A Fast Radio Transmitter Identification System

By

Luotao Sun

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 PAGE

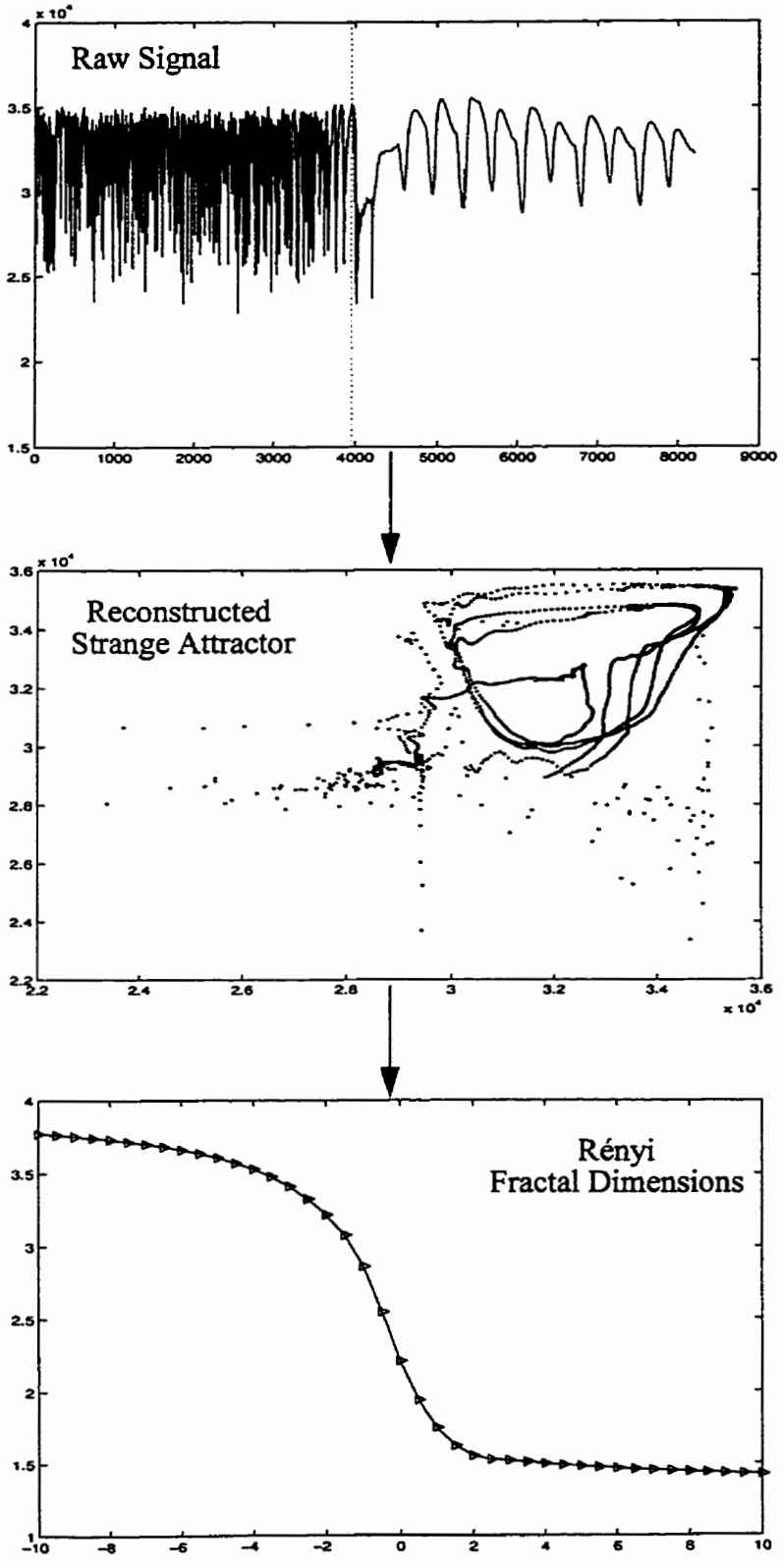
A Fast Radio Transmission Identification System

BY

Luotao Sun

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

MASTER OF SCIENCE


Raw Signal
Reconstructed
Strange Attractor
Rényi
Fractal Dimensions

ABSTRACT

This thesis is concerned with fast, reliable classification and identification of radio transmitters based on an analysis of their turn-on transients. Since the transients are unique, they are called the fingerprints of the corresponding radio transmitters.

Major tasks in a radio transmitter identification system are: separating a transient from the channel noise; extracting important features contained in the transient; and classifying the transient based on these features. A system developed in this thesis achieves noise segmentation using the variance fractal dimension trajectory, with a modified triggering technique which improves the segmentation consistency. A variance fractal amplification technique is used for feature enhancement. Multifractal modelling of transients based on their strange attractors is also used in this thesis to investigate the suitability of such compact representations in transient classification. Preprocessing of the probabilistic neural network (PNN) using the principal component analysis (PCA) and the self-organizing feature map (SOFM) is performed to reduce the dimensionality of the PNN inputs and to cluster inputs, thus improving the training and classification speed.

Experimental results show that the radio transmitter identification system is faster than the previous implementations. More specifically, the training time for a network consisting of 400 transients can be reduced to about a half of the original training time, 241 seconds. The system can classify radio transmitters not only according to their manufacturers and models, but also serial numbers, with the average classification rate around 97%.

ACKNOWLEDGMENTS

First I would like to thank my advisor, Dr. W. Kinsner for introducing this interesting research topic to me and giving me his time and support throughout the course of this thesis.

I would like to thank Dr. N. Serinken of the Communications Research Centre (CRC) for valuable discussions and inputs, and for providing a computer and all the transients used in this research. I would also like to acknowledge the financial support provided by the CRC during a part of this work.

The friendship and discussions provided by my colleagues in the Delta Research Group are also acknowledged. Special thanks to Richard Dansereau, Alexis Denis, Tina Ehtiati, and Rasekh Rifaat for their help during the writing of this thesis.

I would like to thank everyone in my family for their constant support, encouragement and patience. This thesis is devoted to my mother, Liling Sun.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BP	back-propagation
CRC	Communications Research Centre
DWT	discreet wavelet transform
GUI	graphical user interface
MLFN	multiple layer feedforward network
PCA	principal component analysis
pdf	probability density function
PNN	probabilistic neural networks
RF	radio frequency
SDI	single document interface
SOFM	self-organizing feature map
STFT	short-time Fourier transform
TAC-MM	Transient Analyser and Classifier using Multifractal Modelling
vels	volume elements, also known as hypervolume
WT	wavelet transform

LIST OF SYMBOLS

$\alpha(t)$	learning rate of a neural network
η	adjustment parameter for relative difference triggering
$B(t)$	continuous temporal signal
$B(t_j)$	sampled temporal signal
C	autocorrelation function
Cov	covariance matrix
$C_r^{(q)}$	generalized correlation integral
δ	time delay
$d(\mathbf{x}, \mathbf{x}_r)^2$	squared Euclidean distance between an unknown sample, $\mathbf{x}$, and the training case, $\mathbf{x}_r$
Δt	time increment
D	fractal dimension
D_I	information dimension
D_C	correlation dimension
D_q	generalized (Rényi) dimension
D_σ	variance dimension
$D_\sigma(t)$	variance dimension trajectory
E	embedding Euclidean dimension
$f_c(\mathbf{x})$	pdf for a class c
$g_c(x)$	estimated pdf for a class c
h_c	prior probability of a class c
$h_{c(x),i}$	neighbourhood function in the SOFM
H	Hurst exponent
H_1	Shannon's entropy
H_2	the second-order entropy
H_q	Rényi generalized entropy
l_c	loss function associated with misclassifying a sample of the class c
L	diagonal matrix whose diagonal values equal to the eigenvalues of the weight matrix, **W**
m	embedding dimension
$\mathbf{m}_i$	reference vector on the self-organized map
M	mutual information

$\mu_{D_\sigma(t)}$	mean of variance dimensions
n_c	number of training cases for a given class, c
N	number of samples taken
N_c	topological neighbourhood for a self-organized map
N_v	number of vels (volume elements) with a size of v
p	number of principal components
p_j	relative frequency
P_c	classification rate of the PNN
q	moment order in the generalized entropy function
s	slope of a log-log plot
$\Theta(x)$	the Heaviside step function
τ	threshold for triggering transient
σ	width of a function
σ^2	variance of a sampled temporal signal's amplitude
$\sigma_{D_\sigma(t)}$	standard deviation of a variance dimension trajectory
v	size of vel
$\mathbf{w}_i$	weight vector associated with neuron i
$\mathbf{W}$	weight matrix
$W(x)$	weight function
$\mathbf{x}$	neural network input vector
z_i	the value of a pattern neuron i

CHAPTER I
INTRODUCTION

1.1 Background and Motivation

Classification and identification of radio transmitters is very important because of the increasing number of cases where the electromagnetic spectrum is used illegally or improperly. It has been reported that radio transmitters are used to interfere with other transmitters, resulting in either a security problem, or radio frequency (RF) traffic jams. In addition, locating a radio transmitter geographically can be very valuable for military use.

Many studies have been conducted on the classification and identification of unknown radio transmitters by analyzing *turn-on transient signals* from the transmitters [CPYS95] [HiPa96] [Paya95] [Shaw97] [Toon97]. The transient signal is generated when the user pushes the push-to-talk button to activate the transmitter, its phase-locked loop encounters a transition from some initial frequency to the pre-defined carrier frequency. The circuit's response to this discontinuity generates frequency and amplitude transients which decay to a final steady-state condition. These transient signals are called the finger-prints of radio transmitters in that they exhibit characteristics unique to the transmitters. The uniqueness is due to both different designs and different tolerances of components used in the transmitters. The procedure of identifying a radio transmitter automatically is called the *fingerprinting* of the transmitter. The concept of transmitter transient finger-printing was proposed by Kinsner starting from 1993 and has been studied in his research

group on algorithmic, hardware and software aspects [Diet94], [Ruda94], [Shaw94], [Ande95], [Khan95], [Kwok95], [Toon95], [Toon97], [Shaw97].

The structure of a transmitter fingerprinting system is shown in Fig. 1.1.

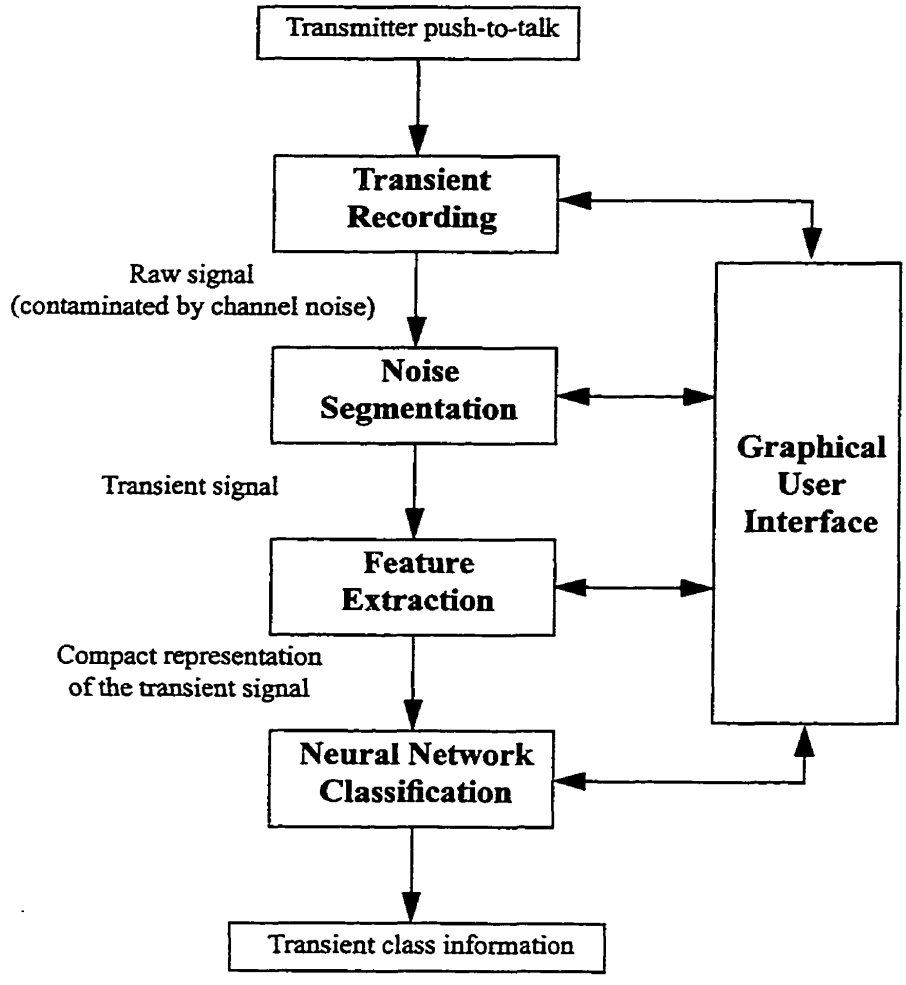


Fig. 1.1. The structure of a radio transmitter fingerprinting system.

This thesis focuses mainly on the following modules: *noise segmentation*, *feature extraction*, *neural network classification*, and *graphical user interface design*. A brief overview of recent effort in dealing with these issues is presented next.

In general, real-time transmitter fingerprinting is very challenging for the following reasons [CPYS95]:

a) Short duration of the radio transmitter transients makes classical frequency analysis difficult;

b) Nonstationary nature of transients; and

c) Wide interclass variation due to large variations in the structures and systems generating the transients.

For noise segmentation, variance fractal dimension trajectory analysis has proved to be very effective [Shaw97] because noise and transient are distinguishable in terms of fractality. To extract features from the transient signal, the *short-time Fourier transform* (STFT) [Kola98], the *discrete wavelet transform* (DWT) [CPYS95] [HiPa96] [Paya95] [Toon97] and the *variance fractal amplification* [Kins94a] [Shaw97] are employed. The STFT technique is limited in providing successful real-time, robust, consistent results. It also suffers from the artifact of window selection [Kins94a]. The DWT is a better approach than the STFT by allowing localization in both frequency and time. The DWT also provides the flexibility to choose the particular wavelet function for a specific application. The problem with the DWT is that the computational overhead is quite large, and since the number of extracted features can be very large, it is not suitable for the near real-time speed requirement in our case. The variance fractal amplification has demonstrated its ability to characterize nonstationary signals such as transients, while the computation overhead is reasonable, or even in real-time [Kins94b]. *Back-propagation* (BP) neural networks [Toon97], and *probabilistic neural networks* (PNN) [Shaw97] are used for classifi-

cation purposes. The PNN is preferred to the BPNN in this thesis mainly because of its fast training and well-defined architecture.

Still, some problems with these approaches exist. More specifically, the following issues are the major concerns in this thesis:

a) Inconsistent noise segmentation: The variance fractal dimension trajectory technique can separate the noise from the transient if there is a distinct difference between them in the variance fractal dimension domain. In some cases, there is variable noise component in the transient portion due to inconsistent recording, or the transient itself is very complicated. This results in fluctuations on the dimension trajectory around the transition moment where the noise ends and the transient starts. A simple thresholding scheme (e.g., the absolute difference triggering [Shaw97]) may detect the transition incorrectly due to such fluctuations;

b) Inefficient feature extraction: The variance fractal amplification technique relies on the existence of consistent and unique multifractality to produce a sufficient characterization of a signal. When the noise is not separated consistently, as a consequence, there may be some noise remaining in the transient, or the transient is not preserved completely. In this case, feature extraction based on this technique may lead to failure in further classification; and

c) Slow training speed with a large training set: The training speed of the PNN is faster than that of the BPNN. However, when a large number of transients are used for training and classification, the PNN training speed can slow down

significantly. This is because the entire training set must be stored in the network and processed during the training and classification. Therefore, it is not suitable for real-time applications [Mast93]. In this thesis, a large network is required to perform complicated classification, thus reducing the PNN training time is very important for practical purposes.

This thesis investigates new techniques to solve the first two problems using *multifractal analysis* for noise segmentation and feature extraction. More specifically, the Rényi generalized dimensions and multifractal characterization of strange attractors are studied. To speed up the training and classification procedure of the PNN, the Kohonen *self-organizing feature maps* (SOFM), and the *principal component analysis* (PCA) techniques are employed to reduce the size of training sets and the dimensionality of PNN inputs, respectively.

1.2 Thesis Statement and Objectives

The general goal to achieve in this thesis is to develop a fast radio transmitter identification system which is able to model and classify the transient signal from a transmitter, thus achieving reliable and fast radio transmitter fingerprinting. Based on the goal as stated, the objectives are:

a) A technique for consistent noise segmentation, especially for the cases where there are fluctuations in the variance fractal dimension trajectory due to the existence of noise or the complexity of transients;

b) A technique for robust feature extraction which focuses on the important transient portion of the signal;

c) An improvement of the PNN processing to achieve faster training; and

d) An interactive, *graphical user interface* (GUI) environment to embed the above processing procedure.

1.3 Thesis Organization

The organization of the thesis reflects the logical development of the thesis work.

Chapter 2 provides the basis of fractal analysis, with emphasis on applications for transient signals. The characteristics of temporal nonstationary signals are discussed first. Then, the concepts of fractals and multifractals are introduced to model such signals. Theory on chaos and dynamic nonlinear systems is also provided and the relationship between chaos and fractals is discussed. The implementation of these procedures for noise segmentation and transient feature extraction is also presented.

Chapter 3 gives an overview of the underlying scheme for PNN classification. An introduction to neural network classification is provided, with the emphasis on the PNN. Some issues related to PNN training, processing, and improvement are discussed. Two techniques of PNN preprocessing are examined, namely the SOFM and the PCA.

Chapter 4 discusses the experimental implementation of the radio transmitter identification system. It describes the major tasks to be accomplished, as well as the software tool design concerns. The specific functions to be implemented in each module are described in detail.

Chapter 5 presents the experimental work and discussion. We begin with an overview of the signals studied in this thesis. The issues related to consistent noise segmentation are discussed in Section 5.2. The next section deals with approaches used for feature extraction. Section 5.4 focuses on the transient classification processing, including preprocessing and classification procedures. Experimental results are discussed in each section and a summary is provided at the end of this chapter.

Chapter 6 provides conclusions based on the experimental results and the contributions made by this thesis. Recommendations for further research are also given.

CHAPTER II
FRACTAL ANALYSIS ON TRANSIENT SIGNALS

2.1 Introduction

This chapter presents the basic ideas about fractal analysis, with emphasis on applications for transient signals. First, the characteristics of nonstationary signals are discussed, and an overview of techniques to model such signals is provided. Then, the concepts of fractals and multifractals are introduced as alternatives to nonstationary signal processing. Fractal analysis of transients is usually conducted in terms of fractal dimensions as a function of time if they are nonstationary, resulting in a fractal trajectory. An even better characterization of transient signals is achieved through time-varying multifractals analysis (multifractal surface). Since transient signals can be considered as outputs from chaotic dynamical systems, theory on chaos and dynamic nonlinear systems is introduced. Multifractal analysis is then applied to such systems in phase space based on strange attractor reconstruction. Noise segmentation is achieved using fractal measures because noise and signal have different fractal complexities, thus resulting in different fractal dimensions. Feature extraction is performed by selecting a set of fractal dimensions as the unique representation of a signal. The implementation of these procedures is also presented.

