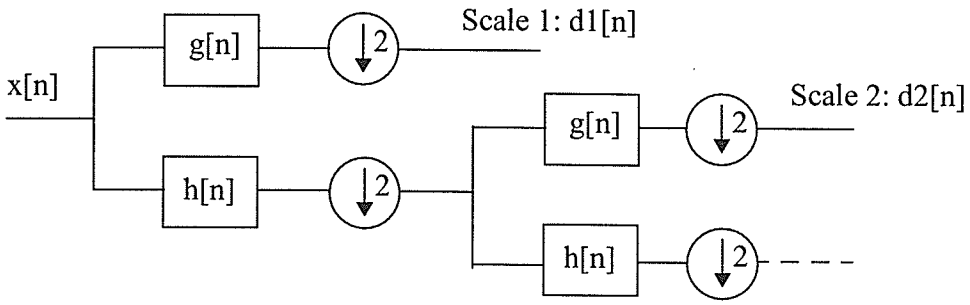

Fig. 2.1. Multi-stage filter bank DWT Implementation.

2.2.3 Choice of Mother Wavelet

There are several wavelet systems to implement wavelet transforms in modern signal processing. These include Haar, Daubechies, and etc. A signal can be decomposed by different families of wavelets. The choice of mother wavelet and wavelet parameters depends on the particular application. For example, for signal processing in real time, computational efficiency may be the essential feature to be considered. For classification of power system disturbance signals, the choice hinges on discrimination between various types of transient disturbances and computationally feasibility, and so on. In our case, low amplitude, short duration, and fast decaying and oscillatory types of signals are our interested. One of the most popular othonormal wavelets, Daubechies' wavelet, has been shown the abilities to detect and characterize such power transients. This thesis selects the

Daubechies wavelets to analyze power transient signals. Considering computational efficiency in our case, the Daub4 is appropriate in this thesis [Daub92]. Equations (2.11) and (2.12) are Daub4 scaling and mother wavelet functions, respectively [WiCo96].

$$\Phi(t) = c_0\Phi(2t) + c_1\Phi(2t-1) + c_2\Phi(2t-2) + c_3\Phi(2t-3) \tag{2.11}$$

$$\Psi(t) = -c_3\Phi(2t) + c_2\Phi(2t-1) - c_1\Phi(2t-2) + c_0\Phi(2t-3) \tag{2.12}$$

where $c_0 = (1+\sqrt{3})/4$, $c_1 = (3+\sqrt{3})/4$, $c_2 = (3-\sqrt{3})/4$, and $c_3 = -(\sqrt{3}-1)/4$.

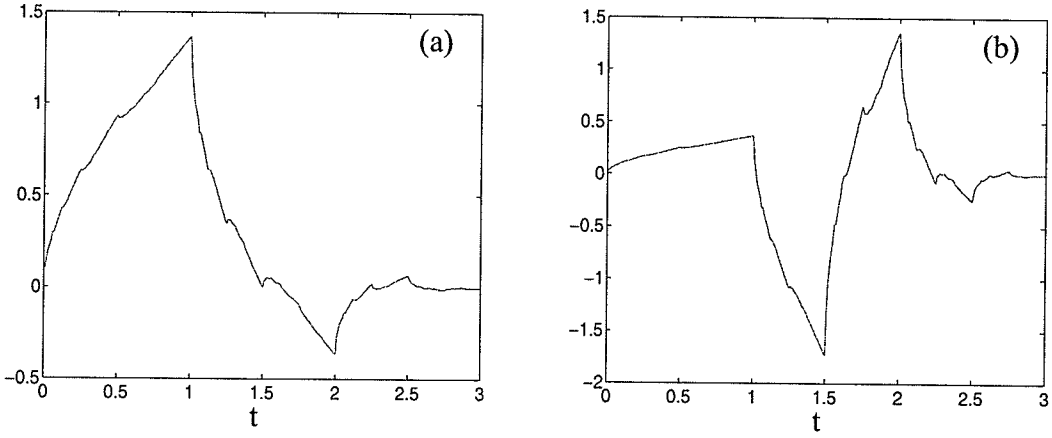

Fig. 2.2. (a) Scaling function and (b) mother wavelet function for the Daub4

Generally, it is not possible to solve directly for $\Phi(t)$. The obvious approach is to solve for $\Phi(t)$ iteratively until $\Phi_j(t)$ is very nearly equal to $\Phi_{j-1}(t)$, where j is the jth iteration. The scaling function $\Phi(t)$ and mother wavelet $\Psi(t)$ for the Daub4 are shown in Figs. 2.2 (a) and (b), respectively, generated from (2.11) and (2.12) after 8 iterations.

2.3 Wavelet Analysis on Power Transients

In our experiment, discrete Daub4 wavelet transform has been used to characterize several types of simulated transients such as impulsive transients, oscillatory transients, voltage sags or swells, momentary interruption, and so on. The wavelet transform is accomplished via the MSD technique.

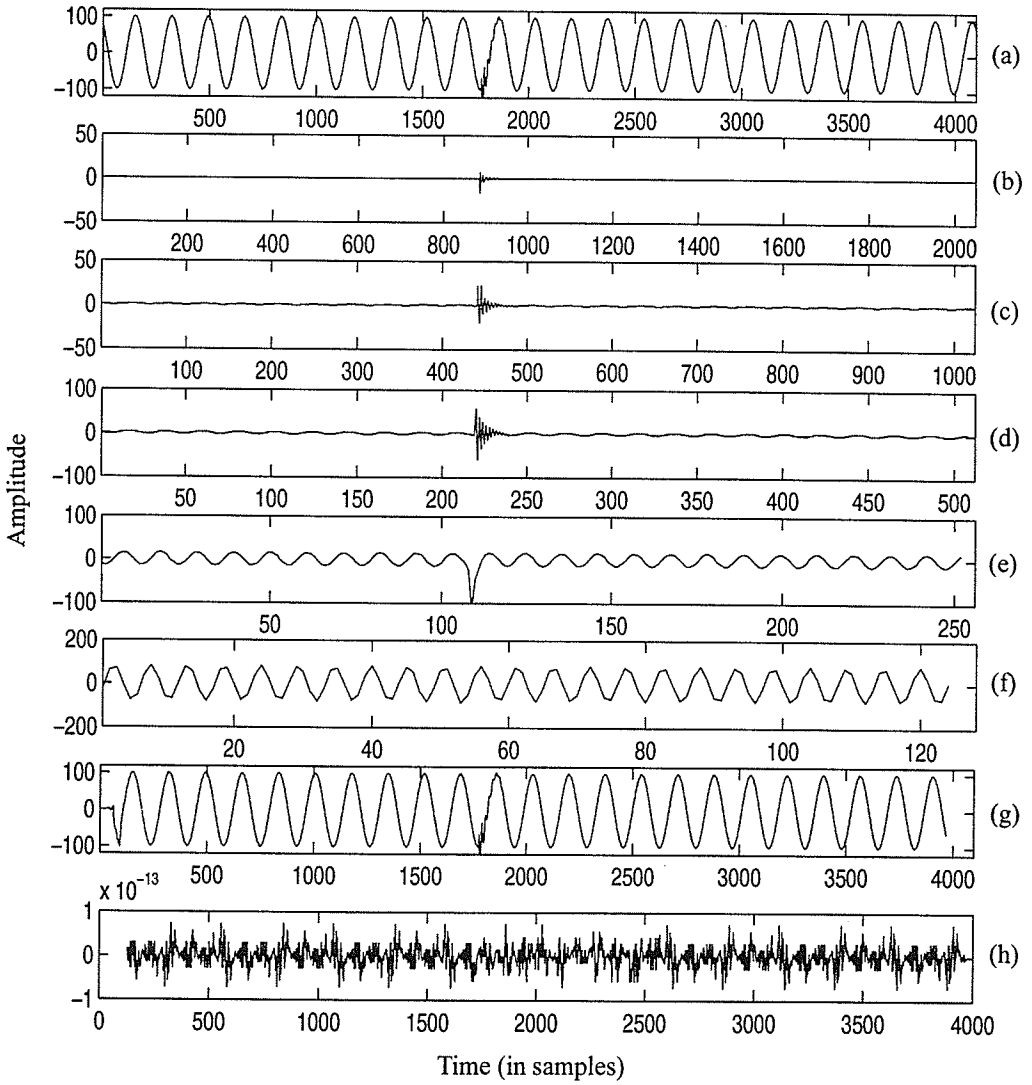

Fig. 2.3. The DWT on a power transient. (a) A simulated oscillatory transient, *s*; The detailed signals: (b) *d1*, (c) *d2*, (d) *d3*, (e) *d4*, and (f) *d5*; (g) Reconstructed signal *s** from components (b) to (f); (h) Reconstructed Residual $r = s - s^*$.

Figure 2.3 shows an example of using discrete Daub4 wavelet to analyze a typical capacitor switching event. Figure 2.3 (a) is a simulated oscillatory transient sampled at 10,240 Hz with data length of 4096 points. The original signal is transformed into detailed signals as identified in Figs. 2.3 (b) to (f) and named detail $d1$, $d2$, $d3$, $d4$, and $d5$, respectively. The abscissa for each waveform in Figs. 2.3 (b) to (f) represents time, whereas the ordinate is an amplitude scale, called wavelet coefficients.

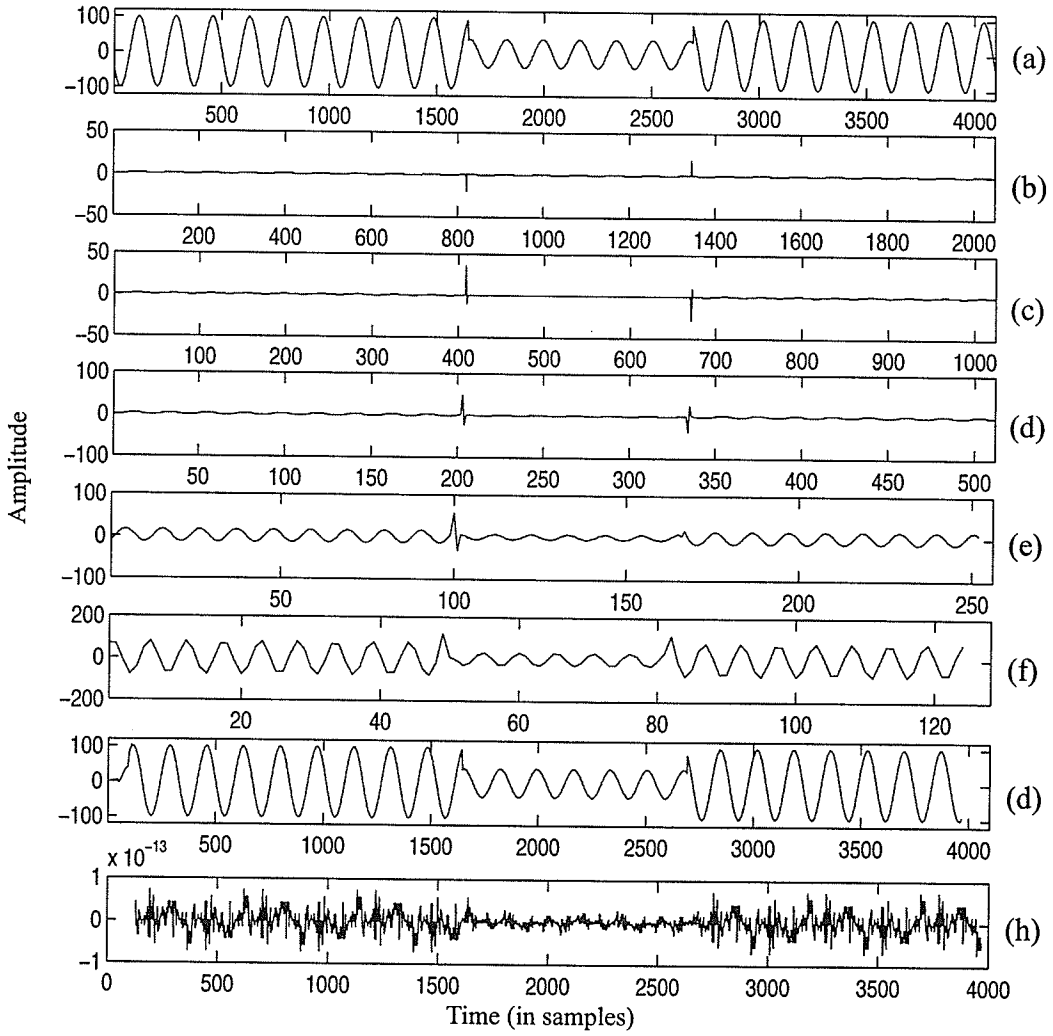

Fig. 2.4. The DWT on a power transient. (a) Simulated voltage sags, s; The detailed signals: (b) *d1*, (c) *d2*, (d) *d3*, (e) *d4*, and (f) *d5*; (g) Reconstructed signal s^* from components (b) to (f); (h) Reconstructed Residual $r = s - s^*$.

Another example is given for wavelet analysis on a voltage sag event. Similarly, Fig. 2.4 (a) is a simulated voltage sag event. Through DWT, it is decomposed into detailed signals as identified $d1$ to $d5$ shown in Figs. 2.4 (b) to (f), respectively. From the results, we can see that by decomposing transients into the detailed signals that contain the sharp edges, transitions, and jumps, it is relatively easy to localize, detect, and recognize the transients, thus leading to efficient feature extraction in wavelet domain. We only select detailed signals $d1 \sim d5$ for feature extraction because the further decomposed components are dominated by the power signal. However, even if only five detailed signals are studied to classify the power transients, the number of wavelet coefficients is still very large. Therefore, further analysis of these coefficients is necessary. Section 5.3.1 gives the detail of feature extraction from wavelet domain.

2.4 Summary

This chapter presents wavelet analysis applied to nonstationary temporal signals, especially transients. Discrete Daub4 wavelet transform has been used to investigate simulated disturbances. The implementation of the DWT is accomplished via the MSD technique. The essential idea underlying wavelet analysis is to decompose power transients into a series of wavelet components, which are useful for analyzing and classifying various transients. Wavelet analysis is feasible and practical in the area of power quality problems. The advantages using it over Fourier-based methods are: (i) it is efficient in detecting and localizing various types of power system disturbances, (ii) it can zoom the area of interest for better visualization of signal characteristics, (iii) it gives both frequency and time information simultaneously, and (iv) it is relatively easy to extract the

features from wavelet time-scale domain. Therefore, wavelet analysis is appropriate in the study of transients because of the nature of such signals and considered to be a good approach for feature extraction of power transient signals. The next chapter deals with another novel technique, mutilfratal analysis on transient signals.

CHAPTER III
MULTIFRACTAL ANALYSIS ON POWER TRANSIENTS

3.1 Introduction

The previous chapter presents wavelet transform on power transients. Wavelet transform characterizes the transients from time-frequency multiresolution point of view. This chapter introduces another technique, multifractal analysis, to study such nonstationary signals. Multifractal analysis characterizes the transients from fractal complexity point of view and in the form of fractal dimensions. Particularly, multifractal analysis of the transients is conducted in term of the variance fractal dimension trajectory (VFDT).

Analysis for nonstationary signals is challenging for traditional signal processing techniques, as stated in Section 2.1. Fractal technique has ability to characterize such signals. Kinsner's research group suggests several novel fractal approaches for: (i) image segmentation of breast cancer which outperforms traditional image segmentation techniques [Chen97]; (ii) electrocardiogram segmentation [HuKi01]; and (iii) noise segmentation and feature extraction of turn-on transients from radio transmitters [Shaw97] [Sun99]. Based on the features extracted using VFDT approach for multifractal analysis, the identification system for radio transmitters can achieve the average classification rate around 97%. There are several important advantages of the VDFT approach over other techniques: (i) it can achieve input data normalization automatically; (ii) it can be formulated either for a batch or real-time computation; (iii) it does not require a window in the Fourier sense, and

therefore does not introduce the window artifact; (iv) it is immune to noise; and (v) high data compression ratio can be achieved [Kins94b].

Based on the above mentioned advantages and successful applications, the VDFT approach is used to analyze power transients for feature extraction, which is performed by selecting a set of fractal dimensions as the unique complexity representation of a transient signal.

In this chapter, Section 3.2 introduces the concepts of fractal and fractal dimensions. Section 3.3 presents the detailed procedure of calculating the variance fractal dimension (VDF) that is chosen as an appropriate model for this application. Section 3.4 shows the experimental results of calculating the VFDT for power transients along with the discussion of the parameter setting such as the sliding window size and its displacement for feature extraction purpose. Section 3.5 provides a summary about this chapter.

3.2 Fractal and Fractal Dimensions

To describe complex objects in nature that Euclid left as formless or amorphous, the concept of fractal and fractal dimension was introduced. In Euclidean description, it is well known that topological (Euclidean) dimension of an object is considered to be the smallest possible non-negative integer space in which the object can be embedded. For instance, Euclidean dimension is 0 for a point, 1 for a line, 2 for a surface, and 3 for a volume [Kins94a]. However, the definition of fractal dimensions was motivated by the observations that such values are not appropriate to describe more complex objects. The appearance of objects such as the Cantor set [Cant83] triggered a search for a better defini-

tion of dimension. As a result, the concept of dimension was further advanced from integer to fractional.

A fractal is a subset in $\mathbf{R}^n$ which is self-similar and whose fractal dimension exceeds its topological dimension [Deva92]. A fractal dimension can be interpreted simply as the "degree of meandering" (or roughness or brokenness, or irregularity) of an object [Kins95], [Mand82]. Another interpretation of a fractal dimension is a critical exponent that makes a measure of the object constant [Kins94a]. For example, although a coastline is immeasurable in terms of length, it has a certain characteristic (often constant) degree of meandering under the expression as a power law relationship between the measurement and the scales, in which the critical exponent value is defined as the fractal length dimension for the coastline when it converges to a constant [Kins94a].

Fractal based modelling can be classified into the general four categories: morphological-based, entropy-based, spectrum-based and variance-based dimensions [Kins94a]. We will focus on variance-based dimensions because it is suited for fractal modelling of temporal signal, such as the power transients being analyzed in this thesis.

3.3 Variance Fractal Dimension

The variance fractal dimension (VFD) is used for characterizing the fractal components of a time series. A time series representing a chaotic or nonchaotic process can be analyzed directly in time by computing the spread of the increments in the signal amplitude (i.e., through its variance, σ^2) [Kins94b].

Now we introduce how to calculate the variance dimension from a given time series. Let us assume that the signal $x(t)$ is either continuous or discrete in time t. The variance, σ^2, of its amplitude increments over a time increment Δt is related to the time increment according to the following power law

$$Var[x(t_2) - x(t_1)] \sim |t_2 - t_1|^{2H} \tag{3.1}$$

where H is the Hurst exponent. By setting $\Delta t = |t_2 - t_1|$ and $(\Delta x)_{\Delta t} = x(t_2) - x(t_1)$, the exponent H can be calculated from

$$H = \lim_{\Delta t \to 0} \frac{1}{2} \frac{\log[Var(\Delta x)_{\Delta t}]}{\log(\Delta t)} \tag{3.2}$$

Finally, for the embedding Euclidean dimension E (i.e., the number of independent variables in the observed signal), the variance dimension can be computed from

$$D_\sigma = E + 1 - H \tag{3.3}$$

In practical computation, The *lim* in (3.2) is translated into the slope of a log-log plot with a finite sequence of time increments, $\{\Delta t_1, \Delta t_2, \ldots, \Delta t_k\}$. In order to spread the points on the log-log plot equally, the sequence is usually chosen to be b-adic, as shown in Fig. 3.1. The following log values are calculated for the plot

$$X_k = \log_b(\Delta t_k) \tag{3.4}$$

$$Y_k = \log_b[Var(\Delta x)_k] \tag{3.5}$$

where k is in the range $[k1, k2]$ which corresponds to the optimal linear range in the log-log plot.

A polynomial line fitting approach is used to determine the slope s from these points, with careful attention given to outliers [Kins94a]. The slope is computed from the log-log plot from

$$s = \frac{(k2-k1)\sum_{i=k1}^{k2} X_i Y_i - \sum_{i=k1}^{k2} X_i \sum_{i=k1}^{k2} Y_i}{(k2-k1)\sum_{i=k1}^{k2} X_i^2 - \left(\sum_{i=k1}^{k2} X_i\right)^2} \tag{3.6}$$

The Hurst exponent H is obtained by

$$H = s/2 \tag{3.7}$$

If the time series is stationary, this analysis produces a single VFD.

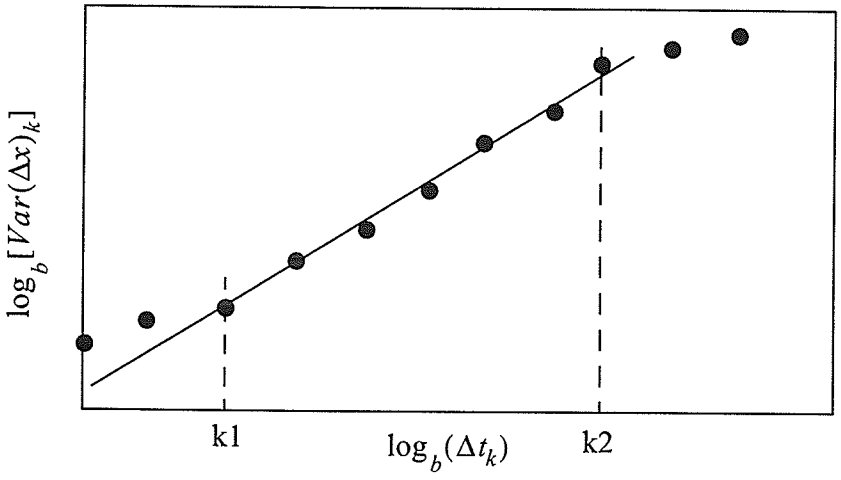

Fig. 3.1. A typical log-log plot. The outer points are not included in the fit.

3.4 Multifractal Analysis on Power Transients

3.4.1 Multifractal

Before applying multifractal analysis on power transients, the concept of multifractal is introduced. A multifractal refers to an object with different local fractal dimension. A local fractal dimension refers to a dimension value calculated on a limited area of the fractal object. A common approach to calculating local fractal dimensions is by using a moving window that selects data only within this window. For a temporal signal, we use a rectangular window that slides along the time axis.

If the local fractal dimension is the same everywhere, then the fractal object is a pure fractal, otherwise it is called multifractal [Vics92]. For example, a sine wave and a completely uncorrelated white noise are pure fractal with the local variance dimensions of 1.0 and 2.0 everywhere, respectively. However, a transient or other nonstationary signal is not pure fractal because its fractality is time varying. Such a signal could be more appropriately characterized by a model consisting of several local fractal dimensions taken at different points in time. Multifractality deals with signals or objects which have varying local fractal dimensions.

3.4.2 Variance Fractal Dimension Trajectory

Due to the nonstationarity of power transients, multifractal analysis of them is suited and conducted in terms of variance fractal dimensions as a function of time, thus resulting in the VFDT.

The VFDT is generated by calculating local variance fractal dimensions for a rectangular sliding window that moves along the entire time axis. It is straightforward to apply

this approach to power transient signals. Thus, the transient signal is modelled and characterized by selecting a set of local fractal dimension values. When the VFDT is used, the sliding window is shifted a large amount along the time axis, resulting in a smaller number of points on the trajectory. The local variance dimensions corresponding to these points are taken as extracted features of the power transient signal.

In the procedure of feature extraction using the VFDT approach, a special type of fractal modelling, fractal amplification [Kins94a] is used. Fractal amplification refers to emphasizing the fractality from a computational point of view. The variance dimension is calculated by using a dyadic sequence of time interval, instead of a linear sequence. This results in relatively large values and more variations of the dimension, which benefits transient signals classification [Shaw97], [Sun99].

3.4.3 Parameter Setting for Feature Extraction

A number of parameters need to be considered in computing the VFDT, including the rectangular sliding window size for the local variance dimension calculation, N_w, and its displacement, *NS*. First, the appropriate selection of N_w is important. If the N_w is too large, it will cause the dimensions of locally distinct fractals to be buried in the dimensions of their most significant adjacent fractals, and at the same time will be computationally time consuming. On the other hand, the relatively small N_w will not provide sufficient data to cover the elementary components of the transients for the analysis. Second, it is also important to select an appropriate spacing, *NS*, between successive windows. To ensure the entire signal to be considered, the *NS* should be in the range from a single sample up to the width of the window. If the *NS* is too small, e.g., one sample step, the successive win-

dows will be overlapped too much, thus causing too much correlation and time-consuming computation. If the *NS* is too large, important details in the trajectory may be lost. Notice that the *NS* directly determines the amount of data reduction, i.e., feature number for classification. It is important that this parameter should be selected conservatively so that significant fractal characteristics are not neglected in the modelling process. Generally, the parameter setting should be dependent on the nature of the signal being analyzed and can be studied experimentally to give the classification result for transients as optimal as possible [Sun99], [Shaw97], [ChKi00].

We shall now describe experiments for finding appropriate values of the N_w and *NS* (in sample number) for calculation of the VFDT of power transients. The experiments are done by changing: (i) the N_w to view the VFDT with given *NS* and (ii) the *NS* to view the VFDT with given N_w. The analyzed signal is a simulated oscillatory transient with 4096 samples obtained at 10,240 sps and shown in Fig. 3.2 (a).

First, we investigate the effect of the N_w on the VFDT with given $NS = 128$. By setting the N_w as 128, 256, 512 and 1024, the corresponding VFDTs of the transients are calculated and shown in Figs. 3.2 (b) to (e), respectively. We observe that a relatively small N_w results in more variation in the VFDT as shown in Fig. 3.2 (b), whereas a large N_w causes smooth trajectory as shown in Fig. 3.2 (e). For our purpose of classification, it is found that the N_w of 256 is reasonable because it represents the significant features of power transients while avoiding too much details.

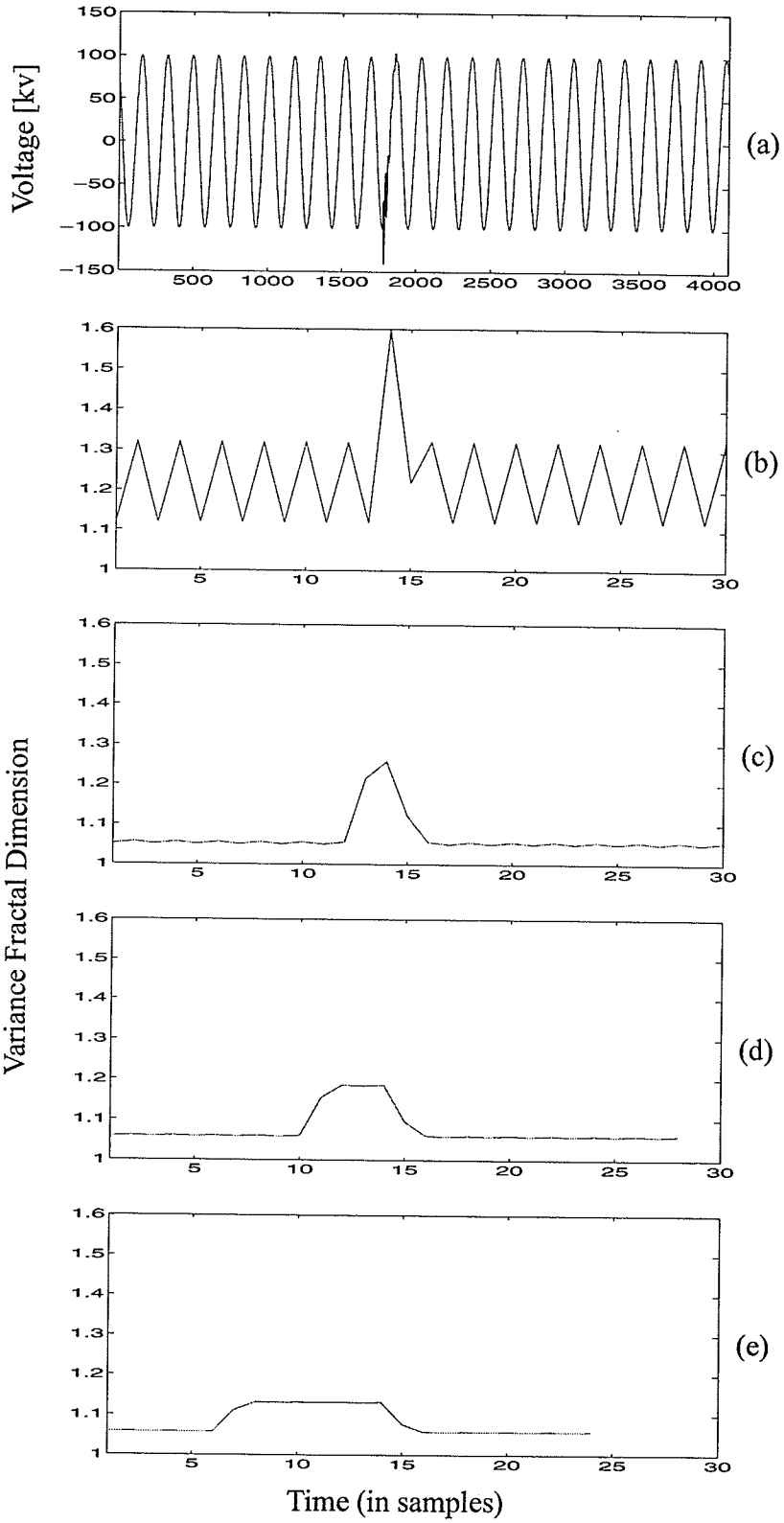

Fig. 3.2. The effect of window size on the VFDT (*NS* is 128). (a) is an oscillatory transient. The N_w is (b) 128, (c) 256, (d) 512, and (e) 1024.

Second, we investigate the effect of the *NS* on the VFDT with given $N_w = 256$. By setting the *NS* as 64, 128, and 256, the corresponding VFDTs of the transients are shown in Figs. 3.3 (a) to (c). One can see that the effect of the *NS* is similar to that of the N_w on the VFDT. For our purpose of classification, it is found that the *NS* of 128 is appropriate. Notice that the high data compression ratio can be achieved by variance dimension technique just by increasing the *NS*. For instance, if the transient length is 4096 samples, the N_w and *NS* are set to 256 and 128, respectively. In this case, we get 30 fractal dimensions extracted from the original signal.

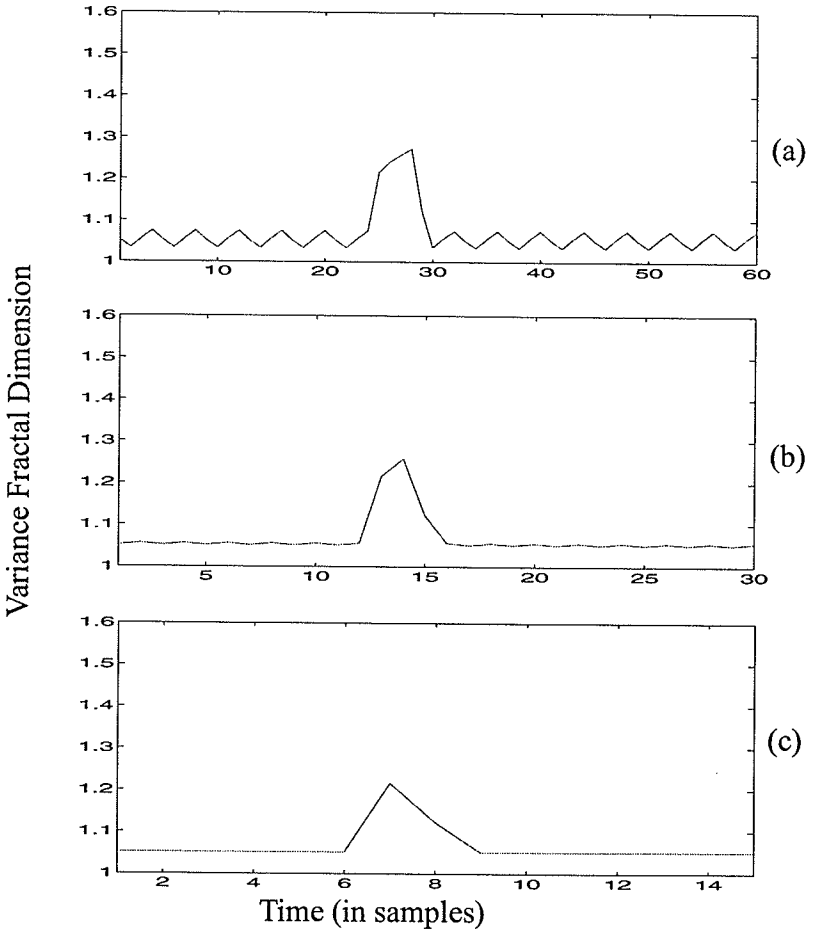

Fig. 3.3. The effect of displacement parameter on the VFDT (N_w is 256). The *NS* is (a) 64, (b) 128, and (c) 256.