2.2 Nonstationary Signals and Their Processing

A temporal signal $B(t)$ can be sampled into $B(t_j)$ based on the Nyquist sampling theorem. Each sample is then quantized (16 bits/sample in our study). A set of such

digital samples forms a digital signal, also called a time series, denoted by $\{B(t_j) \mid j = 1, \ldots, N\}$. Here t_j is the time at which the sample $B(t_j)$ takes place and N is the number of samples taken. To simplify the notation, we use B_t interchangeable with $B(t_j)$, where $t = 1, \ldots, N$ denotes the time by sample number. The signal is said to be strict *stationary* if the probabilistic structure of B_t is unaffected by a shift in the time origin [Digg90]. Temporal *nonstationary* signals refer to the signals with time-varying signal statistics [Papo84].

In practice, many signals are nonstationary and many techniques have been developed to analyze them. An overview of these techniques is provided below, which can be categorized into (a) time domain modelling, (b) time-frequency modelling, and (c) fractal modelling.

In time domain modelling, signal processing schemes often assume some consistent and stable properties of the signal under study in order to establish a suitable mathematical model. For nonstationary signals, they are usually modified with some statistical means to fit in the model. These methods include (i) decomposition method; and (ii) differencing (ARIMA model).

The *decomposition method* [BrDa87] assumes that an observed time series B_t can be expressed as

$$B_t = m_t + s_t + y_t \tag{2.1}$$

where m_t is called the *trend component*, s_t is the *seasonal component*, and y_t is the sta-

tionary *random noise component.* The trend component, m_t, can be described in the form of a polynomial as

$$m_t = a_0 + a_1 t + a_2 t^2 + \ldots \tag{2.2}$$

The objective of this method is to estimate and extract the deterministic components m_t and s_t, so that the residual y_t is stationary, and we can find a probabilistic model for y_t to analyze its properties. The major limitation of this method is the difficulty in modelling m_t and s_t from an observed short time series such as transients.

A more flexible technique to trend and seasonality removal is by differencing repeatedly the original time series until we obtain a stationary process. This technique is called *autoregressive integrated moving average* (ARIMA) [BoJe70]. ARIMA models assume that the studied time series can be reduced to stationarity by differencing finitely many times, which then can be modelled by the *autoregressive moving average* process (ARMA). A time series $\{B_t\}$ is said to be an ARMA(p,q) process if $\{B_t\}$ is stationary and if for every t,

$$B_t = \sum_{i=1}^{p} \phi_i B_{t-i} + Z_t + \sum_{j=1}^{q} \theta_j Z_{t-j} \tag{2.3}$$

where the polynomials ϕ and θ are referred to as the autoregressive and moving average polynomials, and $\{Z_t\}$ is a white noise sequence. The limitation with the ARIMA scheme is that it only considers a rather special form of nonstationarity [Prie81] [Prie88]; i.e., the second-order properties of such processes vary over time, but the complete evolu-

tion of the process is governed by a fixed set of parameters. More general classes of non-stationary processes have freely varying time-dependent parameters rather than fixed parameter models. Transient signals, in particular, cannot be well characterized by these parametric signal models because of their high degree of nonstationarity.

The time-frequency modelling approach includes (i) the STFT, and (ii) the Wigner-Ville method, and (iii) the DWT.

The STFT uses a window sliding along the time axis and applies the Fourier transform on the small portion of the time series contained in the window. Thus, once a window has been chosen, time-frequency resolution is fixed. The major limitation with this technique is that it does not allow high time and frequency resolution simultaneously. This is a serious drawback when analyzing transient signals, which are constrained in time but have a wide band in frequency. It also suffers from the artifact of choosing the proper window size.

The Wigner-Ville method is based on the idea that, in theory, an instantaneous correlation function can be computed. The corresponding spectral density function is calculated for each instance. It also has the fundamental limitation of the inherent trade-off between time and frequency resolution [Newl98].

In contrast, the DWT has a variable time-frequency resolution, i.e., a greater time resolution at high frequencies and a smaller resolution at low frequencies. Therefore, it allows good localization in both time and frequency domains. A problem with the DWT is that the computational overhead is quite considerable, and further feature selection is

required on the wavelet coefficients [Toon97]. Therefore, it is not suitable for the near real-time speed requirement of a software implementation in this thesis.

In the next section, the concepts of fractals and multifractals are introduced to provide alternatives to nonstationary signal modelling. Such modellings capture information about the nonstationarity of studied signals in the form of fractal complexity (dimension). It is also found that multifractal analysis, which considers non-constant fractal dimension, is a better approach than the single fractal measure for transient signals.

2.3 Fractals, Fractal Dimensions, and Multifractals

2.3.1 Definitions

The concept of fractal and fractal dimension was introduced and has been studied for the purpose of characterizing complex objects which are difficult to describe by Euclidean geometry, in which non-negative integer values of dimension are used. For example, the Euclidean (or topological) dimension of a line is one, that of a surface is two. The definition of fractal dimensions was motivated by the observations that such values are not appropriate to describe more complex objects. Lets consider an example, the Cantor set. As shown in Fig. 2.1, we start from a line segment (called the initiator) of unit length. The next step is to remove the middle part which is $\frac{1}{3}$ in length, leaving two shortened line segments (thus establishing the generator). The same operation is repeated on these two segments, leading to four segments, each having a length of $\left(\frac{1}{3}\right)^2$. If we keep repeating the process, we will end up with an object consisting of an infinite number of points, each having dimension 0. On the other hand, the infinite number of points could fill out a line, whose dimension would be 1. We can clearly see, however, that the Cantor set

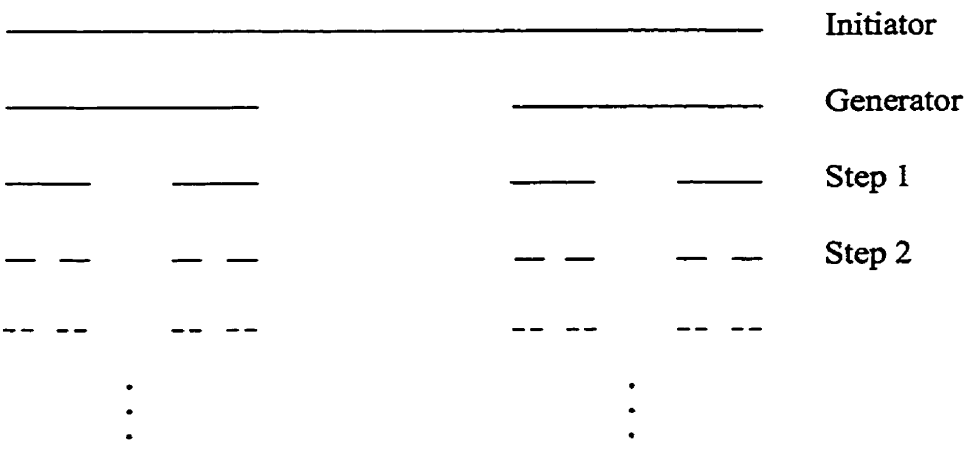


Fig. 2.1. Cantor set (the fractal dimension is 0.63, which is between 0 and 1).

is neither a point nor a line segment. Instead, it is an object with a definite self-similar structure.

It seems the traditional definition of dimension gives us ambiguous results. The ambiguity comes from the detail (or the scale of the object) considered. Studies of such cases led to the generalization of defining dimensions, which takes into account the relationship between the scale and some measurement on the object. Thus, for some complex objects, the dimension values may not necessarily be integer. Fractional dimensions were proposed and studied by L. Brouwer, F. Hausdorff, A. Besivovitch, A. Kolmogorov, and B. Mandelbrot [Kins95], and are referred to as fractal dimensions.

A fractal dimension, D, can be interpreted as the "degree of meandering" (or roughness, or brokenness, or irregularity) of an object [Kins95]. A fractal is a set for which the fractal dimension strictly exceeds the topological dimension [Mand82]. Another interpretation of a fractal dimension is a critical exponent that makes a measure of the

object constant. *Self-similarity* (i.e., invariance against changes in scale or size) is the essence of fractals. It is usually expressed as some power law relationship between the scale and the measurement. Fractal based modelling can be classified into four categories based on the type of fractal dimension one chooses, that is, *morphological, entropy, spectrum and variance based* dimensions [Kins94a]. We will focus mainly on variance based and entropy based generalized dimensions because of our particular application in this thesis.

Variance-based Dimensions

Assume a time series $B(t_j)$, where the sample time t_j is distributed uniformly in time. The variance, σ^2, of its amplitude changes over a time increment, $\Delta t = |t_2 - t_1|$, is related to that time increment according to the following power law

$$Var[B(t_2) - B(t_1)] \sim |t_2 - t_1|^{2H} \tag{2.4}$$

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

$$H = \lim_{\Delta t \to 0} \frac{1}{2} \frac{\log_b[Var(\Delta B)_{\Delta t}]}{\log_b(\Delta t)} \tag{2.5}$$

Finally, the variance dimension can be calculated from

$$D_\sigma = E + 1 - H \tag{2.6}$$

where E is the embedding Euclidean dimension and it is equal to one if the temporal sig-

nal has a single independent variable.

In practice, we translate the *lim* in Eq. 2.2 into the slope of a log-log plot. A finite sequence of time increments, $\{\Delta t_1, \Delta t_2, \ldots, \Delta t_k\}$, is used. Note that the sequence is usually chosen to be b-adic in order to spread the points on the log-log plot equally. The slope s is then determined from these points by a polynomial line fitting method. The underlying assumption is that the measured data points are distributed around a line with errors due to fluctuations which are normally distributed. However, in practice there are upper and lower cutoffs at large and small scales. The determination of the slope is then performed only within an appropriate range, as shown in Fig. 2.2.

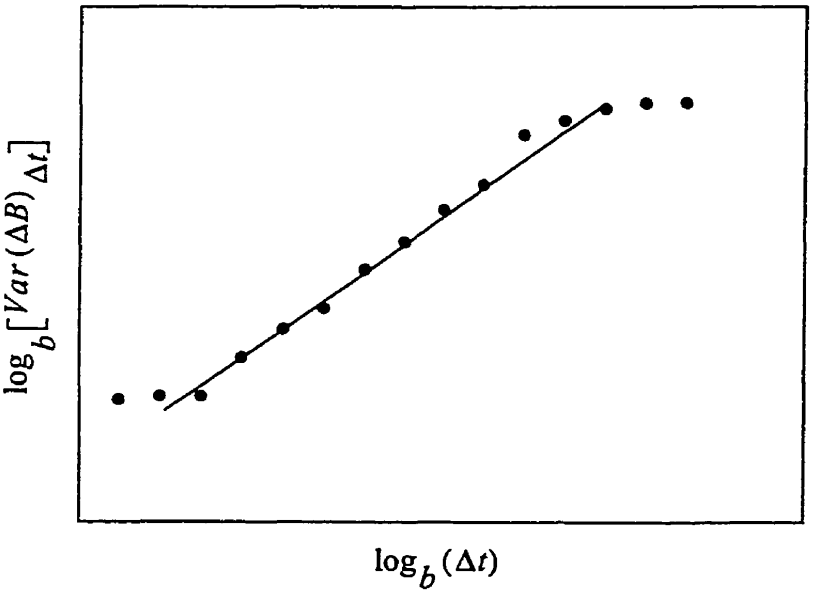


Fig. 2.2. Line-fitting on log-log plot.

Variance and spectrum based dimensions are appropriate for a time series with a single independent variable. For the spectrum-based dimension calculation, artifacts may be introduced when the short-time Fourier transform is performed. In contrast, the vari-

ance-based fractal dimension calculation does not have such artifacts; it operates directly on a time series; and the computation could be done in real-time [Kins94a]. Therefore it is well suited for transient signals analysis.

Entropy-based Dimensions

Entropy-based dimensions are motivated by the fact that some objects have different distributions on different subregions, thus the critical exponents of the scaling property may not always be the same for the entire object. Entropy-based dimensions take the distribution of a fractal into consideration, as well as the geometrical properties.

In information theory, *entropy* (i.e., average self-information) is the amount of information needed to specify the state of a system to a certain resolution. The well-known Shannon entropy H_1 is defined as

$$H_1 = -\sum_{j=1}^{N_v} p_j \log p_j \tag{2.7}$$

where N_v is the number of vels (volume elements [Kins94a]) covering the fractal, each of size v, and p_j is the relative frequency n_j with which the fractal enters (intersects) the j-th vel of the covering to the total number N of intersects of the fractal with all the vels.

$$p_j = \lim_{N_v \to \infty} \frac{n_j}{N} \tag{2.8}$$

where

$$N = \sum_{j=1}^{N_v} n_j \tag{2.9}$$

If we assume the power-law relationship exists

$$c^{H_1} \sim v^{-D_I} \tag{2.10}$$

then we have the *information dimension*, D_I, defined as

$$D_I = \lim_{v \to 0} \frac{H_1}{\log(1/v)} \tag{2.11}$$

An improvement of the concept of information dimension is the *correlation dimension*, D_C. It considers the correlation between pairs of neighbouring points on the fractal and is given as

$$D_C = \lim_{v \to 0} \frac{H_2}{\log(1/v)} \tag{2.12}$$

where H_2 is the second-order entropy function and is defined as

$$H_2 = -\sum_{j=1}^{N_v} p_j^2 \tag{2.13}$$

In 1955, Alfred Rényi introduced the generalized entropy concept [Rény55]. He generalized the entropy from Shannon entropy (i.e., the measure of disorder) and the second-order entropy based on squared probabilities, to any exponent as

$$H_q = \frac{1}{q-1}\log \sum_{j=1}^{N_v} p_j^q \qquad -\infty < q < \infty \tag{2.14}$$

where q is called the moment order. Based on the entropy function in Eq. 2.11, the Rényi generalized dimension is given by

$$D_q = \lim_{v \to 0} \frac{H_q}{\log (v)} \tag{2.15}$$

For $q = 1$, D_q is equivalent to the information dimension, and for $q = 2$, D_q is the correlation dimension. In general, the following holds

$$D_{q_2} \leq D_{q_1} \text{ for } q_1 < q_2 \tag{2.16}$$

Thus, D_q is a monotonic nonincreasing function of q, as shown in Fig. 2.3. The concept of generalized dimensions is very useful when the fractal object is very complex and a single-valued dimension cannot describe all the information contained in the object. This leads to the concept of multifractals.

2.3.2 Multifractals

A fractal object is called *multifractal* if there is more than one corresponding Rényi dimensions D_q for a range of q [Kins94a]. This is because the fractal has varying distributions. The difference between dimensions of different order q measures the degree of inhomogeneity (or nonuniformity) of a multifractal in the sense of whether its different subsets are visited with equal frequency. If the fractal is strictly self-similar, D_q does not

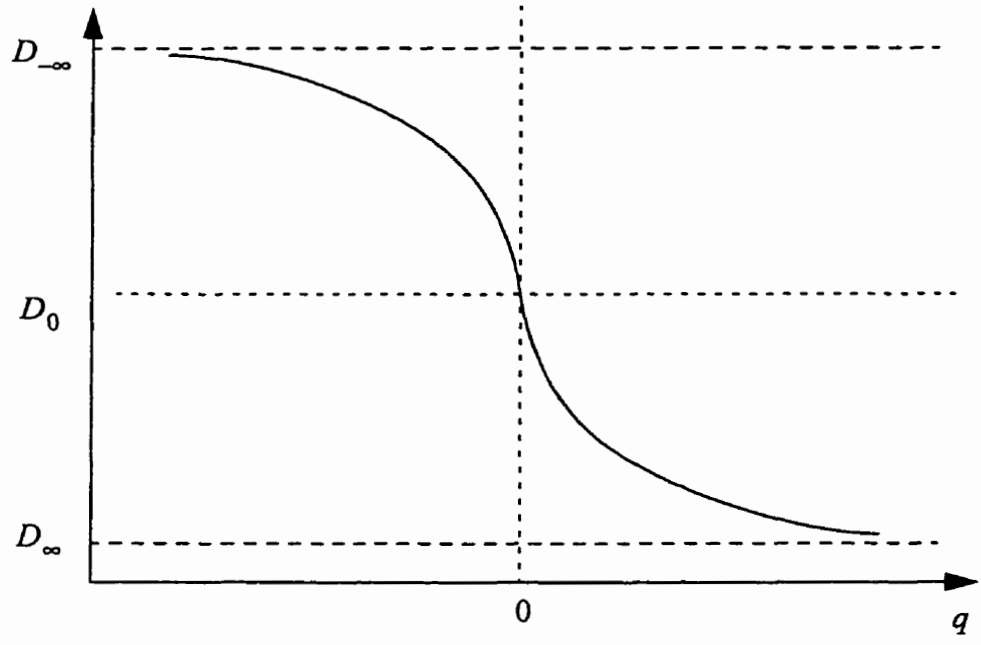


Fig. 2.3. Monotonic nonincreasing curve of D_q.

change with q. Otherwise there exists a Rényi spectrum, while the spread of the spectrum indicates the complexity of the fractal.

A multifractal also refers to an object with different local fractal dimensions. A local fractal dimension is a dimension value calculated on a subregion of the object. For a temporal signal, local fractal dimensions are obtained by using a moving rectangular window along the time axis that selects data from only within its boundaries for calculation. If the local fractal dimension is the same everywhere, then the fractal object is a pure fractal. Otherwise it is called multifractal [Vics92]. If a single definition of dimension is used, the multifractal measure can be visualized as a *fractal dimension trajectory.* A *multifractal surface* is obtained if we consider the Rényi spectrum as the local fractal measure along the time axis.

Transient signals are multifractals due to their nonstationarity, which means they have non-flat fractal dimension trajectory or spectrum. This observation provides us with a basis for transient signal processing throughout this thesis.

2.4 Chaos and Strange Attractors

2.4.1 Background on Chaos Theory

Most natural phenomena are produced by nonlinear dynamical systems. The behaviour of a nonlinear dynamical system can fall into three categories: stable, unstable, and chaotic [JoSm87]. *Stable* behavior means that after some transient period such systems settle in a periodic or a steady-state motion. If the trajectories in phase space are aperiodic and unbounded, it is called *unstable*. *Chaos* refers to the apparent randomness, or irregularity, or unpredictability that arises in deterministic dynamical systems [Kins95].

The important properties associated with a chaotic system are: unpredictable, indecomposable, and yet contains regularity. A chaotic system is unpredictable because it is very sensitive to initial conditions. The sensitivity is due to the critical region where the dynamical system operates. On the other hand, it is deterministic, i.e., they have equations governing their behavior, usually a set of differential equations. This property results in the regularity of such a system.

Let us consider an example of a chaotic system, the Chua's circuit, as shown in Fig. 2.4. The circuit is described by the following state equations

$$\frac{dv_{C_1}}{dt} = \frac{1}{RC_1}(v_{C_2} - v_{C_1}) - \frac{1}{C_1}g(v_{C_1}) \tag{2.17a}$$

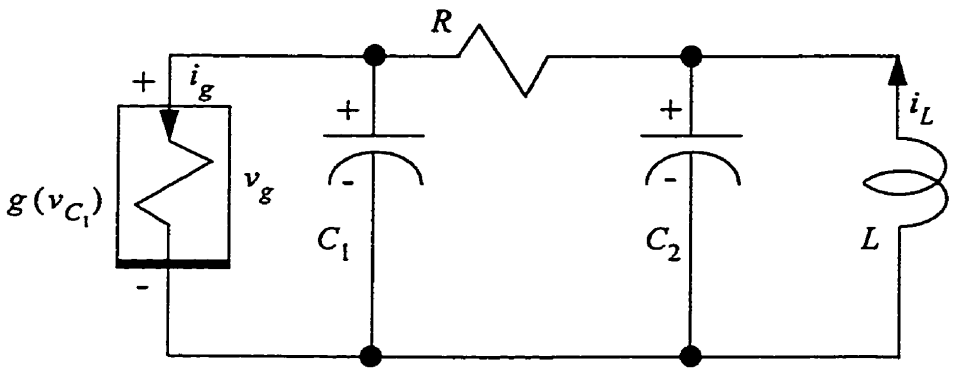


Fig. 2.4. Chua's circuit (after [Kins95]).

$$\frac{dv_{C_2}}{dt} = \frac{1}{RC_2}(v_{C_1} - v_{C_2}) + \frac{1}{C_2} i_L \tag{2.17b}$$

$$\frac{di_L}{dt} = -\frac{1}{L} v_{C_2} \tag{2.17c}$$

where the nonlinear conductance $g(v_{C_1})$ is chosen to be a three piecewise symmetrical linear approximation as illustrated in Fig. 2.5.

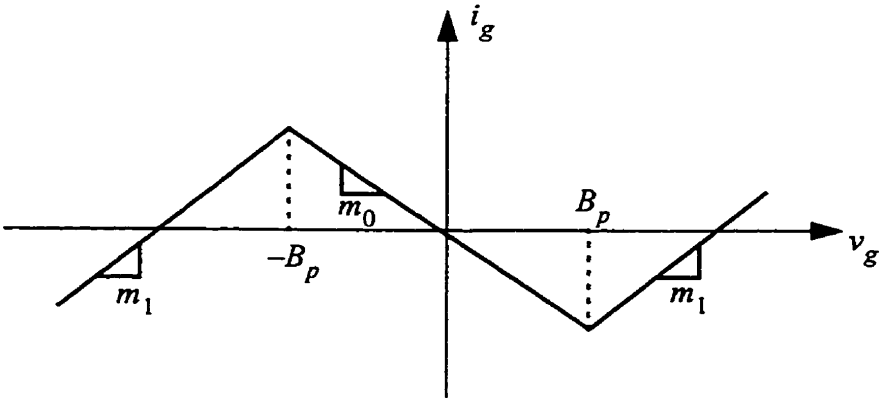


Fig. 2.5. *v-i* characteristic for Chua's circuit.

Then we choose the setting of the parameters of the circuit as

$$\alpha = C_1/C_2 = 9 \tag{2.18a}$$

$$\beta = C_2/\left(LG^2\right) = 100/7 \tag{2.18b}$$

$$m_0 = -1/7 \tag{2.18c}$$

$$m_1 = 2/7 \tag{2.18d}$$

The behavior of Chua's circuit is illustrated in Fig. 2.6.

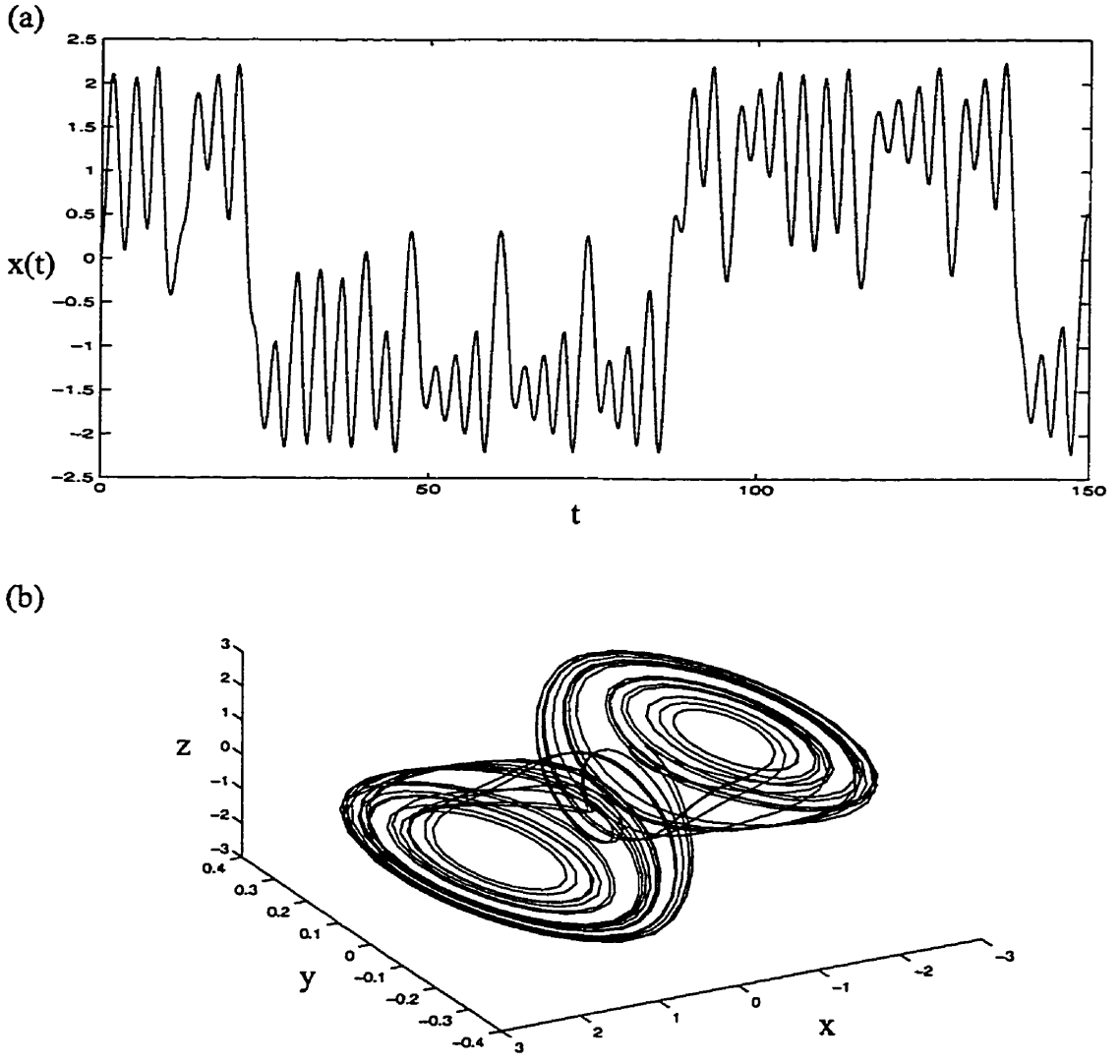


Fig. 2.6. Chaotic behavior of Chua's circuit. (a) The wave form of component x. (b) The xyz trajectory of the double scroll. (given $x = v_{C_1}/B_p$, $y = v_{C_2}/B_p$ and $z = i_L/(B_pG)$)

We can see from Fig. 2.6 that the time waveform exhibits somewhat random behavior, while the trajectory in state space clearly shows the existence of a bounded object. This object is also referred to as *strange attractor* because the attractor is a one of infinite numbers of Cantor-type dust with a fractal dimension [Kins95].

2.4.2 Relationship Between Chaos and Fractals

Strictly speaking, dynamical systems and fractal geometry are independent fields of study. Dynamics is the study of objects in motion, while fractals study static geometric objects. However, strange attractors are the point where chaos and fractals meet in an unavoidable and most natural fashion: as geometrical pattern, they are fractals, as dynamic objects, they are chaotic [PeJS92].

Therefore, an alternative to fractal modelling of nonstationary signals is to treat them as outputs of chaotic systems, leading to phase space characterization. One can associate a fractal (i.e. a strange attractor) with a chaotic system, thus characterizing the system quantitatively in terms of fractal dimensions. A problem existing in practice is that we usually only have access to the signal, not directly to the underlying strange attractor. This leads to the issue of strange attractor reconstruction from a measured time series, which will be discussed in the following section.

2.4.3 Strange Attractor Reconstruction

The reconstruction of a strange attractor is necessary when we obtain a time series in experimental settings. In this case, only one component of the state is measured, not the full states of the original attractor. The objective of this procedure is to retrieve the geometric structure of the underlying strange attractor to some faithful extend.

Time Delay, δ

Given a dynamical system described by a set of m equations, we say that the strange attractor is embedded in m-dimensional phase space. However, the experimental observation of such systems is only one-dimensional, which is in the form of a single time series. Ideally, the strange attractor should be reconstructed out of successive co-ordinates $\{x, \dot{x}, \ddot{x}, \dddot{x}, \ldots\}$ [PCFS80]. However, the measured time series usually does not have such high resolution to allow for these higher order differentials. A practical solution is to embed the attractor in an m-dimensional phase space and the m-dimensional co-ordinates, x_i, of the attractor are constructed from the original sampled time series, x_i ($i = 1, 2, 3, \ldots N$), according to

$$\begin{aligned} x_i &= (x_i, x_{i+\delta}, x_{i+2\delta}, \ldots, x_{i+(m-1)\delta}) \\ i &= 1, 2, 3, \ldots, N-(m-1)\delta \end{aligned} \tag{2.19}$$

where δ is the time delay, m is the embedding dimension, and N is the length of the time series. The *method of time delays* provides a relatively simple way of constructing an attractor from a single experimental time series. Theoretically, the time delay can be an arbitrary value provided that we have a very long and noise-free time series [Addi97] [PeJS92]. In practice, the condition is seldom satisfied, and the choice of the time delay is very important as we want the reconstructed attractor to exhibit the basic structure of the real attractor, with similar dynamical properties. If it is too small, the reconstructed attractor is restricted to the diagonal of the phase space. If it is too large, then the points are uncorrelated and the structure of the attractor disappears. If it is close to some periodicity in the system, the component at that period will be under-represented in the reconstruction

[PaCh87]. We should make the reconstructed attractor in the phase space as spread-out as possible to reduce the linear dependency, thus giving a good representation of the *m*-variable attractor geometry. This is especially true when a small amount of noise is present in the signal. A common approach is by the use of the *autocorrelation function, C*. The autocorrelation function is defined as

$$C = \frac{\sum_{i=1}^{N-\delta} (x_i')\,(x_{i+\delta}')}{\sum_{i=1}^{N-\delta} (x_i')^2} \tag{2.20}$$

where

$$x_i' = x_i - \bar{x_i} \tag{2.21}$$

and $\bar{x_i}$ is the mean of the time series. It measures the linear dependency of two points in the time series, which is generally decreasing with the time delay, δ, as shown in Fig. 2.7. The time delay for reconstruction is chosen to be the time at which the autocorrelation function falls below a specific threshold value, such as one half.

Embedding Dimension, *m*

After selecting the time delay, the minimum embedding dimension of the strange attractor should be decided as well. According to Takens' theorem [Take81], when there is only a single measured variable from a dynamical system, it is possible to reconstruct a state space that is equivalent to the original state space composed of all the dynamical variables. The embedding space dimension, *m*, is so chosen that $m > 2n$, where n is the

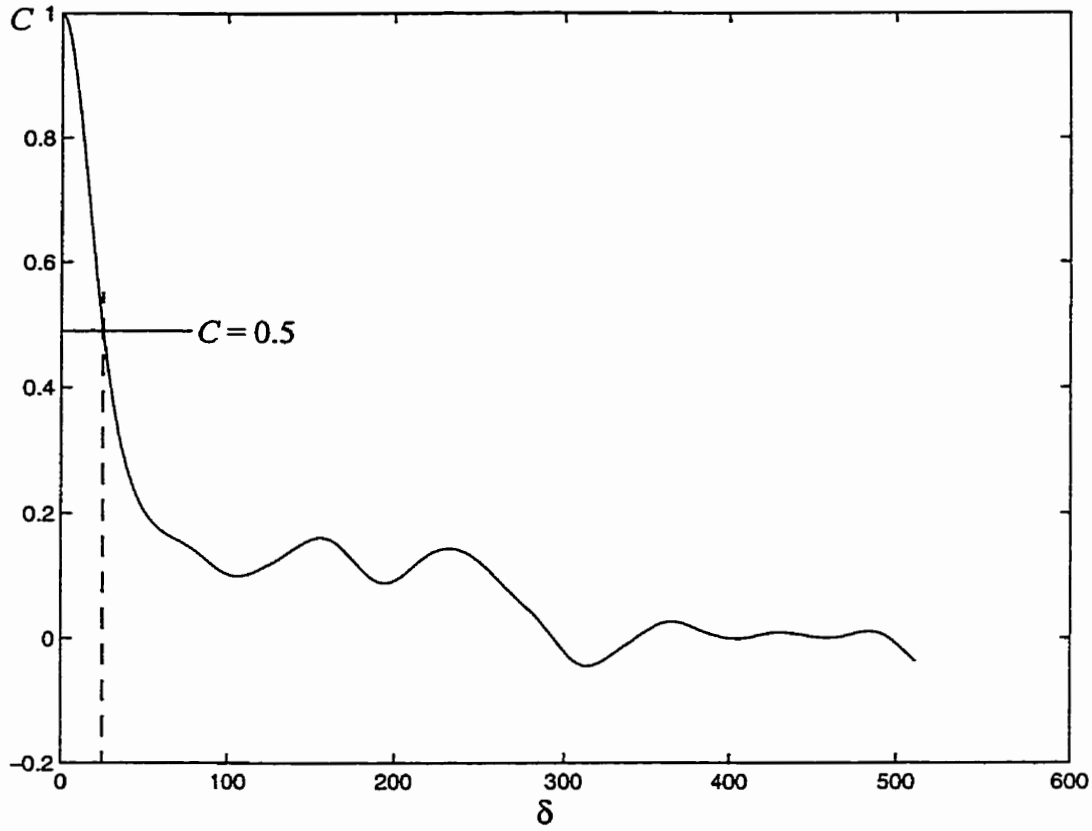


Fig. 2.7. The autocorrelation function for a strange attractor and the choice of the time delay.

dimension of the original embedding space. This guarantees that each point in the projected attractor corresponds to one and only one point in the original attractor. In other words, we have a truthful representation in the sense that the embedding space is large enough for the attractor to untangle itself. It should be noted that Takens' requirement for m is a sufficient but not necessary condition for the reconstruction. In practice, the correlation dimensions (D_C) of the reconstructed attractor are calculated in embedding spaces of successively larger dimensions. At first, the value of D_C increases with the increasing embedding dimension. After a certain point, D_C saturates and no more increase in D_C could be observed. The minimal embedding dimension is then chosen at the saturation point, as shown in Fig. 2.8.

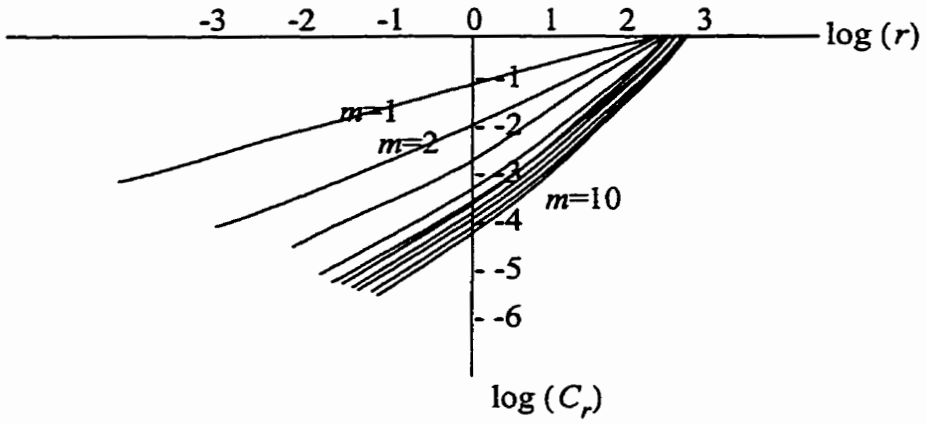


Fig. 2.8. A correlation dimension plot for a reconstructed strange attractor in embedding dimension m from 1 to 10.

Generalized Dimension Calculation for Strange Attractors

The generalized dimensions can be calculated by the concept of generalized correlation integral [PaSc87], which is defined as

$$C_r^{(q)} = \lim_{r \to 0} \{ \frac{1}{N} \sum_i \left[\frac{1}{N} \sum_j \Theta (r - |x_i - x_j|) \right]^{q-1} \}^{\frac{1}{q-1}} \tag{2.22}$$

where $\Theta(x)$ is the Heaviside step function, and N is the size of the attractor. $\Theta(x)$ is used to count how many pairs of points (x_i and x_j) on the attractor fall within the distance r. Then the generalized dimension in Eq. 2.12 can be written as

$$D_q = \lim_{r \to 0} \frac{\log C_r^{(q)}}{\log r} \tag{2.23}$$

2.5 Fractal Analysis on Transient Signals

2.5.1 Noise Segmentation

Fractal analysis measures the complexity of an object or a signal. Noise is distinguishable from a transient because noise has more randomness, while the transient carries more meaningful information, thus is more correlated. It is concluded that noise and the transient exhibit different degrees of complexity (or singularity), i.e., noise has a higher degree of complexity and the transient has lower degree of complexity. If there is a significant change in the fractal measurement, it indicates the transition from one to the other. Therefore, noise segmentation is achieved.

2.5.2 Feature Extraction

Modelling and characterizing a transient signal is achieved by selecting a set of fractal dimension values for the signal. More specifically, when the variance fractal dimension trajectory is used, the moving window is shifted a larger amount along the time axis, resulting in a smaller number of points on the trajectory. The variance dimension values corresponding to these points are considered as extracted features of the original signal. An important gain of this technique is signal normalization, which is achieved automatically because there are theoretical lower and upper bounds for fractal dimension values. The variance dimension for a temporal signal, in particular, has the lower bound of 1.0 and the upper bound of 2.0, as implied in Eq. 2.3. Further discussions on the importance of signal normalization is provided in Chapter 3.

In the procedure of feature extraction using the variance fractal dimension trajectory, a special type of fractal modelling, *fractal amplification* [Kins94a] is used. Fractal

amplification refers to emphasizing the fractality from a computational point of view. It is implemented by using a dyadic sequence of time interval in the calculation of the variance dimension, instead of a linear sequence. This results in relatively larger values and more variations of the dimension. For the purpose of feature extraction, fractal amplification is preferred [KiGr94] because a higher degree of detail about the fractality of the signal can be achieved.

Another way of characterizing transients is to treat transients as outputs from chaotic dynamical systems [Lang96]. The strange attractor of a transient is reconstructed and multifractal analysis is applied to the attractor, thus obtaining multifractal features of the transient indirectly. Since a strange attractor is specific to the dynamical system, not to a particular transient output, robust characterization of transients may be achieved.

2.6 Summary

This chapter summarizes fractal analysis applied to nonstationary temporal signals, especially transient signals. It is the core of this thesis. Nonstationary signals can be modelled and quantitatively characterized in terms of fractal dimensions. As we shall see, multifractal analysis is more appropriate in the study of transients because of the nature of such signals. We have described a fractal trajectory and a multifractal surface as available means of multifractal analysis. Such signals can also be considered as the outputs from dynamical chaotic systems, therefore chaos theory can be applied. To use chaos theory, a strange attractor needs to be reconstructed from a temporal signal first. Several related issues on strange attractor reconstruction are covered. Fractal analysis is considered to be a good approach for noise segmentation and feature extraction..