3.5 Summary

In this chapter, we introduce the application of fractal technique on analyzing non-stationary transients in power system. The VFD is chosen as an appropriate modelling method to characterize the fractal components of power transients. Due to the nonstationarity of power transients, multifractal analysis of them is suited and conducted in terms of variance fractal dimensions as a function of time, thus resulting in the VFDT. The parameter setting of the sliding window size and its displacement for calculation of the VFDT is discussed. In our case, we find that the window size of 256 and the displacement of 128 (in sample number) are suitable for the purpose of feature extraction of power transients. Our experiments show that multifractal analysis of transients based on the VFDT is efficient for characterizing most of transient types. As we have demonstrated, the fractal method can achieve the normalization of input data automatically due to theoretical lower and upper bounds of dimension value. In our case, this value should be in the range between 1.0 and 2.0. In addition, data reduction can be realized by simply changing the displacement of the sliding window.

After wavelet and fractal analysis of transients, the next stage deals with the classification. This is done by a particular neural network, namely probabilistic neural network. Its basis is described in the next chapter.

CHAPTER IV
THE PROBABILISTIC NEURAL NETWORK

4.1 Introduction

After wavelet and multifractal analysis of transients are completed, classification is done by inputting the features of transients into a classifier for classification. We use a neural network classifier since neural networks have the capability to learn complex mappings, linear or nonlinear, from the input space to output space.

Neural networks are a statistical analysis tool, that is, they let us build behaviour models starting from a collection of examples (defined by a series of numeric or textual descriptive variables) of this behaviour. They are comprised of a number of processing units that are linked via weighted interconnections. A processing unit is essentially an equation which is often referred to as a "transfer function". A neural network uses previous examples (training set) to establish (learn) a relationship between the input variables and the output variables by setting these weights. Once the relationship is established (the neural networks is trained), the neural network can be presented with new variables and it will generate predicted outputs.

A typical neural network is the multilayer feedforward network, often called the *backpropagation network* (BPN). The problems with using this network are: (i) the training time becomes too long if the searching space is large and (ii) it is difficult to decide how many layers and nodes are required for a practical application. A more suitable candidate for classification of power transients is the probabilistic neural network (PNN), which

overcomes the limitations of the BPN [Shaw97], [MoKi98]. The PNN is based on the Bayes classification rule, which minimizes the expected risk of misclassification for a given decision surface, thus achieving optimal classification. Furthermore, due to its modular architecture and fast learning speed, it is suited for evolving learning systems as found in on-line classification [Spec88]. Therefore, the PNN is applied for power transients classification.

The basic idea behind the PNN is the Bayes classification rule and Parzen's method of density estimation. To understand the PNN paradigm, we begin with a discussion of the Bayes decision strategy and Parzen's estimator of probability density functions. Then the architecture of the PNN is provided.

4.2 The PNN Basic

4.2.1 Bayes Classification Strategy

An accepted norm for decision rules or strategies used to classify patterns is that they do so in the way that minimizes the "expected risk." Such strategies are called Bayes strategies.

Consider a C-class situation where we have a collection of samples belonging to C different classes denoted as $k = 1, 2, \ldots, C$. Each of these samples is a p-dimensional vector $\mathbf{x} = (x_1, \ldots, x_p)$. The parameter h_k is defined as the prior probability of a class k. The parameter l_k is defined as the loss function associated with misclassifying a sample, which originally belongs to class k, into one of other classes. The $f_k(\mathbf{x})$ is the *probability density function* (PDF) for the class k. The Bayes decision rule is then stated as follows:

An unknown sample **x** is classified into class k if

$$\text{if } h_k l_k f_k(\mathbf{x}) > h_j l_j f_j(\mathbf{x}) \tag{4.1}$$

for all classes j not equal to k.

The above decision rule does in fact minimize the expected cost of misclassification. The prior probabilities h_k are often known or can be estimated accurately. In many or most applications, the prior probabilities are treated as being equal for each class. The same is true with the loss functions l_k. Therefore, the key to using (4.1) is the ability to estimate the PDFs $f_k(\mathbf{x})$ based on the training set. By using Parzen's method, we can get reasonable estimated PDFs when there is a comprehensive training set.

4.2.2 Parzen's Method of Density Estimation

Parzen (1962) presented an excellent method for estimating a univariate PDF from random samples. Parzen's PDF estimator converges asymptotically to the true density as the number of samples increases. Parzen's PDF estimator uses a weight function (kernel) $W(d)$, which has its largest value at $d = 0$ and which decreases rapidly as the absolute value of d increases. The PDF estimator is a scaled average of that function across the training set. Let us state Parzen's method mathematically.

Assuming that there are n_k training cases for a given class k, then the estimated PDF for that class, $g_k(x)$, is

$$g_k(x) = \frac{1}{n_k \sigma} \sum_{r=1}^{n_k} W\left(\frac{x - x_r}{\sigma}\right) \tag{4.2}$$

where $W(x)$ is the weight function (kernel) and the scaling parameter sigma, σ, defines the width of the kernel that surrounds each training case.

The Gaussian function is simply a well behaved, easily computed function that satisfies the conditions required by Parzen's method, and it has been shown in practice to perform well. Therefore, a common choice for the weight function, $W(x)$, is the (normalized for unit area) Gaussian function

$$W(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{x^2}{2\sigma^2}\right)} \tag{4.3}$$

Proper choice of the value of σ is critical to the performance of the PNN. If σ is too small, individual training cases will be considered only in isolation and we will be left with essentially a nearest-neighbour classifier [Mast93]. In such case, individual cases exert too much influence, thus losing the benefit of aggregate information [Mast95]. However, a too large value of σ causes so much blurring that the details of the density are lost, often distorting the density estimate badly. Figure 4 shows the effect of different values of σ on the PDF estimation. In this figure, we have eight sample points from a single class in the training set. Each of these points is the center of a Gaussian window function. Thus, Parzen's density approximation would be the sum of the eight functions multiplied by a constant. In Fig. 4 (a), one can see that a too small value of σ causes the estimated density function to have distinct modes corresponding to the locations of the training samples, which essentially leads to a nearest-neighbour classifier [Mast93]. Figure 4 (c) shows that a large value of σ causes the details of the density to be smoothed out and the estimated

density will be close to Gaussian regardless of the true underlying distribution. The value of σ is just about right in Fig. 4 (d).

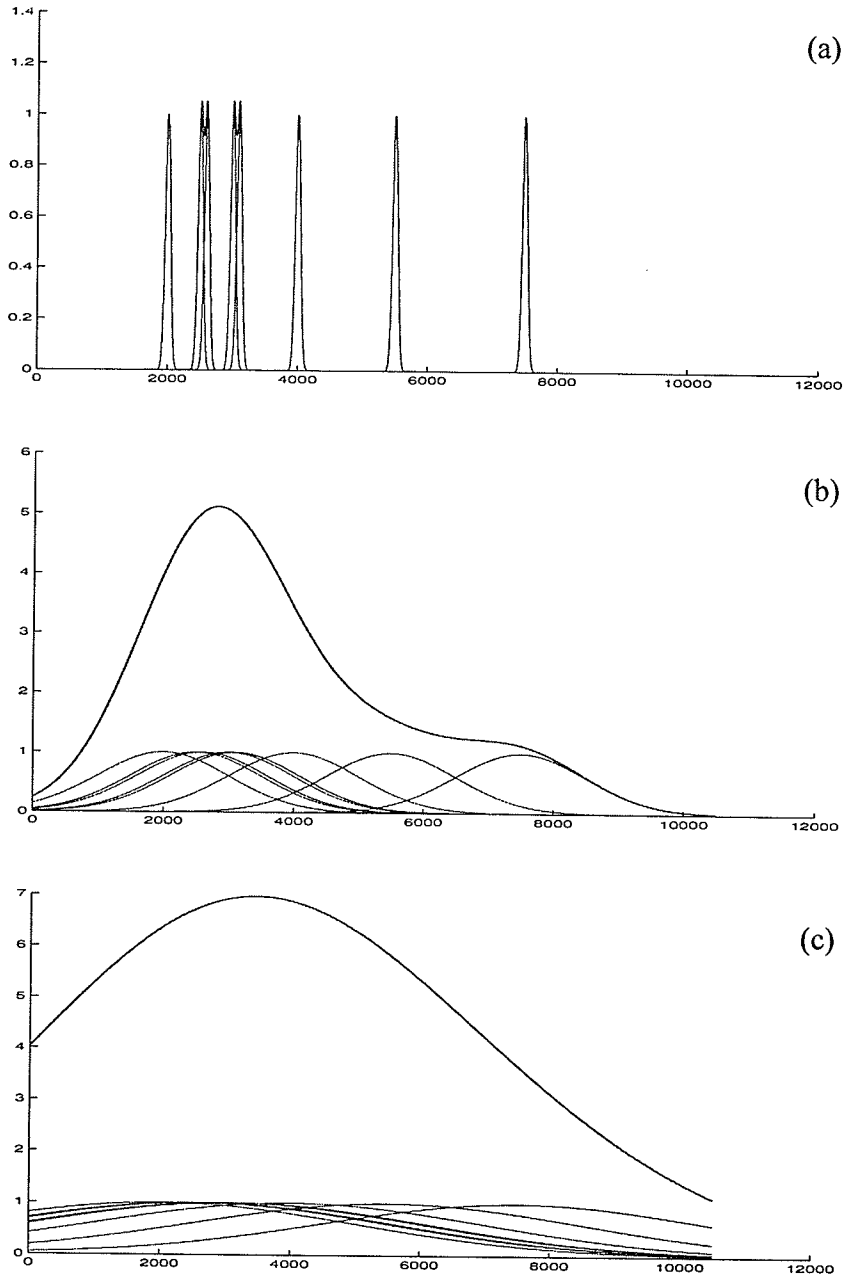

Fig. 4.1. The effect of different σ values on the PDF estimation. The σ is: (a) too small, (b) just about right, and (c) too large.

4.2.3 Multivariate Extension of Parzen's Method

Since the PNN processes p input variables (i.e., p features in our case), the PDF estimator must consider multivariate input. Cacoullos (1966) has extended Parzen's method to the multivariate case [Caco66]. It is much more complicated because each variable x_i is allowed to have its own scaling factor σ_i, $i = 1, \ldots, p$. This fully general density estimator for a class k is given by

$$g_k(x_1, \ldots, x_p) = \frac{1}{n_k \sigma_1 \ldots \sigma_p} \sum_{r=1}^{n_k} W\left(\frac{x_1 - x_{r,1}}{\sigma_1}, \ldots, \frac{x_p - x_{r,p}}{\sigma_p}\right) \tag{4.4}$$

where $\mathbf{x_r} = (x_{r,1}, \ldots, x_{r,p})$ means the rth sample from the class k and n_k is the number of the samples belonging to this class.

For a simple case, we may assume that all scaling parameters are equal, i.e. $\sigma_1 = \sigma_2 = \ldots = \sigma_p = \sigma$. That assumption comes to our basic or traditional PNN model [Mast93]. However, this one-sigma-fits-all approach is a handicap. Firstly, the poor user is burdened with the task of providing the network with variables whose variations are all commensurable. If that is not done, good performance is unlikely. Secondly, worse still is the fact that even when care is taken to properly scale the data, performance may still suffer. This is because the user probably does not know in advance which variables are important and which are not. The presence of variables whose variation is meaningless has a dilute effect on the useful variables. Ideally, we would want the variation of unimportant variables to be small so that they exert minimal influence on the distance measure computed between an unknown sample and each training case. This distance measure should be dominated by the important variables [Mast95]. However, it is unreasonable to

expect a user to decide in advance which variables are important and which are not. The solution to this problem is to use a separate σ weight for each variable [Mast95].

Let us examine the modelled power transients shown in Fig. 2.3, Fig. 2.4, and Fig. 3.2 (c). One can see that the features extracted from wavelet and multifractal modelling are not all commensurable. The difference is also large among the values of features extracted from wavelet transform. Therefore, good performance could not be achieved using a single σ value to scale all of the variable. This has also been proved in our experiments that the PNN does not achieve better performance with one-sigma-fits-all than with multi-sigma. Therefore, in this thesis, we will focus on the multi-sigma PNN which uses multivariate extension of Parzen's method to estimate the PDFs for each class.

We can achieve economy by letting the multivariate weight function be the product of the univariate weight function, as shown by

$$W(x_1, \ldots, x_p) = \prod_{i=1}^{p} W_i(x_i) \tag{4.5}$$

Now, with this simplification along with the Gaussian kernel in (4.3), the density estimator as described in (4.4) is rewritten as

$$g_k(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}\sigma_1\sigma_2\ldots\sigma_p n_k} \sum_{r=1}^{n_k} e^{-\sum_{i=1}^{p} \frac{(x_i - x_{r,i})^2}{2\sigma_i^2}} \tag{4.6}$$

where $\mathbf{x} = (x_1, \ldots, x_p)$ is the input vector.

A conjugate gradient method is used in this thesis as a multivariate optimization scheme for finding multiple values of σ in estimation of the PDFs [Mast95]. Although there is no universal, mathematically rigorous method for choosing the "best" values of σ, we can choose them as optimal as possible based on the PNN classification purpose. The discussion for choice of the scaling parameters will be in Section 4.3.

4.2.4 The PNN Architecture

As stated, the Bayes decision rule assigns pattern **x** to the class that has a *maximum posterior probability* (MAP). MAP-rule is optimal in the sense that no other rules can yield a lower error rate. By using this method for classification, density function for each class is needed. We also know the multivariate extension of Parzen's method for estimating the PDFs from samples. Thus, a good classifier is available. Donald Specht discovered that the Bayes-Parzen classifier could be cast in the form of a neural network. The Specht's PNN architecture is shown in Fig. 4.2. It is a four-layer organization.

In the input layer, the number of input neurons is equal to the dimensional number p of feature space of an input vector $\mathbf{x} = (x_1, \ldots, x_p)$ that is used to describe the objects to be classified. These neurons merely distribute input variables to the next layer. In the pattern layer, one neuron contains one training case from training set and it computes a distance measure between the unknown input **x** and the training case represented by that neuron. An activation function, known as the Parzen window, is then applied to the distance measure. In the summation layer, there is one neuron for each of the C classes. These neurons sum the values of the pattern layer neurons corresponding to that class to obtain an estimated density function of the class, which is desired by Bayes discriminant

criterion. The set of σ, $\sigma = (\sigma_1, \ldots, \sigma_p)$, of the Parzen window are the only parameters needed to be decided by the PNN, which shows the simplicity of the PNN. In the output layer, the number of neurons is also equal to the number of classes, *C*. The output layer is often a simple threshold discriminator which activates a single neuron to represent the projected class of the unknown sample.

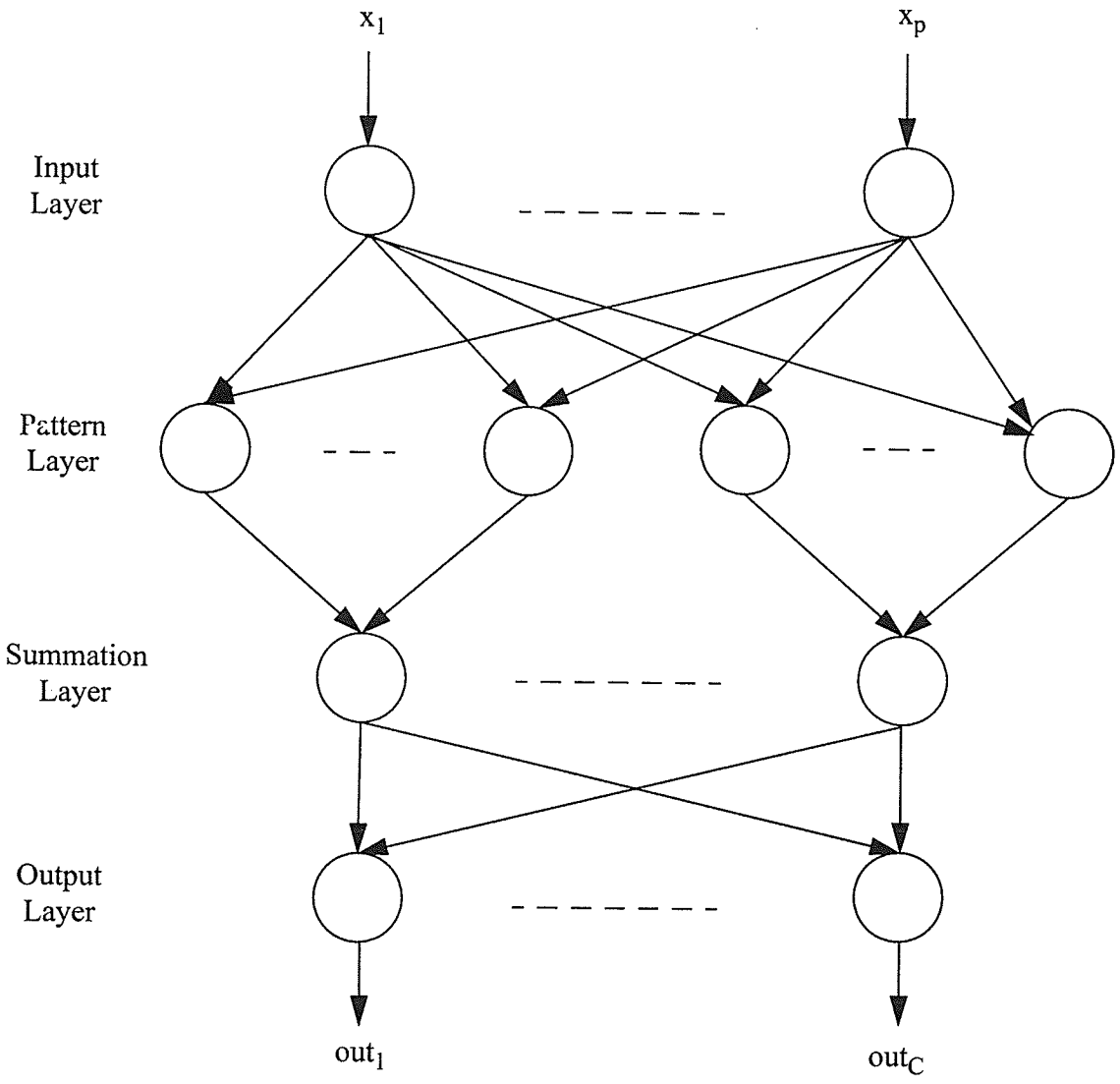

Fig. 4.2. The PNN Architecture

4.2.5 The PNN Computation

Assume an unknown case, $\mathbf{x} = (x_1, \ldots, x_p)$, is given at the input layer. In the pattern layer, each neuron calculates the scaled squared Euclidean distance between the unknown case $\mathbf{x}$ and a training case $\mathbf{x_r} = (x_{r,1}, \ldots, x_{r,p})$, given by

$$D(\mathbf{x}, \mathbf{x_r}) = \sum_{i=1}^{p} \left(\frac{x_i - x_{r,i}}{\sigma_i} \right)^2 \tag{4.7}$$

Then the weight function described in (4.3) is applied to this distance as

$$W(\mathbf{x}, \mathbf{x_r}) = e^{-D(\mathbf{x}, \mathbf{x_r})} \tag{4.8}$$

In the summation layer, the summation neurons then compute the PDF for the class they represent by

$$g_k(\mathbf{x}) = \frac{1}{n_k} \sum_{r=1}^{n_k} e^{-D(\mathbf{x}, \mathbf{x_r})} \tag{4.9}$$

Notice that the common multiplicative constant factors are ignored for achieving economy in (4.7) and (4.9).

In the output layer, all the outputs from the neurons of summation layer are compared and the largest value is selected. The output neuron corresponding to this value is activated and the index of this activated neuron in the output layer represents the predicted class to which the unknown case is classified.

Now, given a training set and set of σ weights for the corresponding p variables, we can compute density estimations for each class at any given point. Therefore, we can conduct classification when an unknown case is input to the PNN. Then, how do we determine the set of optimal σ and what efficient optimization algorithms do we use? This brings us to the next topic.

4.3 The PNN Training

Before a neural network is used, it should be trained to establish a model for a particular application. Neural networks are trained through learning from a given training set. As known, neural networks have the capability to learn complex mappings from the input space to output space. For the PNN training, we employ a supervised learning scheme, which means that during the training (or learning) process, selected training cases are presented to the neural network along with the correct output. The classifier then adapts itself to produce the correct output when encountering similar cases. In our case, we need to train the PNN in order to find a set of optimal σ. To introduce an efficient optimization algorithm, a continuous error criterion is introduced first.

4.3.1 A Continuous Error Criterion

Suppose there are C classes for a given training set. We define a delta function that indicates whether or not a given case in the training set belongs to a particular class.

$$\delta_k(\mathbf{x_r}) = \begin{cases} 1 \text{ if training case } \mathbf{x_r} \text{ is a member of class } k \\ 0 \text{ otherwise} \end{cases} \tag{4.10}$$

A class vector $\mathbf{y}$, $\mathbf{y} = (y_1, \ldots, y_C)$, is used to represent the output from the PNN. Using (4.10), the class vector for training case $\mathbf{x_r}$ is $\mathbf{y} = (\delta_1(\mathbf{x_r}), \ldots, \delta_C(\mathbf{x_r}))$, which represents the fact that a given case belongs to class k by means of a C-vector that is entirely 0 except for a single 1 in the kth position. Now, the PNN can be considered as a mapping function between the input vector and the output vector.

Suppose we are evaluating the PNN for some observation $\mathbf{x}$ whose class membership is known. C activation functions, one for each class, are defined by

$$u_k(\mathbf{x}) = \sum_{r=1}^{n_k} \delta_k(\mathbf{x_r}) e^{-D(\mathbf{x}, \mathbf{x_r})} \tag{4.11}$$

where the scaled squared Euclidean distance $D(\mathbf{x}, \mathbf{x_r})$ has been defined in (4.7). Then, the Bayesian confidences are defined as

$$q_k(\mathbf{x}) = \frac{u_k(\mathbf{x})}{s(\mathbf{x})} \tag{4.12}$$

where

$$s(\mathbf{x}) = \sum_{k=1}^{C} u_k(\mathbf{x}) \tag{4.13}$$

Assume that $\mathbf{x}$ is known to belong to class k. We thus hope that

$$q_j(\mathbf{x}) = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases} \tag{4.14}$$

The continuous error attributed to this observation is shown as

$$e_k(\mathbf{x}) = [1 - q_k(\mathbf{x})]^2 + \sum_{\substack{j=1 \\ j \neq k}}^{C} [q_j(\mathbf{x})]^2 \tag{4.15}$$

4.3.2 The PNN Training Based on Error Function

The training process of the PNN is to minimize the continuous error in (4.15). The minimal error corresponds to the set of the optimal σ. A conjugate gradient method is employed to find the minimal error for our PNN with the help of global search and golden section techniques.

Conjugate gradient methods are one of the most popular and effective families of multivariate optimization algorithms. There is an almost beauty in conjugate gradient methods. They have nearly (or entirely) the power and convergence speed of full 2nd-order gradient methods which are optimal to minimize error for a data set under a model. Yet, they require neither the computation nor even the storage of any 2nd-order derivative information. As known, when there are more than a handful of variables to be optimized, computing inverse matrixes of 2nd-order derivatives becomes prohibitively expensive. The conjugate gradient methods overcome this limitation. The process of a conjugate gradient method for the multi-sigma PNN training is as follows:

(1) Renew conjugate gradient $\mathbf{d}_i$ by the previous one $\mathbf{d}_{i-1}$ and current negative gradient $\mathbf{g}_i$ of the error function,

$$\mathbf{d}_i = \mathbf{g}_i + \gamma \mathbf{d}_{i-1} \tag{4.16}$$

where γ is decided by current negative gradient $\mathbf{g}_i$ and previous one $\mathbf{g}_{i-1}$

$$\gamma = \frac{(\mathbf{g}_i - \mathbf{g}_{i-1}) \cdot \mathbf{g}_i}{\mathbf{g}_{i-1} \cdot \mathbf{g}_{i-1}} \tag{4.17}$$

(2) Renew parameters σ_i by previous σ_{i-1} and current conjugate gradient $\mathbf{d}_i$

$$\sigma_i = \sigma_{i-1} + \lambda \boldsymbol{d}_i, \lambda \in \mathbf{R} \tag{4.18}$$

where λ is a length along the conjugate direction for an optimal renewal of the σ_i. It will be found by global search and golden section techniques.

It is time-consuming to search an optimal λ from a continuous real space. The following procedure is for reducing the search time:

(1) Discretize λ into discrete variable.

(2) Apply the global search on such a discrete space to find out a nearly optimal λ_j.

(3) Treat $[\lambda_{j-1}, \lambda_{j+1}]$ as a continuous space again and apply golden section technique on this continuous segment to find the optimal λ_{opt}.

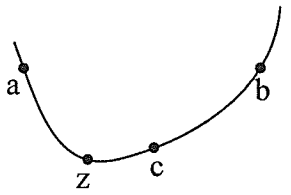

Fig. 4.3. The triplet and test point in golden section minimization

The golden section technique considers the triple of points a, c, and b, as shown in Fig. 4.3. The error is computed at an intermediate point, z, which is a fraction 0.38197 into the larger of the two intervals from the current minimum point c. For instance, in this figure, the interval between a and c is selected because it is larger than that between c and b. Thus, the point z is a fraction 0.38197 into this interval from the c.

4.4 Summary

This chapter presents the multi-sigma PNN as a classifier. The PNN is found to be appropriate for our application due to its fast training and mathematically sound solutions. The structure of the PNN is given and the PNN processing is described, particularly the Bayes classification strategy, Parzen's method, and multivariate extension of Parzen's method for density estimation. This chapter also introduces the PNN training with the conjugate gradient method in order to find a set of optimal σ for the PNN.

Now we conclude the theoretical background part of this thesis. The next objective is to implement the techniques experimentally. Several issues on experimental design and software implementation are discussed in the next chapter.

CHAPTER V
SYSTEM DESIGN AND SOFTWARE SIMULATION

5.1 Introduction

The previous chapters mainly focus on presenting the principle of the proposed approaches. This chapter is concerned with the experimental design of the power transients classification system and the software implementation. As stated in Section 1.1, the thesis deals with three main modules: transients simulation, feature extraction, and features classification. The specific functions to be implemented in each module are described in detail in the following sections.

5.2 Simulation of Power Transients

The proposed approaches for transients identification system need to be evaluated on several kinds of electrical disturbances. The transients simulation module generates a large number of simulated electromagnetic transient disturbances for test and verification.

As stated before, it is imperative to be able to distinguish the various types of disturbances accurately. In this thesis, five kinds of typical voltage disturbances associated with switching and fault transients have been taken into consideration including voltage sags, momentary interruption, voltage swells, impulsive transient, and oscillatory transient. The power transient signals used in this thesis are simulated in Matlab code. The sampling frequency used is 10,240 sps and the sample number is 4096. A brief description for each type of disturbances is followed [CuDR99]:

- The voltage sags describe a drop of 10~90 percent of the rated system voltage

lasting for 0.5 cycles to minute, usually caused by a remote fault somewhere on the power system;

- The voltage swells are temporary increase above normal level of voltage, often appearing on the unfaulted phases of a three-phase circuit where a single-phase short circuit occurs;
- The momentary interruption is a complete loss of voltage on a power system lasting for 0.5 cycles to 1 minute, usually caused by fault induced operation of circuit breaker or fuses;
- The impulsive transient is a sudden, non-power frequency change in the steady state condition of voltage, unidirectional in polarity (primarily either positive or negative), most commonly caused by lightning; and
- The oscillatory transient is a sudden, non-power frequency change in the steady state condition of voltage, including both positive and negative polarity values.

A oscillatory transient with a primary frequency component less than 5 kHz and a duration from 0.3 ms to 50 ms is considered a low frequency transient. Low frequency oscillatory transient is our concern because the most frequent event is capacitor switching which typically results in an oscillatory voltage transient with a primary frequency between 300 Hz and 900 Hz.

Some papers related to classification issue suggest that the voltage sags and the momentary interruption can be treated as one class, and so do the impulsive transient and

the oscillatory transient. In this thesis, further down to the individual disturbance, each one is treated as one class.

After the simulated transients were generated, they were stored on disk. Each file is corresponding to a case that belongs to one of the above mentioned five classes of disturbances. While a transient did exist in all of the files, its start point could be anywhere within the 4096 samples. Since wavelet and multifractal analysis have the ability to capture the transient part and extract features through analysis as shown in Section 2.3 and Section 3.4, it is unnecessary to extract the superimposed distortions on the fundamental component of system voltages in time domain. Therefore, a simulated disturbance is input directly to the next module for analysis and feature extraction.

In practice, there is always noise due to a complicated environment. Therefore, the effect of noise on this classification system is investigated in this thesis. We deal with white Gaussian noise, pink noise, and brown noise. The transient signals contaminated by noises are obtained by adding noise to simulated pure power signals with certain *signal to noise ratio* (SNR) according to our experimental requirements.

5.3 Feature Extraction of Power Transients

The feature extraction module is to yield compact representative models of transients while remaining the important information for classification of transients. As a result, fast training and on-line classification can be achieved. Each disturbance has its own specific signatures in light of time-frequency and complexity characteristics in the classification system. In this thesis, 60 features from 4096 data points of the original input

signal are extracted for classification, in which 30 features derived from wavelet analysis and the other 30 obtained from the variance fractal dimension trajectory (VFDT). The software simulation for feature extraction of the transients is implemented in Matlab 5.3. The following subsections provide the process of feature extraction in detail.

5.3.1 Wavelet Analysis and Feature Extraction

5.3.1.1 Discrete Daub4 Wavelet Transform

As known, the essential idea underlying the wavelet analysis is to decompose a given signal into different scales with different levels of resolution by dilating a single prototype function. In our experiment, discrete *Daubechies 4* (Daub4) wavelet transform has been used and performed via the multiresolution signal decomposition (MSD) technique. A transient signal can be fully decomposed into smoothed signals and detailed signals until L wavelet levels, given by $N = 2^L$, where N is the total number of data points in the original input signal. Since in our case, the transient for analysis has 4096 sample points, there are 12 dyadic wavelet levels the signal can be fully decomposed to. Each of these wavelet levels corresponds to a frequency band given by the equation:

$$f = 2^{v}\left(\frac{f_s}{N}\right) \tag{5.1}$$

where f is the higher frequency limit of the frequency band represented by the level v, and f_s is the sampling frequency [PaCo01].