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

CHAPTER III
NEURAL NETWORK PREPROCESSING AND CLASSIFICATION

3.1 Introduction

Neural networks are commonly used for the purpose of classification. In this chapter, an introduction to neural network classification is provided, with the emphasis on a particular type, the PNN. We prefer the PNN to the MLFN (*multiple layer feedforward network*) because of the advantages unique to the PNN, such as a strictly defined architecture, mathematically sound confidence level, and very short training time, which make it suitable for transient classification. Some issues related to PNN training, processing, and improvement are discussed. Preprocessing is usually required when the speed of neural network classification is of critical concern. Two techniques are examined here, namely the SOFM and the PCA. The SOFM tries to reduce the number of cases in the training class set through clustering, while the PCA reduces the dimensionality of each input case by removing redundant information in the original input case components.

3.2 Overview of Neural Networks

In many real-world applications, we encounter problems that are difficult to solve by the use of *programmed computing*, in which algorithms are designed and subsequently implemented using the currently dominant architecture. These problems are characterized by some or all of the following: a high-dimensional problem space; complex, unknown, or mathematically intractable interactions between problem variables; and a solution space that may be empty, contain a unique solution, or contain a number of useful solutions. The

concept of *neural computing* was introduced in the effort of solving such problems. It is based on the computing scheme in biological systems. Some key aspects of neural computing are [Scha97]:

1. The overall computational model consists of a reconfigurable interconnection of simple elements;
2. Individual units implement a local function, and the overall network of interconnected units displays a corresponding functionality;
3. The system knowledge, experience, or training is stored in the form of network interconnection;
4. To be useful, neural systems must be capable of storing information (i.e., they must be "trainable");
5. A neural network is a dynamic system; its state (e.g. unit outputs and interconnection strengths) changes over time in response to external inputs or an initial (unstable) state.

Because of these features, neural networks have been found very effective in solving classification problems. The applications of neural networks include image processing and computer vision, signal processing, pattern recognition, planning, control, power systems, and artificial intelligence.

There exist many forms of architecture of neural networks, ranging from very simple ones such as a perceptron to much more complex models. In this thesis, the PNN is used. An overview of the PNN is provided next.

3.3 PNN Basics

3.3.1 PNN Architecture

The PNN architecture is shown in Fig. 3.1. It is a four-layer organization. The input units are merely distribution units that pass the input values to the pattern units, and the number is equal to the size of an input vector $\mathbf{x}$. Each pattern unit stores one training case and it computes a distance measure between the input and the training case it represents by forming a dot product of the input pattern vector $\mathbf{x}$ with a weight vector $\mathbf{w}_i$, $z_i = \mathbf{x} \cdot \mathbf{w}_i$. Then it performs a nonlinear operation on z_i before outputting its activation

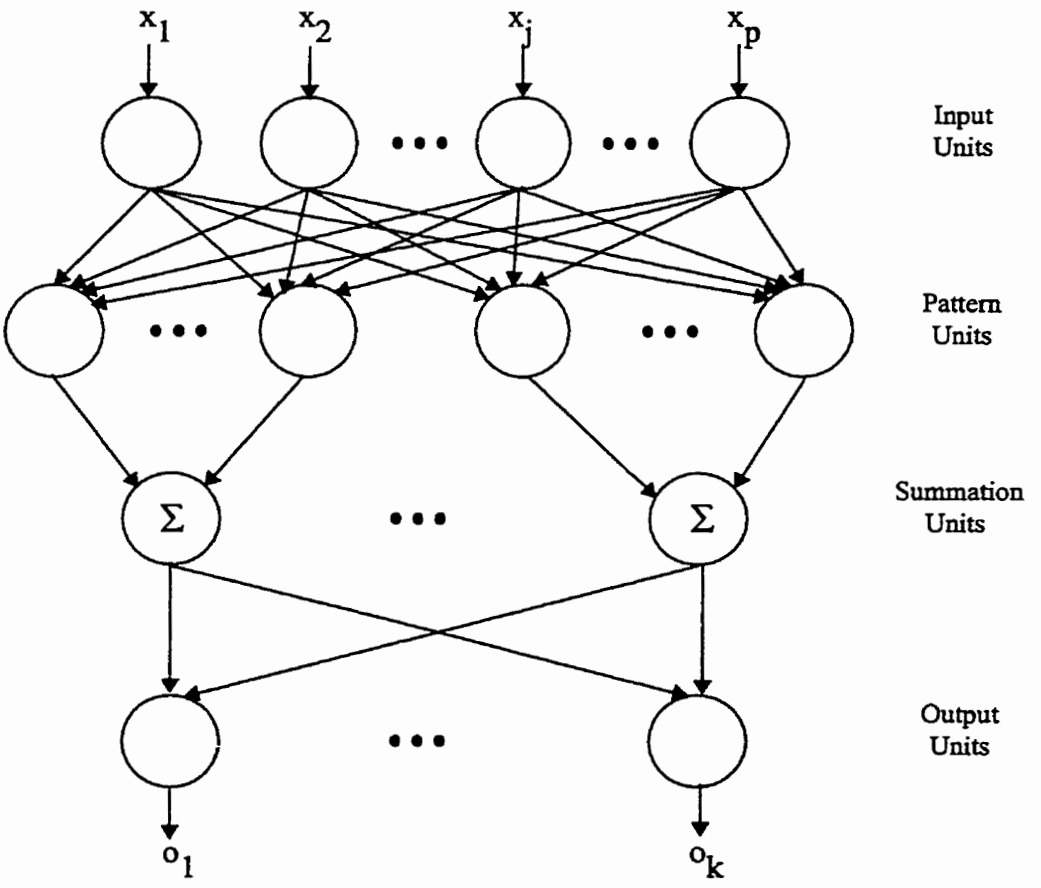


Fig. 3.1. The PNN Architecture (after [Spec90a]).

level to the summation unit. Instead of the sigmoid activation function, $1/\left(1+e^{-z_i}\right)$, commonly used for the back-propagation neural network (BPNN), the nonlinear operation used here is an exponential function, $\exp\left[(z_i-1)/\sigma^2\right]$. The summation units sum the inputs from the pattern units that correspond to the category from which the training pattern is selected. There is one summation unit for each class. The number of output (or decision) units is also the same as the number of classes. This layer is often a simple threshold discriminator which activates the unit having the largest output value to represent the projected class of the unknown sample.

To understand the PNN paradigm in more detail, a discussion on the Bayes decision strategy and Parzen's estimation of probability density functions (pdfs) is required. It is provided in the next two subsections.

3.3.2 Bayes Strategy for Classification

Bayes strategies for pattern classification are the decision rules or strategies used to minimize the expected risk of misclassification for a given decision surface.

Consider a C-class situation where we have a collection of samples from C different classes denoted as $c = 1, 2, \ldots, C$. Each sample is denoted by a p-dimensional vector $\mathbf{x} = (x_1, x_2, \ldots, x_p)$. Let us first define the following parameters:

- h_c : the prior probability of a class, c;
- l_c : the loss function associated with misclassifying a sample of the class c as other classes;
- $f_c(\mathbf{x})$: the pdf for the class c.

The Bayes decision rule is then stated as

$$d(\mathbf{x}) = c \text{ if } h_c l_c f_c(\mathbf{x}) > h_i l_i f_i(\mathbf{x}) \tag{3.1}$$

which means the unknown sample $\mathbf{x}$ is classified into class c if the above holds for all classes i not equal to c. The key to using Eq. 3.1 is the ability to estimate the PDF, $f_c(\mathbf{x})$, based on training cases. Often the priori probabilities h_c are known or can be estimated accurately, and the loss functions l_c require subjective evaluation. However, if the pdfs are unknown, and all that is given is a set of training samples, then these samples provide the only clue to the unknown underlying probability densities. Parzen has shown that a class of pdf estimators asymptotically approaches the underlying parent density as the number of samples increases towards a fully comprehensive representation of the class data [Parz62].

3.3.3 Parzen's Method of Density Estimation

Let us assume that there are n_c training cases for a given class, c. The estimated PDF for that class, $g_c(x)$, is given as

$$g_c(x) = \frac{1}{n_c \cdot \sigma} \sum_{i=1}^{n_c} W\left(\frac{x - x_i}{\sigma}\right) \tag{3.2}$$

where $W(x)$ is the weight function (kernel) and σ defines the width of the kernel centered at each training case.

The weight function, $W(x)$, is usually chosen to be the normalized Gaussian function

$$W(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{x^2}{2\sigma^2}\right)} \tag{3.3}$$

The effect of σ is illustrated in Fig. 3.2. It is demonstrated that, a small value of σ causes the estimated parent density function to have distinct modes corresponding to the locations of the training samples, which essentially leads to a nearest-neighbour classifier [Mast93]. A larger value of σ produces a greater degree of interpolation. However, if σ is too large, details of the density will be smoothed out and the estimated density will be Gaussian regardless of the true underlying distribution [Spec90a]. The proper value for σ is critical to the performance of the PNN.

Cacoullos has extended Parzen's results to cover the multivariate case [Caco66]. The density estimator for a class, *c*, is defined as

$$g_c(x_1, \ldots, x_p) = \frac{1}{n_c \sigma_1 \ldots \sigma_p} \sum_{i=1}^{n_c} W\left(\frac{x_1 - x_{1,i}}{\sigma_1}, \ldots, \frac{x_p - x_{p,i}}{\sigma_p}\right) \tag{3.4}$$

Some simplifications can be made to Eq. 3.4. First, let us assume all smoothing parameters are equal, i.e. $\sigma_1 = \sigma_2 = \ldots = \sigma$. Second, the multivariate weight function is assumed equal to the product of the univariate weight function, as

$$W(x_1, \ldots, x_p) = \prod_{i=1}^{n_c} W_i(x_i) \tag{3.5}$$

Thus, the density estimator as described in Eq. 3.4 is rewritten as

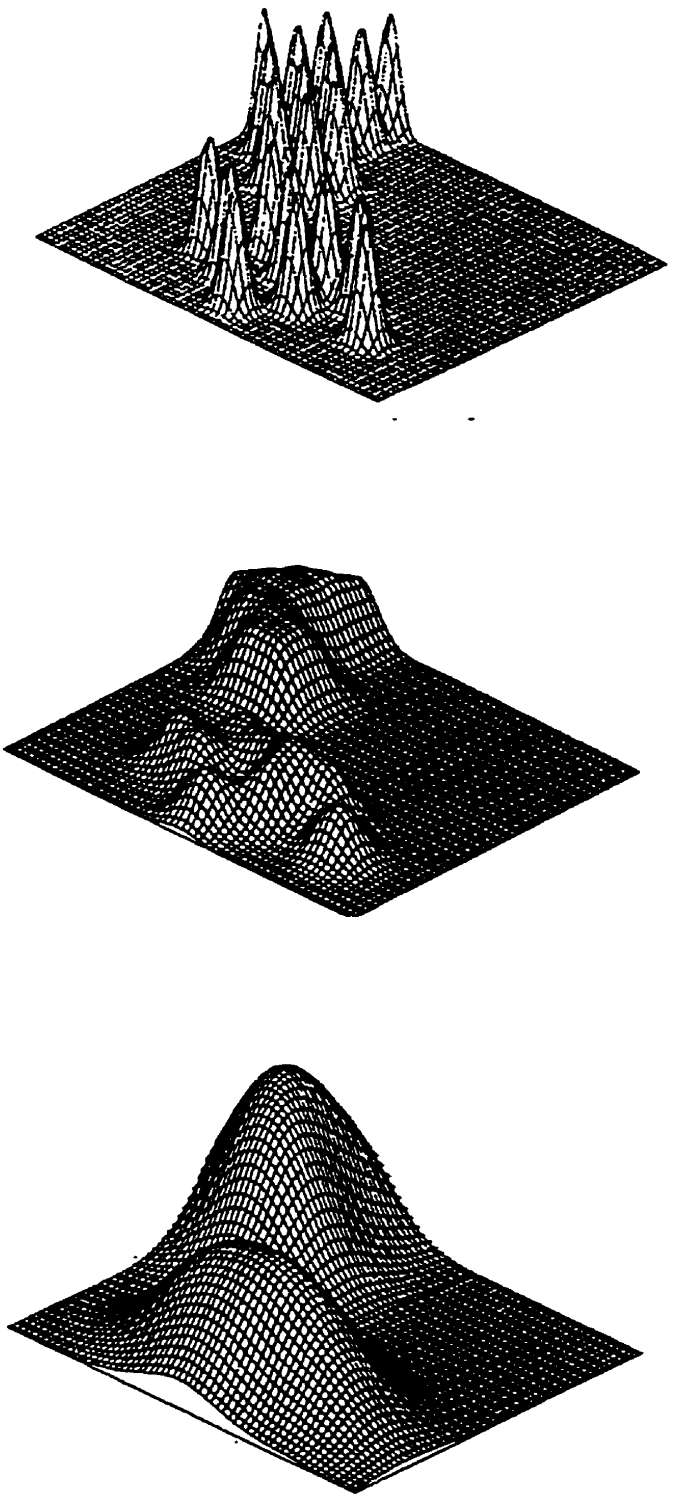


Fig. 3.2. The smoothing effect of different values of σ on a PDF estimation: (a) a small value of σ; (b) a larger value; and (c) an even larger value (after [Meis72]).

$$g_c(\mathbf{x}) = \frac{1}{n_c \cdot \sigma^p \cdot (2\pi)^{p/2}} \sum_{i=1}^{n_c} e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}} \tag{3.6}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_p)$ is the input vector.

3.3.4 PNN Processing in Detail

Let us assume an unknown case, $\mathbf{x} = (x_1, x_2, \ldots, x_p)$ is given at the input layer. The weight function described in Eq. 3.3 can be replaced by

$$W(\mathbf{x}, \mathbf{x}_r) = e^{-d(\mathbf{x}, \mathbf{x}_r)^2} \tag{3.7}$$

where $d(\mathbf{x}, \mathbf{x}_r)^2$ is the scaled, squared Euclidean distance between the unknown sample, $\mathbf{x}$, and the training case, $\mathbf{x}_r$, and is obtained by

$$d(\mathbf{x}, \mathbf{x}_r)^2 = \frac{1}{\sigma^2} \sum_{i=1}^{p} |x_i - x_{r,i}|^2 \tag{3.8}$$

In the pattern layer, each unit computes the distance between the training case it represents, and the unknown sample, according to Eq. 3.8. The weight function is then applied to this distance using Eq. 3.7.

The summation units then calculate the PDF for the class they represent according to Eqs 3.2 or 3.4.

In the output layer, all the results from the summation layer are compared and the largest value is selected. The output unit corresponding to this value is activated as the result. The unknown sample is classified to the class the activated output unit represents.

3.3.5 Input Normalization

Normalization of input data for the neural network provides an equal range for different input variables, and the variables are scaled to match the range of the input neurons. In general, input normalization is important for neural network performance. First, if an input is used to train output neurons, and the output neurons have an activation function with bounded range, then normalization is necessary. For some neural network models, such as the Kohonen self-organizing maps, there is a strict limit on their input values, and a normalization procedure is required. Another reason for normalization is to equalize initially the importance of variables. If input variables have quite different ranges of value, the variation in weight vectors during the training procedure can be very significant, which may result in failure in learning. Third, most training algorithms minimize the total error of all outputs. If no normalization is conducted, those output neurons with larger variabilities will be favored, as they will dominate the error sum. This can have profound negative consequences on the classification and is a problem that should always be considered.

There are some needs and advantages of input data normalization specific to the PNN. First of all, in the distance calculation in Eq. 3.8, variables having a much larger range of values will dominate in the distance measure. For the case in this thesis, variables having large variations are more likely the remaining noise component due to inconsistent

noise segmentation. The effect of remaining noise is reduced by the normalization procedure. Second, in the probability density function estimation as defined in Eq. 3.4, the processing can be simplified by assuming that all smoothing parameters are equal, if the input variables are normalized approximately to the same range and the same variation. Otherwise, a smoothing parameter needs to be determined and optimized for each input variable, which could be very time-consuming.

3.4 Neural Network Preprocessing

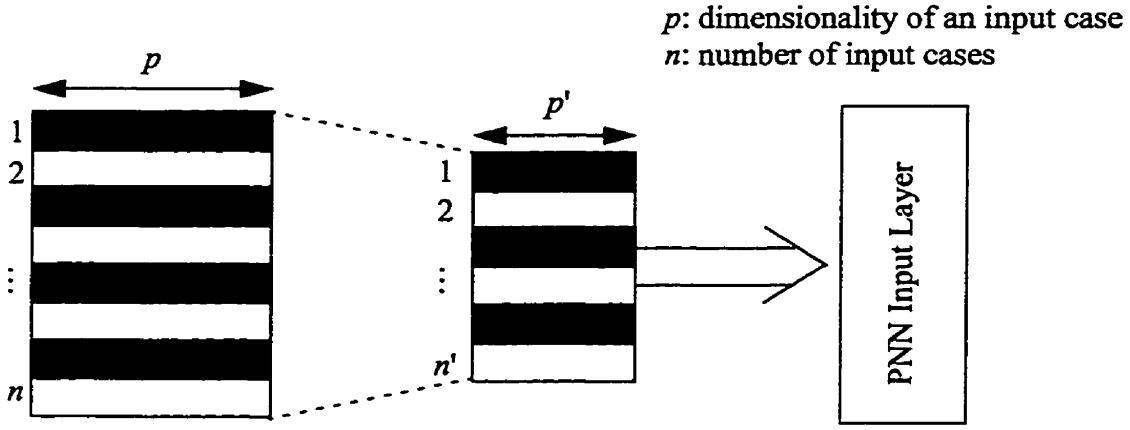


Fig. 3.3. Illustration of neural network preprocessing: dimensionality reduction and clustering.

3.4.1 Motivation

To accelerate the basic PNN processing, preprocessing is performed in this thesis. It consists of two procedures, one is dimensionality reduction on each input case, and the other is clustering to reduce the number of cases in the training set.

Dimensionality reduction refers to further feature extraction among variables of input vectors, resulting in fewer and relatively independent variables. One serious prob-

lem with having a large dimensionality of input vectors is the slow execution speed. Another problem is overfitting [Mast95]. Especially when there is a large number of variables compared with the number of training cases, neural networks may focus on individual training cases instead of generalizing beyond the training set, the ability which is very important in most cases. Third, there is usually redundant information in these variables because of the existence of correlation between them, and such redundancy has effect on the performance of the PNN. The PCA provides an effective solution to the above problems by accounting for the maximum possible variation between input variables.

In order to provide the best generalization of the neural networks, we usually supply as many training cases as possible. This may result in many duplicate training cases, which causes waste in storing space and slow training speed. Elimination of duplicate training cases and selection of training cases can be achieved by some *clustering* techniques. The objective here is to form a cluster for all training cases from each class, then choose the cases closest to the cluster centroid for further training, while discarding those which are far from the centroid. The assumption is that the training cases closer to the cluster centroid are more representative cases for the corresponding class than the others. The SOFM is used as for the clustering purpose because of its ability to map high-dimensional input data to a low-dimensional representation while preserving the topological similarity.