Table 5.1 gives the frequency band information for the different levels for the wavelet analysis, where the sampling frequency f_s is 10,240 Hz. In our study, the wavelet

levels below 6 level (or above the 5 scales) are ignored due to too low frequency that is close to the fundamental component of system voltages, 60 Hz, which is beyond our interest for transients analysis. Therefore, only the first five detailed signals ($d1 \sim d5$) are studied for feature extraction of power transients in our thesis.

Table 5.1: Frequency band vs. wavelet level with the f_s of 10,240 Hz.

Wavelet level	Frequency band	Center Frequency
0 (a11)	DC-2.5Hz	1.25Hz
1 (d11)	2.5-5Hz	3.75Hz
2 (d10)	5-10Hz	7.5Hz
3 (d9)	10-20Hz	15Hz
4 (d8)	20-40Hz	30Hz
5 (d7)	40-80Hz	60Hz
6 (d6)	80-160Hz	120Hz
7 (d5)	160-320Hz	240Hz
8 (d4)	320-640Hz	480Hz
9 (d3)	640-1280Hz	960Hz
10 (d2)	1280-2560Hz	1920Hz
11 (d1)	2560-5120HZ	3840Hz

5.3.1.2 Feature Extraction from Wavelet Components

Even if only five octaves (detailed signals) are studied to classify the transients, the number of wavelet coefficients, $2^{11}+2^{10}+2^{9}+2^{8}+2^{7}= 3968$, is still very large. Therefore, further analysis of these coefficients is necessary. Since our goal is classification of the transients, the higher classification rate (or the lower classification error) the system can achieve, the better the approach is to extract the features of the transients. In our study, three approaches of feature extraction were studied including (i) *wavelet trans-*

form modulus maxima (WTMM) [MaHw92], (ii) *bin mean* (BM) [PhMS00], and (iii) *bin root mean square* (BRMS) approaches [PaCo01].

The first technique of feature extraction we use in this thesis is the WTMM approach. Although Meyer proved that the local maxima of a wavelet transform modulus do not characterize uniquely a function [Mery91], it has been shown numerically that the signal can be reconstructed, with a good approximation, from the local maxima of such modulus. For a two-dimensional signal, the WTMM technique first calculates modulus of the wavelet coefficients of the signal, then searches the local maximum modulus from the all modulus to represent the object. For a one-dimensional signal, the local modulus maxima correspond to local extrema in wavelet domain. Since the WTMM can represent the signal approximately and especially compactly, it may be used as features for classification purpose. To keep the number of the features as the same for various transients, the first m maxima of the WTMM in each octave are taken. As a result, $5 \times m$ features are extracted from wavelet domain for the five-level decomposition. The number m is decided by the experiment. In this thesis, the m of 6 is appropriate. Therefore, 30 time-frequency features are derived using the WTMM approach.

In the BM approach, each of the five detailed signal is divided into m equal length bins [PhMS00]. In each bin, there is a mean of the wavelet coefficients that fall within that bin. We take these mean values of all bins as features, thus obtaining m features for one detailed signal. Since we have five detailed signals for study, $5 \times m$ features are obtained. Similarly, we set the bin number m to 6. Therefore, 30 time-frequency features are derived using the BM approach.

The procedure of feature extraction using the BRMS approach is same with the BM approach except that instead of taking the mean value in each bin, we take the root mean square (RMS) value of the coefficients that fall within that bin as a feature. In power systems, the RMS is an acronym for Root Mean Square which represents the operation of taking the square root of the time average (mean) of the value squared. The RMS value of a power signal is useful because it facilitates average power calculations. Similarly, 30 RMS values are calculated as our features derived from time-frequency analysis through wavelet decomposition. More specifically, the RMS value $\|v_i\|$ of the ith bin in a detailed signal is given by

$$\|v_i\| = sqrt\left(\frac{1}{N_i}\sum_{j=1}^{N_i} d_i^{\,2}[j]\right), \qquad i = 1, \ldots, m \tag{5.2}$$

where m is the number of bins in each octave. N_i is the number of data points of the coefficients in that bin, given by $N_i = N/m$, and the N is the total number of coefficients in that octave. $d_i[j]$ is wavelet coefficient.

Among the above-mentioned three approaches, the BRMS approach is the best in extracting power transients features in light of classification experiments. The features extracted by the WTMM approach can yield a little bit higher classification rate than by the BM approach. However, when the signal is corrupted in a noisy environment, the WTMM technique is very time-consuming to extract features because a large number of local maxima may be detected. In addition, the performance of classification by the WTMM is poorer than by the BRMS. Therefore, the BRMS approach is chosen to extract features from wavelet domain of power transients in this thesis.

5.3.2 Feature Extraction from Multifractal Analysis

As previously stated, the variance fractal dimension trajectory (VFDT) is a multi-fractal model of the nonstationary power transient signal. It reveals the multifractal nature of the transient. The VDFT is generated by calculating local variance fractal dimensions for a rectangular sliding window that moves along the entire time axis. A number of parameters need to be considered in computing the VFDT, including the rectangular sliding window size for the local variance dimension calculation and its displacement. The parameter setting has been discussed in Section 3.4.2 in the role of feature extraction. Generally, it should be dependent on the nature of the signal being analyzed and can be studied experimentally to give the classification result for transients as optimal as possible [ChKi00], [Sun99], [Shaw97]. For the purpose of classification in this thesis, it is found by experiments that when the transient size is 4096 sample points obtained at 10,240 sps, the window size of 256 and the window displacement of 128 (in sample number) are reasonable, thus, we get 30 variance fractal dimensions as features of the transient signal. This represents a significant data reduction. The detail of how to calculate the VFDT has been stated in Chapter 3.

Worth to mention is that for the transients of pure momentary interruption, computing local variance fractal dimensions may fail in a certain duration where there are zero values of voltage. These zeros values will result in zero of the denominator in the formula of calculating the dimension. In such cases, the variance dimensions are set to 2, though not real dimensions, but just for distinguishing from other transients based on the classification purpose.

5.3.3 Combination Features

We conducted three experiments to classify power transients by using the wavelet miltiresolution time-frequency features, the VFDT-based features, and combination features of them, respectively.

Firstly, the multi-sigma probabilistic neural network (PNN) classified the miltiresolution time-frequency features of pure transients and noisy transients, respectively. The average classification rate was around 96% for pure in-sample data set and 89% for noisy transients contaminated by additive white Gaussian noise at 10 dB SNR. Secondly, classification of the VFDT-based features was conducted and the results showed poorer classification performance than that using features extracted by wavelet transform. Therefore, it was found that satisfactory classification of power transients could not be achieved using either the miltiresolution time-frequency features or the VFDT-based features alone. Therefore, we considered to combine these two-group features and expected them to complement each other. The experiment was repeated again using the combination features. The classification results were very satisfactory. The identification system can achieve classification rate of 100% using in-sample data set (i.e. training set), around 99% using pure out-of-sample data set (i.e. testing set), and around 91%, 87%, and 92% using transients contaminated by additive white Gaussian, pink, and brown noises at 10, 10, -5dB SNR, respectively. The detailed experimental results are shown in Section 6.3.

In general, more features result in good classification rate. However, too many features may lead to a degradation of the classification performance, which is curse of dimensionality. In our case, we found when we increased the number of wavelet features from

30 to 60 by increasing the number of bins, or increased the number of fractal features from 30 to 60 by decreasing the window displacement, the performance of this identification system did not improve anymore. Therefore, the increase of feature number does not always improve classification performance. The number of features should be decided by experiments. In addition, there is a trade-off between classification performance and speed. Considering these factors, we think that 60 features are appropriate in our thesis.

5.4 The PNN Classification

Once the feature extraction process is completed, classification is done by inputting the features of the transient signal into a classifier to decide which class it belongs to. We use the multi-sigma PNN as the classifier. In preliminary work, the multi-sigma PNN intends to classify five classes of power transients: voltage sags (labeled class 0), impulsive transients (class 1), momentary interruption (class 2), voltage swells (class 3), and oscillatory transient (class 4). More specifically, consider an input vector, $\mathbf{x} = (x_1, \ldots, x_{60})$, where x_i representing the ith feature of the 60 features. The PNN produces an output vector, $\mathbf{y} = (y_0, \ldots, y_4)$. After classification by the PNN, $\mathbf{y}$ will be filled with 0s except for a single 1 which indicates the output transient class. For example, if input is the oscillatory transient, the correct output vector of the classifier is $\mathbf{y} = (0, 0, 0, 0, 1)$. The simulations for the multi-sigma PNN training and classification are programmed in C++ language.

5.4.1 The PNN Training

The PNN training utilizes a supervised training scheme. The multiple scaling parameters sigmas, $\sigma = (\sigma_1, \ldots, \sigma_{60})$, in the Parzen density estimation, need to be

trained. The process of determining the multiple sigmas of the PNN based on the data set is called learning or training. The data set used in training is generally referred as training set and the data set used in testing as testing set. The multi-sigma PNN training is based on the conjugate gradient method (one of the most popular multivariate optimization methods), which uses the derivative information and average error measure to update sigmas iteratively. Detail of training the multi-sigma PNN has been discussed in Section 4.3. The iteration is finished when a sufficient accuracy has been achieved or successive improvement becomes negligible. Here, sigma threshold T_s and error threshold T_e are set for the iteration. When the distance between new sigmas and last one is less than the T_s, or the difference between new classification error and the last one is less than the T_e, the iteration is stopped. Otherwise, the iteration is repeated up to the maximum iteration number, set to 50. In the thesis, we set $T_s = T_e = 1.0e^{-10}$.

Notice that when a training case is compared to itself in the pattern layer, maximum activation in the output is produced due to zero distance measure, thus a bias is produced. The holdout method [Spec90b] is used to avoid this problem. When a member of training set is used as input, it is temporarily removed from the pattern layer.

5.4.2 The PNN Classification and Rejection

After finding a set of optimal σ weights based on the training set, classification tests should be conducted through the multi-sigma PNN. In this thesis, we classify pure simulated out-of-sample power disturbances. In addition, we also classify simulated power disturbances contaminated by white, pink, and brown noises under different SNR levels. The classification ability of the PNN is evaluated based on the confusion matrix

and confidential level, as well as average correct classification rate. Details of these tests and the experimental results are presented in Section 6.3 and Appendix C.

The rejection ability of the PNN classifier is also considered. When a completely new transient is input to the PNN classifier, it should be able to discard it, thus preventing the PNN to produce a possibly incorrect classification result. This achieves simply by providing the neural network a rejection threshold of $1.0e^{-10}$ on the summation neuron activation value. When all the summation activations are less than the threshold, it triggers an unknown transient.

5.5 Summary

This chapter describes the details of each module to be accomplished in the thesis including transients simulation, feature extraction, and the multi-sigma PNN classification. Based on the system design, the software simulation is realized.

Chapter 6 presents the experimental work as the realization and verification for the classification system design we have developed. Tests are conducted, results are reported, and conclusions are followed.

CHAPTER VI
EXPERIMENTAL RESULTS AND DISCUSSION

After describing the theoretical background and system design of the transients classification system in previous chapters, experiments are accordingly conducted to realize and investigate the proposed approach using data sets with and without noise. First, this chapter begins with a summary of transient signals studied in this thesis. The next section deals with approaches used for feature extraction. Section 6.3 focuses on the transient classification procedure, along with experimental results and discussion. Finally, a summary is presented at the end of this chapter.

6.1 Overview of the Transient Signals

A large number of simulated electromagnetic transient disturbances were generated for test and verification of the system. More specifically, the database for pure signals consists of 750 power disturbances, belonging to 5 types of disturbances, namely voltage sags, momentary interruption, voltage swells, impulsive transient, and oscillatory transient. Each type is treated as an individual class. There is an identical set of transient signals for each class, i.e., 150 cases for each class. A MATLAB code was used to generate these disturbances numerically. A transient signal is acquired with a sampling rate of 10,240 Hz and a length of 4096 samples for each case. Typical waveform for each class is shown in Figs 6.1 (a) to (e), respectively.

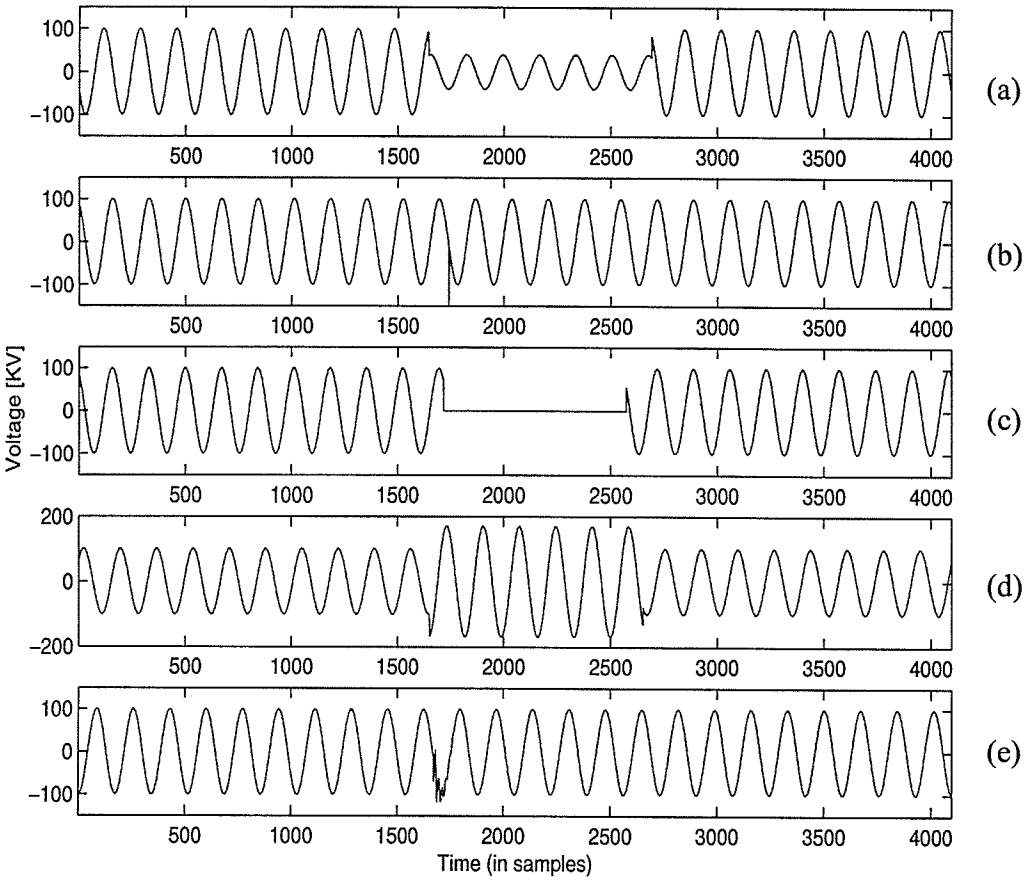

Fig. 6.1. Transients Simulation: (a) voltage sags, (b) impulsive transient, (c) momentary interruption, (d) voltage swells, and (e) oscillatory transient.

In practice, there is always noise due to complicated environments. Therefore, the effect of noise on this classification system is investigated in this thesis. We conducted the analysis of transient signals contaminated by noises such as white Gaussian noise, pink noise, and brown noise. The effect of color noise at different SNR levels is also investigated. The following is rational measure of SNR [Kins98]

$$SNR = \left(\sum_{i=1}^{N} x[i]^2 \right) / \left(\sum_{i=1}^{N} n[i]^2 \right) \tag{6.1}$$

or

$$SNR_{dB} = 10\log_{10}(SNR) \text{ [dB]} \tag{6.2}$$

where $x[i]$ is a pure transient signal, $n[i]$ is a noise, and N is the sample length of the transient signal and noise.

Our database for noise consists of 9 recordings from three kinds of noises, namely, white Gaussian noise, pink noise, and brown noise. The white, pink, and brown noise were generated by Matlab function, by a white-to-pink noise filter, and by integrating white noise, respectively. Each kind of noises has three recordings for the purpose of testing the consistency of noise effect. Each noise will be added to the pure simulated signals for evaluating the performance of the classification system in a noisy environment.

6.2 Feature Extraction

As stated in Section 5.3, 60 features from 4096 data points of the original input signal are extracted for classification, in which 30 features derived from wavelet transform and the other 30 obtained from variance fractal dimension trajectory (VFDT) approach. The software simulation of feature extraction of transients is implemented in Matlab 5.3.

To extract features from wavelet domain, discrete Daub4 wavelet transform has been used and performed via the multiresolution signal decomposition (MSD) technique. The features are extracted by bin root mean square (BRMS) approach from the first five detailed signals in wavelet domain. For instance, for the disturbances shown in Fig. 6.1, their corresponding multiresolution time-frequency features are shown in Fig. 6.2.

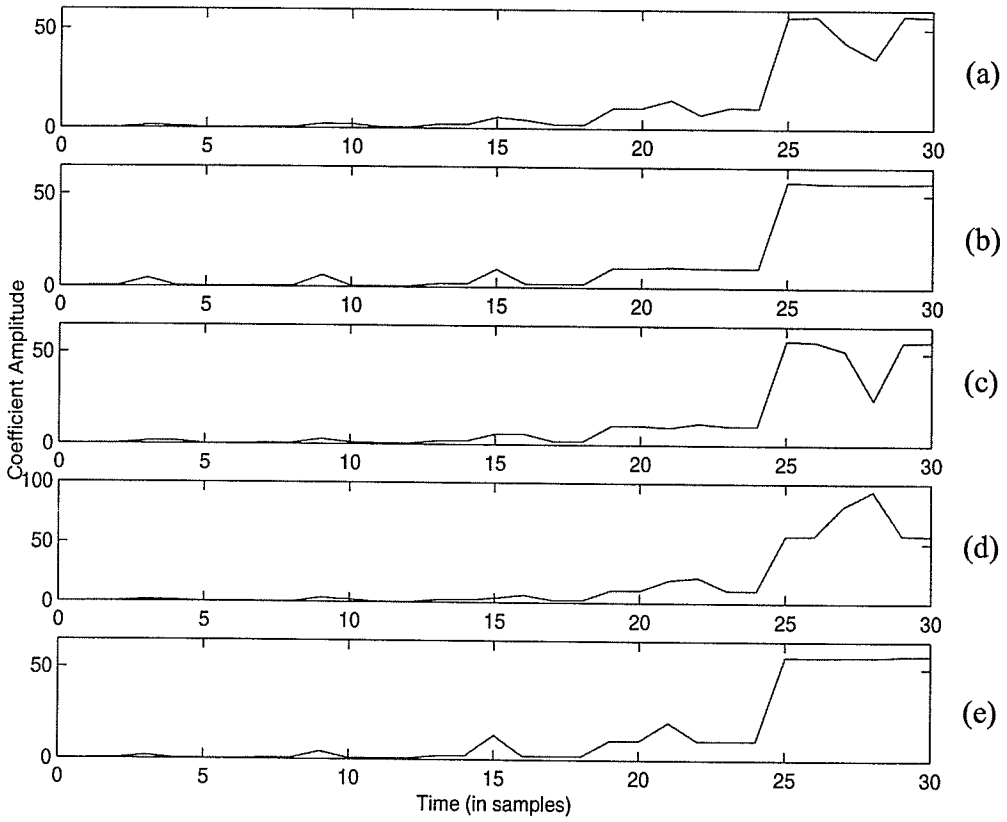

Fig. 6.2. Multiresolution time-frequency features for (a) voltage sags, (b) impulsive transient, (c) momentary interruption, (d) voltage swells, and (e) oscillatory transient.

In the procedure of feature extraction using the VFDT technique, we found that when the transient size is 4096 sample points obtained at 10,240 sps, a window size of 256 and a window displacement of 128 are suitable, thus leading to 30 features. For the signals shown in Fig. 6.1, their corresponding VFDT-based features are shown in Fig. 6.3. As mentioned before, the variance fractal dimension (VFD) for power transients is between 1.0 and 2.0. Notice that in the VFDT of the pure momentary interruption disturbance, several local variance dimensions are set to 2, which are not real dimensions, just for the classification purpose. We do this due to zero values of signal which result in failure for computation of some local VFDs because of zero values as denominator.

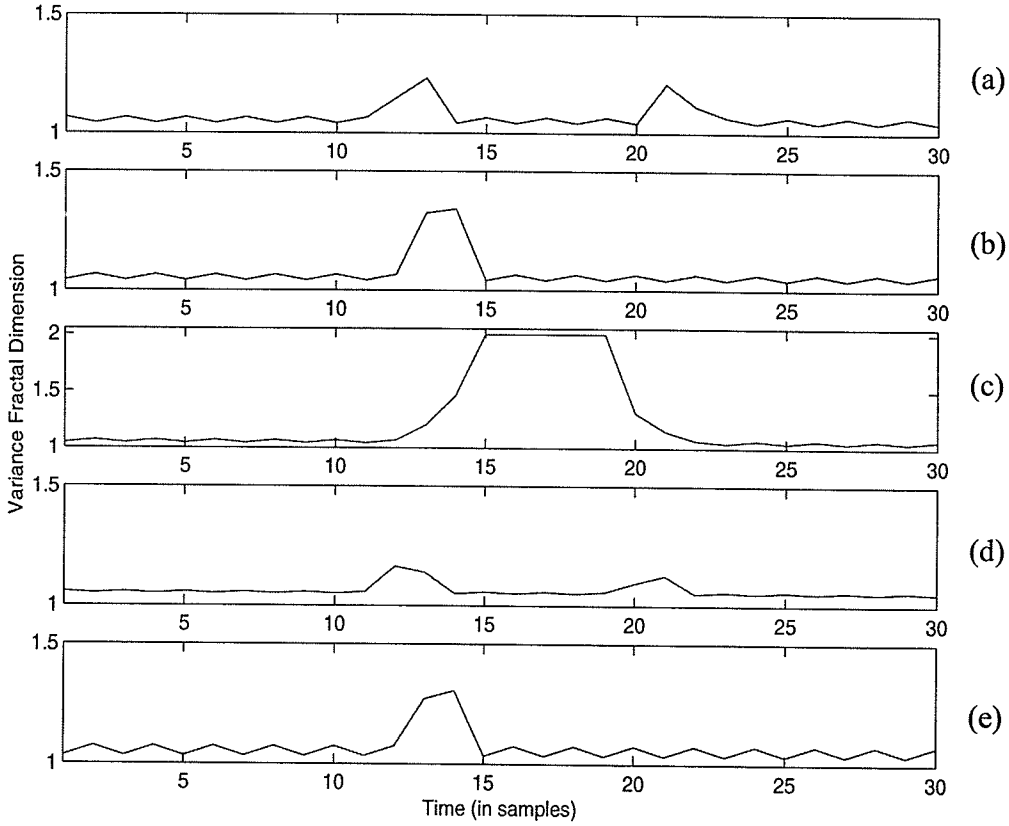

Fig. 6.3. The VFDT-based features for (a) voltage sags, (b) impulsive transient, (c) momentary interruption, (d) voltage swells, and (e) oscillatory transient.

6.3 The PNN Classification

6.3.1 Training and Testing the PNN on Pure Signal Data Set

A summary for all pure and noisy transients used for this thesis are provided in Table 6.1 and Table 6.2, respectively. Both tables show the class identification number assigned to each class. This number is for use by the classification stage of the system. In Table 6.1, the first 50 recordings of each class are used for training and the rest for classification. Therefore, there are 250 recordings in the training set and 500 in the testing set. In

Table 6.2, all cases are used as testing set. Therefore, there are 2250 cases with different noises at different SNR levels for testing the classification system in noisy environments.

Table 6.1: Summary of pure transients in experiments.

Disturbance Name	Number of disturbances	Class Identifier
Voltage Sags	150	0
Impulsive Transient	150	1
Momentary Interruption	150	2
Voltage Swells	150	3
Oscillatory Transient	150	4

Table 6.2: Summary of noise transients in experiments.

Disturbance Name	Number of cases with white noise at 20, 15 and 10 dB SNR	Number of cases with pink noise at 20, 15, and 10 dB SNR	Number of cases with brown noise at 0.25, -5.0, and -10 dB SNR	Class Identifier
Sags	50	50	50	0
Impulsive	50	50	50	1
Interruption	50	50	50	2
Swells	50	50	50	3
Oscillatory	50	50	50	4

After features of the 250 transients are ready, the next step is to train the multi-sigma probabilistic neural network (PNN) to get a set of optimal σ. The software implementation for training the PNN is accomplished in C++ language. It took about one quarter running the training program to complete the training for 60-sigma PNN consisting of 250 transients with Pentium (R) III of 600 M Hz. More detail about the training can be found in Section 4.3 and Appendix A for the neural network parameter setting. Since we

do not conduct an exhaustive search, no claim of absolute optimality will be made for these parameters. Generally, these parameters provide a suitable balance between speed, data reduction, and favorable classification results. In this thesis, classification performance is our most concern, so near optimal sigmas are trained. Finally, when the training is done, the PNN classifier is ready to classify transient signals.

As this level of training, the multi-sigma PNN can correctly classify 250/250 disturbances in the training set, using the holdout method. Using the same trained sigmas, the PNN classifier can classify 496/500 disturbances in the testing set. This means that the correct classification rate achieves 99%. Notice that we tried a single value of the scaling parameter σ of the PNN. Although the training speed is much faster than that using the multi-sigmas, the classification performance is poorer. Even for the training set, the correct classification rate is only around 90%.

We can describe our classification results of this experiment in terms of confusion matrix, which is a standard tool for testing any type of classifier. Table 6.3 shows the confusion matrix along with the classification results achieved from this experiment. Notice that the testing set consists of 500 transients without noise.

From the table, one can see that the confusion matrix has one row and one column for each class. The row represents the original class and the column means the predicted class by the PNN classification. The number in the matrix shows the various patterns of misclassification that are obtained from a testing set. For instance, the number in the Class 1 row and the Class 4 column is the number of cases that originally belong to Class 1 but

have been misclassified into Class 4 in the testing set. Ideally, the confusion matrix for a correct classification rate of 100% in a testing set would be a pure diagonal.

Table 6.3: Classification results of the PNN for pure data set.

Class	0	1	2	3	4
0	**100**				
1		**98**			2
2			**100**		
3				**100**	
4		2			**98**
Unknown					
Total	100	100	100	100	100
P_c	**1.0**	**.98**	**1.0**	**1.0**	**.98**

The classification results are perfect for class 0, 2, 3, and quite satisfactory for class 1, 4. The average classification rate (P_c) of 0.99 is achieved.

6.3.2 Training with Pure Data and Testing with Noisy Data

In this section, a noisy testing set is used to test this classification system. Notice that the PNN has been trained (i.e., optimal multiple sigmas have been obtained) based on the above-mentioned pure training set.

First, the experiments regarding the effect of white Gaussian noise on classification result were conducted. Figures 6.3 (a) to (b) show pure and contaminated oscillatory transients with additive white Gaussian noise at 20, 15, and 10 dB SNR, respectively. When SNR is 20 dB, the classification is still perfect, i.e., the confusion matrix is strictly diago-

nal. The confusion matrixes for classification results at 15 and 10 dB SNR are shown in Table 6.4 and 6.5, respectively.

Table 6.4 shows that the classification results are quite satisfactory with an average classification rate (P_c) of 96%. It is also observed that classification performance begins to degrade for class 1.

Table 6.4: Classification result of the PNN with white noise added at 15 dB SNR.

Class	0	1	2	3	4
0	**50**				
1		**49**			1
2			**50**		
3				**50**	
4		11			**39**
Unknown					
Total	50	60	50	50	40
P_c	**1.0**	**.82**	**1.0**	**1.0**	**.98**

Table 6.5: Classification result of the PNN with white noise added at 10 dB SNR.

Class	0	1	2	3	4
0	**50**				
1		**49**			1
2	4		**46**		
3				**50**	
4		27			**23**
Unknown					
Total	54	76	46	50	24
P_c	**.93**	**.64**	**1.0**	**1.0**	**.96**

Table 6.5 shows that an average classification rate (P_c) is 91%. One can see that classification performance is poor for class 1. It is found from the table that some of the transients of Class 4 are misclassified to Class 1. The reason for this is that the oscillatory transients corrupted by white Gaussian noise are easy to be confused with the impulsive transients, especially under heavy noise. By visualizing Fig. 6.4 (c), we think that misclassification is likely occurred. Notice that the confusion matrix is cluster and the performance of classification degrades with SNR decrease, which is reasonable. In addition, classification experiments for the transients with other recordings of white noise added were conducted and consistent results were given.

The experiments were repeated using the transients with pink noise added. Figures 6.5 (a) to (d) show pure and contaminated oscillatory transient with addictive pink noise added under the SNR of 20, 15, and 10 dB, respectively. Table 6.6, 6.7, and 6.8 show the confusion matrixes of classification results at 20, 15, and 10 dB SNR, respectively. Their corresponding classification rates are 99%, 96%, and 87%, respectively.

Table 6.6: Classification result of the PNN with pink noise added at 20 dB SNR

Class	0	1	2	3	4
0	**50**				
1		**48**			2
2			**50**		
3				**50**	
4		1			**49**
Unknown					
Total	50	49	50	50	51
P_c	**1.0**	**.98**	**1.0**	**1.0**	**.96**

Table 6.7: Classification result of the PNN with pink noise added at 15 dB SNR

Class	0	1	2	3	4
0	**50**				
1		**46**			4
2			**50**		
3				**49**	1
4		6			**44**
Unknown					
Total	50	52	50	49	49
P_c	**1.0**	**.88**	**1.0**	**1.0**	**.90**

Table 6.8: Classification result of the PNN with pink noise added at 10 dB SNR.

Class	0	1	2	3	4
0	**50**				
1		**44**			6
2	2		**48**		
3				**49**	1
4		27			**23**
Unknown					
Total	52	71	48	49	30
P_c	**.96**	**.62**	**1.0**	**1.0**	**.77**

The same experiments were repeated using the transients with brown noise added. Figures 6.6 (a) to (d) show pure and contaminated oscillatory transient with addictive brown noise added under 0.25, -5, and -10 dB SNR, respectively. Notice that with additive brown noise, even for a small SNR value, classification performance is still good. For

instance, when SNR is 0.25 dB, classification is still perfect and the confusion matrix is a strictly diagonal. The confusion matrixes of classification results are shown in Table 6.9 and 6.10 at -5 and -10 dB SNR, respectively. Their corresponding classification rates are 92% and 91%, respectively.

Table 6.9: Classification result of the PNN with brown noise added at -5 dB SNR.

Class	0	1	2	3	4
0	**50**				
1		**45**			5
2	7		**43**		
3				**49**	1
4		8			**42**
Unknown					
Total	57	53	43	49	48
P_c	**.88**	**.85**	**1.0**	**1.0**	**.88**

Table 6.10: Classification result of the PNN with brown noise added at -10 dB SNR.

Class	0	1	2	3	4
0	**50**				
1		**49**			1
2	4		**46**		
3				**50**	
4		27			**23**
Unknown					
Total	54	76	46	50	24
P_c	**.93**	**.64**	**1.0**	**1.0**	**.96**

Based on the above permutation experiments, one can see that this classification system is accurate when there is no noise existed and quite satisfactory in noisy environments. When noise is very heavy, classification performance becomes poor. In addition, this system is much more robust with additive brown noise than white Gaussian or pink noise. Notice that for testing the consistency of noise effect on the system, other recordings of noises were tried and the results were consistent.

6.3.3 Training and Testing PNN on Noisy Data

The above section deals with pure power transients as the training set. In this section, we will use noisy training set with additive white Gaussian noise of 15 dB SNR to train the multi-sigma PNN. Then, pure testing set and noisy testing sets with different noises are respectively classified for testing the PNN. Table 6.11 shows a confusion matrix with the classification result from one of permutation experiments, i.e., classification of the transients contaminated by brown noise at -10 dB SNR. The average classification rate is 92%.

Table 6.11: Classification result of the PNN with brown noise added at -10 dB SNR.

Class	0	1	2	3	4
0	**50**				
1		**49**			1
2	4		**46**		
3				**50**	
4		27			**23**
Unknown					
Total	54	76	46	50	24
P_c	**.93**	**.64**	**1.0**	**1.0**	**.96**

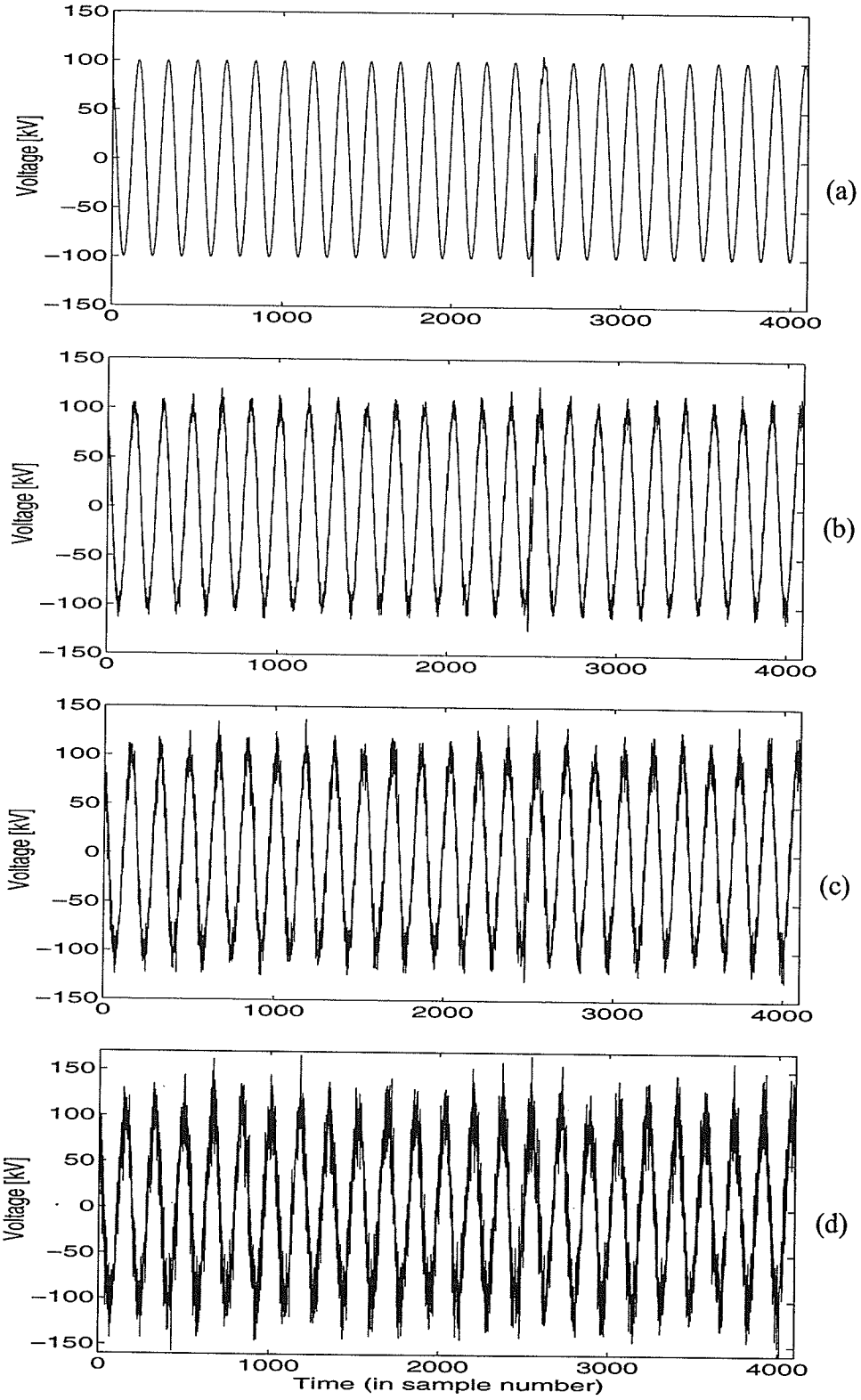

Fig. 6.4. An oscillatory transient with additive white Gaussian noise under SNR of: (a) 0 dB (pure), (b) 20 dB, (c) 15 dB, and (d) 10 dB

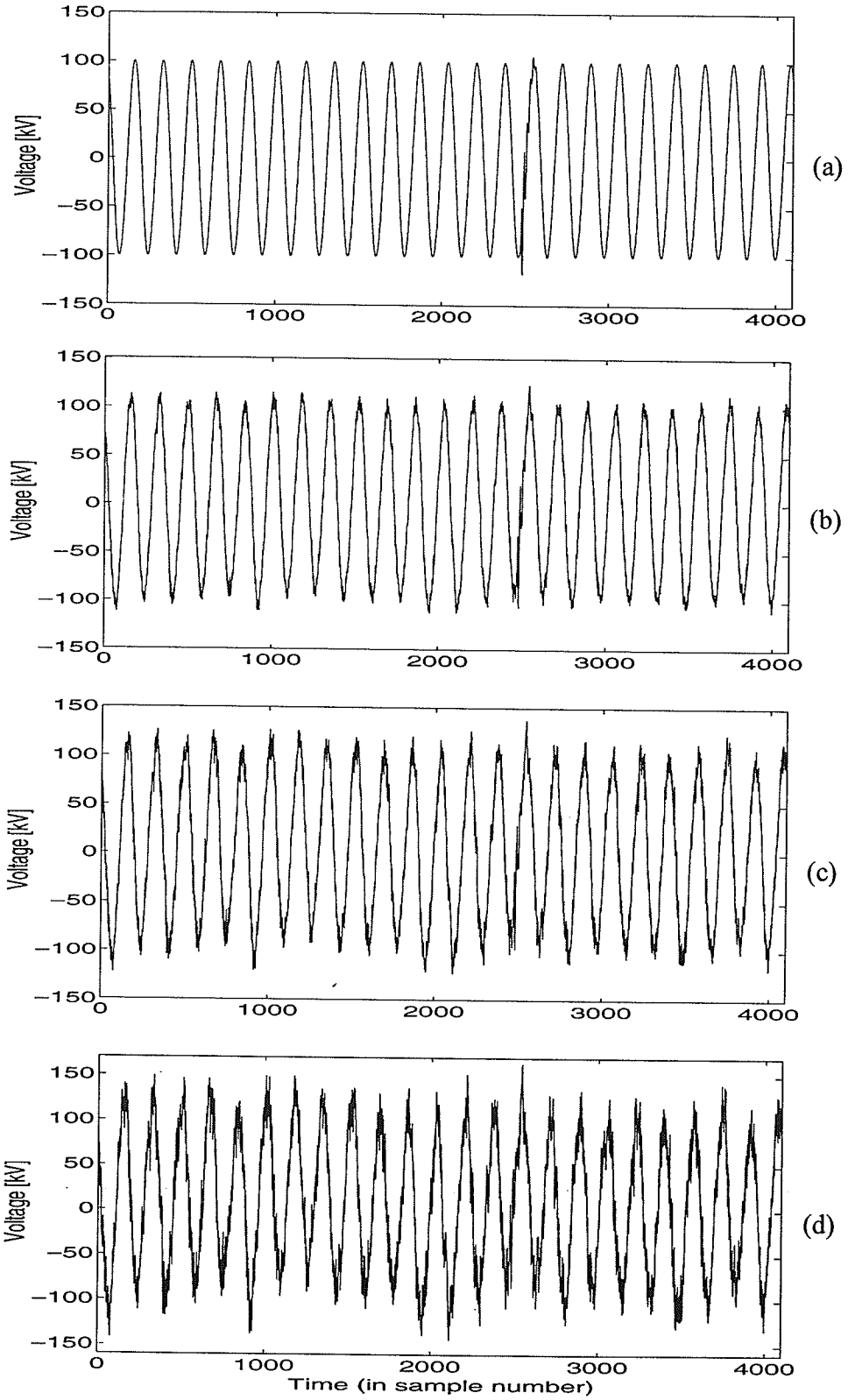

Fig. 6.5. An oscillatory transient with additive pink noise under SNR of: (a) 0 dB (pure), (b) 20 dB, (b) 15 dB, and (c) 10 dB

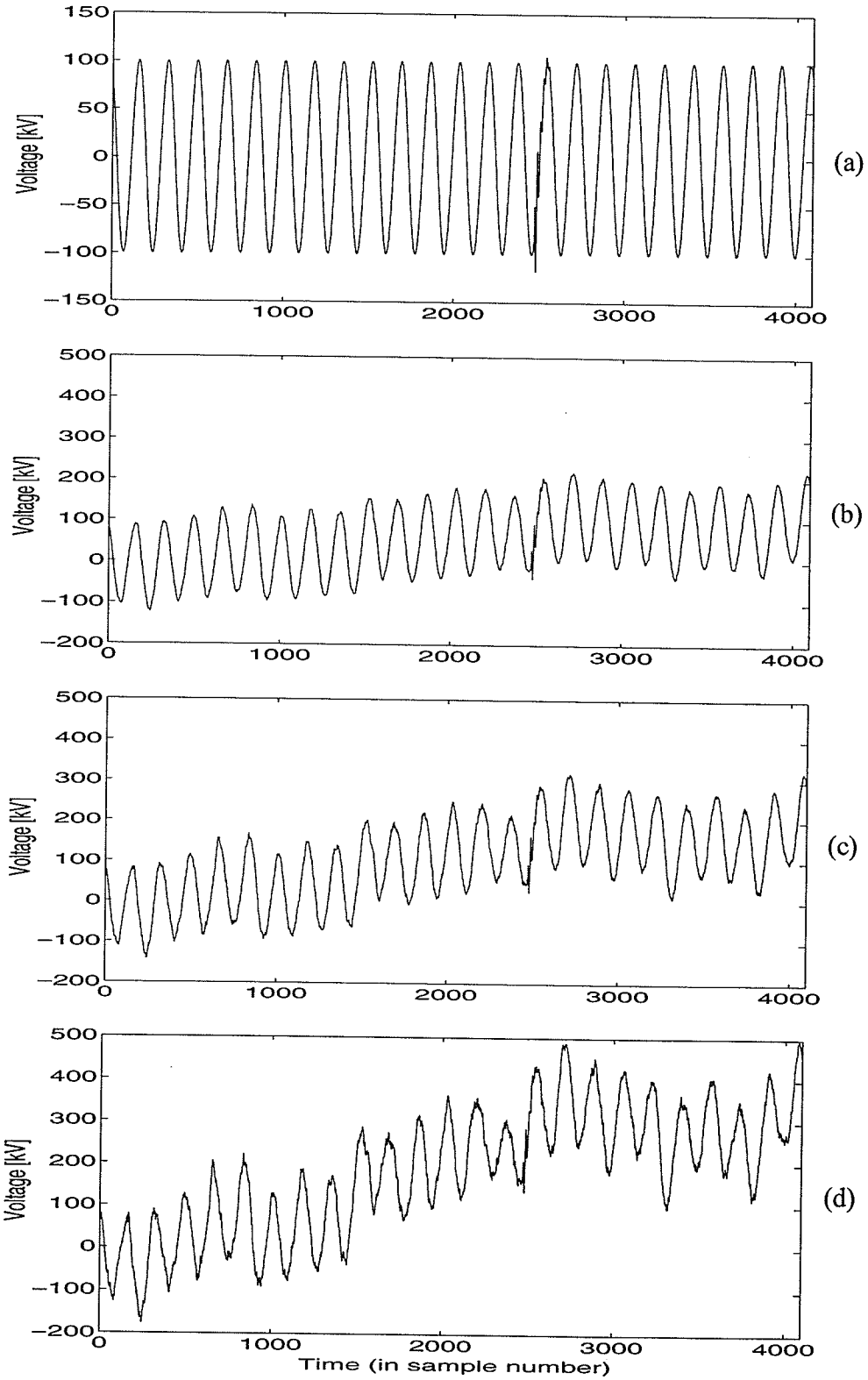

Fig. 6.6. An oscillatory transient with additive brown noise under SNR of: (a) 0 dB (pure), (b) 0.25 dB, (c) -5 dB, and (d) -10 dB.

6.4 Summary

The experimental results are presented in this chapter in a sequential order. First, power transient signals are summarized for test and evaluation of the proposed approaches. Then, wavelet transform and multifractal analysis are applied to extract combination features of transient signals, i.e, multiresolution time-frequency features and the VDFT-based features.

Finally, the multiple-sigma PNN is constructed for classification based on the extracted features. A confusion matrix is used to describe the classification results. For pure power transients, the average classification rate is 100% for training set and around 99% for testing set.

In practice, there is always noise due to complicated environments. Therefore, the effect of noise on this classification system is investigated in this thesis. Experiments show that the performance of this classification system is satisfactory when there is white, pink, or brown noise existed, respectively. It is found that the system is much more robust with additive brown noise than white Gaussian or pink noise. The performance in a pink noise environment is relatively poorest. However, even with pink noise at low SNR value such as 10 dB, the average classification can achieve 87%.

CHAPTER VII
CONCLUSIONS AND RECOMMENDATIONS

7.1 Conclusions

In this thesis, an accurate and reliable system for classification of power transients is developed. This system includes three major modules: transients simulation, feature extraction, and features classification.

Wavelet transform (WT) and multifractal analysis have been employed to model the transients for feature extraction from time-frequency multiresolution and fractal complexity points of view, respectively. A probabilistic neural network (PNN) is used as a classifier.

The discrete Daub4 wavelet transform has been shown the ability to characterize power transients in a compact representation. More specifically, it can: (i) detect and localize various transient disturbances; (ii) zoom the area of interest for better visualization of signal characteristics; (iii) provide both time and frequency information simultaneously; and (iv) easily extract the features from wavelet domain. In our case, the first five scales of wavelet decomposition are sufficient for analysis of power transients (based on the sampling rate, sampling points, and nature of power transient signals). Compared to the wavelet transform modulus maxima (WTMM) and the bin mean (BM) of wavelet coefficients approaches, the bin root mean square (BRMS) approach is the best for further extracting feature from a large number of wavelet coefficients in classification sense.

Multifractal analysis of transients is conducted in terms of variance fractal dimension trajectory (VFDT) approach. It is found that the window size of 256 and the displacement of 128 samples are suitable in computing the VFDT for the purpose of feature extraction of power transients. Our experiments show that multifractal analysis of transients based on the VFDT is efficient for characterizing most of transient types. As we have demonstrated, the fractal method can achieve the normalization of input data automatically due to theoretical lower and upper bounds of dimension value. In our case, the dimension values for power transients are in the range between 1.0 and 2.0. In addition, data reduction can be realized by simply changing the displacement of the sliding window.

As a result, 60 combination features from 4096 data points of the original input signal are extracted for classification, in which 30 features derived from wavelet analysis and the other 30 obtained from the variance fractal dimension trajectory (VFDT). Thus, a data reduction ratio of 68:1 is achieved to provide sufficient and compact representation of the transients.

Finally, the modelled power transient, i.e. sixty combination features, are presented to a neural network for classification. The PNN is found to be appropriate for our application due to its fast training and mathematically sound solutions. It has also been proved that the multi-sigma PNN achieves better performance than the one-sigma-fits-all PNN. For the PNN training, the conjugate gradient method is effective to find a set of optimal σ parameters needed for the PNN classifier.

Using the above mentioned approaches for feature extraction and classification, the experimental results show that this identification system is precise and fast in discrim-

inating the type of transient events with the classification rate of 100% using in-sample data set and around 99% using pure out-of-sample data set. Experiments also show that the classification system is robust in a noisy environment with the classification rate of around 91%, 87%, and 92% for the transients contaminated by additive white Gaussian, pink, and brown noises at 10, 10, and -5 dB SNR, respectively. In particular, this system is much more robust with additive brown noise than white Gaussian or pink noise. The average processing time for completing feature extraction and classification is about 2 second per transient with Pentium (R) III of 600 M Hz.

Overall, the experimental results have demonstrated that the objectives of this thesis are achieved successfully.

7.2 Contributions

This thesis has made the following contributions:

a) The BRMS approach has been successfully applied to power transients for further extracting features in wavelet time-scale domain;

b) Multifractal analysis on nonstationary signals is conducted in terms of the VFDT, which provides complexity degree information of the signals, directly used as features of the transients;

c) The multi-sigma PNN has been implemented for accurate classification of modelled power transients; and

d) The power transient classification system has been developed based on the combination features from wavelet transform and multifractal analysis.

7.3 Recommendations

To do further research on this topic, recommendations are suggested as follows:

a) Compare the Daubeches mother wavelets with other mother wavelets and choose optimal mother wavelet and wavelet parameters for this application based on classification performance;

b) Conduct more permutation experiments to investigate the effect of dimensional number on the PNN classification performance in order to find an optimal dimensional number, i.e., the number of features for this work;

c) Test a large database of transients and provide a suitable training set;

d) Work on real power transients instead of simulated ones; and

e) Develop a user friendly software package.

REFERENCES

[ADDT98] L. Angrisani, P. Daponte, M. D'Apuzzo, and A. Testa, "A measurement method based on the wavelet transform for power quality analysis," *IEEE Trans. on Power Delivery*, vol. 13, no. 4, pp. 990-998, April 1992.

[Caco66] T. Cacoullos, "Estimation of a multivariate density," *Annals of the Institute of Statistical Mathematics* (Tokyo), vol. 18, no. 2, pp. 179-189, 1966.

[Cant83] G. Cantor, "Über unendliche, lineare Punktmannigfaltigkeiten V," *Mathematische Annalen*, vol. 21, pp. 545-591, 1883.

[Chen97] H. Chen, *Accuracy of Fractal and Multifractal Measures for Signal Analysis*. M.Sc. Thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, Canada, 1997, 193 pp.

[ChKi00] J. Chen and W. Kinsner, "Multifractal analysis of transients in power systems," *Proc. of IEEE CCECE'00*, Halifax, Canada, pp. 307-311, May, 2000.

[CuDR99] N. Cumbria, M. DeRegt, and N. D. Rao, "Effects of power disturbances on sensitive loads," *Proc. IEEE 1999 Can. Conf. Electrical & Computer Engineering*, pp. 1181-1186, May 1999.

[Daub92] I. Daubechies, *Ten Lectures on Wavelets*. Philadephia, PA: Capital City, 1992, 376 pp.

[Deva92] Robert L. Devaney. *A first Course in Chaotic Dynamical Systems: Theory and Experiment*. Reading, MA: Addison-Wesley, 1992, 302 pp.

[HuHH99] S. J. Huang, C. T. Hsieh, and C. L. Huang, "Application of Morlet wavelet to supervise power system disturbances," *IEEE Trans. on Power Systems*, vol. 14, no. 1, pp. 235-241, Jan. 1999.

[HuKi01b] B. Huang and W. Kinsner, "New domain block partitioning based on complexity measure of ECG," Proc. *IEEE-EMBS Conf. on Biomedical Engineering*, Istanbul, Turkey, October 25-28, 2001.

[JBWCR99] F. Jiang, Z. Q. Bo, G. Weller, P. S. M. Chin, and M. A. Redfern, "A GPS based fault location scheme for distribution line using wavelet transform technique," *International Conf. on Power Systems Transients*, Budapest, Hungary, pp. 224-227, June 1999.

[KaMI00] M. Karimi, H. Mokhtari, and M. R. Iravani, "Wavelet based on-line disturbance detection for power quality applications," *IEEE Trans. on Power Systems*, vol. 15, no. 4, pp. 1212-1220, Oct. 2000.

[Kins94a] W. Kinsner, "Fractal dimensions: morphological, entropy, spectrum, and vari-

ance classes," *Technical Report, DEL 94-4*, Department of Electrical and Computer Engineering, University of Manitoba, May 1994, 146 pp.

[Kins94b] W. Kinsner, "Batch and real-time computation of a fractal dimension based on variance of a time series," *Technical Report, DEL 94-6*, Department of Electrical and Computer Engineering, University of Manitoba, May 1994, 22 pp.

[Kins95] W. Kinsner, "Self similarity: fractals, chaos, scaling and their applications," *Technical Report DEL 95-2*, Department of Electrical and Computer Engineering, University of Manitoba, Jan. 1995, 113 pp.

[LiMo99] T. B. Littler and D. J. Morrow, "Wavelets for the analysis and compression of power system disturbances," *IEEE Trans. on Power Delivery*, vol. 14, no. 2, pp. 358-364, April 1999.

[MaAB99] P L Mao, R K Aggarwal, and Z Q Bo, "An ANN based electromagnetic transients identification technique for power transformer systems," *IPST'99-International Conf. on Power Systems Transients*, Budapest, Hungary, pp. 207-212, June 1999.

[MaHw92] S. Mallar and W. L. Hwang, "Singularity detection and processing with wavelets," *IEEE Trans. Information Theory*, vol. 38, no. 2, pp. 617-641, March 1992.

[Mand82] B.B. Mandelbrot, *The Fractal Geometry of Nature*, New York, NY: W.H.Freeman, 1982, 465 pp.

[Mast93] T. Masters, *Practical Neural Network Recipes in C++*, San Diego, CA: Academic Press, 1993, 511 pp.

[Mast95] T. Masters, *Advanced Algorithms for Neural Networks: A C++ Sourcebook*, New York, NY: John Wiley & Sons, 1995, 448 pp.

[Mery91] Y. Meyer, "Un contre-exemple a la conjecture de Marr et a celle de S. Mallat," preprint, 1991.

[Moki97] F. Mo and W. Kinsner, "Wavelet modelling of transients in power systems," *Proc. IEEE 1997 Can. Conf. Communications, Power and Computing,* Winnipeg, Canada, May 1997.

[MoKi98] F. Mo and W. Kinsner, "Probabilistic neural networks for power line fault classification", *Proc. IEEE 1998 Can. Conf. Communications, Power and Computing,* pp. 585-588, *May* 1998.

[PaCo01] C. Parameswariah and M. Cox, "Frequency characteristics of wavelets," *Technical Report*, Department of Electrical Engineering, Louisiana Tech University, Ruston, USA, download from website, 19 pp.

[PaSa98] S. K. Pandey and L. Satish, "Multiresolution signal decomposition: a new tool for fault detection in power transformers during impulse tests," *IEEE Trans. on Power Delivery*, vol. 13, no. 4, pp. 1194-1200, Oct. 1998.

[PhMS00] F. Phan, E. Micheli-Tzanakou, and S. Sideman, "Speaker identification using neural networks and wavelets," *IEEE Engineering in Medical and Biology Magazine*, vol. 19, pp. 92-101, Jan./Feb. 2000.

[PiBh96] P. Pillay and A. Bhattacharjee, "Application of wavelets to model short-term power system disturbances," *IEEE Trans. on Power Systems*, vol. 11, no. 4, pp. 2031-2037, Nov. 1996.

[RCMG96]D. C. Robertson, O. I. Camps, J. S. Mayer, and W. B. Gish, "Wavelets and electromagnetic power system transients," *IEEE Trans. on Power Delivery*, vol. 12, no. 3, pp. 1250-1255, July 1997.

[SaPG97] S. Santoso, E. J. Powers, and W. M. Grady, "Power quality disturbance data compression using wavelet transform methods," *IEEE Trans. on Power Systems*, vol. 11, no. 4, pp. 2031-2037, Nov. 1996.

[Shaw97] D. B. Shaw, "Classification of transmitter transients using fractal measures and probabilistic neural networks," *M. Sc. Thesis*, Department of Electrical and Computer Engineering, University of Manitoba, 1997, 375 pp.

[Spec88] D. F. Specht, "Probabilistic neural networks for classification, mapping, or associative memory," *Proc. IEEE Int. Conf. Neural Networks,* vol. 1 (San Diego, CA), pp. 525-532, July 1988.

[Spec90a] D.F. Specht, "Probabilistic neural networks," *Neural Networks*, vol. 3, pp. 109-118, 1990.

[Spec90b] D.F. Specht, "Probabilistic neural networks and the polynomial adaline as complementary techniques for classification," *IEEE Trans. on Neural Networks*, vol. 1, no. 1, pp. 111-121, 1990.

[SPGH96] S. Santoso, E. J. Powers, W. M. Grady, and P. Hofmann, "Power quality assessment via wavelet transform analysis," *IEEE Trans. on Power Delivery*, vol. 11, no. 2, pp. 924-930, April 1996.

[Sun99] L. Sun, "A fast radio transmitter identification system," *M. Sc. Thesis*, Department of Electrical and Computer Engineering, University of Manitoba, 1999, 314 pp.

[Vics92] T. Vicsek, *Fractal Growth Phenomena*, Singapore: World Scientific, 1992 (2nd ed.), 507 pp.

APPENDIX A
PARAMETER SETTINGS FOR THE SYSTEM

Transient File Parameters

- Sampling Rate: 10,240 Hz
- Transient File Size: 46,862
- Transient Size: 4096
- Number of Transients Classes: 5
- Number of Transients: 3000

Parameters in Multifractal Analysis for Feature Extraction

- Window Size: 256
- Window Displacement: 128

Parameters in Wavelet Analysis for Feature Extraction

- Bin Number: 6
- Wavelet Levels: 5

Neural Network Training Parameters

- Sigma Threshold: $1.0e^{-10}$
- Error Threshold: $1.0e^{-10}$
- Maximum Iteration Number: 50

Neural Network Rejection Threshold: $1.0e^{-10}$

APPENDIX B
SOURCE CODE FOR THE SYSTEM

Table B-1: List of source files.

TransientGen.m

```
% Generate Voltage Sags (or Dips).
clear

% Initial parameters setting.
P=12;
datalength=2^P;
fs=10240;
AN=100;
PNO=50;
a=2*pi*60/fs;

% Enter the loop for generating the PNO files.
for patternno=1:PNO
    fai=360*rand(1,1);
    length_dip=800+(1500-800)*rand(1,1);
    i0=floor(100+(2000-100)*rand(1,1));
    i1=floor(i0+length_dip);
    AD=AN*(0.4+(0.7-0.4)*rand(1,1));

    clear y;
    clear name;
    for i=1:datalength
        y(i)=AN*cos(a*i+fai);
    end

    for i=i0:i1
        y(i)=AD*cos(a*i+fai);
    end

    name=int2str(patternno);
    name1=strcat('\cjin\simulation\dip\dip_4t',name);
    name1=strcat(name1, '.dat')
    fid =fopen (name1, 'wt');
    fprintf(fid,'%f\n', y);
    fclose(fid);

end
% End of loop

% Generate Impulsive Signal.
clear

% Initial parameters setting.
P=12;
datalength=2^P;
fs=10240;
AN=100;
PNO=50;
a=2*pi*60/fs;

% Enter the loop for generating the PNO files.
```

```
for patternno=1:PNO
    fai=360*rand(1,1);
    i0=floor(100+(3000-100)*rand(1,1));
    AD=AN*(1.3+(1.6-1.3)*rand(1,1));

    clear y;
    clear name;
    for i=1:datalength
        y(i)=AN*cos(a*i+fai);
    end

    if( y(i0)>=0 )
        y(i0)=y(i0)+AD;
    else
        y(i0)=y(i0)-AD;
    end

    name=int2str(patternno);
    name1=strcat('\cjin\simulation\impulsive\impul_4t',name);
    name1=strcat(name1, '.dat')
    fid =fopen (name1, 'wt');
    fprintf(fid,'%f\n', y);
    fclose(fid);

end
% End of loop

% Generate Voltage Interruption
clear

% Initial parameters setting.
P=12;
datalength=2^P;
fs=10240;
AN=100;
PNO=50;
a=2*pi*60/fs;

% Enter the loop for generating the PNO files.
for patternno=1:PNO
    fai=360*rand(1,1);
    length_in=800+(1500-800)*rand(1,1);
    i0=floor(100+(2000-100)*rand(1,1));
    i1=floor(i0+length_in);

    clear y;
    clear name;
    for i=1:datalength
        y(i)=AN*cos(a*i+fai);
    end

    for i=i0:i1
        y(i)=0;
    end
```

```
    name=int2str(patternno);
    name1=strcat('\cjin\simulation\interruption\inter_4t',name);
    name1=strcat(name1, '.dat')
    fid =fopen (name1, 'wt');
    fprintf(fid,'%f\n', y);
    fclose(fid);

end
% End of loop

% Generate Voltage Swells
clear

% Initial parameters setting.
P=12;
datalength=2^P;
fs=10240;
AN=100;
PNO=50;
a=2*pi*60/fs;

% Enter the loop for generating the PNO files.
for patternno=1:PNO
    fai=360*rand(1,1);
    length_sw=800+(1500-800)*rand(1,1);
    i0=floor(100+(2000-100)*rand(1,1));
    i1=floor(i0+length_sw);
    AD=AN*(1.3+(1.7-1.3)*rand(1,1));

    clear y;
    clear name;
    for i=1:datalength
        y(i)=AN*cos(a*i+fai);
    end

    for i=i0:i1
        y(i)=AD*cos(a*i+fai);
    end

    name=int2str(patternno);
    name1=strcat('\cjin\simulation\swell\sw_4t',name);
    name1=strcat(name1, '.dat')
    fid =fopen (name1, 'wt');
    fprintf(fid,'%f\n', y);
    fclose(fid);

end
% End of loop


% Generate Tranisent
clear

P=12;
```

```
datalength=2^P;
fs=10240;
AN=100;
PNO=50;
sa_t=1/fs;
r=0.7;
a=2*pi*60;

% Enter the loop for generating the PNO files.
for patternno=1:PNO
    fai=360*rand;
    AT=45+(50-45)*rand(1,1);
    B=350+(450-350)*rand(1,1);
    br=-B*r;
    i0=100+(3500-100)*rand(1,1);
    t0=i0*sa_t;
    os_T = 0.00160 + (0.001690-0.00160)*rand(1,1);
    ft=1/os_T; as=2*pi*ft;

    clear oy;
    clear sy;
    clear y;
    clear name;
    for i=1:datalength
        t=i*sa_t;
        sy(i)=AN*sin(a*t+fai);
        if(t<t0)
                oy(i)=0;
        else if( t>=t0 & t<=t0+os_T)
                oy(i)=AT*sin(as*(t-t0)+fai);
        else if(t>t0+os_T)
                oy(i)=AT*sin(as*(t-t0)+fai)*exp(br*(t-t0));
        end
        end
        end
        y(i)=sy(i)+oy(i);
    end

    name=int2str(patternno);
    name1=strcat('\cjin\simulation\transient\tran_4t',name);
    name1=strcat(name1, '.dat')
    fid =fopen (name1, 'wt');
    fprintf(fid,'%f\n', y);
    fclose(fid);

end
% End of loop
```