3.4.2 Principal Component Analysis (PCA)

PCA is used for extracting the useful information present in a large set of variables while removing the redundant information. The basic idea is to account for as much of the

variation in input cases as possible by means of as few new variables as possible. Each successive principal component is computed in such a way that it has the maximum possible variance, while it is independent of all previous principal components. As the result, the number of variables is reduced and the redundancy among the variables is eliminated.

Assume we have a weight vector $\mathbf{w} = (w_1, w_2, \ldots, w_p)'$, where the prime ($'$) indicates the transpose operation. An input case would be $\mathbf{x} = (x_1, x_2, \ldots, x_p)'$. Then each principal component is computed for the input case $\mathbf{x}$ as shown in Eq. 3.9.

$$y = \mathbf{w}'\mathbf{x} = w_1 x_1 + w_2 x_2 + \ldots + w_p x_p \tag{3.9}$$

Equation 3.9 concerns a single principal component. In practice, we usually work with several principal components. In general, for each case consisting of p variables, we extract p' principal components, and usually $1 \leq p' \ll p$.

Thus, we can modify Eq. 3.9 as

$$\mathbf{y} = \mathbf{W}'\mathbf{x} \tag{3.10}$$

where $\mathbf{y}$ is a p'-vector of principal components, and $\mathbf{W}$ is a $p \times p'$ weight matrix, each of whose p' column is a weight vector defining a different principal component. In practice, principal components are centered so that we have zero mean for each variable in the input vector. Therefore, Eq. 3.10 is modified as

$$\mathbf{y} = \mathbf{W}'(\mathbf{x} - \mu) \tag{3.11}$$

with $\mu = (\mu_1, \mu_2, \ldots, \mu_p)'$.

To calculate principal components, we need to compute the **W** matrix to be able to perform the dot product in Eq. 3.10. The algorithm is listed below.

- Center input cases by computing the mean of each variable j, $j = 1, \ldots, p$

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij} \tag{3.12}$$

where n is the number of cases and p is the dimensionality of each case.

- Fill the covariance matrix, ***Cov***, according to

$$cov_{ij} = cov_{ji} = \frac{1}{n}\sum_{k=1}^{n} (x_{ki}-\mu_i)\,(x_{kj}-\mu_j) \tag{3.13}$$

- Compute the eigenstructure of ***Cov***. Note that the eigenvalues of the matrix are the variances of the principal components, and n should be equal or greater than p to perform the eigenstructure calculation.
- Fill **W** in such a way: the eigenvector corresponding to the largest eigenvalue is the first column of **W**, the second-largest eigenvalue is the second column, and so on.
- Stop adding columns when the eigenvalues become small enough that we no longer consider them an important contribution to the variation in the training set.

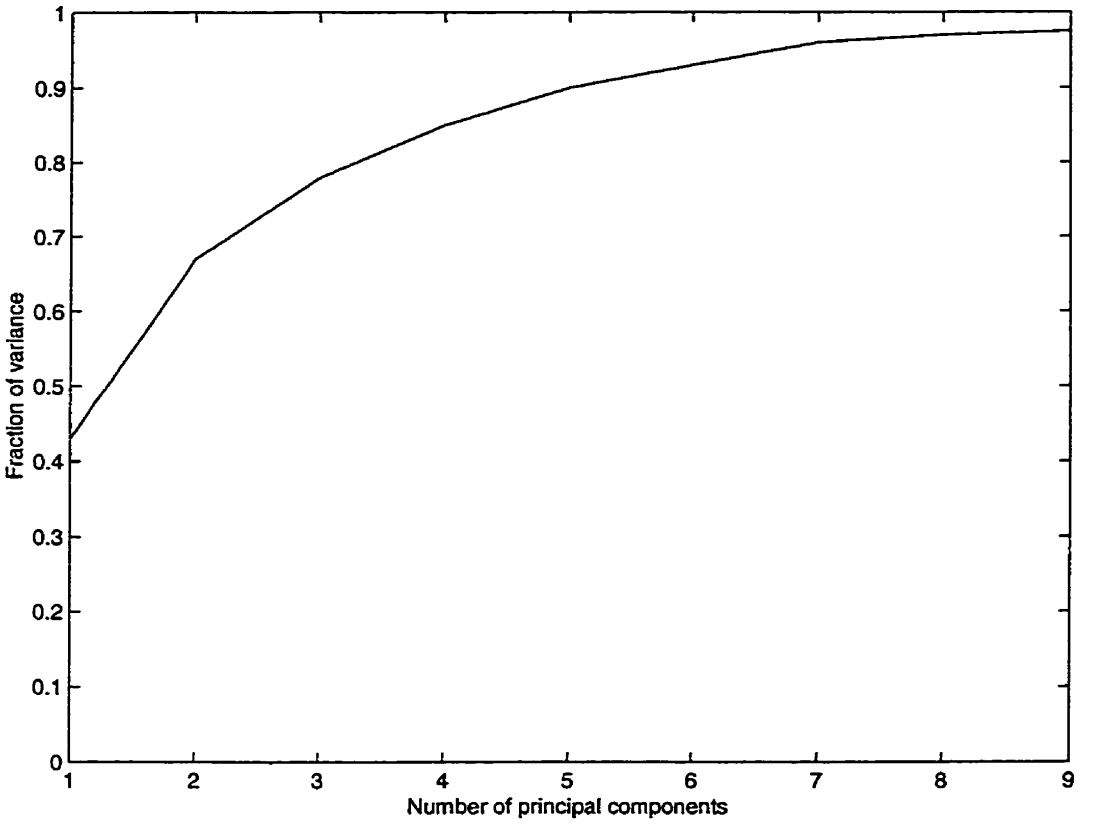


Fig. 3.4. Successively accounting for variance (after [Mast95]).

A criterion of choosing the number of principal component, p', is to plot the fraction of the total variance as a function of p', then choose a fraction value that is considered satisfactory. A typical form of the function is shown in Fig. 3.4.

Another practical issue to be considered is the scaling of the principal components. The first principal component has the maximum variance, and each succeeding one has less variance. This tends to emphasize the importance of the first few components, which sometimes may not be desirable. The solution to this problem is scaling them to unit vari-

ance. Let **L** be a $p' \times p'$ diagonal matrix, with the diagonal values equal to the eigenvalues of **W**. Then Eq. 3.10 is modified as below to give unit variance as well as zero mean

$$\mathbf{y} = \mathbf{L}^{-\frac{1}{2}}\mathbf{W}'(\mathbf{x} - \mu) \tag{3.14}$$

3.4.3 Self-Organizing Feature Map (SOFM)

Kohonen has demonstrated a neural learning structure involving networks that converts the feature space to topologically ordered similarity graphs, called the self-organizing feature map (SOFM) [Koho82]. The SOFM defines a mapping from the input data space R^n onto a regular (usually one or two-dimensional) array of nodes. With each node i, a parametric reference vector $\mathbf{m}_i \in R^n$ is associated. An input vector is compared with the $\mathbf{m}_i$, and the best match is defined as the "response"; the input is thus mapped onto this node.

In contrast with other techniques, SOFM training implements a mechanism called *competitive learning*. With each training case, the winning output unit is first found by comparing the input and reference vectors. Then the winning reference vector and also vectors near it, i.e., within a certain radius in the map, are moved towards the input vector according to a neighborhood function. The radius and the learning rate factor of the neighborhood function are decreasing monotonically towards the end of learning. After the training phase the map is labeled with known examples of input vectors. At the end of training, the output layer is organized into a meaningful order map in which similar models are close to each other and dissimilar models far from each other.

A summary of the SOFM algorithm is provided as follows.

For each sample $\mathbf{x}(t)$, first the winner index c (best match) is identified by the condition

$$\|\mathbf{x}(t) - \mathbf{m}_c(t)\| \leq \|\mathbf{x}(t) - \mathbf{m}_i(t)\|, \text{ for } \forall i \tag{3.15}$$

where $t = 1,2,...$ is the step index.

After that, the updating takes place, defined as

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{c(x),i}(\mathbf{x}(t) - \mathbf{m}_i(t)) \quad \text{if } i \in N_c \tag{3.16}$$

where N_c is the topological neighbourhood and is defined such that all units which lie within a certain radius from unit c are included in N_c. $h_{c(x),i}$ is the neighbourhood function, a decreasing function of the distance between the ith and cth nodes on the map grid. A simple neighbourhood function is the bubble function which is defined as

$$h_{c(x),i} = \alpha(t) \tag{3.17}$$

where $\alpha(t)$ is the learning rate. Another widely applied neighborhood function is the Gaussian function

$$h_{c(x),i} = \alpha(t) \cdot \exp\left(\frac{\|r_c - r_i\|^2}{2\sigma^2(t)}\right) \tag{3.18}$$

where $\alpha(t)$ is the learning rate and $\sigma(t)$ defines the width of the function.

To illustrate the clustering result, we employ the concept of *SOM density map* [ZhLi93], which is similar to the self-organizing map with the number of mapped input vectors (images) written on each node of the map. A simple example of some two-dimensional input data clustering is given in Fig. 3.5.

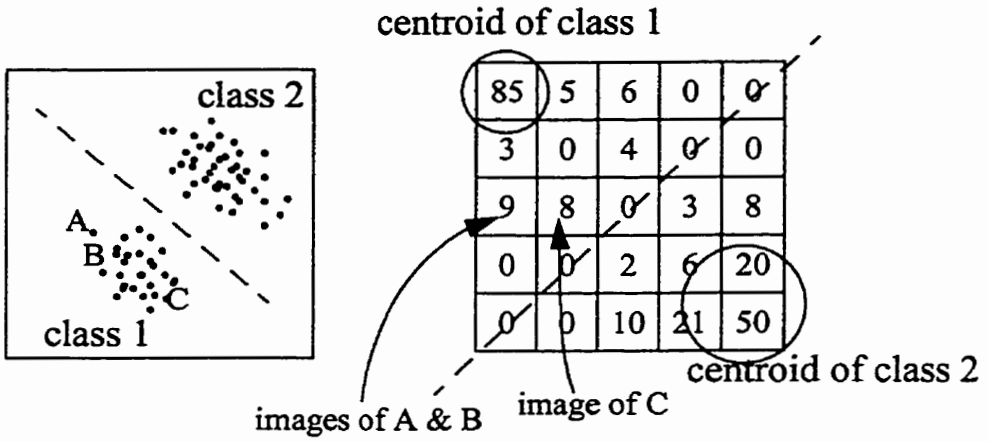


Fig. 3.5. SOFM clustering example (after [ZhLi93]). (a) The original data from two classes. (b) The density map showing the clustering result.

In this example, the SOFM clustering technique is applied to input data (patterns) from two different classes. The topology of the map is chosen to be a 5 by 5 lattice. The SOM density map clearly shows two distinct clusters, which means the similarity relations in the original input data is well preserved. Patterns that are closer to each other in the original space will "crowd" their images on the density map. For instance, patterns A and B are closer to each other while pattern C is far from them in the original input space. This results in that A and B are mapped to the same node on the density map while pattern C is mapped to another node. We can also determine the centroid of a cluster on the density

map. Consequently, an immediate way to reduce the number of training sets and to select the most representative training cases is to keep those training cases closer to the centroid of a cluster while discarding those far from the centroid.

3.5 Summary

This chapter has presented the PNN and related issues for transient classification. The structure of the PNN is examined and PNN processing is described, particularly the Bayes classification strategy and Parzen's method of density estimation. The PNN is found to be appropriate for our needs for it provides fast training and mathematically sound solutions. The classification speed of the PNN still needs to be improved for real-time applications. The proposed solutions are dimensionality reduction and clustering of training cases. The PCA achieves dimensionality reduction by removing the redundancy between variables of input vectors. The SOFM can reduce the number of training cases through clustering, thus the most representative cases are retained for training.

This concludes the 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 IV
SYSTEM DESIGN AND SOFTWARE IMPLEMENTATION

4.1 Introduction

This chapter is devoted to the experimental design of the radio transmitter identification system. It describes the major tasks to be accomplished, as well as the software tool design concerns. As stated in Chapter 1, the thesis deals with four modules of the transmitter identification system, namely noise segmentation, feature extraction, neural network classification, and user interface design. The specific functions to be implemented in each module are described in detail in the following sections.

4.2 Noise Segmentation

In the noise segmentation module, the input is a raw transient recording from a communication receiver discriminator channel. The recording procedure is explained in Toonstra's thesis [Toon97]. Suffice it to say that the recording is contaminated by ambient channel noise, which is followed by the transient part. The pre-transient noise portion of each recording should be removed first because it does not contain useful information for classification and may lead to classification failure. Techniques used here include the variance dimension trajectory and Rényi generalized dimension spectrum analysis. The processing procedure in detail is provided in the following subsections.

The effect of parameter settings for these techniques should also be studied and an optimal setting for practical purposes should be suggested. The output of this module is the separated transient of a raw signal.

4.2.1 Variance Fractal Trajectory

The variance fractal dimension is used for characterizing transient signals because it applies directly to temporal signals, and computation can be done in near real-time [Kins94b]. The variance fractal trajectory is obtained by calculating local variance fractal dimensions for a rectangular sliding window. The window moves along the entire signal, thus producing a trajectory. In our case, the noise precedes the transient, therefore the trajectory should exhibit a high to low transition, because the fractality of noise is higher than that of the transient. According to the study by Shaw [Shaw97], the dimension should be assigned to the starting point of the window to ensure proper detection of transient. The segmentation is achieved by finding the transition point by a difference threshold scheme, as discussed in Section 5.2. If a large enough change in the dimension occurs, it triggers the start of the transient and a certain number of samples from this point is separated as the transient for further processing.

4.2.2 Multifractal Analysis

An improvement of variance fractal trajectory for noise segmentation is suggested by the generalization of the variance fractal trajectory. It is motivated by the fact that transient signals are nonstationary signals, therefore, they are inherently multifractal. Instead of a single fractal dimension value (variance fractal dimension) corresponding to a sliding window, the Rényi generalized dimension spectrum is obtained. This results in a time-evolving multifractal dimension surface. A multifractal dimension surface can be considered as a collection of fractal trajectories. The multifractal surface contains far more information about the transient signal than the variance dimension trajectory. Consequently, it

is proposed that one should be able to choose the trajectory with the biggest transition from noise to transient, thus giving more confidence in the noise segmentation results.

4.2.3 Parameter Setting

The parameters involved in the noise segmentation module should be studied experimentally to give solutions as optimal as possible. The most obvious parameter is the rectangular window size we consider both for the local variance dimension calculation and the Rényi generalized dimension calculation. A relatively large window size causes the dimensions of locally distinct fractals to be buried within the dimension of their most significant neighbouring fractal, and at the same time it is computationally expensive. However, a too small window size provides insufficient data for the analysis. The window size should also be in the range where the signal can be considered stationary over the length of the window. The window shift parameter for the sliding window should be set to 1 to provide good localization of the start of the transient [Shaw97] [Toon97].

Another important issue is the triggering scheme. A threshold scheme is employed to detect the transition in the fractality from noise to transient. The scheme used in [Shaw97] is to find the earliest time when the dimension, $D_\sigma(t)$, compared with the mean, $\mu_{D_\sigma(t)}$, of the portion of the raw signal containing noise is sufficiently different as follows:

$$\left|D_\sigma(t) - \mu_{D_\sigma(t)}\right| > (\tau \cdot \mu_{D_\sigma(t)} + \sigma_{D_\sigma(t)}) \tag{4.1}$$

where τ is a certain threshold and $\sigma_{D_\sigma(t)}$ is the standard deviation of the signal. Note that both $\mu_{D_\sigma(t)}$ and $\sigma_{D_\sigma(t)}$ are calculated only for the noise portion of the signal. A questions

naturally arises: how do we calculate $\mu_{D_\sigma(t)}$ and $\sigma_{D_\sigma(t)}$ only for the noise portion without knowing the transient start first? The assumption here is that we consider the transient recording to be somewhat consistent, i.e., sufficient data are sampled prior to the start of the transient. Therefore, we can perform the calculation on a fixed number of samples which are in the noise portion for all the transient recordings.

4.3 Transient Feature Extraction

The transient feature extraction module produces a compact representation of a transient while retaining the important information for classification. As a result, the required storage space is significantly decreased and the classification can be achieved much more effectively.

Feature extraction is achieved using a similar technique as the variance fractal dimension trajectory for noise segmentation. Instead of shifting the window by one sample, a much bigger window shift is used. For example, if the transient size is 2048 samples in length, the window size is 512, and the window shift is set to 32, then we have 64 dimensions. The window shift is determined by the data reduction ratio to be achieved. It should be noted that the value should be selected carefully so that significant fractal characteristics are not neglected. The calculation of the variance dimension is different too. Fractal amplification [Kins94a] should be applied to reveal a higher degree of detail about the signal. This is implemented by using a dyadic sequence of time intervals to calculate the slope of the log-log plot instead of a linear sequence, resulting in relatively larger values and more variations of the dimension. For the purpose of feature extraction, fractal

amplification is preferred [KiGr95] because a higher degree of detail about the fractality of the signal can be achieved.

In addition to feature extraction ability, fractal modelling achieves *input data normalization* automatically. This is because there are theoretical lower and upper bounds for fractal dimension values regardless of the actual amplitude spread of the original signal. The variance dimension for a time series, in particular, has the lower bound of 1.0 and the upper bound of 2.0, as implied in Eq. 2.6. It should be noted that the experimental result may exceed the bound slightly due to numerical artifacts.The normalization is critical for the neural network classification, as explained in Chapter 3.

An alternative to transient modelling and feature extracting with variance fractal measures is provided by the multifractal analysis in phase space. By taking the nonuniform distribution into consideration in the multifractal analysis, we can reveal the underlying dynamics in the fractal sense. First, the strange attractor should be reconstructed from a transient signal. The generalized dimension spectrum is then obtained for the strange attractor using the correlation integral as defined in Eq. 2.21. The spectrum provides multifractal characterization of the strange attractor, which indirectly reveals the fractality of the original transient. The time delay and embedding dimension for strange attractor reconstruction should be determined according to the discussion in Section 2.4.3. More specifically, the autocorrelation function with a threshold value of 0.5 is used to find the time delay.

The output of this module should be a set of fractal dimensions as the extracted features and unique representations of the transient signals. These features are now ready to be used as neural network inputs.

4.4 PNN Classification

4.4.1 Preprocessing

Before the PNN classification, dimensionality reduction and clustering procedures are necessary to achieve faster training and classification. The PCA and SOFM techniques are employed for this purpose, as discussed in Chapter 3. The PCA processing results in a more compact representation of a transient because it removes the redundancy between the features we obtained from the previous module. The SOFM reduces the number of training cases through clustering.

The PCA procedure is executed for each training case and unknown cases. The user should be able to specify the number of principal components, that is, the number of new features of a transient. The weight matrix necessary to calculate principal components in Eq. 3.10 is obtained based on the entire training set, because the PCA treats the training set as a single class and any categorization inherent is ignored [Mast95]. Then the elements in the training set are replaced by principal components calculated for each training case. During the classification, principal components are computed first for the unknown input case, then they are used for further classification.

The use of SOFM is activated when the user inputs more than a threshold number of training cases for a class. The threshold value can also be specified by the user. Since

our purpose is simply clustering training cases, not visualization, a one-dimensional map topology with 5 nodes is used in this thesis. The SOFM is then performed individually on each class whose training cases are more than the threshold number. After the SOFM, the number of remaining training cases for that class is equal to the threshold.