FeatureExtr.m

```
clear;

% fin1 =fopen ('\cjin\simulation\noise\white\white1.dat', 'r');
fin1 =fopen ('\cjin\simulation\noise\pink\pink1.dat', 'r');
% fin1 =fopen ('\cjin\simulation\noise\brown\brown1.dat', 'r');

[w, count1] = fscanf(fin1,'%f\n');
count1
fclose(fin1);

Energy_noise=0;
for i=1:count1
    Energy_noise = Energy_noise + w(i)*w(i);
end

SNR=10;
SNR=10^(SNR/10);

% Define the output file name for storing the wavelet features.
waveFeatureName = '\cjin\feature_mean1\t5nn5d_rms_1t.doc';

% Define the output file name for storing the fractal features.
fractalFeatureName = '\cjin\feature_vdf1\t5nn_vdf_1t.doc';

fido2 =fopen (fractalFeatureName, 'a');
fido =fopen (waveFeatureName, 'a');

% Input transient signals being analyzed.
inputNamePrefix = '\cjin\simulation\dip\dip_1t';

fun_wavelet_rms(inputNamePrefix, w, SNR, Energy_noise, fido);
fun_fractal_vfdt(inputNamePrefix, w, SNR, Energy_noise, fido2);

fclose(fido);
fclose(fido2);

Class=5;
PatternNo=50;
fid1 = fopen (fractalFeatureName, 'r');
[x1, count1] = fscanf(fid1,'%f\n');
fclose(fid1);
d1=count1/(Class*PatternNo);

fid2 = fopen (waveFeatureName, 'r');
[x2, count2] = fscanf(fid2,'%f\n');
ave_x2=(abs(max(x2))+abs(min(x2)))/2;
fclose(fid2);
d2=count2/(Class*PatternNo);

d=d1+d2;
j1=1;
```

```
j2=1;
TotalLength=d1+d2;

for ClassNumber=1:Class
    for PatternNumber=1:PatternNo
        j3=(ClassNumber-1)*PatternNo*d+(PatternNumber-1)*d;
        for i=1:d1
            feature(j3+i)=x1(j1)*ave_x2/2;  j1=j1+1;
        end
        for i=d1+1:d1+d2
            eature(j3+i)=x2(j2);  j2=j2+1;
        end
    end
end

size(feature);
figure(3);
plot(feature);

% Define the output file name for combination features.
name3 = '\cjin\feature_combination1\t5nn5d_rms_vdf_1t.doc';
fido3 =fopen (name3, 'w');
fprintf(fido3,'%f\n', feature);
fclose(fido3);

% End
```

fun_wavelet_rms.m

```
function fun_wavelet_rms(inputNamePrefix, w, SNR, Energy_noise, fido);

C=[0.4829629131445341, 0.8365163037378079, 0.2241438680420134, -0.1294095225512604];
patternNo = 50;
Max_scale=5;
Scale_Dimension = 6;
Miss = 2;

for patternno=1:patternNo
clear clear XX; clear X;
    clear X_l; clear X_h;
    clear X_ll; clear X_lh;
    clear X_lll; clear X_llh;
    clear X_llll; clear X_lllh;
    clear X_lllll; clear X_llllh;
    clear d1; clear d2; clear d3; clear d4; clear d5; clear features;

    [XX, count] = fun_inputthispattern(patternno, inputNamePrefix);
    Energy_signal=0;
    for i=1:count
        Energy_signal = Energy_signal + XX(i)*XX(i);
    end
    XX = XX + (sqrt(Energy_signal/(Energy_noise*SNR)))*w;
    X=XX';

    [X_l,X_h]=wavelet_transform(C,X);
    [X_ll,X_lh]=wavelet_transform(C,X_l);
    [X_lll,X_llh]=wavelet_transform(C,X_ll);
    [X_llll,X_lllh]=wavelet_transform(C,X_lll);
    [X_lllll,X_llllh]=wavelet_transform(C,X_llll);

    [d1,d2,d3,d4,d5]=fun_findmean(X_h,X_lh,X_llh,X_lllh,X_llllh,X_lllll,Scale_Dimension,Miss);
    features([1:Scale_Dimension])=d1;
    features([1*Scale_Dimension+1:2*Scale_Dimension])=d2;
    features([2*Scale_Dimension+1:3*Scale_Dimension])=d3;
    features([3*Scale_Dimension+1:4*Scale_Dimension])=d4;
    features([4*Scale_Dimension+1:5*Scale_Dimension])=d5;
    features = features/(max(features)-min(features));
    fprintf(fido,'%f\n', features);

end
```

fun_inputthispattern.m

```
function [y, count]=fun_inputthispattern(patternno, inputNamePrefix)
name=int2str(patternno);
name=strcat(inputNamePrefix, name);
name=strcat(name, '.dat')
fid = fopen (name, 'r');
[y, count] = fscanf(fid,'%f\n');
plot(y);
fclose(fid);
```

wavelet_transform.m

```
function [c11,d11]=wavelet_transform(C,x)
length=size(x);
datalength=length(1,2);

i=1;
for  n=1:2:(datalength-3)
     n1=n+1;n2=n+2;n3=n+3;
     c11(i)=C(1)*x(n)+C(2)*x(n1)+C(3)*x(n2)+C(4)*x(n3);
     d11(i)=C(4)*x(n)-C(3)*x(n1)+C(2)*x(n2)-C(1)*x(n3);
     i=i+1;
end
     c11(i)=C(1)*x(datalength-1)+C(2)*x(datalength)+C(3)*x(1)+C(4)*x(2);
     d11(i)=C(4)*x(datalength-1)-C(3)*x(datalength)+C(2)*x(1)-C(1)*x(2);

% End
```

fun_findmean.m

```
function [Features_d1,Features_d2,Features_d3,Features_d4,Features_d5,Features_c5]=fun_findmean(d1, d2, d3, d4, d5, c5, Scale_Dimension, Miss);

clear Features_d1; clear Features_d2; clear Features_d3; clear Features_d4; clear Features_d5;

N1=size(d1); length_s1=N1(1,2)-Miss; BinNo_s1 = floor(length_s1/Scale_Dimension);
[Features_d1] = fun_submean(Scale_Dimension, BinNo_s1, d1);
N2=size(d2); length_s2=N2(1,2)-Miss; BinNo_s2 = floor(length_s2/Scale_Dimension);
[Features_d2] = fun_submean(Scale_Dimension, BinNo_s2, d2);
N3=size(d3); length_s3=N3(1,2)-Miss; BinNo_s3 = floor(length_s3/Scale_Dimension);
[Features_d3] = fun_submean(Scale_Dimension, BinNo_s3, d3);
N4=size(d4); length_s4=N4(1,2)-Miss; BinNo_s4 = floor(length_s4/Scale_Dimension);
[Features_d4] = fun_submean(Scale_Dimension, BinNo_s4, d4);
N5=size(d5); length_s5=N5(1,2)-Miss; BinNo_s5 = floor(length_s5/Scale_Dimension);
[Features_d5] = fun_submean(Scale_Dimension, BinNo_s5, d5);
```

fun_submean.m

```
function [Feature_d] = fun_submean(Scale_Dimension, BinNo_s, d)
clear Feature_d;

for j=1:Scale_Dimension
    j;
    s=0;
    for k=(j-1)*BinNo_s+1:j*BinNo_s
        s=s+d(k)*d(k);
    end
    Feature_d(j)=sqrt(s/BinNo_s);
    %Feature_d(j)=mean( d([(j-1)*BinNo_s+1:j*BinNo_s]) );
end
```

function fun_fractal_vfdt.m

```
function fun_fractal_vfdt(inputNamePrefix, w, SNR, Energy_noise, fido);

max_2_times = 12;
NT = 2.^max_2_times;
K1=1;
K2=6;
K_total=K2-K1+1;
E=1;
window_with=256;
shift_window=128;

last_location=(NT-window_with);
dimension_number=last_location/shift_window

patternNo=50;
for  patternno=1:patternNo
     clear varB3; clear X; clear Y;
     clear x; clear D;

     [x, count] = fun_inputthispattern(patternno, inputNamePrefix);

     Energy_signal=0;
     for i=1:count
          Energy_signal = Energy_signal + x(i)*x(i);
     end

     % Superpose the noise on each signal. If no noise, use '%' before this statement.
     x = x + (sqrt(Energy_signal/(Energy_noise*SNR)))*w;

     ll=1;
     max_varB3=-1000000;
     for L=1:shift_window:last_location
          for k=K1:K2
               sum1=0;
               sum2=0;
               nk=2.^k;
               Nk=window_with-nk;
               for n=1:Nk
                    daltB=x(L+n+nk)-x(L+n);
                    sum1=sum1+daltB*daltB;
                    sum2=sum2+daltB;
               end
               varB3(k)=(sum1-(sum2*sum2)/Nk);
               varB3(k)=varB3(k)/(Nk-1);
               if( varB3(k)> max_varB3 );
                    max_varB3;
               end
               if( varB3(k)<0.0001 )
                    varB3(k)= max_varB3;
               else
                    varB3(k)=log2(varB3(k));
```

```
            end
            X(k)=k;
            Y(k)=varB3(k);
        end

        s1=0;s2=0;s3=0;s4=0;
        for i=K1:K2
            s1=s1+X(i)*Y(i);
            s2=s2+X(i);
            s3=s3+Y(i);
            s4=s4+X(i)*X(i);
        end

        s=(K_total*s1-s2*s3)/(K_total*s4-s2*s2);
        H=s/2;
        D(ll)=E+1-H;
        ll=ll+1;

    end
    fprintf(fido,'%f\n', D);

end
```

TrainMain.cpp

```
#include <fstream.h>
#include <stdlib.h>

extern double adpnnmain(int StepNumber,
                        int Feature_Dimension,
                        int Class_Number,
                        int Pattern_Number,
                        int *CLass,
                        double *Sigma,
                        int &minErrorScore);
int main()
{

    int i, tempint;
    int Feature_Dimension, Class_Number, Pattern_Number, k, StepNumber;
    double *Sigma, *minSigma;
    int *Class;
    double Error, minError;
    int minErrorScore, lastminErrorScore, bestStepNumber;
    int s[] = {10, 13, 15, 17, 20, 26};

    char filenameGeneralInfo[100]="c:\\cjin\\ne\\adpnn\\1\\g5t_250cases_d60.dat"; // t=5, d=60, each=50
    ifstream fcin1(filenameGeneralInfo, ios::nocreate);
    if(!fcin1){cerr<<"cannot open file1"<<endl; exit(1);}

    char trainingSigma[100]="c:\\cjin\\feature_sigma1\\Sigma_rms_vdf_1t.dat";
    ofstream fout;
    fout.open(trainingSigma);
    fout.setf(ios::fixed);
    fout.setf(ios::showpoint);
    fout.precision(9);

    fcin1>>Feature_Dimension;
    fcin1>>Class_Number;
    cout<<"Feature_Dimension = "<<Feature_Dimension<<", Class_Number = "<<Class_Number<<endl;

    Class = new int[Class_Number];
    if(Class == NULL){cout<<"No enough memory!"; exit(1);}

    Pattern_Number = 0;
    for(i=0; i<Class_Number; i++){
        fcin1>>Class[i];
        cout<<"Class["<<i<<"] = "<<Class[i]<<endl;
        Pattern_Number += Class[i];}

    Sigma = new double[Feature_Dimension];
    if(Sigma == NULL){cout<<"No enough memory!"; exit(1);}
    minSigma = new double[Feature_Dimension];
    if(minSigma == NULL){cout<<"No enough memory!"; exit(1);}
    minError=1.e04;
    lastminErrorScore =0;
```

```
    bestStepNumber=s[1];
    for(i=0; i<5; i++){
        minErrorScore = 0;
        StepNumber = s[i];
        cout<<endl<<"StepNumber = "<<s[i]<<endl;
        for(k=0; k<Feature_Dimension; k++) Sigma[k]=0.0;
        Error = adpnnmain(StepNumber, Feature_Dimension, Class_Number,
            Pattern_Number, Class, Sigma, minErrorScore);

        if(Error<minError){
            minError=Error;
            lastminErrorScore=minErrorScore;
            for(k=0; k<Feature_Dimension; k++) minSigma[k] = Sigma[k];
            bestStepNumber=s[i];}

    }// end for

    cout<<endl<<"------ final result------"<<endl;
    cout<<endl<<"minError="<<minError<<", minErrorScore="<<lastminErrorScore<<",
    "<<lastminErrorScore*100/Pattern_Number<<"%"<<", bestStep="<<bestStepNumber<<endl<<endl;
    for(k=0; k<Feature_Dimension; k++)
        cout<<minSigma[k]<<"";
    cout<<endl<<endl;

    for(k=0;k<Feature_Dimension;k++)
        fout<<minSigma[k]<<endl;

    delete [] minSigma;
    delete [] Sigma;
    delete [] Class;

    return 1;
}
```

adpnnmain.cpp

```
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#include "adpnnmain.h"

int  PrintControl;

double adpnnmain(int StepNumber,
                 int Feature_Dimension,
                 int Class_Number,
                 int Pattern_Number,
                 int *Class,
                 double *Sigma,
                 int &minErrorScore)
{
    int *index;
    double **x;
    int i, k, c, tempint, MaxIterationNo;
    double tempdouble;
    double Threshold1, Threshold2, low_end, high_end, Step;
    double *ibest, *ibestCopy, *Sigma0, *Sigma1;
    double Previous, Error, Error0, t, Gamma, numinator, denum;
    double high, dot1, dot2, scale, mean_Sigma1;
    double *First_Error, *Second_Error, *g0, *g1, *d0, *d1;
    int Iterate, Score, maxScore;
    //int minErrorScore;
    double *Constt, *maxScoreSigma, *minErrorSigma;

    double Refine_t, maxScoreError, minError;
    int New_t, StepNumber1;

    char filenameTrainningData[100]="c:\\cjin\\feature_combination1\\t5nn5d_rms_vdf_1t.doc";
    ifstream fcin2(filenameTrainningData, ios::nocreate);
    if(!fcin2){cerr<<"cannot open file2"<<endl; exit(1);}

    index = new int[Pattern_Number];
    if(index == NULL){cout<<"No enough memory!"; exit(1);}
    i=0;
    for(c=0; c<Class_Number; c++){
        for(k=0; k<Class[c]; k++){
            index[i] = c;
            i++;}}

    x = new double *[Pattern_Number];
    if(x == NULL){cout<<"No enough memory!"; exit(1);}
    for(i=0; i<Pattern_Number; i++){
        x[i] = new double[Feature_Dimension];
        if(x[i]== NULL){cout<<"No enough memory!"; exit(1);
    }}
    for(i=0; i<Pattern_Number; i++){
        for(k=0; k<Feature_Dimension; k++){
```

```
            fcin2>>x[i][k];
}}

Constt = new double[Feature_Dimension];
if(Constt == NULL){cout<<"No enough memory!"; exit(1);}

ibest = new double[Feature_Dimension];
if(ibest == NULL){cout<<"No enough memory!"; exit(1);}
ibestCopy = new double[Feature_Dimension];
if(ibestCopy == NULL){cout<<"No enough memory!"; exit(1);}

maxScoreSigma = new double[Feature_Dimension];
if(maxScoreSigma == NULL){cout<<"No enough memory!"; exit(1);}
Sigma0 = new double[Feature_Dimension];
if(Sigma0 == NULL){cout<<"No enough memory!"; exit(1);}
Sigma1 = new double[Feature_Dimension];
if(Sigma1 == NULL){cout<<"No enough memory!"; exit(1);}
minErrorSigma = new double[Feature_Dimension];
if(minErrorSigma == NULL){cout<<"No enough memory!"; exit(1);}

First_Error = new double[Feature_Dimension];
if(First_Error== NULL){cout<<"No enough memory!"; exit(1);}
Second_Error = new double[Feature_Dimension];
if(Second_Error== NULL){cout<<"No enough memory!"; exit(1);}

g0 = new double[Feature_Dimension];
if(g0== NULL){cout<<"No enough memory!"; exit(1);}
g1 = new double[Feature_Dimension];
if(g1== NULL){cout<<"No enough memory!"; exit(1);}
d0 = new double[Feature_Dimension];
if(d0 == NULL){cout<<"No enough memory!"; exit(1);}
d1 = new double[Feature_Dimension];
if(d1 == NULL){cout<<"No enough memory!"; exit(1);}

ifstream fcin3("initialP1.dat", ios::nocreate);
if(!fcin3){cerr<<"cannot open file3 in adpnnmain.cpp"<<endl; exit(1);}
fcin3>>Threshold1;
fcin3>>Threshold2;
fcin3>>MaxIterationNo;
fcin3>>low_end;
fcin3>>high_end;

Iterate = 1;
for(k=0; k<Feature_Dimension; k++)
    Constt[k]=0.0;

for(k=0; k<Feature_Dimension; k++){
    ibest[k] = Constt[k]+low_end;
    ibestCopy[k] = ibest[k];}
Step = exp(log(high_end/low_end)/(StepNumber-1));
fun_error(Feature_Dimension, Class_Number, Pattern_Number, Class,
    index, x, ibestCopy, Previous, Score);
cout<<"Previous = "<<Previous<<", Score = "<<Score<<endl;
for(k=0; k<Feature_Dimension; k++){
```

```
        Sigma[k] = ibest[k];
}

for(i=1; i<StepNumber; i++){
        for(k=0; k<Feature_Dimension; k++)
                Sigma[k] *= Step;
        fun_error(Feature_Dimension, Class_Number, Pattern_Number, Class,
                index, x, Sigma, Error, Score);
        if(Error<Previous){
                for(k=0; k<Feature_Dimension; k++)
                        ibest[k] = Sigma[k];
                Previous = Error;
        }
}

for(k=0; k<Feature_Dimension; k++){
        Sigma[k] = ibest[k];
        Sigma0[k] = Sigma[k];}

fun_derivative(Feature_Dimension, Class_Number, Pattern_Number, Class,
        index, x, Sigma, Error0, First_Error, Second_Error, Score);

for(k=0; k<Feature_Dimension; k++){
        g0[k] = -First_Error[k];
        d0[k] = -First_Error[k];
        d1[k] = d0[k];}

tempdouble = -100000.0;
for(k=0; k<Feature_Dimension; k++)
        if(Second_Error[k]>tempdouble) tempdouble = Second_Error[k];
t = 1.5/tempdouble;
cout<<"t = "<<t<<endl;

maxScore=0;
minError=1.0e04;
StepNumber1 = 20;
for(k=0; k<Feature_Dimension; k++) maxScoreSigma[k] = 0.0;
PrintControl=0;
for(i=1; i<=MaxIterationNo; i++){
        New_t = fun_global(Feature_Dimension, Class_Number, Pattern_Number, Class,
                index, x, Sigma, d1, t, StepNumber1);

        Refine_t = fun_golden_section(Feature_Dimension, Class_Number, Pattern_Number, Class,
                index, x, Sigma, d1, t, StepNumber1, New_t);

        for(k=0; k<Feature_Dimension; k++) Sigma[k] += Refine_t*d0[k];
        fun_derivative(Feature_Dimension, Class_Number, Pattern_Number, Class,
                index, x, Sigma, Error, First_Error, Second_Error, Score);

        for(k=0; k<Feature_Dimension; k++) g1[k] = -First_Error[k];

        numinator=0.0; denum=0.0;
        for (k=0; k<Feature_Dimension; k++){
                numinator += (g1[k]-g0[k])*g1[k]; denum += g0[k]*g0[k];}
```

```
Gamma = numinator/denum;

for(k=0; k<Feature_Dimension; k++) d1[k] = g1[k]+Gamma*d0[k];
for(k=0; k<Feature_Dimension; k++){
     g0[k]=g1[k];
     d0[k]=d1[k];}

dot1=dot2=0.0;
high = 1.e-4;
for(k=0; k<Feature_Dimension; k++){
     //cout<<""<<Second_Error[k];// test
     if(Second_Error[k]>high)
          high = Second_Error[k];
     dot1 += g1[k]*d1[k];
     dot2 += d1[k]*(Second_Error[k])*d1[k];
}

scale=dot1/dot2;
high=1.5/high;
if(high<1.e-4)
     high=1.e-4;

if(scale<0.0)
     scale=high;
else if(scale<0.1*high)
     scale=0.1*high;
else if(scale>10.0*high)
     scale=10.0*high;

t=2.0*scale;

for(k=0; k<Feature_Dimension; k++) Sigma1[k]=Sigma[k]-Sigma0[k];
Iterate++;

mean_Sigma1=0.0;
for(k=0; k<Feature_Dimension; k++) mean_Sigma1 += fabs(Sigma1[k]);
mean_Sigma1 /= Feature_Dimension;

cout<<"iter="<<i<<" t="<<t<<" Nt="<<New_t<<" Rt="<<Refine_t;
cout<<"  Error="<<Error<<"Score="<<Score<<endl;

if((mean_Sigma1>Threshold1) || (fabs(Error-Error0)>Threshold2)){
     for(k=0; k<Feature_Dimension; k++){
          Sigma0[k]=Sigma[k];
          Error0=Error;}
}
else{
     break;
}

if(maxScore<Score){
     maxScore = Score;
     maxScoreError = Error;
     for(k=0; k<Feature_Dimension; k++) maxScoreSigma[k] = Sigma[k];}
```

```
            if(Error<minError){
                minError = Error;
                minErrorScore = Score;
                for(k=0; k<Feature_Dimension; k++) minErrorSigma[k]=Sigma[k];}

        }

        delete [] index;
        delete [] x;
        delete [] Sigma0;
        delete [] Sigma1;
        delete [] minErrorSigma;
        delete [] maxScoreSigma;
        delete [] ibest;
        delete [] ibestCopy,
        delete [] Constt;
        delete [] First_Error;
        delete [] Second_Error;
        delete [] g0;
        delete [] g1;
        delete [] d0;
        delete [] d1;

        return minError;

} // end main
```

adpnnmain.h

```
extern void fun_error(
    int Feature_Dimension,
    int Class_Number,
    int Pattern_Number,
    int *Class,
    int *index,
    double **x,
    double *Sigma,
    double &Error,
    int &Score);

extern void fun_derivative(
    int Feature_Dimension,
    int Class_Number,
    int Pattern_Number,
    int *Class,
    int *index,
    double **x,
    double *Sigma,
    double &Error,
    double *First_Error,
    double *Second_Error,
    int &Score);

extern int fun_global(
    int Feature_Dimension,
    int Class_Number,
    int Pattern_Number,
    int *Class,
    int *index,
    double **x,
    double *Sigma,
    double *d,
    double t,
    int StepNumber);

extern double fun_golden_section(
    int Feature_Dimension,
    int Class_Number,
    int Pattern_Number,
    int *Class,
    int *index,
    double **x,
    double *Sigma,
    double *d1,
    double t,
    int StepNumber,
    int New_t);
```

fun_derivative.cpp

```
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

void fun_derivative(
     int Feature_Dimension,
     int Class_Number,
     int Pattern_Number,
     int *Class,
     int *index,
     double **x,
     double *Sigma,
     double &Error,
     double *First_Error,
     double *Second_Error,
     int &Score)
{

int i, k, j, c, index_j, tempint;
double *u, *q, *SigmaCopy, *dsqr;
double **First_u, **Second_u, **First_q, **Second_q;
double Scaled_Distance, Difference, s;
double *First_s, *Second_s;
double Max_value, tempdouble;
int Max_position;

Error=0.0; Score=0;

u = new double[Class_Number];
if(u == NULL){cout<<"No enough memory!"; exit(1);}
q = new double[Class_Number];
if(q == NULL){cout<<"No enough memory!"; exit(1);}

First_u = new double *[Class_Number];
Second_u = new double *[Class_Number];
if(First_u == NULL){cout<<"No enough memory!"; exit(1);}
if(Second_u == NULL){cout<<"No enough memory!"; exit(1);}
for(i=0; i<Class_Number; i++){
     First_u[i] = new double[Feature_Dimension];
     if(First_u[i]== NULL){cout<<"No enough memory!"; exit(1);
}}
for(i=0; i<Class_Number; i++){
     Second_u[i] = new double[Feature_Dimension];
     if(Second_u[i]== NULL){cout<<"No enough memory!"; exit(1);
}}

First_q = new double *[Class_Number];
Second_q = new double *[Class_Number];
if(First_q == NULL){cout<<"No enough memory!"; exit(1);}
if(Second_q == NULL){cout<<"No enough memory!"; exit(1);}
for(i=0; i<Class_Number; i++){
```

```
        First_q[i] = new double[Feature_Dimension];
        if(First_q[i]== NULL){cout<<"No enough memory!"; exit(1);
}}
for(i=0; i<Class_Number; i++){
        Second_q[i] = new double[Feature_Dimension];
        if(Second_q[i]== NULL){cout<<"No enough memory!"; exit(1);
}}

First_s = new double[Feature_Dimension];
if(First_s == NULL){cout<<"No enough memory!"; exit(1);}
Second_s = new double[Feature_Dimension];
if(Second_s == NULL){cout<<"No enough memory!"; exit(1);}

dsqr = new double[Feature_Dimension];
if(dsqr == NULL){cout<<"No enough memory!"; exit(1);}

SigmaCopy = new double[Feature_Dimension];
if(SigmaCopy == NULL){cout<<"No enough memory!"; exit(1);}

for(k=0; k<Feature_Dimension; k++) SigmaCopy[k] = Sigma[k];

for(k=0; k<Feature_Dimension; k++){
        First_Error[k] = 0.0;
        Second_Error[k] =0.0;}

for(i=0; i<Pattern_Number; i++){
        s = 0.0;
        for(c=0; c<Class_Number; c++)
                u[c] = 0.0;
        for(c=0; c<Class_Number; c++){
                for(k=0; k<Feature_Dimension; k++){
                        First_u[c][k] = 0.0;
                        Second_u[c][k] = 0.0;}}

        for(j=0; j<Pattern_Number; j++){

                if(i!=j){
                        index_j=index[j];
                        Scaled_Distance=0.0;
                        for(k=0; k<Feature_Dimension; k++){
                                Difference = (x[i][k]-x[j][k])/SigmaCopy[k];
                                dsqr[k] = Difference*Difference;
                                Scaled_Distance += dsqr[k];}

                        Scaled_Distance = exp(-Scaled_Distance);
                        u[index_j] += Scaled_Distance;

                        for(k=0; k<Feature_Dimension; k++){
                                tempdouble = Scaled_Distance * dsqr[k];
                           First_u[index_j][k] += tempdouble*2.0/SigmaCopy[k];
                                Second_u[index_j][k] +=
                                        tempdouble*(2.0*dsqr[k]-3.0)*2.0/(SigmaCopy[k]*SigmaCopy[k]);
                        }
```

```
    }
}

for(c=0; c<Class_Number; c++){
    u[c] /= Class[c];
    for(k=0; k<Feature_Dimension; k++){
        First_u[c][k] /= Class[c];
        Second_u[c][k] /= Class[c];}}

s=0.0;
for(c=0; c<Class_Number; c++)
    s += u[c];

for(k=0; k<Feature_Dimension; k++){
    First_s[k]=0.0;
    for(c=0; c<Class_Number; c++){
        First_s[k] += First_u[c][k];}}

for(k=0; k<Feature_Dimension; k++){
  Second_s[k]=0.0;
  for(c=0; c<Class_Number; c++){
      Second_s[k] += Second_u[c][k];}}

if (s<1.e-40)
    s = 1.e-40;

for(c=0; c<Class_Number; c++)
    q[c] = u[c]/s;

Max_value = q[0]; Max_position = 0;
for(c=1; c<Class_Number; c++){
    if (q[c]>Max_value){
        Max_value = q[c]; Max_position = c;}}

if(Max_position == index[i])
  ++Score;
for(c=0; c<Class_Number; c++){
    for  (k=0; k<Feature_Dimension; k++){
        First_q[c][k] = (First_u[c][k]-q[c]*First_s[k])/s;
        Second_q[c][k]=(2.0/s)*(First_s[k]*(q[c]*First_s[k]-First_u[c][k]));
        Second_q[c][k]=(Second_u[c][k]-q[c]*Second_s[k]+Second_q[c][k])/s;}}

for(c=0; c<Class_Number; c++){
    if(c != index[i])
        Error += q[c]*q[c];
    else
        Error += (1-q[c])*(1-q[c]);}

for  (k=0; k<Feature_Dimension; k++){
for(c=0; c<Class_Number; c++){
 if(c != index[i])
   First_Error[k] += q[c]*First_q[c][k];
 else
        First_Error[k] += (q[c]-1)*First_q[c][k];}}
```

```
    for  (k=0; k<Feature_Dimension; k++){
    for(c=0; c<Class_Number; c++){
      if(c != index[i])
     Second_Error[k] += q[c]*Second_q[c][k]+First_q[c][k]*First_q[c][k];
        else
              Second_Error[k] += (q[c]-1)*Second_q[c][k]+First_q[c][k]*First_q[c][k];}
    }

}

Error /= Pattern_Number;

for(k=0; k<Feature_Dimension; k++){
    First_Error[k] = (First_Error[k]*2.0)/Pattern_Number;
    Second_Error[k] = (Second_Error[k]*2.0)/Pattern_Number;
}

delete [] First_u;
delete [] Second_u;
delete [] First_q;
delete [] Second_q;
delete [] u;
delete [] q;
delete [] SigmaCopy;
delete [] First_s;
delete [] Second_s;
delete [] dsqr;

}
```

fun_error.cpp

```
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

void fun_error(
     int Feature_Dimension,
     int Class_Number,
     int Pattern_Number,
     int *Class,
     int *index,
     double **x,
     double *Sigma,
     double &Error,
     int &Score)
{

int i, j, k, c, index_j, tempint, Max_position;
double *u, *q, *SigmaCopy;
double Scaled_Distance, Difference, s, Max_value;

SigmaCopy = new double[Feature_Dimension];
if(SigmaCopy == NULL){cout<<"No enough memory!"; exit(1);}
u = new double[Class_Number];
if(u == NULL){cout<<"No enough memory!"; exit(1);}
q = new double[Class_Number];
if(u == NULL){cout<<"No enough memory!"; exit(1);}

for(k=0; k<Feature_Dimension; k++) SigmaCopy[k] = Sigma[k];

Error=0.0; Score=0;

for(i=0; i<Pattern_Number; i++){
     for(c=0; c<Class_Number; c++)
          u[c] = 0.0;
     for(j=0; j<Pattern_Number; j++){
          if(i!=j){
               index_j=index[j];
               Scaled_Distance=0.0;
               for(k=0; k<Feature_Dimension; k++){
                    Difference = (x[i][k]-x[j][k])/SigmaCopy[k];
                    Scaled_Distance += Difference*Difference;}

               u[index_j] += exp(-Scaled_Distance);

          }
     }

     for(c=0; c<Class_Number; c++)
          u[c] /=Class[c];

     s=0.0;
```

```
        for(c=0; c<Class_Number; c++)
            s += u[c];

        if(s<1.e-40)
            s=1.e-40;

        for(c=0; c<Class_Number; c++)
            q[c] = u[c]/s;

        Max_value = q[0]; Max_position = 0;
        for(c=1; c<Class_Number; c++){
            if (q[c]>Max_value){
                Max_value = q[c];
                Max_position = c;}}

        if(Max_position == index[i])
          Score++;

        for(c=0; c<Class_Number; c++){
            if(c != index[i])
                Error += q[c]*q[c];
            else
                Error += (1.0-q[c])*(1.0-q[c]);}

}

Error /= Pattern_Number;

delete [] u;
delete [] q;
delete [] SigmaCopy;

}
```

fun_global.cpp

```
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
extern int PrintControl;
extern void fun_error(
      int Feature_Dimension,
      int Class_Number,
      int Pattern_Number,
      int *Class,
      int *index,
      double **x,
      double *Sigma,
      double &Error,
      int &Score);

int fun_global(
      int Feature_Dimension,
      int Class_Number,
      int Pattern_Number,
      int *Class,
      int *index,
      double **x,
      double *Sigma,
      double *d,
      double t,
      int StepNumber)
{

double *Minimal_Sigma, *SigmaCopy, *Step_v;
double Previous, Error;
int Score;
int i, k, jj, tempint, ibest_t;

Step_v = new double[Feature_Dimension];
if(Step_v == NULL){cout<<"No enough memory!"; exit(1);}
for(k=0; k<Feature_Dimension; k++)
     Step_v[k] = d[k]*t/(StepNumber-1);

Minimal_Sigma = new double[Feature_Dimension];
if(Minimal_Sigma == NULL){cout<<"No enough memory!"; exit(1);}
for(k=0; k<Feature_Dimension; k++) Minimal_Sigma[k] = Sigma[k];

SigmaCopy = new double[Feature_Dimension];
if(SigmaCopy == NULL){cout<<"No enough memory!"; exit(1);}

ibest_t=0;
for(k=0; k<Feature_Dimension; k++) SigmaCopy[k] = Sigma[k];
fun_error(Feature_Dimension, Class_Number, Pattern_Number, Class,
          index, x, SigmaCopy, Previous, Score);

if   (PrintControl==1){
```

```
    cout<<"Previous = "<<Previous<<", Score = "<<Score<<endl; cin>>tempint; }

for  (i=1; i<StepNumber; i++){
    for(k=0; k<Feature_Dimension; k++) SigmaCopy[k] += Step_v[k];

    fun_error(Feature_Dimension, Class_Number, Pattern_Number, Class,
        index, x, SigmaCopy, Error, Score);

    if   (PrintControl==1){
        cout<<"i="<<i<<",Previous="<<Previous<<", Error = "<<Error<<", Score = "<<Score<<endl;}

    if(Error < Previous){
        ibest_t = i;
        Previous = Error;}
}

if(ibest_t==0){
    for(jj=1; jj<=50; jj++){
        for(k=0; k<Feature_Dimension; k++)
            SigmaCopy[k] -= Step_v[k];
        fun_error(Feature_Dimension, Class_Number, Pattern_Number, Class,
            index, x, SigmaCopy, Error, Score);
        if(Error<Previous){
            ibest_t--;
            Previous = Error;}
        else
            break;
    }
}

else if(ibest_t == StepNumber-1){
    for(jj=1; jj<=50; jj++){ // 50 is just a loop times.
        for(k=0; k<Feature_Dimension; k++){
            SigmaCopy[k] += Step_v[k];}
        fun_error(Feature_Dimension, Class_Number, Pattern_Number, Class,
        index, x, SigmaCopy, Error, Score);

        if(Error<Previous){
            ibest_t++;
            Previous = Error; }
        else
            break;
    }
}

delete [] Minimal_Sigma;
delete [] Step_v;
delete [] SigmaCopy;

return ibest_t;

}
```