The output of this procedure is a much more compact set of features for a transient and a group of most representative training cases with the number not more than a pre-defined threshold value. It is then passed forward to PNN for training and classification purposes.

4.4.2 PNN Training

The PNN structure is established using a supervised learning scheme. That is, during the training process, selected training cases are presented to the neural network along with the correct output. The network then adapts itself to produce the correct output when encountering similar cases.

The PNN is first initialized with a single value of sigma and the optimal value for sigma is found through a measure of its performance during the training. The measure of performance used in this thesis is the error function defined as the total correct classification cases.

It should be noted that a bias is produced when a training case is compared to itself in the pattern layer, thus producing zero distance measure and maximum activation in the output. The holdout method [Spec90b] is used to solve this problem. When a training case is used as the input, it is temporarily removed from the pattern layer.

4.4.3 PNN Classification and Rejection Test

After the training of the PNN, a test of the PNN classification should be conducted. In this thesis, we try to classify transients to the accuracy of different serial numbers of the same model. Therefore, the structure of the PNN is designed to include radios not only from different manufacturers and models, but also with different serial numbers of the same model. Such structure gives the PNN a complicated decision boundary and should be able to reveal the classification ability of the PNN in most cases.

The neural network's rejection ability is always of practical concern. When a completely new transient is presented to the PNN, it should be able to discard it, thus preventing the PNN from further processing effort and a possibly incorrect classification result. The neural network is provided with a rejection threshold on the summation neuron activation value. When all the summation activations are less than the threshold, it triggers an unknown transient.

4.5 User Interface Design

This work is based on the TAC-MM package designed by D. Shaw [Shaw97] using Visual C++ in MS Windows 95 environment. It is a 32-bit, single document interface (SDI) application. In addition to the implementation of the functions of each module, some modifications in the user interface design take place correspondingly.

Based on the functions performed by each module, the improved or added GUI functions in this thesis are:

- The complete information of the parameter setting is available to the user and

the user can specify individual parameters;

- The output file should include the information of the parameter setting as well;
- The user is able to specify the threshold for executing SOFM and the number of training cases to be reduced to (N). When the number of training cases from a single class exceeds the threshold, the SOFM routine is performed and the N most representative cases are chosen for further processing;
- The user has the option to perform PCA processing, when required. Also the number of new extracted features can be specified by the user.

An overview of the TAC-MM functions is provided in Appendix A.

4.6 Summary

This chapter deals with the major tasks to be accomplished in the thesis. Details of each module's input, processing procedure, and output are provided. Based on the system design we proposed, the software implementation is realized along with the corresponding user interface functions.

Chapter 5 will present the experimental work as the realization and verification of the design issues we have proposed. Tests are conducted and results are presented along with conclusions.

CHAPTER V
EXPERIMENTAL RESULTS AND DISCUSSION

Based on the theory and the system design provided in the previous chapters, experimental work is accordingly conducted in the sequence as illustrated in Fig. 1.1. In this chapter, we first provide an overview of the signals studied in this thesis. The issues related to consistent noise segmentation are discussed in Section 5.2. The next section deals with approaches used for feature extraction. Section 5.4 focuses on the transient classification processing, including preprocessing and classification procedures. Experimental results are discussed in each section and a summary is provided at the end of this chapter.

5.1 Overview of the Transient Signals

The transmitter transient signals used in this thesis are provided by the Communications Research Centre (CRC), Ottawa. The signals are acquired from the discriminator output of an Icom IC-R7000 communications receiver and a SoundBlaster 16 sound card on a PC, with a sampling rate of 44.1 kHz and 16 bits accuracy per sample, as implemented in the early stage of this research [Toon97]. A single receiver is used for all recordings to ensure that the classification and identification are based on transmission alone. The squelch signal of the receiver is used as a marker for the start of a transient. When a state change in the squelch level beyond a certain threshold is detected, a fixed number of pre-trigger samples along with post-trigger samples are stored, resulting in a length of 8,192 samples for each recording. The decaying waveform observed in Fig. 5.1(b) is due to the high pass characteristic of the data recording subsystem.

The recorded transient signal contains ambient channel noise which is followed by the start of the transient. A typical waveform of such a recording is shown in Fig. 5.1.

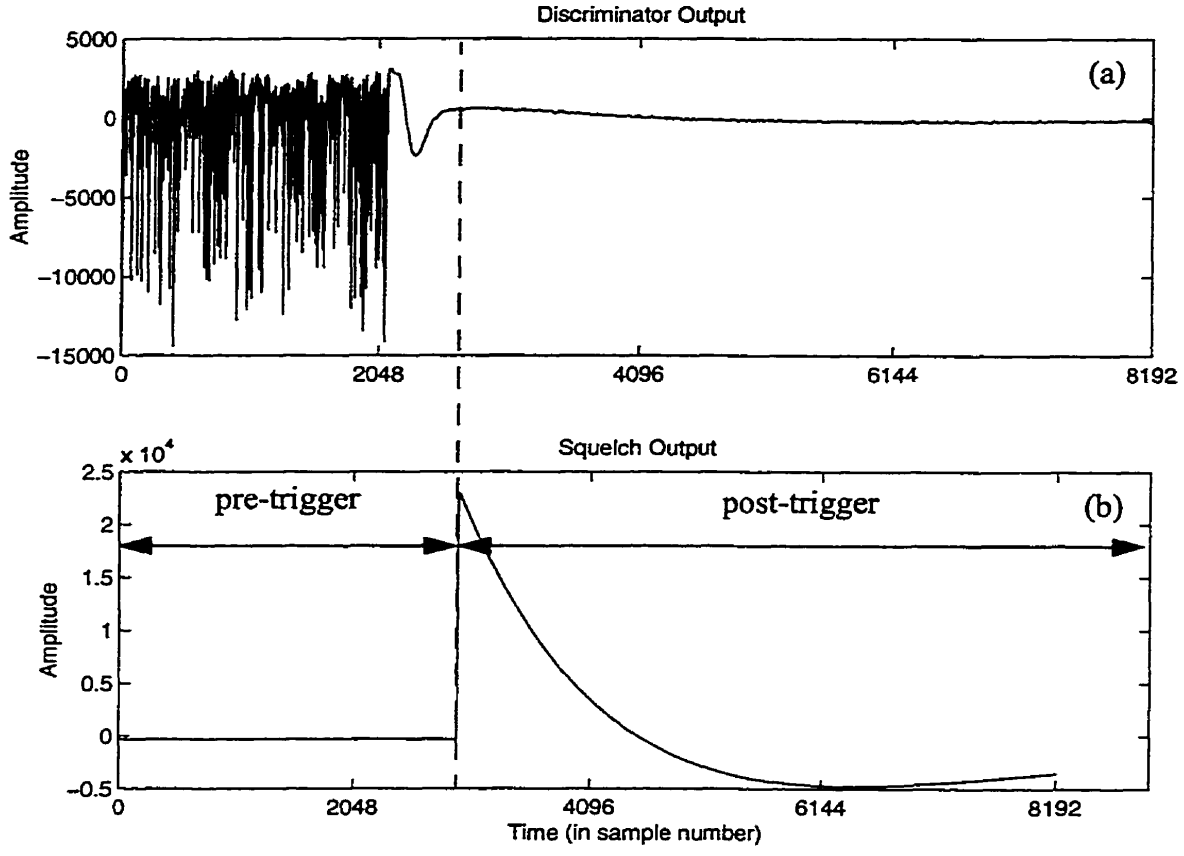


Fig. 5.1. The waveform of a transient recording from discriminator channel (a), along with the squelch channel output (b).

Our database consists of 20 radio transmitters, belonging to 10 different models from five manufacturers, namely, Force, Icom, Kenwood, Motorola, and Yaesu. Since we want to classify them not only by the model, but further down to the individual transmitter, each radio is treated as one class. A summary of all the radios is provided in Table 5.1.

Table 5.1: Summary of radios in experiment.

Manufacturer	Model	Radio Name	Number of Transients	Class Number
Force	CMH350	Force1	31	0
		Force2	31	1
		Force3	40	2
Icom	IC-251A	Icom1	40	3
		Icom2	40	4
	IC-A22	Icom3	100	5
Kenwood	TH25AT	Ken1	40	6
		Ken2	40	7
		Ken3	40	8
	TH21AT	Ken4	40	9
	TH-79A	Ken5	40	10
Motorola	MCX100	Mot1	40	11
		Mot3	40	12
	VISAR	Mot4	40	13
		Mot6	40	14
	HT-1000	Mot5	40	15
		Mot7	40	16
		Mot8	40	17
		Mot9	40	18
Yaesu	FT208R	Yaesu1	31	19

5.2 Noise Segmentation

5.2.1 Threshold Triggering Scheme (Absolute vs. Relative)

As discussed in Chapter 1, the major concern in noise segmentation is the consistent separation of transient from channel noise. The fundamental assumption used here is that noise and transient have different degrees of complexity, thus leading to different

fractality. A natural approach is, therefore, chosen to be a straightforward threshold triggering scheme. The approach used in [Shaw97] compares the variance dimension along the dimension trajectory with the mean of the first quarter of the raw signal, which is the minimum duration of the noise. The start of the transient is determined to be the earliest time when it satisfies the condition

$$\left|D_\sigma(t) - \mu_{D_\sigma(t)}\right| > (\tau \cdot \mu_{D_\sigma(t)} + \sigma_{D_\sigma(t)}) \tag{5.1}$$

where τ is the threshold. As discussed in Section 4.2.3, the mean value of variance dimension trajectory, $\mu_{D_\sigma(t)}$, and the standard deviation $\sigma_{D_\sigma(t)}$ are calculated only for the noise portion of the signal. It is observed that for all transient recordings used in the experiment, there are at least 1800 samples prior to the start of the transient, therefore $\mu_{D_\sigma(t)}$ and $\sigma_{D_\sigma(t)}$ are calculated for the first 1800 samples.

An example of this implementation in operation is shown in Fig. 5.2. From the figure, a problem with the absolute difference triggering scheme in Eq. 5.1 is found that it ignores the transition in dimension values between adjacent locations, but only looks for the first value which falls outside a certain range, i.e., $[(\tau - 1) \cdot \mu_{D_\sigma(t)} + \sigma_{D_\sigma(t)}, (\tau + 1) \cdot \mu_{D_\sigma(t)} + \sigma_{D_\sigma(t)}]$. For some cases where there is considerable fluctuation in the variance dimension trajectory around the start of the transient, due to inconsistent recording or the complexity of the transient itself, this scheme might not give the desired result. An improvement is suggested by the use of relative difference, which takes into account the transition between points on the variance dimension trajectory. It is expressed as

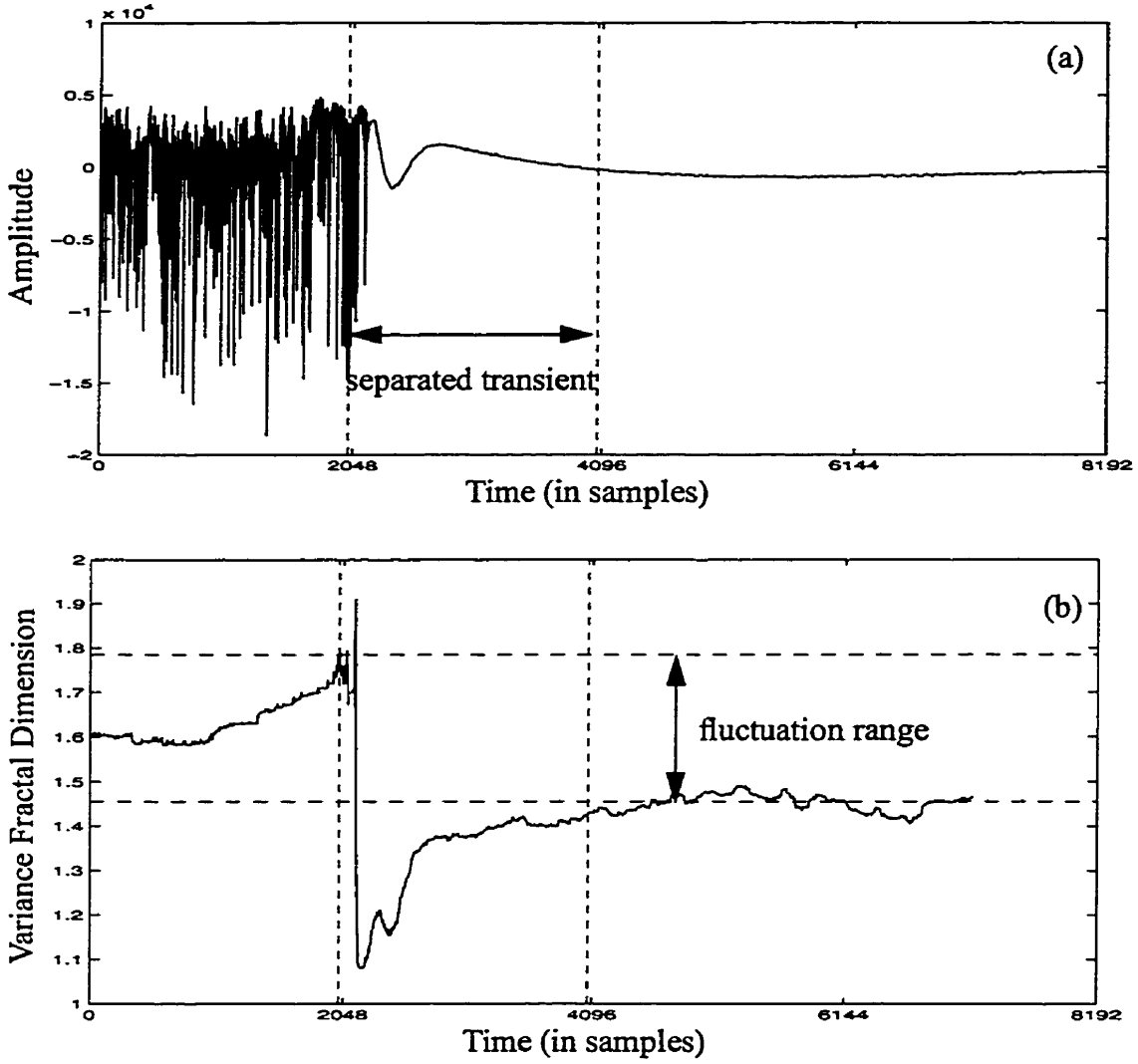


Fig. 5.2. Noise segmentation using absolute difference triggering.
(a) Raw signal. (b) Variance dimension trajectory.

$$\left|D_{\sigma}(t) - D_{\sigma}(t-1)\right| > \eta \cdot (\tau \cdot \mu_{D_{\sigma}(t)} + \sigma_{D_{\sigma}(t)}) \tag{5.2}$$

where η is the adjustment parameter. Based on this scheme, the transient signal in the previous example is separated as shown in Fig. 5.3.

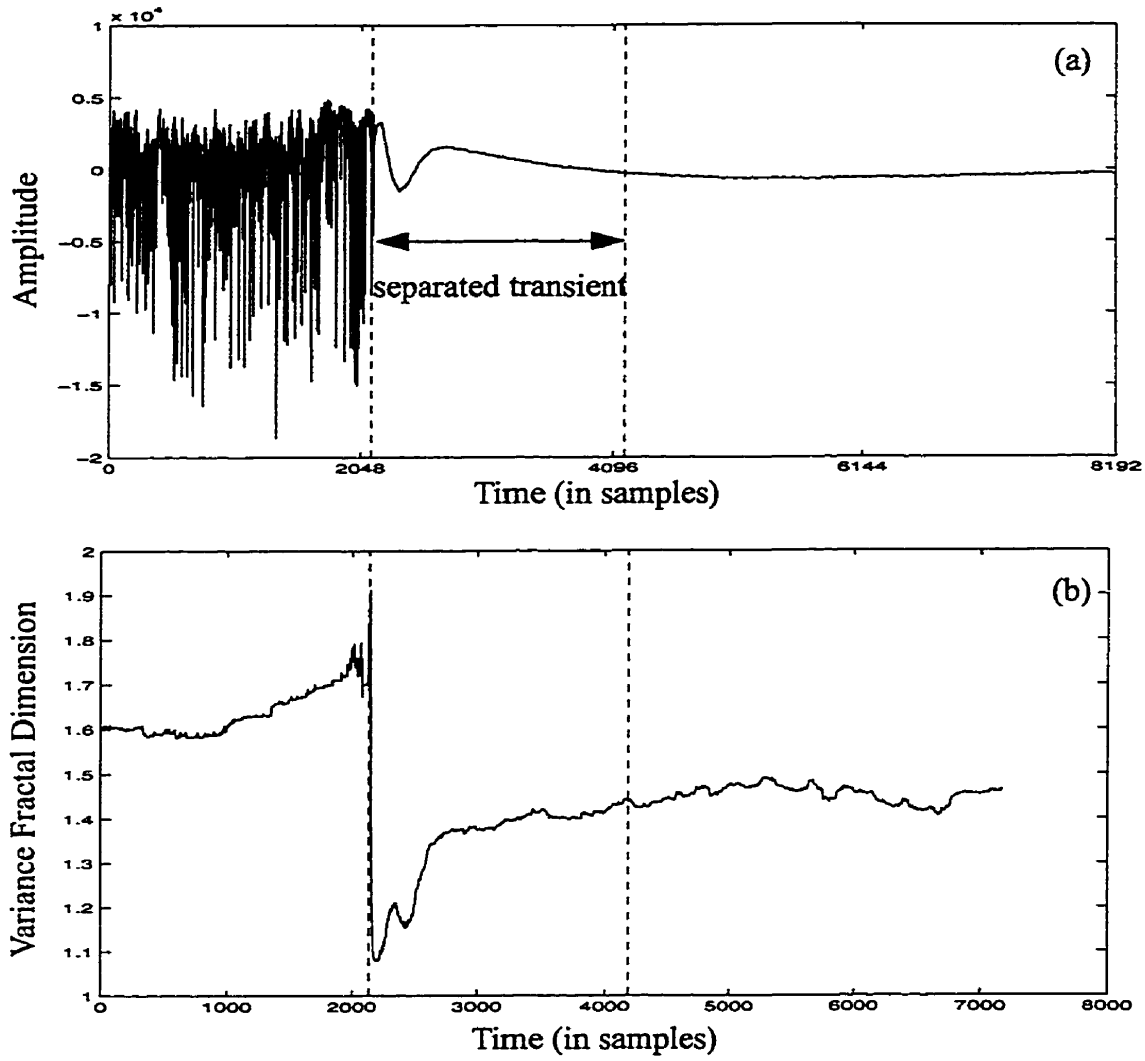


Fig. 5.3. Noise segmentation using relative difference triggering.
(a) Raw signal. (b) Variance dimension trajectory.

The conjecture here is that, by focusing on large transitions in relative difference triggering, more consistent segmentation of transients can be achieved. Further verification in terms of classification performance is presented later in this chapter.

5.2.2 Parameter Setting for Variance Dimension Calculation

There are several parameters involved in variance fractal dimension calculation for separating noise from transient. As discussed in [Shaw97] [Toon97], the shift parameter for the sliding window is set to 1 to provide good localization of the start of the transient.