fun_gloden_section.cpp

```
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

extern void fun_error(
    int Feature_Dimension,
    int Class_Number,
    int Pattern_Number,
    int *Class,
    int *index,
    double **x,
    double *Sigma,
    double &Error,
    int &Score);

double fun_golden_section(
    int Feature_Dimension,
    int Class_Number,
    int Pattern_Number,
    int *Class,
    int *index,
    double **x,
    double *Sigma,
    double *d,
    double t,
    int StepNumber,
    int New_t)
{

double Position1, Position2, Position3, Segment1, Segment2, Th1, Th2;
double New_Position, Step, Error, Error1, Error2, Error3;
double *New_Sigma, *Sigma1, *Sigma2, *Sigma3;
int i, k, Iterate, Score;

Th1 = 0.0001;
Th2 = 0.00001;
Iterate=0;
Step=t/(StepNumber-1);
Position2=Step*New_t;Position1=Position2-Step;Position3=Position2+Step;

Sigma1 = new double[Feature_Dimension];
if(Sigma1 == NULL){cout<<"No enough memory!"; exit(1);}
Sigma2 = new double[Feature_Dimension];
if(Sigma2 == NULL){cout<<"No enough memory!"; exit(1);}
Sigma3 = new double[Feature_Dimension];
if(Sigma3 == NULL){cout<<"No enough memory!"; exit(1);}
New_Sigma = new double[Feature_Dimension];
if(New_Sigma == NULL){cout<<"No enough memory!"; exit(1);}
for(k=0; k<Feature_Dimension; k++){
    Sigma1[k]=Sigma[k]+d[k]*Position1;
    Sigma2[k]=Sigma[k]+d[k]*Position2;
```

```
        Sigma3[k]=Sigma[k]+d[k]*Position3;}
    fun_error(Feature_Dimension, Class_Number, Pattern_Number, Class,
              index, x, Sigma1, Error1, Score);
    fun_error(Feature_Dimension, Class_Number, Pattern_Number, Class,
              index, x, Sigma2, Error2, Score);
    fun_error(Feature_Dimension, Class_Number, Pattern_Number, Class,
              index, x, Sigma3, Error3, Score);
    for(i=1; i<=100; i++){
        Iterate++;
        Segment1 = Position2-Position1;
        Segment2 = Position3-Position2;
        if(Segment1>=Segment2){
            New_Position = Position2-Segment1*0.38197;
            for(k=0; k<Feature_Dimension; k++)
                New_Sigma[k] = Sigma[k]+d[k]*New_Position;
            fun_error(Feature_Dimension, Class_Number, Pattern_Number, Class,
              index, x, New_Sigma, Error, Score);

            if(Error<Error2){
                Position3=Position2;Error3=Error2;
                Position2=New_Position;Error2=Error;}
            else{
                Position1=New_Position;
                Error1=Error;
                break;}
        }
        else{
            New_Position = Position2+Segment2*0.38197;
            for(k=0; k<Feature_Dimension; k++)
                New_Sigma[k] = Sigma[k]+d[k]*New_Position;
            fun_error(Feature_Dimension, Class_Number, Pattern_Number, Class,
              index, x, New_Sigma, Error, Score);
            if(Error<Error2){
                Position1=Position2;Error1=Error2;
                Position2=New_Position;Error2=Error;}
            else {
                Position3=New_Position;
                Error3=Error;
                break;}
        }
        if((Segment1<Th1) || (Segment2<Th1))
            break;
        else if(fabs((Error1-Error2)<Th2) && fabs((Error3-Error2)<Th2))
            break;
    }

        delete [] Sigma1;
        delete [] Sigma2;
        delete [] Sigma3;
        delete [] New_Sigma;

        return Position2;

    }
```

TestpnnMain.cpp

```
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

extern void fun_test(
      int Feature_Dimension,
      int Class_Number,
      int Pattern_Number,
      int *Class,
      int *index,
      double **x,
      double *Sigma,
      double &Error,
      int &Score);

int main() {

      int Feature_Dimension;
      int Class_Number;
      int Pattern_Number;
      int *Class=0;
      int *index;
      double **x;
      double *Sigma;

      int i, k,c;
      double Error;
      int Score;

      // Type the general info file name for open.
      char filenameGeneralInfo[100]="c:\\cjin\\ne\\adpnn\\1\\g5t_250cases_d60.dat"; // t=5, d=60, each=50

      // Type the features of validation set for open.
      char filenameTrainningData[100]="c:\\cjin\\feature_combination1\\t5nn5d_rms_vdf_3t.doc";

      // Open trained Sigma file for getting scaling parameters.
      char trainingSigma[100]="c:\\cjin\\feature_sigma1\\Sigma_rms_vdf_1t.dat";

      ifstream fcin1(filenameGeneralInfo, ios::nocreate);
      if(!fcin1){cerr<<"cannot open file1"<<endl; exit(1);}

      ifstream fcin2(filenameTrainningData, ios::nocreate);
      if(!fcin2){cerr<<"cannot open file2"<<endl; exit(1);}

      ifstream fcin3(trainingSigma, ios::nocreate);
      if(!fcin3){cerr<<"cannot open file3"<<endl; exit(1);}

      fcin1>>Feature_Dimension;
      fcin1>>Class_Number;
      cout<<"Feature_Dimension = "<<Feature_Dimension<<", Class_Number = "<<Class_Number<<endl;
      Class = new int[Class_Number];
```

```
    if(Class == NULL){cout<<"No enough memory!"; exit(1);}

    Pattern_Number = 0;
    for(i=0; i<Class_Number; i++){
        fcin1>>Class[i];
        cout<<"Class["<<i<<"] = "<<Class[i]<<endl;
        Pattern_Number += Class[i];}

    index = new int[Pattern_Number];
    if(index == NULL){cout<<"No enough memory!"; exit(1);}
    i=0;
    for(c=0; c<Class_Number; c++){
        for(k=0; k<Class[c]; k++){
            index[i] = c;
            //cout<<c;
            i++;}}

    // Read the second file for getting test data
    x = new double *[Pattern_Number];
    if(x == NULL){cout<<"No enough memory!"; exit(1);}
    for(i=0; i<Pattern_Number; i++){
        x[i] = new double[Feature_Dimension];
        if(x[i]== NULL){cout<<"No enough memory!"; exit(1);
    }}
    for(i=0; i<Pattern_Number; i++){
        for(k=0; k<Feature_Dimension; k++){
            fcin2>>x[i][k];
    }}

    // Read the training sigma for getting sigma data
    Sigma = new double[Feature_Dimension];
    if(Sigma == NULL){cout<<"No enough memory!"; exit(1);}
        for(k=0; k<Feature_Dimension; k++){
        fcin3>>Sigma[k];
        //cout<<Sigma[k]<<" ";
    }

    // Classify these batch of patterns based on the features from wavelet and fractal domain.
    fun_test(Feature_Dimension, Class_Number, Pattern_Number, Class,
        index, x, Sigma, Error, Score);
    cout<<endl<<"Error="<<Error<<", Score="<<Score<<", Rate_correct = "<<(Score/250.0)*100<<endl;

    delete [] Class;
    delete [] index;
    delete [] x;
    delete [] Sigma;

    return 1;

}
```

fun_test.cpp

```
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

void fun_test(
     int Feature_Dimension,
     int Class_Number,
     int Pattern_Number,
     int *Class,
     int *index,
     double **x,
     double *Sigma,
     double &Error,
     int &Score)
{

int i, j, k, c, index_j, tempint, Max_position;
double *u, *q, *SigmaCopy;
double Scaled_Distance, Difference, s, Max_value, R_Th;
ofstream fout_mis, fout_all;

//Define the file name for the results of all classified cases.
char filename_all[100]="c:\\cjin\\feature_confusion\\rms_vf_3t_1tT_all.dat";

// Define the file name for the results of all misclassified cases.
char filename_mis[100]="c:\\cjin\\feature_confusion\\rms_vf_3t_1tT_mis.dat";

fout_all.open(filename_all, ios::app);
fout_mis.open(filename_mis, ios::app);

fout_mis<<"The misclassification cases:"<<endl;
fout_all<<"The classification results for all cases:"<<endl;

SigmaCopy = new double[Feature_Dimension];
if(SigmaCopy == NULL){cout<<"No enough memory!"; exit(1);}
u = new double[Class_Number];
if(u == NULL){cout<<"No enough memory!"; exit(1);}
q = new double[Class_Number];
if(u == NULL){cout<<"No enough memory!"; exit(1);}

for(k=0; k<Feature_Dimension; k++)
     SigmaCopy[k] = Sigma[k];

R_Th=1.0e-10;
Error=0.0; Score=0;

cout<<endl<<"The pattern number for test: "<<Pattern_Number<<endl;
for(i=0; i<Pattern_Number; i++){
     for(c=0; c<Class_Number; c++)
          u[c] = 0.0;
     for(j=0; j<Pattern_Number; j++){
```

```
            if(i!=j){
                index_j=index[j];
                Scaled_Distance=0.0;
                for(k=0; k<Feature_Dimension; k++){
                    Difference = (x[i][k]-x[j][k])/SigmaCopy[k];
                    Scaled_Distance += Difference*Difference;}
                u[index_j] += exp(-Scaled_Distance);
            }
        }

        for(c=0; c<Class_Number; c++)
            u[c] /=Class[c];
        s=0.0;
        for(c=0; c<Class_Number; c++)
            s += u[c];
        if(s<1.e-40)
            s=1.e-40;
        for(c=0; c<Class_Number; c++)
            q[c] = u[c]/s;

        Max_value = q[0]; Max_position = 0;
        for(c=1; c<Class_Number; c++){
            if (q[c]>Max_value){
                Max_value = q[c];
                Max_position = c;}}

        if( (Max_value*100) <R_Th)
            Max_position=9;

        if(Max_position == index[i])
          Score++;
        else{fout_mis<<"fileNo="<<i<<"class="<<index[i]<<"Predicted="<<Max_position<<
                "Conf(%)="<<Max_value*100<<endl;}

        fout_all<<"fileNo="<<i<<"class="<<index[i]<<"Predicted="<<Max_position<<
                "Conf(%)="<<Max_value*100<<endl;

        for(c=0; c<Class_Number; c++){
            if(c != index[i])
                Error += q[c]*q[c];
            else
                Error += (1.0-q[c])*(1.0-q[c]);}

}

Error /= Pattern_Number;

fout_mis<<endl<<"correctRate(%)="<<(Score/250.0)*100<<"Error="<<Error<<endl;

delete [] u;
delete [] q;
delete [] SigmaCopy;

}
```

APPENDIX C
EXPERIMENTAL RESULTS FOR CLASSIFICATION

C.1 Testing the PNN on Pure Signal Data Set

C.1.1 Training Set as Validation Set

The validation set is a copy of the training set, which consists of 250 pure transients with the detail as following.

- Voltage sags (labelled class 0): transient number from 0 to 49;
- Impulsive transient (labelled class 1): transient number from 50 to 99;
- Momentary interruption ((labelled class 2): transient number from 100 to 149;
- Voltage swells (labelled class 3): transient number from 150 to 199; and
- Oscillatory transient (labelled class 4): transient number from 200 to 249.

List C.1.1 shows the classification results by the PNN for all cases in the training set.

List C.1.1 Training set as validation set

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
0	0	0	100.000
1	0	0	100.000
2	0	0	100.000
3	0	0	100.000
4	0	0	100.000
5	0	0	100.000
6	0	0	100.000
7	0	0	100.000
8	0	0	100.000
9	0	0	100.000
10	0	0	100.000
11	0	0	100.000
12	0	0	100.000
13	0	0	100.000
14	0	0	100.000
15	0	0	100.000
16	0	0	100.000
17	0	0	100.000
18	0	0	100.000
19	0	0	100.000
20	0	0	100.000
21	0	0	100.000
22	0	0	100.000
23	0	0	100.000
24	0	0	100.000
25	0	0	100.000
26	0	0	100.000
27	0	0	100.000
28	0	0	100.000
29	0	0	100.000
30	0	0	100.000
31	0	0	100.000
32	0	0	100.000
33	0	0	100.000
34	0	0	99.9999
35	0	0	100.000
36	0	0	100.000
37	0	0	100.000
38	0	0	100.000
39	0	0	100.000
40	0	0	100.000
41	0	0	100.000
42	0	0	100.000
43	0	0	100.000
44	0	0	100.000
45	0	0	100.000
46	0	0	100.000
47	0	0	100.000
48	0	0	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
49	0	0	100.000
50	1	1	100.000
51	1	1	99.8273
52	1	1	99.9999
53	1	1	99.9987
54	1	1	99.9709
55	1	1	99.4599
56	1	1	99.9957
57	1	1	100.000
58	1	1	99.9859
59	1	1	99.8820
60	1	1	99.9952
61	1	1	99.9998
62	1	1	100.000
63	1	1	99.9923
64	1	1	100.000
65	1	1	99.9739
66	1	1	99.9747
67	1	1	100.000
68	1	1	99.9993
69	1	1	99.9998
70	1	1	99.9995
71	1	1	100.000
72	1	1	99.9999
73	1	1	100.000
74	1	1	100.000
75	1	1	100.000
76	1	1	99.9982
77	1	1	99.9990
78	1	1	99.9881
79	1	1	99.9943
80	1	1	100.000
81	1	1	100.000
82	1	1	97.3558
83	1	1	99.9631
84	1	1	99.9978
85	1	1	99.9873
86	1	1	99.9909
87	1	1	97.2627
88	1	1	98.6942
89	1	1	99.9984
90	1	1	99.9199
91	1	1	99.9976
92	1	1	100.000
93	1	1	100.000
94	1	1	100.000
95	1	1	99.9815
96	1	1	99.9961
97	1	1	100.000
98	1	1	99.9727
99	1	1	99.9295
100	2	2	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
101	2	2	100.000
102	2	2	100.000
103	2	2	100.000
104	2	2	100.000
105	2	2	100.000
106	2	2	100.000
107	2	2	100.000
108	2	2	100.000
109	2	2	100.000
110	2	2	100.000
111	2	2	100.000
112	2	2	100.000
113	2	2	100.000
114	2	2	100.000
115	2	2	100.000
116	2	2	100.000
117	2	2	100.000
118	2	2	100.000
119	2	2	100.000
120	2	2	100.000
121	2	2	100.000
122	2	2	100.000
123	2	2	100.000
124	2	2	100.000
125	2	2	100.000
126	2	2	100.000
127	2	2	100.000
128	2	2	100.000
129	2	2	100.000
130	2	2	100.000
131	2	2	100.000
132	2	2	100.000
133	2	2	100.000
134	2	2	100.000
135	2	2	100.000
136	2	2	100.000
137	2	2	100.000
138	2	2	100.000
139	2	2	100.000
140	2	2	100.000
141	2	2	100.000
142	2	2	100.000
143	2	2	100.000
144	2	2	100.000
145	2	2	100.000
146	2	2	100.000
147	2	2	100.000
148	2	2	100.000
149	2	2	100.000
150	3	3	100.000
151	3	3	100.000
152	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
153	3	3	100.000
154	3	3	100.000
155	3	3	100.000
156	3	3	100.000
157	3	3	100.000
158	3	3	100.000
159	3	3	100.000
160	3	3	100.000
161	3	3	100.000
162	3	3	100.000
163	3	3	100.000
164	3	3	100.000
165	3	3	100.000
166	3	3	100.000
167	3	3	100.000
168	3	3	100.000
169	3	3	100.000
170	3	3	100.000
171	3	3	100.000
172	3	3	100.000
173	3	3	100.000
174	3	3	100.000
175	3	3	100.000
176	3	3	100.000
177	3	3	100.000
178	3	3	100.000
179	3	3	100.000
180	3	3	99.9954
181	3	3	100.000
182	3	3	100.000
183	3	3	100.000
184	3	3	100.000
185	3	3	100.000
186	3	3	100.000
187	3	3	100.000
188	3	3	100.000
189	3	3	100.000
190	3	3	100.000
191	3	3	100.000
192	3	3	100.000
193	3	3	100.000
194	3	3	100.000
195	3	3	100.000
196	3	3	100.000
197	3	3	100.000
198	3	3	100.000
199	3	3	100.000
200	4	4	99.7160
201	4	4	99.9961
202	4	4	99.2882
203	4	4	100.000
204	4	4	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
205	4	4	99.9997
206	4	4	100.000
207	4	4	99.9998
208	4	4	99.9996
209	4	4	99.9998
210	4	4	99.9784
211	4	4	99.9951
212	4	4	99.9780
213	4	4	100.000
214	4	4	100.000
215	4	4	99.9874
216	4	4	99.8766
217	4	4	99.9801
218	4	4	99.9963
219	4	4	100.000
220	4	4	99.9998
221	4	4	98.1232
222	4	4	99.9695
223	4	4	99.9996
224	4	4	99.9597
225	4	4	99.9971
226	4	4	100.000
227	4	4	99.9821
228	4	4	100.000
229	4	4	100.000
230	4	4	99.9981
231	4	4	99.1485
232	4	4	99.9988
233	4	4	99.8549
234	4	4	100.000
235	4	4	99.9943
236	4	4	99.9019
237	4	4	99.9979
238	4	4	99.9561
239	4	4	100.000
240	4	4	94.9323
241	4	4	99.7723
242	4	4	99.9343
243	4	4	99.9360
244	4	4	100.000
245	4	4	99.9999
246	4	4	99.9987
247	4	4	99.9998
248	4	4	100.000
249	4	4	99.5099

C.1.2 Testing Set as Validation Set

The validation set is from the testing set, which consists of 500 pure transients with the detail as following.

- Voltage sags (labelled class 0): transient number from 0 to 99;
- Impulsive transient (labelled class 1): transient number from 100 to 199;
- Momentary interruption ((labelled class 2): transient number from 200 to 299;
- Voltage swells (labelled class 3): transient number from 300 to 399; and
- Oscillatory transient (labelled class 4): transient number from 400 to 499.

List C.1.2 shows the classification results by the PNN for all cases in the testing set.

List C.1.2 Testing set as validation set

Transient #	Original Class	Predicted Class	Bayesian Confidence (%)
0	0	0	100.000
1	0	0	100.000
2	0	0	100.000
3	0	0	100.000
4	0	0	100.000
5	0	0	100.000
6	0	0	100.000
7	0	0	100.000
8	0	0	100.000
9	0	0	100.000
10	0	0	100.000
11	0	0	100.000
12	0	0	100.000
13	0	0	100.000
14	0	0	100.000
15	0	0	100.000
16	0	0	100.000
17	0	0	100.000
18	0	0	100.000
19	0	0	100.000
20	0	0	100.000
21	0	0	100.000
22	0	0	100.000
23	0	0	100.000
24	0	0	99.8763
25	0	0	100.000
26	0	0	100.000
27	0	0	100.000
28	0	0	100.000
29	0	0	100.000
30	0	0	100.000
31	0	0	100.000
32	0	0	100.000
33	0	0	100.000
34	0	0	100.000
35	0	0	100.000
36	0	0	100.000
37	0	0	100.000
38	0	0	100.000
39	0	0	100.000
40	0	0	100.000
41	0	0	100.000
42	0	0	100.000
43	0	0	100.000
44	0	0	100.000
45	0	0	100.000
46	0	0	100.000
47	0	0	100.000
48	0	0	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
49	0	0	100.000
50	0	0	100.000
51	0	0	100.000
52	0	0	100.000
53	0	0	100.000
54	0	0	100.000
55	0	0	100.000
56	0	0	100.000
57	0	0	99.9986
58	0	0	100.000
59	0	0	100.000
60	0	0	100.000
61	0	0	100.000
62	0	0	99.9981
63	0	0	100.000
64	0	0	100.000
65	0	0	100.000
66	0	0	100.000
67	0	0	100.000
68	0	0	100.000
69	0	0	100.000
70	0	0	100.000
71	0	0	100.000
72	0	0	100.000
73	0	0	100.000
74	0	0	100.000
75	0	0	100.000
76	0	0	100.000
77	0	0	100.000
78	0	0	100.000
79	0	0	100.000
80	0	0	100.000
81	0	0	100.000
82	0	0	100.000
83	0	0	100.000
84	0	0	100.000
85	0	0	100.000
86	0	0	100.000
87	0	0	100.000
88	0	0	100.000
89	0	0	100.000
90	0	0	100.000
91	0	0	100.000
92	0	0	100.000
93	0	0	100.000
94	0	0	100.000
95	0	0	100.000
96	0	0	100.000
97	0	0	100.000
98	0	0	100.000
99	0	0	100.000
100	1	1	99.9999

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
101	1	1	99.9948
102	1	1	99.8748
103	1	1	99.9995
104	1	1	99.9859
105	1	1	99.9380
106	1	1	99.9993
107	1	1	99.8010
108	1	1	100.000
109	1	1	99.9754
110	1	1	99.9974
111	1	1	99.9997
112	1	1	100.000
113	1	1	99.9642
114	1	1	99.9990
115	1	1	99.8584
116	1	1	99.9891
117	1	1	99.9655
118	1	1	99.7953
119	1	1	98.1480
120	1	1	100.000
121	1	1	99.9992
122	1	1	99.9978
123	1	1	99.9947
124	1	1	98.9590
125	1	1	99.9985
126	1	1	99.9220
127	1	1	99.9996
128	1	1	100.000
129	1	1	99.5569
130	1	1	99.9998
131	1	1	99.9977
132	1	1	99.9899
133	1	1	99.9987
134	1	1	99.2919
135	1	1	99.9932
136	1	1	99.9996
137	1	1	99.9747
138	1	1	99.9913
139	1	1	99.9995
140	1	1	99.9476
141	1	1	99.9963
142	1	1	99.9997
143	1	1	99.9246
144	1	1	99.9999
145	1	1	99.9999
146	1	1	99.9990
147	1	1	99.9726
148	1	1	99.9998
149	1	1	98.0805
150	1	1	100.000
151	1	1	100.000
152	1	1	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
153	1	1	99.9999
154	1	1	99.9413
155	1	1	100.000
156	1	1	99.9998
157	1	1	99.9998
158	1	1	98.7522
159	1	1	100.000
160	1	1	99.9996
161	1	1	99.8247
162	1	1	99.2625
163	1	1	99.9996
164	1	1	99.9999
165	1	1	99.9941
166	1	1	99.9501
167	1	1	99.9999
168	1	1	99.9999
169	1	1	95.2719
170	1	1	99.7649
171	1	1	99.9932
172	1	1	99.7790
173	1	1	99.9999
174	1	1	99.9819
175	1	1	99.2878
176	1	1	99.9995
177	1	1	99.9836
178	1	1	100.000
179	1	1	100.000
180	1	1	100.000
181	1	4	65.9608
182	1	1	99.9994
183	1	1	100.000
184	1	1	99.7961
185	1	4	86.5215
186	1	1	99.9848
187	1	1	93.2353
188	1	1	99.9561
189	1	1	99.9785
190	1	1	99.9987
191	1	1	100.000
192	1	1	99.9865
193	1	1	100.000
194	1	1	100.000
195	1	1	100.000
196	1	1	100.000
197	1	1	99.9991
198	1	1	99.9996
199	1	1	100.000
200	2	2	100.000
201	2	2	100.000
202	2	2	100.000
203	2	2	100.000
204	2	2	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
205	2	2	100.000
206	2	2	100.000
207	2	2	100.000
208	2	2	0.00369181
209	2	2	100.000
210	2	2	100.000
211	2	2	100.000
212	2	2	100.000
213	2	2	100.000
214	2	2	100.000
215	2	2	100.000
216	2	2	100.000
217	2	2	100.000
218	2	2	100.000
219	2	2	100.000
220	2	2	100.000
221	2	2	100.000
222	2	2	100.000
223	2	2	100.000
224	2	2	100.000
225	2	2	100.000
226	2	2	100.000
227	2	2	100.000
228	2	2	100.000
229	2	2	100.000
230	2	2	100.000
231	2	2	100.000
232	2	2	100.000
233	2	2	100.000
234	2	2	100.000
235	2	2	100.000
236	2	2	100.000
237	2	2	100.000
238	2	2	100.000
239	2	2	100.000
240	2	2	100.000
241	2	2	100.000
242	2	2	100.000
243	2	2	100.000
244	2	2	100.000
245	2	2	100.000
246	2	2	100.000
247	2	2	100.000
248	2	2	100.000
249	2	2	100.000
250	2	2	100.000
251	2	2	100.000
252	2	2	100.000
253	2	2	100.000
254	2	2	100.000
255	2	2	100.000
256	2	2	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
257	2	2	100.000
258	2	2	100.000
259	2	2	100.000
260	2	2	100.000
261	2	2	100.000
262	2	2	100.000
263	2	2	100.000
264	2	2	100.000
265	2	2	100.000
266	2	2	100.000
267	2	2	100.000
268	2	2	100.000
269	2	2	0.0131004
270	2	2	100.000
271	2	2	100.000
272	2	2	100.000
273	2	2	100.000
274	2	2	100.000
275	2	2	100.000
276	2	2	100.000
277	2	2	100.000
278	2	2	100.000
279	2	2	100.000
280	2	2	100.000
281	2	2	100.000
282	2	2	100.000
283	2	2	100.000
284	2	2	100.000
285	2	2	100.000
286	2	2	100.000
287	2	2	100.000
288	2	2	100.000
289	2	2	100.000
290	2	2	2.80383e-010
291	2	2	100.000
292	2	2	100.000
293	2	2	100.000
294	2	2	100.000
295	2	2	100.000
296	2	2	100.000
297	2	2	100.000
298	2	2	100.000
299	2	2	100.000
300	3	3	100.000
301	3	3	100.000
302	3	3	99.9938
303	3	3	100.000
304	3	3	100.000
305	3	3	100.000
306	3	3	100.000
307	3	3	100.000
308	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
309	3	3	100.000
310	3	3	100.000
311	3	3	100.000
312	3	3	100.000
313	3	3	100.000
314	3	3	100.000
315	3	3	100.000
316	3	3	100.000
317	3	3	100.000
318	3	3	100.000
319	3	3	100.000
320	3	3	100.000
321	3	3	100.000
322	3	3	100.000
323	3	3	100.000
324	3	3	100.000
325	3	3	100.000
326	3	3	100.000
327	3	3	100.000
328	3	3	100.000
329	3	3	100.000
330	3	3	100.000
331	3	3	100.000
332	3	3	100.000
333	3	3	100.000
334	3	3	100.000
335	3	3	100.000
336	3	3	100.000
337	3	3	100.000
338	3	3	100.000
339	3	3	100.000
340	3	3	100.000
341	3	3	100.000
342	3	3	100.000
343	3	3	100.000
344	3	3	100.000
345	3	3	100.000
346	3	3	100.000
347	3	3	100.000
348	3	3	100.000
349	3	3	100.000
350	3	3	100.000
351	3	3	100.000
352	3	3	100.000
353	3	3	100.000
354	3	3	100.000
355	3	3	100.000
356	3	3	100.000
357	3	3	99.9978
358	3	3	100.000
359	3	3	100.000
360	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
361	3	3	100.000
362	3	3	100.000
363	3	3	100.000
364	3	3	100.000
365	3	3	100.000
366	3	3	100.000
367	3	3	100.000
368	3	3	100.000
369	3	3	100.000
370	3	3	100.000
371	3	3	100.000
372	3	3	100.000
373	3	3	100.000
374	3	3	100.000
375	3	3	100.000
376	3	3	100.000
377	3	3	100.000
378	3	3	100.000
379	3	3	100.000
380	3	3	100.000
381	3	3	100.000
382	3	3	100.000
383	3	3	100.000
384	3	3	100.000
385	3	3	100.000
386	3	3	100.000
387	3	3	100.000
388	3	3	100.000
389	3	3	100.000
390	3	3	100.000
391	3	3	100.000
392	3	3	100.000
393	3	3	100.000
394	3	3	100.000
395	3	3	100.000
396	3	3	100.000
397	3	3	100.000
398	3	3	100.000
399	3	3	100.000
400	4	4	52.1624
401	4	4	99.9940
402	4	4	100.000
403	4	4	99.9299
404	4	4	100.000
405	4	4	99.9688
406	4	4	99.9810
407	4	4	100.000
408	4	4	99.9929
409	4	4	86.6679
410	4	4	99.9992
411	4	4	100.000
412	4	4	99.9995

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
413	4	1	68.7327
414	4	4	100.000
415	4	4	98.6281
416	4	4	100.000
417	4	4	100.000
418	4	4	99.9910
419	4	4	99.9994
420	4	4	100.000
421	4	4	97.3312
422	4	4	100.000
423	4	4	100.000
424	4	4	99.9648
425	4	4	100.000
426	4	4	96.2366
427	4	1	76.1155
428	4	4	99.9998
429	4	4	100.000
430	4	4	100.000
431	4	4	98.5590
432	4	4	99.8847
433	4	4	99.9226
434	4	4	99.8312
435	4	4	99.9880
436	4	4	99.9995
437	4	4	100.000
438	4	4	99.9996
439	4	4	99.9999
440	4	4	100.000
441	4	4	99.9665
442	4	4	99.9999
443	4	4	100.000
444	4	4	100.000
445	4	4	100.000
446	4	4	99.9965
447	4	4	99.9954
448	4	4	99.9999
449	4	4	99.9135
450	4	4	99.9971
451	4	4	99.9321
452	4	4	100.000
453	4	4	99.9875
454	4	4	99.9930
455	4	4	100.000
456	4	4	99.9674
457	4	4	99.9933
458	4	4	99.9998
459	4	4	81.7835
460	4	4	98.2400
461	4	4	99.8606
462	4	4	99.9999
463	4	4	100.000
464	4	4	99.9999

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
465	4	4	100.000
466	4	4	93.6749
467	4	4	100.000
468	4	4	90.1378
469	4	4	99.1025
470	4	4	99.9793
471	4	4	99.9903
472	4	4	99.9136
473	4	4	100.000
474	4	4	99.9999
475	4	4	99.1353
476	4	4	99.9354
477	4	4	99.6673
478	4	4	99.9829
479	4	4	98.1285
480	4	4	99.9026
481	4	4	99.9798
482	4	4	100.000
483	4	4	99.9956
484	4	4	99.9908
485	4	4	99.9929
486	4	4	94.1931
487	4	4	99.9984
488	4	4	100.000
489	4	4	96.8599
490	4	4	100.000
491	4	4	100.000
492	4	4	99.9959
493	4	4	99.9990
494	4	4	100.000
495	4	4	99.1658
496	4	4	99.8871
497	4	4	99.7646
498	4	4	99.3530
499	4	4	99.9996

C.2 Testing the PNN on Transients Contaminated by White Gaussian Noise

C.2.1 Transients with White Noise of 20 dB as Validation Set

The validation set consists of 250 noisy transients, which are from the training set contaminated by white Gaussian noise of 20 dB with the detail as following.

- Voltage sags (labelled class 0): transient number from 0 to 49;
- Impulsive transient (labelled class 1): transient number from 50 to 99;
- Momentary interruption ((labelled class 2): transient number from 100 to 149;
- Voltage swells (labelled class 3): transient number from 150 to 199; and
- Oscillatory transient (labelled class 4): transient number from 200 to 249.

List C.2.1 shows the classification results by the PNN for all cases in this set.

List C.2.1 Transients with white noise of 20 dB as validation set

Transient #	Original Class	Predicted Class	Bayesian Confidence (%)
0	0	0	100.000
1	0	0	100.000
2	0	0	100.000
3	0	0	100.000
4	0	0	100.000
5	0	0	100.000
6	0	0	100.000
7	0	0	100.000
8	0	0	100.000
9	0	0	100.000
10	0	0	100.000
11	0	0	100.000
12	0	0	100.000
13	0	0	100.000
14	0	0	100.000
15	0	0	100.000
16	0	0	100.000
17	0	0	100.000
18	0	0	100.000
19	0	0	100.000
20	0	0	100.000
21	0	0	100.000
22	0	0	100.000
23	0	0	100.000
24	0	0	100.000
25	0	0	100.000
26	0	0	100.000
27	0	0	100.000
28	0	0	100.000
29	0	0	100.000
30	0	0	100.000
31	0	0	100.000
32	0	0	100.000
33	0	0	100.000
34	0	0	99.9093
35	0	0	100.000
36	0	0	100.000
37	0	0	100.000
38	0	0	100.000
39	0	0	100.000
40	0	0	100.000
41	0	0	100.000
42	0	0	100.000
43	0	0	100.000
44	0	0	100.000
45	0	0	99.9985
46	0	0	100.000
47	0	0	100.000
48	0	0	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
49	0	0	100.000
50	1	1	99.9380
51	1	1	81.6667
52	1	1	99.6847
53	1	1	96.4481
54	1	1	99.8383
55	1	1	96.9354
56	1	1	98.8276
57	1	1	99.9109
58	1	1	99.8611
59	1	1	96.7614
60	1	1	99.8588
61	1	1	99.9787
62	1	1	99.9816
63	1	1	98.8840
64	1	1	97.7938
65	1	1	95.2963
66	1	1	98.1895
67	1	1	99.8721
68	1	1	97.3417
69	1	1	93.9560
70	1	1	99.0501
71	1	1	99.6444
72	1	1	99.6842
73	1	1	99.1495
74	1	1	99.7580
75	1	1	99.9060
76	1	1	99.9651
77	1	1	99.9181
78	1	1	97.1437
79	1	1	99.7651
80	1	1	99.9800
81	1	1	99.5256
82	1	1	91.8166
83	1	1	86.6729
84	1	1	99.9709
85	1	1	94.4695
86	1	1	99.6517
87	1	1	98.6212
88	1	1	90.3897
89	1	1	99.3395
90	1	1	99.4660
91	1	1	99.4402
92	1	1	99.9932
93	1	1	99.9814
94	1	1	99.9835
95	1	1	99.2861
96	1	1	99.8585
97	1	1	99.8095
98	1	1	98.8508
99	1	1	95.3489
100	2	2	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
101	2	2	100.000
102	2	2	100.000
103	2	2	100.000
104	2	2	100.000
105	2	2	100.000
106	2	2	100.000
107	2	2	100.000
108	2	2	100.000
109	2	2	100.000
110	2	2	100.000
111	2	2	100.000
112	2	2	100.000
113	2	2	100.000
114	2	2	100.000
115	2	2	100.000
116	2	2	100.000
117	2	2	100.000
118	2	2	100.000
119	2	2	100.000
120	2	2	100.000
121	2	2	100.000
122	2	2	100.000
123	2	2	100.000
124	2	2	100.000
125	2	2	100.000
126	2	2	100.000
127	2	2	100.000
128	2	2	100.000
129	2	2	100.000
130	2	2	100.000
131	2	2	100.000
132	2	2	100.000
133	2	2	99.9997
134	2	2	100.000
135	2	2	100.000
136	2	2	100.000
137	2	2	100.000
138	2	2	100.000
139	2	2	100.000
140	2	2	100.000
141	2	2	100.000
142	2	2	100.000
143	2	2	100.000
144	2	2	100.000
145	2	2	100.000
146	2	2	100.000
147	2	2	100.000
148	2	2	100.000
149	2	2	100.000
150	3	3	100.000
151	3	3	100.000
152	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
153	3	3	100.000
154	3	3	100.000
155	3	3	100.000
156	3	3	100.000
157	3	3	100.000
158	3	3	100.000
159	3	3	100.000
160	3	3	100.000
161	3	3	100.000
162	3	3	100.000
163	3	3	100.000
164	3	3	100.000
165	3	3	100.000
166	3	3	100.000
167	3	3	100.000
168	3	3	100.000
169	3	3	100.000
170	3	3	100.000
171	3	3	100.000
172	3	3	100.000
173	3	3	100.000
174	3	3	100.000
175	3	3	100.000
176	3	3	100.000
177	3	3	100.000
178	3	3	100.000
179	3	3	100.000
180	3	3	99.8358
181	3	3	100.000
182	3	3	100.000
183	3	3	100.000
184	3	3	100.000
185	3	3	100.000
186	3	3	100.000
187	3	3	100.000
188	3	3	100.000
189	3	3	100.000
190	3	3	100.000
191	3	3	100.000
192	3	3	100.000
193	3	3	100.000
194	3	3	100.000
195	3	3	100.000
196	3	3	100.000
197	3	3	100.000
198	3	3	100.000
199	3	3	100.000
200	4	4	95.9744
201	4	4	93.6306
202	4	4	93.2697
203	4	4	100.000
204	4	4	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
205	4	4	99.6967
206	4	4	100.000
207	4	4	99.9988
208	4	4	99.9812
209	4	4	99.9877
210	4	4	99.0405
211	4	4	97.5513
212	4	4	99.3622
213	4	4	100.000
214	4	4	100.000
215	4	4	92.4784
216	4	4	99.7677
217	4	4	97.8988
218	4	4	99.8350
219	4	4	99.9996
220	4	4	91.7074
221	4	4	68.1440
222	4	4	99.5298
223	4	4	99.5918
224	4	4	99.2020
225	4	4	84.4375
226	4	4	99.9999
227	4	4	93.5526
228	4	4	99.9711
229	4	4	100.000
230	4	4	63.4670
231	4	4	91.0315
232	4	4	99.9495
233	4	4	95.3545
234	4	4	99.9618
235	4	4	99.9160
236	4	4	98.1045
237	4	4	96.9834
238	4	4	99.8783
239	4	4	99.9970
240	4	4	89.6372
241	4	4	99.4989
242	4	4	92.4847
243	4	4	82.5491
244	4	4	100.000
245	4	4	99.9999
246	4	4	98.6649
247	4	4	99.0906
248	4	4	100.000
249	4	4	78.6284

C.2.2 Transients with White Noise of 15 dB as Validation Set

The validation set consists of 250 noisy transients, which are from the training set contaminated by white Gaussian noise of 15 dB with the detail as following.

- Voltage sags (labelled class 0): transient number from 0 to 49;
- Impulsive transient (labelled class 1): transient number from 50 to 99;
- Momentary interruption ((labelled class 2): transient number from 100 to 149;
- Voltage swells (labelled class 3): transient number from 150 to 199; and
- Oscillatory transient (labelled class 4): transient number from 200 to 249.

List C.2.2 shows the classification results by the PNN for all cases in this set.

List C.2.2 Transients with white noise of 15 dB as validation set

Transient #	Original Class	Predicted Class	Bayesian Confidence (%)
0	0	0	100.000
1	0	0	100.000
2	0	0	100.000
3	0	0	100.000
4	0	0	100.000
5	0	0	100.000
6	0	0	100.000
7	0	0	100.000
8	0	0	100.000
9	0	0	100.000
10	0	0	100.000
11	0	0	100.000
12	0	0	100.000
13	0	0	100.000
14	0	0	100.000
15	0	0	100.000
16	0	0	100.000
17	0	0	100.000
18	0	0	100.000
19	0	0	100.000
20	0	0	100.000
21	0	0	100.000
22	0	0	100.000
23	0	0	100.000
24	0	0	100.000
25	0	0	100.000
26	0	0	100.000
27	0	0	100.000
28	0	0	100.000
29	0	0	100.000
30	0	0	100.000
31	0	0	100.000
32	0	0	100.000
33	0	0	100.000
34	0	0	99.8397
35	0	0	100.000
36	0	0	100.000
37	0	0	100.000
38	0	0	100.000
39	0	0	100.000
40	0	0	100.000
41	0	0	100.000
42	0	0	100.000
43	0	0	100.000
44	0	0	100.000
45	0	0	99.9952
46	0	0	100.000
47	0	0	100.000
48	0	0	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
49	0	0	100.000
50	1	1	99.6647
51	1	4	60.6031
52	1	1	97.4013
53	1	1	90.4978
54	1	1	99.6192
55	1	1	97.5800
56	1	1	95.6642
57	1	1	98.0370
58	1	1	99.7850
59	1	1	86.9630
60	1	1	99.5992
61	1	1	95.4811
62	1	1	98.3532
63	1	1	95.5909
64	1	1	91.9528
65	1	1	89.0550
66	1	1	93.7383
67	1	1	95.5714
68	1	1	88.1653
69	1	1	89.7290
70	1	1	93.0615
71	1	1	99.3272
72	1	1	95.6549
73	1	1	89.8836
74	1	1	94.9478
75	1	1	99.5480
76	1	1	99.5695
77	1	1	99.4108
78	1	1	83.1219
79	1	1	99.5271
80	1	1	96.1366
81	1	1	99.3359
82	1	1	80.2903
83	1	1	78.8979
84	1	1	99.6290
85	1	1	80.6791
86	1	1	98.6304
87	1	1	96.6362
88	1	1	98.8069
89	1	1	94.2958
90	1	1	99.4397
91	1	1	93.8297
92	1	1	99.8853
93	1	1	99.8562
94	1	1	98.5180
95	1	1	93.8846
96	1	1	98.6543
97	1	1	96.3945
98	1	1	91.0434
99	1	1	91.8268
100	2	2	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
101	2	2	100.000
102	2	2	100.000
103	2	2	100.000
104	2	2	100.000
105	2	2	100.000
106	2	2	100.000
107	2	2	100.000
108	2	2	100.000
109	2	2	99.9999
110	2	2	100.000
111	2	2	100.000
112	2	2	100.000
113	2	2	100.000
114	2	2	100.000
115	2	2	100.000
116	2	2	100.000
117	2	2	100.000
118	2	2	100.000
119	2	2	100.000
120	2	2	100.000
121	2	2	100.000
122	2	2	100.000
123	2	2	100.000
124	2	2	100.000
125	2	2	100.000
126	2	2	100.000
127	2	2	100.000
128	2	2	100.000
129	2	2	100.000
130	2	2	100.000
131	2	2	100.000
132	2	2	100.000
133	2	2	99.4952
134	2	2	100.000
135	2	2	100.000
136	2	2	100.000
137	2	2	100.000
138	2	2	100.000
139	2	2	100.000
140	2	2	100.000
141	2	2	100.000
142	2	2	100.000
143	2	2	100.000
144	2	2	100.000
145	2	2	99.9999
146	2	2	100.000
147	2	2	100.000
148	2	2	100.000
149	2	2	100.000
150	3	3	100.000
151	3	3	100.000
152	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
153	3	3	100.000
154	3	3	100.000
155	3	3	100.000
156	3	3	100.000
157	3	3	100.000
158	3	3	100.000
159	3	3	100.000
160	3	3	100.000
161	3	3	100.000
162	3	3	100.000
163	3	3	100.000
164	3	3	100.000
165	3	3	100.000
166	3	3	100.000
167	3	3	100.000
168	3	3	100.000
169	3	3	100.000
170	3	3	100.000
171	3	3	100.000
172	3	3	100.000
173	3	3	100.000
174	3	3	100.000
175	3	3	100.000
176	3	3	100.000
177	3	3	100.000
178	3	3	100.000
179	3	3	100.000
180	3	3	99.9334
181	3	3	100.000
182	3	3	100.000
183	3	3	100.000
184	3	3	100.000
185	3	3	100.000
186	3	3	100.000
187	3	3	100.000
188	3	3	100.000
189	3	3	100.000
190	3	3	100.000
191	3	3	100.000
192	3	3	100.000
193	3	3	100.000
194	3	3	100.000
195	3	3	100.000
196	3	3	100.000
197	3	3	100.000
198	3	3	100.000
199	3	3	100.000
200	4	4	58.1728
201	4	1	92.9686
202	4	4	55.5727
203	4	4	100.000
204	4	4	99.9993

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
205	4	4	95.0460
206	4	4	100.000
207	4	4	99.9932
208	4	4	98.9399
209	4	4	99.3971
210	4	4	58.7282
211	4	1	67.5510
212	4	4	97.9405
213	4	4	100.000
214	4	4	100.000
215	4	1	73.0546
216	4	4	99.0293
217	4	4	56.4280
218	4	4	97.1491
219	4	4	99.9966
220	4	1	71.3035
221	4	1	74.7887
222	4	4	99.5475
223	4	4	87.8537
224	4	4	83.8049
225	4	1	81.8605
226	4	4	99.9993
227	4	4	81.7285
228	4	4	80.7317
229	4	4	100.000
230	4	1	85.0999
231	4	1	83.6993
232	4	4	98.8870
233	4	4	51.4319
234	4	4	94.7914
235	4	4	99.9584
236	4	4	68.8058
237	4	1	72.2165
238	4	4	99.8651
239	4	4	99.9632
240	4	4	58.7282
241	4	4	97.2601
242	4	4	51.3304
243	4	1	73.7846
244	4	4	100.000
245	4	4	99.9997
246	4	4	99.6999
247	4	4	82.6376
248	4	4	100.000
249	4	1	81.7380

C.2.3 Transients with White Noise of 10 dB as Validation Set

The validation set consists of 250 noisy transients, which are from the training set contaminated by white Gaussian noise of 10 dB with the detail as following.

- Voltage sags (labelled class 0): transient number from 0 to 49;
- Impulsive transient (labelled class 1): transient number from 50 to 99;
- Momentary interruption ((labelled class 2): transient number from 100 to 149;
- Voltage swells (labelled class 3): transient number from 150 to 199; and
- Oscillatory transient (labelled class 4): transient number from 200 to 249.

List C.2.3 shows the classification results by the PNN for all cases in this set.

List C.2.3 Transients with white noise of 10 dB as validation set

Transient #	Original Class	Predicted Class	Bayesian Confidence (%)
0	0	0	100.000
1	0	0	99.9999
2	0	0	100.000
3	0	0	100.000
4	0	0	100.000
5	0	0	99.9952
6	0	0	100.000
7	0	0	100.000
8	0	0	100.000
9	0	0	100.000
10	0	0	100.000
11	0	0	100.000
12	0	0	100.000
13	0	0	100.000
14	0	0	100.000
15	0	0	100.000
16	0	0	100.000
17	0	0	100.000
18	0	0	100.000
19	0	0	100.000
20	0	0	100.000
21	0	0	100.000
22	0	0	100.000
23	0	0	100.000
24	0	0	100.000
25	0	0	100.000
26	0	0	100.000
27	0	0	100.000
28	0	0	100.000
29	0	0	100.000
30	0	0	99.9999
31	0	0	100.000
32	0	0	100.000
33	0	0	100.000
34	0	0	99.9946
35	0	0	100.000
36	0	0	100.000
37	0	0	100.000
38	0	0	100.000
39	0	0	100.000
40	0	0	100.000
41	0	0	100.000
42	0	0	100.000
43	0	0	100.000
44	0	0	100.000
45	0	0	99.9949
46	0	0	100.000
47	0	0	100.000
48	0	0	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
49	0	0	100.000
50	1	1	98.1636
51	1	4	70.9023
52	1	1	88.8412
53	1	1	79.2360
54	1	1	98.0629
55	1	1	93.5145
56	1	1	87.1355
57	1	1	91.9191
58	1	1	98.3510
59	1	1	82.8166
60	1	1	98.0673
61	1	1	84.9637
62	1	1	86.5261
63	1	1	78.5845
64	1	1	80.2660
65	1	1	69.0564
66	1	1	81.7040
67	1	1	78.5821
68	1	1	75.1654
69	1	1	94.0124
70	1	1	80.2471
71	1	1	98.2973
72	1	1	73.1231
73	1	1	60.6231
74	1	1	87.6505
75	1	1	95.6675
76	1	1	98.6052
77	1	1	91.6162
78	1	1	59.8334
79	1	1	97.7654
80	1	1	68.0654
81	1	1	95.1921
82	1	1	58.1062
83	1	1	63.9097
84	1	1	95.9241
85	1	1	72.9528
86	1	1	93.8955
87	1	1	77.9394
88	1	1	98.8125
89	1	1	87.7292
90	1	1	94.7479
91	1	1	77.1637
92	1	1	99.1053
93	1	1	98.7353
94	1	1	91.4261
95	1	1	77.5606
96	1	1	84.0326
97	1	1	88.7978
98	1	1	86.5886
99	1	1	86.3413
100	2	2	99.2920

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
101	2	2	100.000
102	2	2	100.000
103	2	2	100.000
104	2	2	100.000
105	2	0	99.7396
106	2	2	100.000
107	2	2	100.000
108	2	2	100.000
109	2	2	99.6461
110	2	2	100.000
111	2	2	100.000
112	2	2	100.000
113	2	2	99.9990
114	2	0	86.8536
115	2	2	100.000
116	2	2	100.000
117	2	2	100.000
118	2	2	100.000
119	2	2	99.9567
120	2	2	100.000
121	2	2	100.000
122	2	2	99.9012
123	2	2	100.000
124	2	2	100.000
125	2	2	99.9943
126	2	2	99.9997
127	2	2	100.000
128	2	2	100.000
129	2	2	100.000
130	2	2	100.000
131	2	2	100.000
132	2	2	99.9935
133	2	0	99.8690
134	2	2	99.9994
135	2	2	88.9077
136	2	2	99.9998
137	2	2	98.5554
138	2	2	100.000
139	2	2	99.9632
140	2	2	99.4804
141	2	2	100.000
142	2	2	99.9959
143	2	2	100.000
144	2	2	100.000
145	2	2	65.6289
146	2	2	100.000
147	2	2	100.000
148	2	2	100.000
149	2	0	99.9997
150	3	3	100.000
151	3	3	100.000
152	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
153	3	3	100.000
154	3	3	100.000
155	3	3	100.000
156	3	3	100.000
157	3	3	100.000
158	3	3	100.000
159	3	3	100.000
160	3	3	99.9999
161	3	3	100.000
162	3	3	100.000
163	3	3	100.000
164	3	3	100.000
165	3	3	100.000
166	3	3	100.000
167	3	3	100.000
168	3	3	100.000
169	3	3	100.000
170	3	3	100.000
171	3	3	100.000
172	3	3	100.000
173	3	3	100.000
174	3	3	100.000
175	3	3	100.000
176	3	3	100.000
177	3	3	100.000
178	3	3	100.000
179	3	3	100.000
180	3	3	99.9997
181	3	3	100.000
182	3	3	100.000
183	3	3	100.000
184	3	3	100.000
185	3	3	100.000
186	3	3	100.000
187	3	3	100.000
188	3	3	100.000
189	3	3	100.000
190	3	3	100.000
191	3	3	100.000
192	3	3	100.000
193	3	3	100.000
194	3	3	100.000
195	3	3	100.000
196	3	3	100.000
197	3	3	100.000
198	3	3	100.000
199	3	3	100.000
200	4	1	97.1743
201	4	1	96.6359
202	4	1	99.1135
203	4	4	100.000
204	4	4	77.3264

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
205	4	4	57.5004
206	4	4	99.8334
207	4	4	99.0582
208	4	1	98.1982
209	4	4	78.7709
210	4	1	73.9179
211	4	1	70.3718
212	4	4	83.2091
213	4	4	100.000
214	4	4	99.9999
215	4	1	92.7432
216	4	4	92.5561
217	4	1	96.2307
218	4	4	60.9211
219	4	4	99.9994
220	4	1	91.2481
221	4	1	69.0709
222	4	4	99.2749
223	4	1	94.9175
224	4	1	69.3967
225	4	1	75.3582
226	4	4	99.9998
227	4	1	52.6262
228	4	1	96.8489
229	4	4	100.000
230	4	1	88.7118
231	4	1	82.4355
232	4	4	70.5747
233	4	1	72.8337
234	4	1	96.6597
235	4	4	99.9623
236	4	1	90.3009
237	4	1	99.1808
238	4	4	96.4766
239	4	4	63.9573
240	4	1	94.9099
241	4	1	56.4099
242	4	1	73.3518
243	4	1	88.2257
244	4	4	99.9750
245	4	4	99.9300
246	4	4	96.4160
247	4	1	61.7777
248	4	4	99.9995
249	4	1	74.2267

C.3 Testing the PNN on Transients Contaminated by Pink Noise

C.3.1 Transients with Pink Noise of 20 dB as Validation Set

The validation set consists of 250 noisy transients, which are from the training set contaminated by pink noise of 20 dB with the detail as following.

- Voltage sags (labelled class 0): transient number from 0 to 49;
- Impulsive transient (labelled class 1): transient number from 50 to 99;
- Momentary interruption ((labelled class 2): transient number from 100 to 149;
- Voltage swells (labelled class 3): transient number from 150 to 199; and
- Oscillatory transient (labelled class 4): transient number from 200 to 249.

List C.3.1 shows the classification results by the PNN for all cases in this set.

List C.3.1 Transients with pink noise of 20 dB as validation set

Transient #	Original Class	Predicted Class	Bayesian Confidence (%)
0	0	0	100.000
1	0	0	100.000
2	0	0	100.000
3	0	0	100.000
4	0	0	100.000
5	0	0	100.000
6	0	0	100.000
7	0	0	100.000
8	0	0	100.000
9	0	0	100.000
10	0	0	100.000
11	0	0	100.000
12	0	0	100.000
13	0	0	100.000
14	0	0	100.000
15	0	0	100.000
16	0	0	100.000
17	0	0	100.000
18	0	0	100.000
19	0	0	100.000
20	0	0	100.000
21	0	0	100.000
22	0	0	100.000
23	0	0	100.000
24	0	0	99.9999
25	0	0	100.000
26	0	0	100.000
27	0	0	100.000
28	0	0	100.000
29	0	0	100.000
30	0	0	100.000
31	0	0	100.000
32	0	0	100.000
33	0	0	100.000
34	0	0	99.9116
35	0	0	100.000
36	0	0	100.000
37	0	0	100.000
38	0	0	100.000
39	0	0	100.000
40	0	0	100.000
41	0	0	99.9999
42	0	0	100.000
43	0	0	100.000
44	0	0	100.000
45	0	0	99.9999
46	0	0	100.000
47	0	0	100.000
48	0	0	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
49	0	0	99.9998
50	1	1	99.9973
51	1	1	88.3322
52	1	1	99.9679
53	1	1	96.1613
54	1	1	99.9148
55	1	1	92.6648
56	1	1	99.7739
57	1	1	100.000
58	1	1	99.9801
59	1	4	53.7369
60	1	1	99.9871
61	1	1	99.9869
62	1	1	99.9998
63	1	1	99.3551
64	1	1	98.4691
65	1	1	72.8636
66	1	1	98.7602
67	1	1	99.9867
68	1	1	94.6246
69	1	1	99.9894
70	1	1	99.9494
71	1	1	99.9669
72	1	1	99.7943
73	1	1	83.8329
74	1	1	99.9995
75	1	1	100.000
76	1	1	99.9928
77	1	1	99.8753
78	1	1	99.9110
79	1	1	99.8671
80	1	1	99.9437
81	1	1	99.9991
82	1	4	61.5400
83	1	1	94.9753
84	1	1	99.8991
85	1	1	99.8287
86	1	1	99.9596
87	1	1	99.4952
88	1	1	95.9926
89	1	1	98.6173
90	1	1	99.2090
91	1	1	99.9915
92	1	1	100.000
93	1	1	99.9996
94	1	1	99.9998
95	1	1	99.1714
96	1	1	99.8437
97	1	1	99.9907
98	1	1	99.3307
99	1	1	91.5003
100	2	2	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
101	2	2	100.000
102	2	2	100.000
103	2	2	100.000
104	2	2	100.000
105	2	2	100.000
106	2	2	100.000
107	2	2	100.000
108	2	2	100.000
109	2	2	100.000
110	2	2	100.000
111	2	2	100.000
112	2	2	100.000
113	2	2	100.000
114	2	2	100.000
115	2	2	100.000
116	2	2	100.000
117	2	2	100.000
118	2	2	100.000
119	2	2	100.000
120	2	2	100.000
121	2	2	100.000
122	2	2	100.000
123	2	2	100.000
124	2	2	100.000
125	2	2	100.000
126	2	2	100.000
127	2	2	100.000
128	2	2	100.000
129	2	2	100.000
130	2	2	100.000
131	2	2	100.000
132	2	2	100.000
133	2	2	100.000
134	2	2	100.000
135	2	2	100.000
136	2	2	100.000
137	2	2	100.000
138	2	2	100.000
139	2	2	100.000
140	2	2	100.000
141	2	2	100.000
142	2	2	100.000
143	2	2	100.000
144	2	2	100.000
145	2	2	100.000
146	2	2	100.000
147	2	2	100.000
148	2	2	100.000
149	2	2	100.000
150	3	3	100.000
151	3	3	100.000
152	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
153	3	3	100.000
154	3	3	100.000
155	3	3	100.000
156	3	3	100.000
157	3	3	100.000
158	3	3	100.000
159	3	3	100.000
160	3	3	100.000
161	3	3	100.000
162	3	3	100.000
163	3	3	100.000
164	3	3	100.000
165	3	3	100.000
166	3	3	100.000
167	3	3	100.000
168	3	3	100.000
169	3	3	100.000
170	3	3	100.000
171	3	3	100.000
172	3	3	100.000
173	3	3	100.000
174	3	3	100.000
175	3	3	100.000
176	3	3	100.000
177	3	3	100.000
178	3	3	100.000
179	3	3	100.000
180	3	3	99.5911
181	3	3	99.9995
182	3	3	100.000
183	3	3	100.000
184	3	3	100.000
185	3	3	100.000
186	3	3	100.000
187	3	3	100.000
188	3	3	100.000
189	3	3	100.000
190	3	3	100.000
191	3	3	100.000
192	3	3	100.000
193	3	3	100.000
194	3	3	100.000
195	3	3	100.000
196	3	3	100.000
197	3	3	100.000
198	3	3	100.000
199	3	3	100.000
200	4	4	99.1856
201	4	4	96.8094
202	4	4	97.7971
203	4	4	100.000
204	4	4	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
205	4	4	99.5625
206	4	4	100.000
207	4	4	99.9835
208	4	4	98.7566
209	4	4	99.9989
210	4	4	99.3389
211	4	4	98.8722
212	4	4	99.9702
213	4	4	100.000
214	4	4	99.9996
215	4	4	98.8054
216	4	4	99.9735
217	4	4	99.6503
218	4	4	99.6131
219	4	4	99.9999
220	4	4	99.9811
221	4	4	99.2819
222	4	4	94.4465
223	4	4	99.9190
224	4	4	99.5566
225	4	4	96.8239
226	4	4	100.000
227	4	4	99.2566
228	4	4	100.000
229	4	4	100.000
230	4	4	96.3251
231	4	1	51.3053
232	4	4	99.9908
233	4	4	99.7868
234	4	4	99.9974
235	4	4	99.1592
236	4	4	98.6686
237	4	4	99.4430
238	4	4	99.8908
239	4	4	100.000
240	4	4	99.7094
241	4	4	99.8724
242	4	4	99.8902
243	4	4	91.0302
244	4	4	99.9983
245	4	4	100.000
246	4	4	99.4033
247	4	4	99.9772
248	4	4	100.000
249	4	4	66.8458

C.3.2 Transients with Pink Noise of 15 dB as Validation Set

The validation set consists of 250 noisy transients, which are from the training set contaminated by pink noise of 15 dB with the detail as following.

- Voltage sags (labelled class 0): transient number from 0 to 49;
- Impulsive transient (labelled class 1): transient number from 50 to 99;
- Momentary interruption ((labelled class 2): transient number from 100 to 149;
- Voltage swells (labelled class 3): transient number from 150 to 199; and
- Oscillatory transient (labelled class 4): transient number from 200 to 249.

List C.3.2 shows the classification results by the PNN for all cases in this set.