The size of the sliding window also needs to be determined. First of all it should meet the requirement that the signal portion contained in the window can be considered stationary. Our study shows a window size in the range of 128 to 1024 samples is appropriate (with the sampling rate of 44,100 kHz). A study on the effect of the window size is shown in Fig. 5.4. Generally speaking, a smaller window size results in more variations in the trajectory, while a larger window size produces smoother trajectory. For our purpose here, we are looking for a large transition from noise to transient, not the fine detail, therefore it is more appropriate to choose a larger window size. In this thesis, it is found that a window size around 1024 (samples) gives good noise segmentation results.

The threshold, τ, and the adjustment parameter, η, are found by trial and error. In most cases, a value of τ around 0.08 and η around 0.5 lead to successful noise segmentation.

5.2.3 Multifractal Surface

In the previous chapters, we have concluded that a single definition of fractal dimension is not sufficient to describe nonstationary signals such as transients. Instead, Rényi generalized fractal dimensions allow us to consider different distributions of the fractal object on different subareas. For each sliding window along the time axis, a multifractal spectrum is obtained instead of a single variance fractal dimension, resulting in a

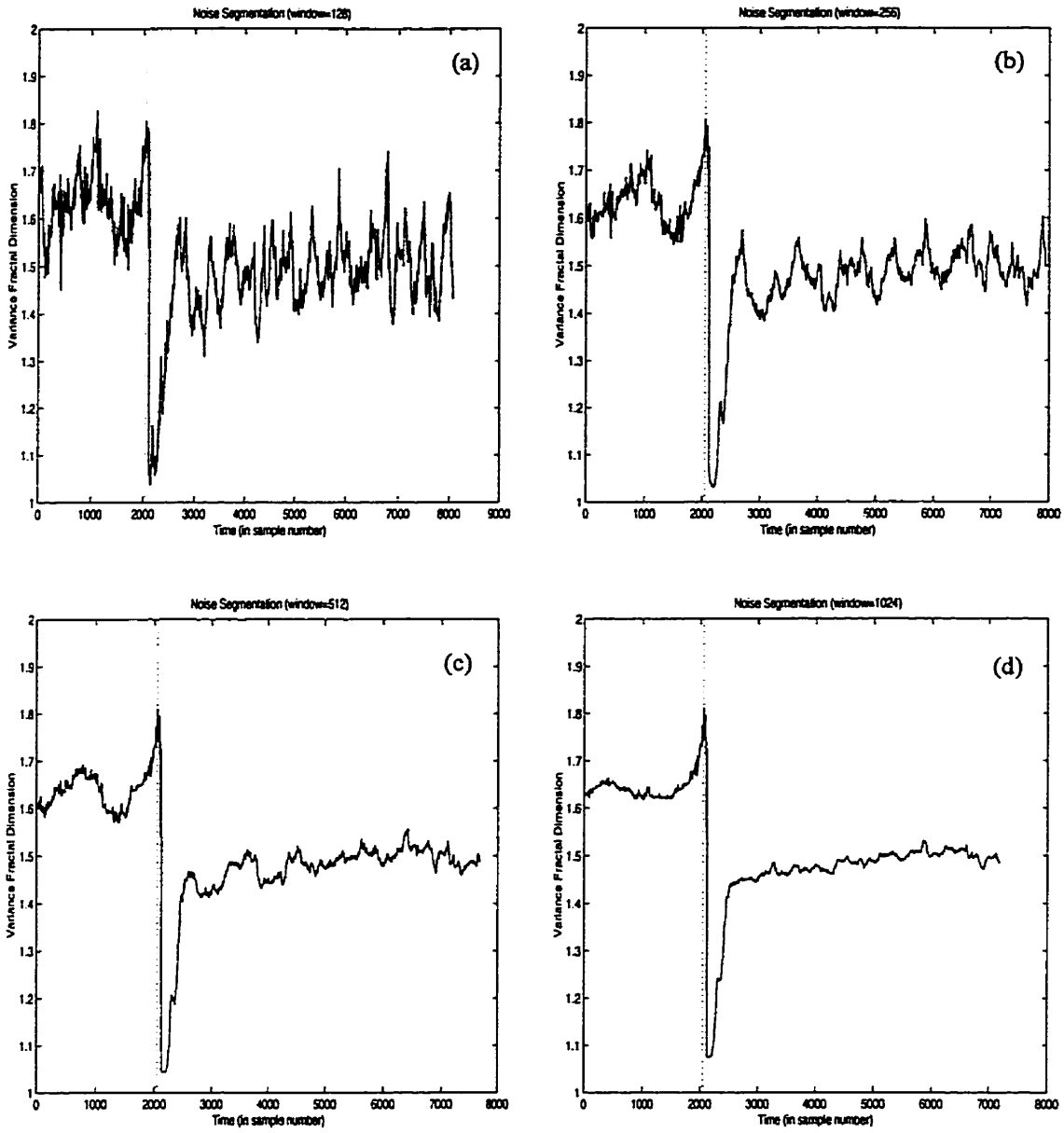


Fig. 5.4. The effect of window size for variance dimension trajectory calculation. Window size (in number of samples) is (a) 128; (b) 256; (c) 512; and (d) 1024.

multifractal surface. Similar to variance dimension trajectory, the multifractal surface reveals the time-evolving multifractality of the transient signal, thus allowing localization in time, which is essential for noise segmentation. Figure 5.5 shows a multifractal surface for a transient signal.

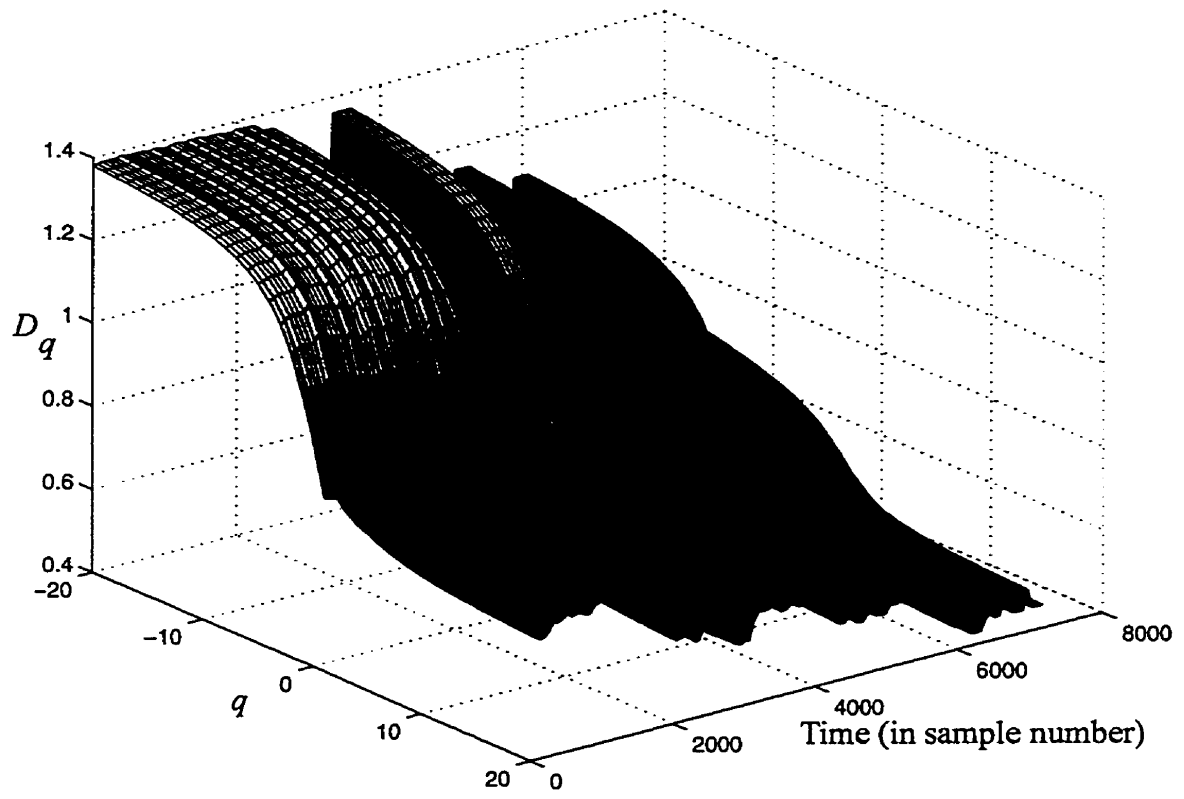


Fig. 5.5. Multifractal surface for a transient signal.

It is important to observe from the figure that there exists a much bigger transition from noise to transient in the part of the surface corresponding to the negative moment order q. A reasonable speculation is that we might be able to segment noise consistently by choosing the trajectory which always has the largest transition from noise to transient. [SuKi98]. From Fig. 5.5, it is also observed that the spread of the multifractal spectrum is quite large at the beginning, while it is smaller at the end of the signal. This means the degree of fractality of noise and transient is large while that of the portion after the transient is low. The multifractal surface provides us with much more information about the underlying fractal for a transient signal than a single fractal dimension trajectory, since the

variance dimension trajectory can be considered a special case of the surface at q around 2. However, the surface contains a much larger amount of data, therefore some feature extraction procedure would be required to reduce the dimensionality and use these features to separate and characterize transients. Due to the time constraint, this study has not been completed.

5.2.4 Discussion of Results

The noise segmentation module performs the fractal and multifractal analysis, and in particular the variance fractal dimension trajectory and multifractal surface. The variance fractal dimension trajectory provides us with a practical and straightforward means of separating noise from transient, using a thresholding mechanism. The relative difference thresholding scheme is considered to detect the transient start more consistently than the absolute difference thresholding in most cases, therefore it provides better quality of inputs to the neural network. The effect of parameters used in the calculation are also studied and it is found a window size of 1024, a threshold value of 0.08 and 0.5 for the adjustment parameter give satisfactory results.

The Rényi generalized fractal dimensions are studied as an alternative to the variance dimension trajectory analysis. The multifractal surface reveals more information about the signal, and it allows localization in time since it is time-evolving multifractal characterization. It is possible to select the value of q which gives a larger transition from noise to transient, thus achieving more consistent noise segmentation. Further studies should be conducted to explore the rich information provided by the multifractal surface.

5.3 Feature Extraction

5.3.1 Parameter Setting for Variance Fractal Amplification

The shift between sliding windows is determined by the number of features we want to obtain. It is suggested that it is selected conservatively so that significant fractal characteristics are not neglected [Shaw97]. For a transient size of 1536 samples, a window size of 512 and a window shift of 32 or 48 are found to be suitable, which lead to 48 or 32 features.

The window size should be about the same as the size used in noise segmentation to give statistical validity while retaining important fractal information.

5.3.2 Second and Third Run of Fractal Dimension Trajectory

This is motivated by looking closer at the variance fractal dimension trajectory we obtained in Section 5.2. It is observed that the trajectory itself exhibits some fine details and it is suspected to be fractal again. Therefore, a second run of fractal dimension calculation is performed on this trajectory as a "differencing" operator to explore higher-order fractality information about the original signal. Similarly, we can do the same calculation on this trajectory to obtain the third-run variance dimension trajectory, and so on. The importance of the higher order runs is that they reveal how complex the time series is in the fractal domain. The waveforms of these trajectories are shown in Fig. 5.6.

There are some interesting characteristics in the second and third run of variance fractal dimension trajectories. First, the existence of non-flat second and third run fractal dimension trajectories demonstrates that transients are very complex in the fractal domain,

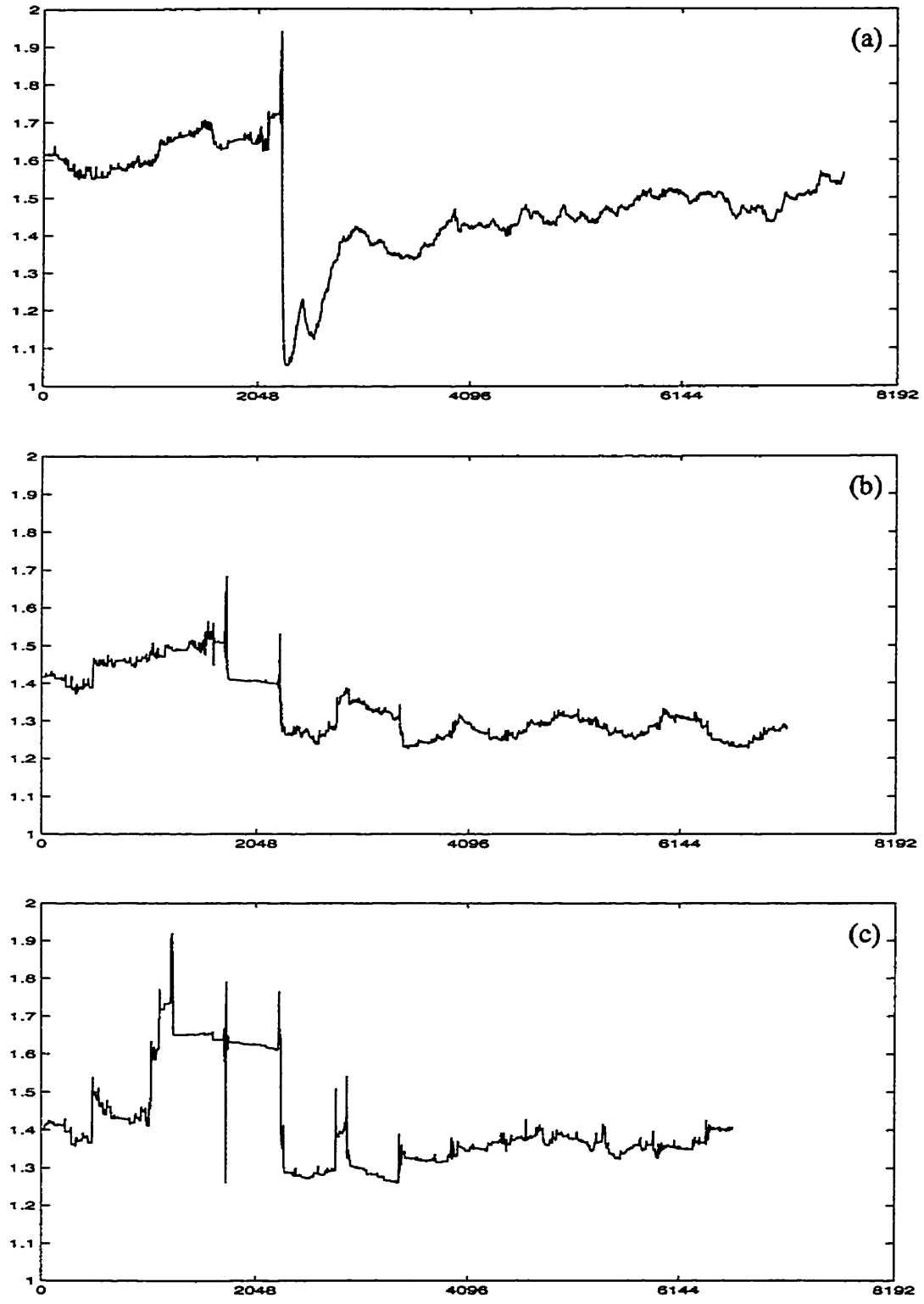


Fig. 5.6. Variance fractal trajectories. (a) First run: based on the original transient signal. (b) Second run: based on the trajectory in (a). (c) Third run: based on the trajectory in (b).

as their fractality is higher order. In comparison with the original trajectory, average values of the fractal dimension are smaller; i.e., the complexity of the trajectory is lower than the signal itself. It is also shown that there are more flat portions in these trajectories, which is similar to the effect of the differencing operation. This differencing effect suggests we may achieve more compact representation of the original signal by reducing the order of the variance dimension trajectory. It is seen that there are also more spikes in these trajectories, and the source of these spikes is considered as a noise component in the trajectories and numerical artifacts. In general, the existence of non-flat second and third run fractal dimension trajectories demonstrates that a transient signal is very complex in the fractal domain, as the fractality is higher order.

We have performed classification based on the second-run variance dimension trajectory. The classification result is found to be generally poorer than what the first-run trajectory provides, as shown in Table 5.2. However, more study is needed in the future to look into the potential of extracting features from the second-run trajectory.

Table 5.2: Classification results based on the second-run variance dimension trajectory.

Radio Name	mot5	mot7	mot8	mot9
mot5	**15**	1	2	0
mot7	0	**9**	5	5
mot8	1	3	**2**	5
mot9	4	7	11	**10**
Unknown	0	0	0	0
Total	20	20	20	20
P_c(first-run)	1.0	0.55	0.35	0.5
P_c	**0.75**	**0.45**	**0.1**	**0.5**

5.3.3 Strange Attractor Reconstruction and Transient Characterization

Most of studies on nonstationary temporal signal analysis are conducted in time or frequency or time-frequency domain. This is because we can get immediate and direct interpretation of the original signal. In comparison, chaos analysis examines the underlying dynamical system responsible for such signals, thus characterizing them indirectly. The analysis is performed on strange attractors, which are transformations of original temporal signals from the time domain to the phase space domain. The objective is to investigate the validity of this analysis on transient signals and the possibility of using strange attractors to characterize transients.

In an experimental environment, we only have a measured time series instead of recordings of a set of independent variables. The strange attractor reconstruction is therefore needed. There are two issues that need to be addressed in strange attractor reconstruction. First, the representation of a strange attractor is dependant on the location of the first sample from which it is reconstructed. This dependance is very strong when the number of samples is small, and it diminishes exponentially as the number increases. Consequently, a sufficiently small number of samples can always be determined beyond which the error is negligible, as demonstrated by other research in this group [Chen97]. Furthermore, as discussed in Chapter 1, since consistent segmenting the transient from noise cannot be guaranteed on some transmitters, there will be additional dependance on the reconstructed strange attractor. Still another issue is that, since transients are nonstationary, their strange attractors evolve in time. In our study, we assume that the reconstructed strange attractor is an "average" representation existing for a transient signal in the sense that we take the whole transient into consideration.