List C.3.2 Transients with pink noise of 15 dB as validation set

Transient #	Original Class	Predicted Class	Bayesian Confidence (%)
0	0	0	100.000
1	0	0	100.000
2	0	0	100.000
3	0	0	100.000
4	0	0	100.000
5	0	0	100.000
6	0	0	100.000
7	0	0	100.000
8	0	0	100.000
9	0	0	100.000
10	0	0	100.000
11	0	0	100.000
12	0	0	100.000
13	0	0	100.000
14	0	0	100.000
15	0	0	100.000
16	0	0	100.000
17	0	0	100.000
18	0	0	100.000
19	0	0	100.000
20	0	0	100.000
21	0	0	100.000
22	0	0	100.000
23	0	0	100.000
24	0	0	100.000
25	0	0	100.000
26	0	0	100.000
27	0	0	100.000
28	0	0	100.000
29	0	0	100.000
30	0	0	100.000
31	0	0	100.000
32	0	0	100.000
33	0	0	100.000
34	0	0	99.7091
35	0	0	100.000
36	0	0	100.000
37	0	0	100.000
38	0	0	100.000
39	0	0	100.000
40	0	0	100.000
41	0	0	99.9983
42	0	0	100.000
43	0	0	100.000
44	0	0	100.000
45	0	0	99.9996
46	0	0	100.000
47	0	0	100.000
48	0	0	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
49	0	0	99.9999
50	1	1	99.9308
51	1	4	52.1349
52	1	1	98.7211
53	1	1	89.0208
54	1	1	99.1295
55	1	1	94.1590
56	1	1	96.9656
57	1	1	99.9983
58	1	1	99.9534
59	1	4	69.2035
60	1	1	99.9002
61	1	1	96.5524
62	1	1	99.9847
63	1	1	92.2494
64	1	1	80.1931
65	1	4	64.4476
66	1	1	91.3800
67	1	1	99.3662
68	1	1	74.0132
69	1	1	99.9978
70	1	1	99.2340
71	1	1	99.7956
72	1	1	93.3143
73	1	1	54.4852
74	1	1	99.9638
75	1	1	99.9951
76	1	1	99.9992
77	1	1	98.5453
78	1	1	97.8327
79	1	1	99.5967
80	1	1	62.8291
81	1	1	99.8911
82	1	4	62.4215
83	1	1	64.6554
84	1	1	98.8533
85	1	1	99.0671
86	1	1	99.2503
87	1	1	96.8060
88	1	1	99.8757
89	1	1	77.7378
90	1	1	99.7177
91	1	1	99.9522
92	1	1	99.9997
93	1	1	99.9848
94	1	1	99.5094
95	1	1	81.7933
96	1	1	97.0991
97	1	1	97.8102
98	1	1	85.1720
99	1	1	82.4440
100	2	2	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
101	2	2	100.000
102	2	2	100.000
103	2	2	100.000
104	2	2	100.000
105	2	2	100.000
106	2	2	100.000
107	2	2	100.000
108	2	2	100.000
109	2	2	100.000
110	2	2	100.000
111	2	2	100.000
112	2	2	100.000
113	2	2	100.000
114	2	2	100.000
115	2	2	100.000
116	2	2	100.000
117	2	2	100.000
118	2	2	100.000
119	2	2	100.000
120	2	2	100.000
121	2	2	100.000
122	2	2	100.000
123	2	2	100.000
124	2	2	100.000
125	2	2	100.000
126	2	2	100.000
127	2	2	100.000
128	2	2	100.000
129	2	2	100.000
130	2	2	100.000
131	2	2	100.000
132	2	2	100.000
133	2	2	99.9999
134	2	2	100.000
135	2	2	100.000
136	2	2	100.000
137	2	2	100.000
138	2	2	100.000
139	2	2	100.000
140	2	2	100.000
141	2	2	100.000
142	2	2	100.000
143	2	2	100.000
144	2	2	100.000
145	2	2	100.000
146	2	2	100.000
147	2	2	100.000
148	2	2	100.000
149	2	2	99.9992
150	3	3	100.000
151	3	3	100.000
152	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
153	3	3	100.000
154	3	3	100.000
155	3	3	100.000
156	3	3	100.000
157	3	3	100.000
158	3	3	100.000
159	3	3	100.000
160	3	3	100.000
161	3	3	100.000
162	3	3	100.000
163	3	3	100.000
164	3	3	100.000
165	3	3	100.000
166	3	3	99.9992
167	3	3	100.000
168	3	3	100.000
169	3	3	100.000
170	3	3	100.000
171	3	3	100.000
172	3	3	100.000
173	3	3	100.000
174	3	3	100.000
175	3	3	100.000
176	3	3	100.000
177	3	3	100.000
178	3	3	100.000
179	3	3	100.000
180	3	4	64.8937
181	3	3	99.9995
182	3	3	100.000
183	3	3	100.000
184	3	3	100.000
185	3	3	100.000
186	3	3	100.000
187	3	3	100.000
188	3	3	100.000
189	3	3	100.000
190	3	3	99.9999
191	3	3	100.000
192	3	3	100.000
193	3	3	100.000
194	3	3	100.000
195	3	3	100.000
196	3	3	100.000
197	3	3	100.000
198	3	3	100.000
199	3	3	100.000
200	4	4	89.9241
201	4	1	92.4206
202	4	4	88.1884
203	4	4	100.000
204	4	4	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
205	4	4	69.7333
206	4	4	99.9972
207	4	4	99.9885
208	4	1	98.9614
209	4	4	99.9639
210	4	4	59.3819
211	4	4	93.8425
212	4	4	99.4708
213	4	4	100.000
214	4	4	99.9991
215	4	4	93.7840
216	4	4	99.9036
217	4	4	54.1504
218	4	4	89.7449
219	4	4	99.9992
220	4	4	99.9525
221	4	4	85.9555
222	4	4	58.5011
223	4	4	94.5694
224	4	4	84.6374
225	4	4	75.8654
226	4	4	100.000
227	4	4	86.4111
228	4	4	99.9975
229	4	4	100.000
230	4	4	71.0701
231	4	1	96.3326
232	4	4	99.8946
233	4	4	92.2467
234	4	4	99.2111
235	4	4	99.9822
236	4	1	51.3688
237	4	4	75.8755
238	4	4	98.8195
239	4	4	99.9668
240	4	4	97.7824
241	4	4	99.9538
242	4	4	99.4260
243	4	1	79.4528
244	4	4	99.9228
245	4	4	99.9997
246	4	4	99.8621
247	4	4	97.3662
248	4	4	99.4949
249	4	1	71.8400

C.3.3 Transients with Pink Noise of 10 dB as Validation Set

The validation set consists of 250 noisy transients, which are from the training set contaminated by pink noise of 10 dB with the detail as following.

- Voltage sags (labelled class 0): transient number from 0 to 49;
- Impulsive transient (labelled class 1): transient number from 50 to 99;
- Momentary interruption ((labelled class 2): transient number from 100 to 149;
- Voltage swells (labelled class 3): transient number from 150 to 199; and
- Oscillatory transient (labelled class 4): transient number from 200 to 249.

List C.3.3 shows the classification results by the PNN for all cases in this set.

List C.3.3 Transients with pink noise of 10 dB as validation set

Transient #	Original Class	Predicted Class	Bayesian Confidence (%)
0	0	0	100.000
1	0	0	100.000
2	0	0	100.000
3	0	0	100.000
4	0	0	100.000
5	0	0	100.000
6	0	0	100.000
7	0	0	100.000
8	0	0	100.000
9	0	0	100.000
10	0	0	100.000
11	0	0	100.000
12	0	0	100.000
13	0	0	100.000
14	0	0	100.000
15	0	0	100.000
16	0	0	100.000
17	0	0	100.000
18	0	0	100.000
19	0	0	100.000
20	0	0	100.000
21	0	0	100.000
22	0	0	100.000
23	0	0	99.9998
24	0	0	100.000
25	0	0	100.000
26	0	0	100.000
27	0	0	100.000
28	0	0	100.000
29	0	0	100.000
30	0	0	100.000
31	0	0	100.000
32	0	0	100.000
33	0	0	100.000
34	0	0	99.9958
35	0	0	99.9989
36	0	0	100.000
37	0	0	100.000
38	0	0	100.000
39	0	0	100.000
40	0	0	99.9995
41	0	0	100.000
42	0	0	100.000
43	0	0	100.000
44	0	0	100.000
45	0	0	100.000
46	0	0	100.000
47	0	0	100.000
48	0	0	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
49	0	0	100.000
50	1	1	99.7118
51	1	4	78.6524
52	1	1	93.8198
53	1	1	81.4810
54	1	1	95.0766
55	1	1	94.7406
56	1	1	91.6747
57	1	1	98.7100
58	1	1	97.6662
59	1	4	56.9804
60	1	1	99.5185
61	1	1	77.1125
62	1	1	90.4946
63	1	1	82.0437
64	1	1	57.4435
65	1	4	67.3877
66	1	1	73.2969
67	1	1	94.6465
68	1	1	63.3324
69	1	1	99.8427
70	1	1	89.9277
71	1	1	99.6300
72	1	1	67.2216
73	1	1	51.4386
74	1	1	91.6223
75	1	1	99.1453
76	1	1	99.8045
77	1	1	87.3065
78	1	1	64.6839
79	1	1	98.6043
80	1	4	50.4161
81	1	1	98.5363
82	1	4	57.0666
83	1	4	51.2536
84	1	1	89.9245
85	1	1	83.3800
86	1	1	97.4379
87	1	1	81.0900
88	1	1	99.6738
89	1	1	62.9497
90	1	1	96.7017
91	1	1	98.8299
92	1	1	99.9539
93	1	1	97.6376
94	1	1	93.0855
95	1	1	58.4536
96	1	1	81.2498
97	1	1	78.7923
98	1	1	56.4690
99	1	1	76.0632
100	2	2	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
101	2	2	100.000
102	2	2	100.000
103	2	2	100.000
104	2	2	100.000
105	2	2	99.9770
106	2	2	100.000
107	2	2	100.000
108	2	2	100.000
109	2	2	100.000
110	2	2	100.000
111	2	2	100.000
112	2	2	100.000
113	2	2	100.000
114	2	0	86.9305
115	2	2	100.000
116	2	2	100.000
117	2	2	100.000
118	2	2	100.000
119	2	2	99.9283
120	2	2	100.000
121	2	2	100.000
122	2	2	99.9994
123	2	2	100.000
124	2	2	100.000
125	2	2	100.000
126	2	2	100.000
127	2	2	100.000
128	2	2	100.000
129	2	2	100.000
130	2	2	100.000
131	2	2	100.000
132	2	2	100.000
133	2	2	95.2890
134	2	2	100.000
135	2	2	100.000
136	2	2	100.000
137	2	2	99.9989
138	2	2	100.000
139	2	2	100.000
140	2	2	99.9930
141	2	2	99.8203
142	2	2	100.000
143	2	2	100.000
144	2	2	100.000
145	2	2	100.000
146	2	2	100.000
147	2	2	100.000
148	2	2	100.000
149	2	0	100.000
150	3	3	100.000
151	3	3	100.000
152	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
153	3	3	100.000
154	3	3	100.000
155	3	3	100.000
156	3	3	100.000
157	3	3	100.000
158	3	3	100.000
159	3	3	100.000
160	3	3	100.000
161	3	3	100.000
162	3	3	100.000
163	3	3	100.000
164	3	3	100.000
165	3	3	100.000
166	3	3	99.9984
167	3	3	100.000
168	3	3	100.000
169	3	3	100.000
170	3	3	100.000
171	3	3	100.000
172	3	3	100.000
173	3	3	100.000
174	3	3	100.000
175	3	3	100.000
176	3	3	100.000
177	3	3	100.000
178	3	3	100.000
179	3	3	100.000
180	3	4	89.9320
181	3	3	100.000
182	3	3	100.000
183	3	3	100.000
184	3	3	100.000
185	3	3	100.000
186	3	3	100.000
187	3	3	100.000
188	3	3	100.000
189	3	3	100.000
190	3	3	100.000
191	3	3	100.000
192	3	3	100.000
193	3	3	100.000
194	3	3	100.000
195	3	3	100.000
196	3	3	100.000
197	3	3	100.000
198	3	3	100.000
199	3	3	100.000
200	4	1	96.2962
201	4	1	93.8225
202	4	1	91.8260
203	4	4	98.5906
204	4	4	99.8186

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
205	4	1	83.2826
206	4	4	66.1256
207	4	4	98.8672
208	4	1	99.5206
209	4	4	91.1837
210	4	1	60.9330
211	4	4	52.5972
212	4	4	86.8112
213	4	1	91.8142
214	4	4	99.9622
215	4	4	75.4802
216	4	4	98.4683
217	4	1	97.8709
218	4	1	62.8730
219	4	4	99.9563
220	4	4	60.4207
221	4	1	57.8927
222	4	1	97.3092
223	4	1	91.1076
224	4	1	61.3011
225	4	1	59.1691
226	4	4	100.000
227	4	1	56.6662
228	4	4	99.5685
229	4	1	57.1588
230	4	1	81.6834
231	4	1	81.4674
232	4	4	81.8133
233	4	1	55.9194
234	4	1	91.2418
235	4	4	99.7589
236	4	1	93.6003
237	4	1	61.7209
238	4	4	80.7142
239	4	1	86.8922
240	4	1	84.8288
241	4	4	97.8787
242	4	4	92.5589
243	4	1	99.6080
244	4	4	85.5801
245	4	4	99.8860
246	4	4	99.3895
247	4	1	51.5132
248	4	4	82.1388
249	4	1	72.1108

C.4 Testing the PNN on Transients Contaminated by Brown Noise

C.4.1 Transients with Brown Noise of 0.25 dB as Validation Set

The validation set consists of 250 noisy transients, which are from the training set contaminated by brown noise of 0.25 dB with the detail as following.

- Voltage sags (labelled class 0): transient number from 0 to 49;
- Impulsive transient (labelled class 1): transient number from 50 to 99;
- Momentary interruption ((labelled class 2): transient number from 100 to 149;
- Voltage swells (labelled class 3): transient number from 150 to 199; and
- Oscillatory transient (labelled class 4): transient number from 200 to 249.

List C.4.1 shows the classification results by the PNN for all cases in this set.

List C.4.1 Transients with brown noise of 0.25 dB as validation set

Transient #	Original Class	Predicted Class	Bayesian Confidence (%)
0	0	0	100.000
1	0	0	100.000
2	0	0	100.000
3	0	0	100.000
4	0	0	100.000
5	0	0	100.000
6	0	0	100.000
7	0	0	100.000
8	0	0	100.000
9	0	0	100.000
10	0	0	100.000
11	0	0	100.000
12	0	0	100.000
13	0	0	100.000
14	0	0	100.000
15	0	0	100.000
16	0	0	100.000
17	0	0	100.000
18	0	0	100.000
19	0	0	100.000
20	0	0	100.000
21	0	0	100.000
22	0	0	100.000
23	0	0	100.000
24	0	0	100.000
25	0	0	100.000
26	0	0	100.000
27	0	0	100.000
28	0	0	100.000
29	0	0	100.000
30	0	0	100.000
31	0	0	100.000
32	0	0	100.000
33	0	0	100.000
34	0	0	99.9987
35	0	0	100.000
36	0	0	100.000
37	0	0	100.000
38	0	0	100.000
39	0	0	100.000
40	0	0	100.000
41	0	0	100.000
42	0	0	100.000
43	0	0	100.000
44	0	0	100.000
45	0	0	100.000
46	0	0	100.000
47	0	0	100.000
48	0	0	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
49	0	0	100.000
50	1	1	100.000
51	1	1	98.8953
52	1	1	99.9999
53	1	1	99.9561
54	1	1	99.9655
55	1	1	98.1710
56	1	1	99.9988
57	1	1	100.000
58	1	1	99.9957
59	1	1	95.9741
60	1	1	99.9900
61	1	1	99.9994
62	1	1	100.000
63	1	1	99.9971
64	1	1	99.9977
65	1	1	99.6493
66	1	1	99.9742
67	1	1	100.000
68	1	1	99.9898
69	1	1	99.9995
70	1	1	99.9974
71	1	1	100.000
72	1	1	99.9999
73	1	1	99.9903
74	1	1	100.000
75	1	1	100.000
76	1	1	100.000
77	1	1	99.9990
78	1	1	99.8982
79	1	1	99.9667
80	1	1	99.9977
81	1	1	100.000
82	1	1	70.4586
83	1	1	99.9936
84	1	1	99.9822
85	1	1	99.9234
86	1	1	99.9995
87	1	1	99.2893
88	1	1	99.0584
89	1	1	99.8832
90	1	1	99.9769
91	1	1	99.9999
92	1	1	100.000
93	1	1	100.000
94	1	1	100.000
95	1	1	99.1944
96	1	1	99.9920
97	1	1	100.000
98	1	1	99.9973
99	1	1	99.0306
100	2	2	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
101	2	2	100.000
102	2	2	100.000
103	2	2	100.000
104	2	2	100.000
105	2	2	100.000
106	2	2	100.000
107	2	2	100.000
108	2	2	100.000
109	2	2	99.9998
110	2	2	100.000
111	2	2	100.000
112	2	2	100.000
113	2	2	100.000
114	2	2	100.000
115	2	2	100.000
116	2	2	100.000
117	2	2	100.000
118	2	2	100.000
119	2	2	100.000
120	2	2	100.000
121	2	2	100.000
122	2	2	100.000
123	2	2	100.000
124	2	2	100.000
125	2	2	100.000
126	2	2	100.000
127	2	2	100.000
128	2	2	100.000
129	2	2	100.000
130	2	2	100.000
131	2	2	100.000
132	2	2	100.000
133	2	2	98.4917
134	2	2	100.000
135	2	2	100.000
136	2	2	100.000
137	2	2	100.000
138	2	2	100.000
139	2	2	100.000
140	2	2	100.000
141	2	2	100.000
142	2	2	100.000
143	2	2	100.000
144	2	2	100.000
145	2	2	100.000
146	2	2	100.000
147	2	2	100.000
148	2	2	100.000
149	2	2	100.000
150	3	3	100.000
151	3	3	100.000
152	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
153	3	3	100.000
154	3	3	100.000
155	3	3	100.000
156	3	3	100.000
157	3	3	100.000
158	3	3	100.000
159	3	3	100.000
160	3	3	100.000
161	3	3	100.000
162	3	3	100.000
163	3	3	100.000
164	3	3	100.000
165	3	3	100.000
166	3	3	100.000
167	3	3	100.000
168	3	3	100.000
169	3	3	100.000
170	3	3	100.000
171	3	3	100.000
172	3	3	100.000
173	3	3	100.000
174	3	3	100.000
175	3	3	100.000
176	3	3	100.000
177	3	3	100.000
178	3	3	100.000
179	3	3	100.000
180	3	3	99.8829
181	3	3	100.000
182	3	3	100.000
183	3	3	100.000
184	3	3	100.000
185	3	3	100.000
186	3	3	100.000
187	3	3	100.000
188	3	3	100.000
189	3	3	100.000
190	3	3	100.000
191	3	3	100.000
192	3	3	100.000
193	3	3	100.000
194	3	3	100.000
195	3	3	100.000
196	3	3	100.000
197	3	3	100.000
198	3	3	100.000
199	3	3	100.000
200	4	4	99.4298
201	4	4	99.8946
202	4	4	99.9051
203	4	4	100.000
204	4	4	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
205	4	4	99.9998
206	4	4	100.000
207	4	4	99.9999
208	4	4	99.9609
209	4	4	100.000
210	4	4	99.9929
211	4	4	99.9950
212	4	4	99.9960
213	4	4	100.000
214	4	4	100.000
215	4	4	99.9498
216	4	4	99.7823
217	4	4	99.9557
218	4	4	99.9969
219	4	4	100.000
220	4	4	99.9999
221	4	4	99.6440
222	4	4	99.2898
223	4	4	99.9995
224	4	4	99.9918
225	4	4	99.9887
226	4	4	100.000
227	4	4	99.9858
228	4	4	100.000
229	4	4	100.000
230	4	4	99.9726
231	4	4	68.9935
232	4	4	99.9952
233	4	4	99.9930
234	4	4	100.000
235	4	4	99.4022
236	4	4	99.9280
237	4	4	99.9999
238	4	4	99.9661
239	4	4	100.000
240	4	4	93.7082
241	4	4	98.8266
242	4	4	99.9716
243	4	4	90.8086
244	4	4	100.000
245	4	4	100.000
246	4	4	99.9755
247	4	4	100.000
248	4	4	100.000
249	4	4	97.4276

C.4.2 Transients with Brown Noise of -5 dB as Validation Set

The validation set consists of 250 noisy transients, which are from the training set contaminated by brown noise of -5 dB with the detail as following.

- Voltage sags (labelled class 0): transient number from 0 to 49;
- Impulsive transient (labelled class 1): transient number from 50 to 99;
- Momentary interruption ((labelled class 2): transient number from 100 to 149;
- Voltage swells (labelled class 3): transient number from 150 to 199; and
- Oscillatory transient (labelled class 4): transient number from 200 to 249.

List C.4.2 shows the classification results by the PNN for all cases in this set.

List C.4.2 Transients with brown noise of -5 dB as validation set

Transient #	Original Class	Predicted Class	Bayesian Confidence (%)
0	0	0	100.000
1	0	0	100.000
2	0	0	100.000
3	0	0	100.000
4	0	0	100.000
5	0	0	100.000
6	0	0	100.000
7	0	0	100.000
8	0	0	100.000
9	0	0	100.000
10	0	0	100.000
11	0	0	100.000
12	0	0	100.000
13	0	0	100.000
14	0	0	100.000
15	0	0	100.000
16	0	0	100.000
17	0	0	100.000
18	0	0	100.000
19	0	0	100.000
20	0	0	100.000
21	0	0	100.000
22	0	0	100.000
23	0	0	100.000
24	0	0	100.000
25	0	0	100.000
26	0	0	100.000
27	0	0	100.000
28	0	0	100.000
29	0	0	100.000
30	0	0	100.000
31	0	0	100.000
32	0	0	100.000
33	0	0	100.000
34	0	0	99.9955
35	0	0	100.000
36	0	0	100.000
37	0	0	100.000
38	0	0	100.000
39	0	0	99.9999
40	0	0	100.000
41	0	0	100.000
42	0	0	100.000
43	0	0	100.000
44	0	0	100.000
45	0	0	99.9999
46	0	0	100.000
47	0	0	100.000
48	0	0	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
49	0	0	100.000
50	1	1	100.000
51	1	1	96.8583
52	1	1	99.9976
53	1	1	93.5905
54	1	1	99.9982
55	1	1	93.5699
56	1	1	99.9745
57	1	1	100.000
58	1	1	99.9995
59	1	4	51.8375
60	1	1	99.9990
61	1	1	99.7224
62	1	1	100.000
63	1	1	99.9882
64	1	1	96.6523
65	1	1	86.6038
66	1	1	99.8911
67	1	1	99.9996
68	1	1	99.3497
69	1	1	100.000
70	1	1	99.9937
71	1	1	100.000
72	1	1	99.9999
73	1	1	90.8490
74	1	1	100.000
75	1	1	100.000
76	1	1	100.000
77	1	1	99.9995
78	1	1	99.2366
79	1	1	99.9896
80	1	1	98.6934
81	1	1	99.9993
82	1	4	59.7074
83	1	1	99.9958
84	1	1	99.8548
85	1	1	99.5733
86	1	1	99.9969
87	1	1	99.7765
88	1	1	99.2238
89	1	1	99.1475
90	1	1	99.9982
91	1	1	100.000
92	1	1	100.000
93	1	1	99.9998
94	1	1	99.9998
95	1	1	98.0792
96	1	1	99.9765
97	1	1	99.9999
98	1	1	99.9652
99	1	1	96.8200
100	2	2	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
101	2	2	100.000
102	2	2	100.000
103	2	2	100.000
104	2	2	100.000
105	2	2	99.9004
106	2	2	100.000
107	2	2	100.000
108	2	2	100.000
109	2	2	83.1047
110	2	2	100.000
111	2	2	100.000
112	2	2	100.000
113	2	2	99.9999
114	2	2	99.9999
115	2	2	100.000
116	2	2	100.000
117	2	2	100.000
118	2	2	100.000
119	2	2	100.000
120	2	2	100.000
121	2	2	100.000
122	2	2	100.000
123	2	2	100.000
124	2	2	100.000
125	2	2	100.000
126	2	2	99.9966
127	2	2	100.000
128	2	2	100.000
129	2	2	100.000
130	2	2	100.000
131	2	2	100.000
132	2	2	100.000
133	2	2	97.4089
134	2	2	100.000
135	2	2	100.000
136	2	2	100.000
137	2	2	99.6039
138	2	2	100.000
139	2	2	100.000
140	2	2	98.6180
141	2	2	100.000
142	2	2	100.000
143	2	2	100.000
144	2	2	100.000
145	2	2	99.8038
146	2	2	100.000
147	2	2	100.000
148	2	2	100.000
149	2	2	100.000
150	3	3	100.000
151	3	3	100.000
152	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
153	3	3	100.000
154	3	3	100.000
155	3	3	100.000
156	3	3	100.000
157	3	3	100.000
158	3	3	100.000
159	3	3	100.000
160	3	3	100.000
161	3	3	100.000
162	3	3	100.000
163	3	3	100.000
164	3	3	100.000
165	3	3	100.000
166	3	3	100.000
167	3	3	100.000
168	3	3	100.000
169	3	3	100.000
170	3	3	100.000
171	3	3	100.000
172	3	3	100.000
173	3	3	100.000
174	3	3	100.000
175	3	3	100.000
176	3	3	100.000
177	3	3	100.000
178	3	3	100.000
179	3	3	100.000
180	3	3	87.3946
181	3	3	100.000
182	3	3	100.000
183	3	3	100.000
184	3	3	100.000
185	3	3	100.000
186	3	3	100.000
187	3	3	100.000
188	3	3	100.000
189	3	3	100.000
190	3	3	100.000
191	3	3	100.000
192	3	3	100.000
193	3	3	100.000
194	3	3	100.000
195	3	3	100.000
196	3	3	100.000
197	3	3	100.000
198	3	3	100.000
199	3	3	100.000
200	4	4	99.8180
201	4	4	68.7249
202	4	4	99.6884
203	4	4	100.000
204	4	4	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
205	4	4	99.9971
206	4	4	100.000
207	4	4	99.9987
208	4	1	77.1977
209	4	4	99.9999
210	4	4	99.9818
211	4	4	99.9856
212	4	4	99.9995
213	4	4	100.000
214	4	4	100.000
215	4	4	98.4977
216	4	4	99.2912
217	4	4	99.7342
218	4	4	99.9848
219	4	4	100.000
220	4	4	100.000
221	4	4	98.8828
222	4	4	94.8280
223	4	4	99.9968
224	4	4	99.9879
225	4	4	99.6804
226	4	4	100.000
227	4	4	99.9761
228	4	4	100.000
229	4	4	100.000
230	4	4	99.4933
231	4	1	99.4263
232	4	4	99.9552
233	4	4	99.9958
234	4	4	99.9998
235	4	1	96.6753
236	4	4	99.5530
237	4	4	99.9937
238	4	4	99.9747
239	4	4	99.9919
240	4	4	99.7973
241	4	4	95.9670
242	4	4	99.9682
243	4	4	80.4829
244	4	4	100.000
245	4	4	100.000
246	4	4	99.9512
247	4	4	99.9998
248	4	4	100.000
249	4	4	65.9432

C.4.3 Transients with Brown Noise of -10 dB as Validation Set

The validation set consists of 250 noisy transients, which are from the training set contaminated by brown noise of -10 dB with the detail as following.

- Voltage sags (labelled class 0): transient number from 0 to 49;
- Impulsive transient (labelled class 1): transient number from 50 to 99;
- Momentary interruption ((labelled class 2): transient number from 100 to 149;
- Voltage swells (labelled class 3): transient number from 150 to 199; and
- Oscillatory transient (labelled class 4): transient number from 200 to 249.

List C.4.3 shows the classification results by the PNN for all cases in this set.

List C.4.3 Transients with brown noise of -10 dB as validation set

Transient #	Original Class	Predicted Class	Bayesian Confidence (%)
0	0	0	100.000
1	0	0	99.9786
2	0	0	100.000
3	0	0	100.000
4	0	0	100.000
5	0	0	100.000
6	0	0	100.000
7	0	0	100.000
8	0	0	100.000
9	0	0	100.000
10	0	0	100.000
11	0	0	100.000
12	0	0	100.000
13	0	0	100.000
14	0	0	100.000
15	0	0	100.000
16	0	0	100.000
17	0	0	99.9994
18	0	0	100.000
19	0	0	100.000
20	0	0	100.000
21	0	0	100.000
22	0	0	100.000
23	0	0	100.000
24	0	0	100.000
25	0	0	99.9996
26	0	0	100.000
27	0	0	100.000
28	0	0	100.000
29	0	0	99.9689
30	0	0	99.5869
31	0	0	100.000
32	0	0	100.000
33	0	0	99.9999
34	0	0	99.9996
35	0	0	99.9298
36	0	0	100.000
37	0	0	99.9998
38	0	0	100.000
39	0	0	100.000
40	0	0	100.000
41	0	0	99.9990
42	0	0	100.000
43	0	0	100.000
44	0	0	100.000
45	0	0	99.9550
46	0	0	99.9999
47	0	0	100.000
48	0	0	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
49	0	0	100.000
50	1	1	99.9998
51	1	1	92.5646
52	1	1	99.8326
53	1	1	72.5992
54	1	1	99.9929
55	1	1	75.6409
56	1	1	99.2716
57	1	1	100.000
58	1	1	99.9952
59	1	4	56.5103
60	1	1	99.9978
61	1	4	51.9848
62	1	1	100.000
63	1	1	99.9005
64	1	1	86.7779
65	1	4	88.0091
66	1	1	87.9902
67	1	1	99.9920
68	1	1	97.4605
69	1	1	100.000
70	1	1	99.9671
71	1	1	100.000
72	1	1	99.9985
73	1	1	71.6430
74	1	1	99.9998
75	1	1	99.9961
76	1	1	99.9998
77	1	1	99.9954
78	1	1	77.4549
79	1	1	99.9963
80	1	4	57.5180
81	1	1	99.9313
82	1	4	77.2414
83	1	1	99.9709
84	1	1	98.9546
85	1	1	94.0681
86	1	1	99.6289
87	1	1	99.9529
88	1	1	99.9972
89	1	1	97.2400
90	1	1	99.9693
91	1	1	100.000
92	1	1	100.000
93	1	1	99.9457
94	1	1	99.7270
95	1	1	94.0985
96	1	1	99.9435
97	1	1	99.9914
98	1	1	88.3614
99	1	1	92.1910
100	2	2	99.9985

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
101	2	2	96.6459
102	2	2	100.000
103	2	2	100.000
104	2	2	100.000
105	2	0	80.9301
106	2	2	99.9770
107	2	2	100.000
108	2	2	100.000
109	2	0	99.9917
110	2	2	100.000
111	2	2	100.000
112	2	2	100.000
113	2	2	99.8169
114	2	0	86.4640
115	2	2	100.000
116	2	2	98.0412
117	2	2	100.000
118	2	2	100.000
119	2	0	95.4354
120	2	2	100.000
121	2	2	100.000
122	2	2	99.9514
123	2	2	100.000
124	2	2	100.000
125	2	2	99.9356
126	2	2	99.9488
127	2	2	100.000
128	2	2	100.000
129	2	2	100.000
130	2	2	100.000
131	2	2	100.000
132	2	2	99.9997
133	2	0	85.6895
134	2	2	99.4818
135	2	2	100.000
136	2	2	100.000
137	2	0	97.4493
138	2	2	100.000
139	2	2	100.000
140	2	0	88.2665
141	2	2	100.000
142	2	2	100.000
143	2	2	100.000
144	2	2	100.000
145	2	2	65.2893
146	2	2	100.000
147	2	2	100.000
148	2	2	100.000
149	2	2	60.4275
150	3	3	100.000
151	3	3	100.000
152	3	3	100.000

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
153	3	3	100.000
154	3	3	100.000
155	3	3	100.000
156	3	3	100.000
157	3	3	100.000
158	3	3	100.000
159	3	3	100.000
160	3	3	100.000
161	3	3	100.000
162	3	3	100.000
163	3	3	100.000
164	3	3	100.000
165	3	3	100.000
166	3	3	100.000
167	3	3	100.000
168	3	3	100.000
169	3	3	100.000
170	3	3	100.000
171	3	3	100.000
172	3	3	100.000
173	3	3	100.000
174	3	3	100.000
175	3	3	100.000
176	3	3	100.000
177	3	3	100.000
178	3	3	100.000
179	3	3	100.000
180	3	4	99.4691
181	3	3	100.000
182	3	3	100.000
183	3	3	100.000
184	3	3	100.000
185	3	3	100.000
186	3	3	100.000
187	3	3	100.000
188	3	3	100.000
189	3	3	100.000
190	3	3	100.000
191	3	3	100.000
192	3	3	100.000
193	3	3	100.000
194	3	3	100.000
195	3	3	100.000
196	3	3	100.000
197	3	3	100.000
198	3	3	100.000
199	3	3	100.000
200	4	4	99.2368
201	4	1	99.9396
202	4	4	91.9479
203	4	4	100.000
204	4	4	99.9998

Transient No.	Original Class	Predicted Class	Bayesian Confidence (%)
205	4	4	99.5791
206	4	4	100.000
207	4	4	98.9220
208	4	1	97.6658
209	4	4	99.9992
210	4	4	80.6028
211	4	4	99.5162
212	4	4	99.9899
213	4	4	100.000
214	4	4	99.9988
215	4	4	60.6808
216	4	4	95.6332
217	4	1	50.3271
218	4	4	97.3941
219	4	4	100.000
220	4	4	99.9999
221	4	4	92.4892
222	4	4	56.0252
223	4	4	99.3407
224	4	4	99.4366
225	4	4	96.7890
226	4	4	99.9985
227	4	4	99.7467
228	4	4	99.9997
229	4	4	100.000
230	4	4	96.9827
231	4	1	96.9975
232	4	4	92.8753
233	4	4	99.6436
234	4	4	99.9585
235	4	1	97.4907
236	4	4	64.8158
237	4	4	99.2953
238	4	4	99.7654
239	4	1	99.3623
240	4	4	95.5394
241	4	4	67.6847
242	4	4	99.8527
243	4	1	71.7648
244	4	4	100.000
245	4	4	99.9998
246	4	4	99.7409
247	4	4	99.9799
248	4	4	99.9999
249	4	1	86.6205