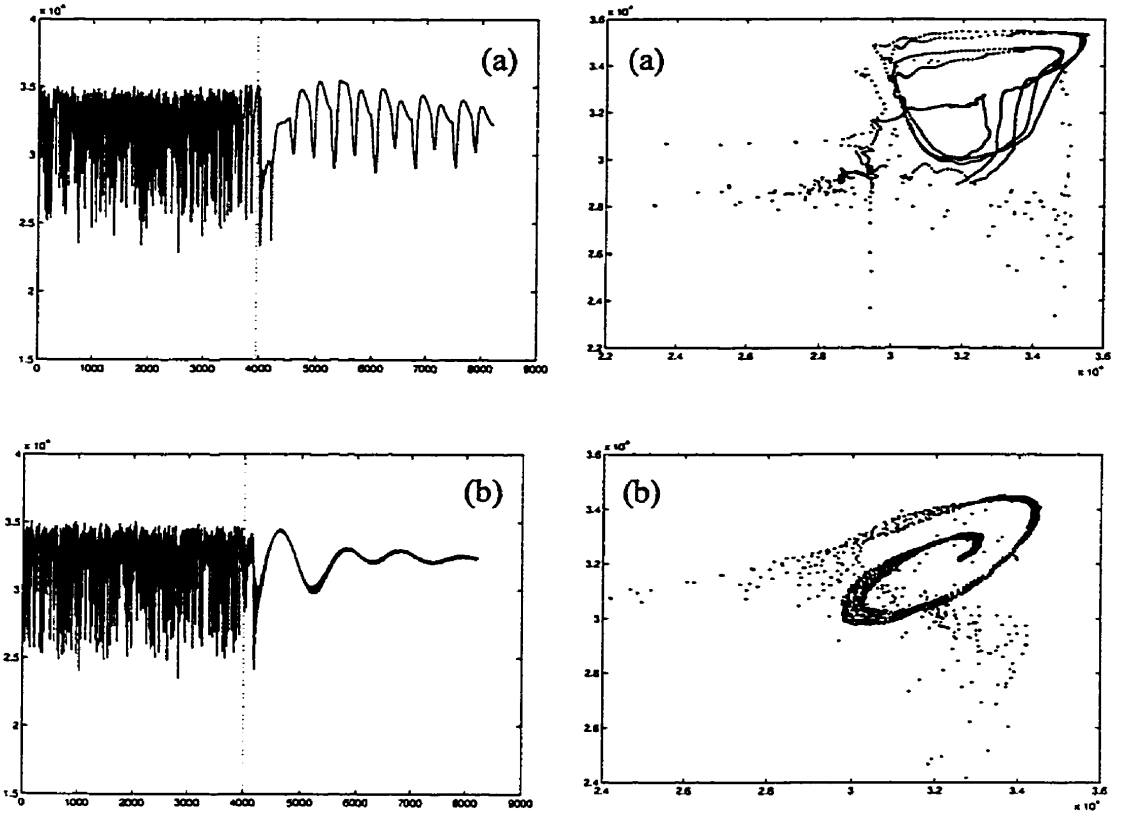


Fig. 5.7. Raw transient signals and the reconstructed strange attractors (shown in 2-D phase space). (a) Class 1 with delay = 48. (b) Class 2 with delay = 133. (after [SuKS99]).

The two parameters to be determined for strange attractor reconstruction are the time delay and the embedding dimension.

Time Delay, δ

The strange attractor is reconstructed using the time delay method. The objective here is to spread out the strange attractor in phase space in order to reveal the actual attractor geometry. The choice of time delay is made by the autocorrelation function criterion, using Eq. 2.17. The time delay is selected as the value which first gives the autocorrelation function, C, equal to 0.5, as suggested in [Addi97]. Figure 5.7 shows the time delay for

two classes of transient signals based on this criterion. The corresponding strange attractors are then plotted in 2-D phase space to show the basic structure of the attractor. Note that the actual embedding dimension is usually larger than 2, as discussed next.

Embedding Dimension, *m*

The method we use in this thesis to determine the embedding dimension for the transient signal is to calculate the correlation dimensions for the reconstructed attractor in the embedding spaces of successively larger dimensions. The dimension values initially increase with the embedding dimension, reaching a limiting value when the embedding space is large enough for the attractor to untangle itself. The experimental results show that there exists a low-dimensional strange attractor for radio transmitter transients, as the correlation dimensions saturate when we keep increasing the embedding dimension, as shown in Fig. 5.8. The saturation also demonstrates that a few thousand data samples (2048 samples in our case) is statistically valid to reconstruct the strange attractor and reveal the chaotic behaviour of the underlying dynamics.

The embedding dimension for transients is found to be a value between 5 and 8. Note that in Fig. 5.8., after a certain point where the correlation curves converge, the curves diverge again and the slopes increase. This is due to the noise component in the attractor [Addi97]. If the signal is a pure noise, there is no tendency of convergence with increasing embedding dimension, as shown in Fig. 5.9.

Generalized Dimensions for Reconstructed Strange Attractors

First, the generalized correlation integral is calculated according to Eq. 2.21. Generalized dimensions are then obtained accordingly from the log-log plot using the linear

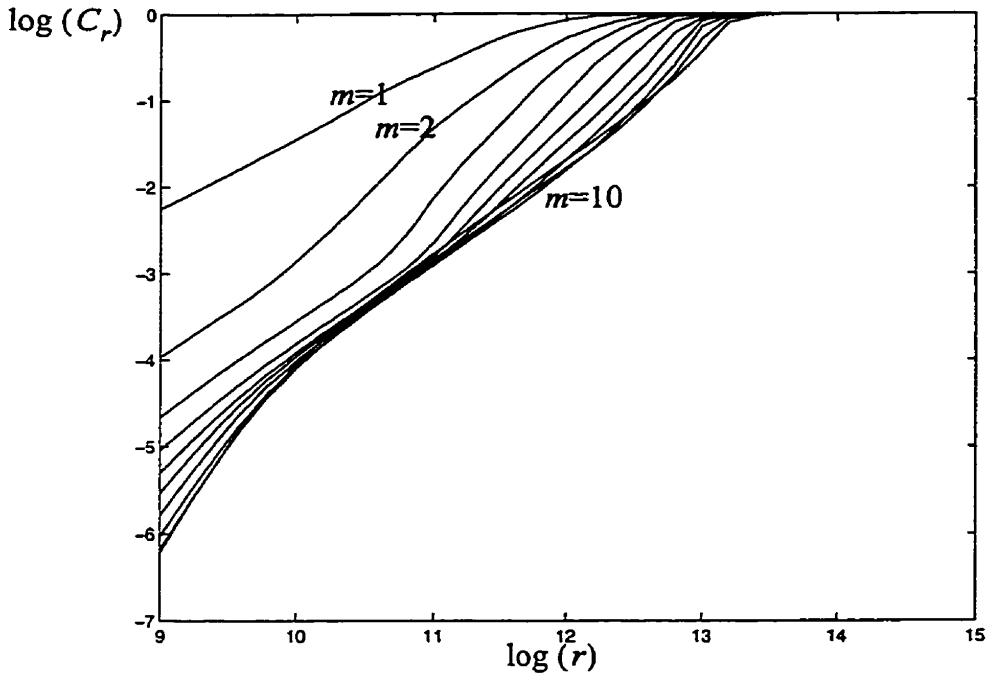


Fig. 5.8. Log-log plot for correlation dimension calculation with increasing embedding dimension, m (m=1 to 10) (after [SuKS99]).

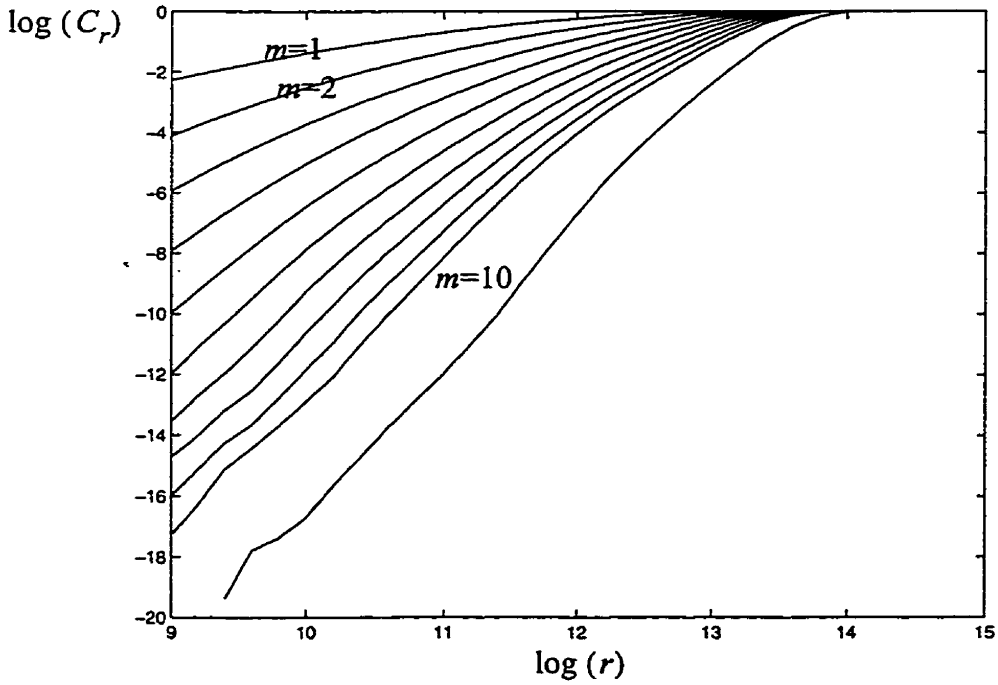


Fig. 5.9. Log-log plot of pure noise for correlation dimension calculation: no convergence with increasing embedding dimension, m (after [SuKS99]).

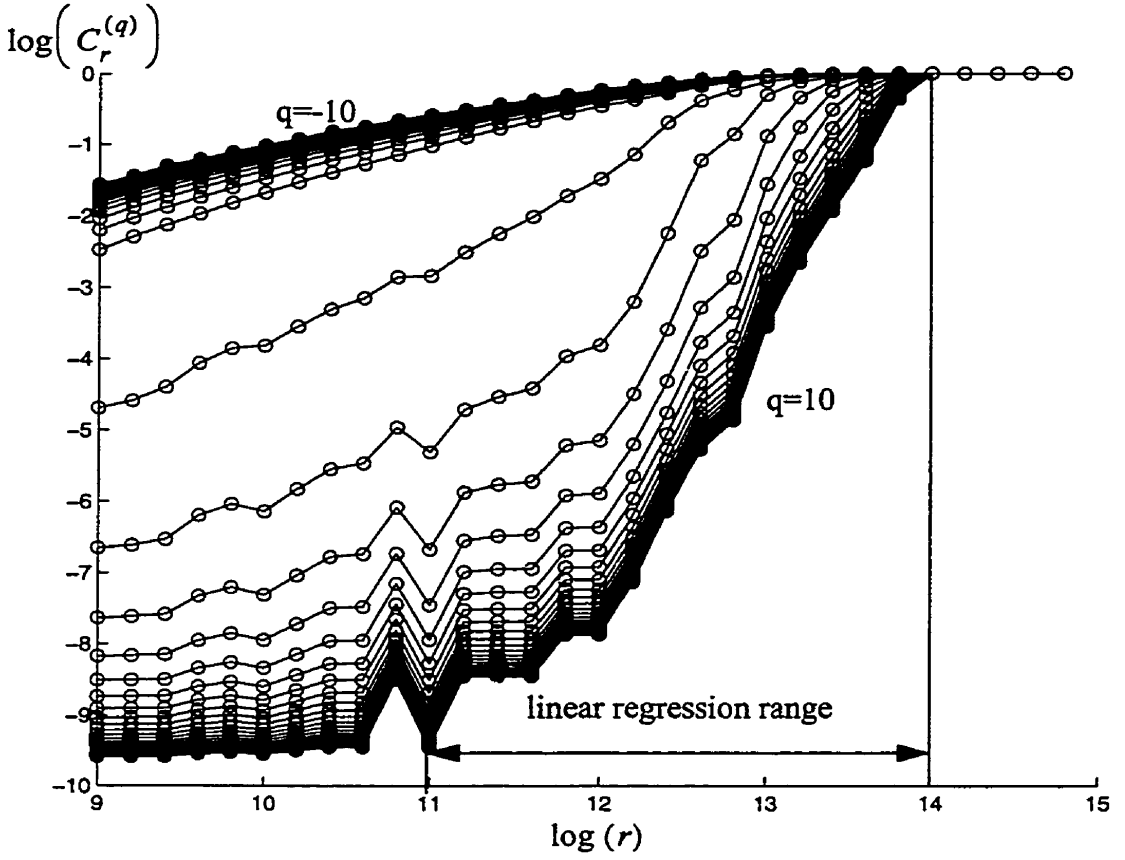


Fig. 5.10. Log-log plot for a reconstructed strange attractor (q=-10 to 10).

regression method. A typical log-log plot for a strange attractor reconstructed from a transient signal is shown in Fig. 5.10.

We need to choose the proper range where the log-log curve is close to linear in order to use the linear regression method, thus excluding the end effects due to saturation of the log-log plot. A practical way of doing it is to start from the first upper point where the log-log curves join, and end with the lower saturation beginning point. This range is selected to perform linear regression for slope calculation, and then fractal dimensions.

Based on the calculation consideration discussed above, the strange attractors for three classes of radio transmitters are reconstructed. There are *Force*, *Motorola*, and

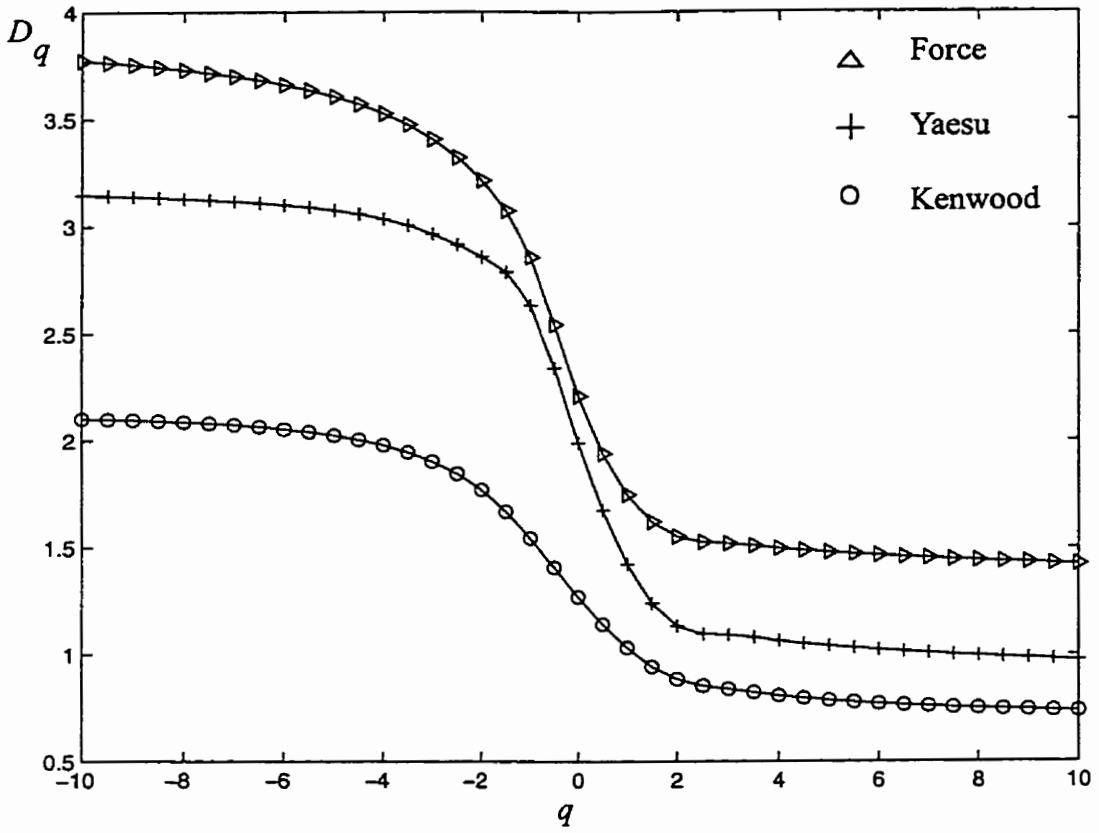


Fig. 5.11. Multifractal spectrum for three classes of radio.

Yaesu. The multifractal spectra of strange attractors from these three classes are obtained from the log-log plots, as shown in Fig. 5.11. It is very important to observe that the multifractal spectrum is separable for the three radio transmitters. This means the multifractal spectrum for the reconstructed strange attractor is capable of revealing the uniqueness of a radio transmitter, thus providing us with a means of feature extraction for the transient signals.

Generally speaking, multifractal measures of strange attractors provide us with the first insight of its ability to characterize a nonstationary signal, and it shows it is worth further study. Some problems exist with this technique. First, even though there is saturation

in fractal dimensions when we increase the embedding dimension, the small number of samples for a transient cannot provide us with a sufficiently dense strange attractor. Therefore, the scaling property of the strange attractor is limited to certain resolutions. Small number of samples for the transient also results in the sensitivity of strange attractor representation and the difficulty of determinating dimension values on the log-log plot. The *bootstraping* technique is usually used for assessing the accuracy of a parameter estimator in small data-sample situations. The assumption using the bootstraping is that the signal under study can be characterized by a well-defined statistical model. Since transient signals are nonstationary, this assumption is not satisfied in most cases where the underlying dynamics are complex. The use of the bootstraping technique is suggested only when the chaos is preliminary, which means some approximation of a statistical model can be achieved. Further study on the validity of this technique should be performed. Another practical issue is the difficulty in determining the linear regression range. Currently the linear regression range is determined visually. Another disadvantage is that the computation time is much longer than that of variance dimension trajectory calculation.

5.3.4 Discussion of Results

Several approaches to extract features from separated transients are studied. Variance fractal feature extraction uses a similar procedure as in noise segmentation, and the window shift is found to be 32 or 48 samples for a transient size of 1536 samples. A larger shift might result in possible loss of information while smaller values will be computationally expensive and will produce redundant information.

The second and third run of variance fractal dimension trajectories are calculated in order to obtain information on higher-order fractality of the underlying signal. The differencing effect achieved by these trajectories suggests another approach to modelling transients and extracting constant features from theses signals.

Multifractal analysis is used for feature extraction in terms of strange attractor fractal characterization in phase space. First the strange attractor for a transient signal is reconstructed using the time delay method. The time delay is determined by the autocorrelation criterion. It shows there exists a low-dimensional strange attractor for transients, and the embedding dimension for the reconstruction is found to be some value around 8, where the fractal dimension of a strange attractor saturates. Then a multifractal spectrum is obtained for the strange attractor from a log-log plot. The spectrum demonstrate the strange attractor is a multifractal and it can be used as a unique representation of the original signal.

5.4 PNN Classification

5.4.1 Basic PNN Classification

As shown in Table 5.1, there are 20 radios from 10 models in our database, and each radio is identified as a distinct class. Therefore, the PNN is designed to classify not only radios from different models, but also radios having different serial numbers from the same model. The first 20 recordings of each radio are used for training and the rest are for classification, that is, there are 400 recordings in the training set and 433 in the classification set. We use the absolute triggering scheme and the suggested parameter settings as discussed earlier in this chapter. A single value of the scaling parameter σ for the PNN is

used. More details can be found in Appendix B for the network parameters. The confusion matrix is used to describe the classification results. The classification results are shown in Tables 5.3. and Table 5.4.

Table 5.3: Classification results of the PNN with absolute triggering scheme.

Class	0	1	2	3	4	5	6	7	8	9	10
0	**7**										
1		**11**	1								
2			**14**								
3				**19**	1						
4					**13**						
5				1		**79**					
6							**20**				
7								**20**			
8									**20**		
9										**20**	
10			2								**20**
11											
12					4						
13											
14					2						
15	4										
16											
17											
18											
19			3								
Unknown						1					
Total	11	11	20	20	20	80	20	20	20	20	20
P_c	**.64**	**1.0**	**.70**	**.95**	**.65**	**.99**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**

Table 5.4: Classification results of the PNN with absolute triggering scheme (cont'd).

Class	11	12	13	14	15	16	17	18	19
0									
1		1							1
2									1
3		1							
4		4							
5									
6									
7									
8									
9									
10	1								1
11	**19**								
12		**6**							
13		1	**8**	1		2	2		
14		2		**16**					
15					**20**				
16		2	5			**11**	10	11	
17		1	4	3		4	**7**	2	
18			3			3	1	**7**	
19									**7**
Unknown		2							1
Total	20	20	20	20	20	20	20	20	11
P_c	**.95**	**.30**	**.40**	**.80**	**1.0**	**.55**	**.35**	**.35**	**.64**

The classification results are quite satisfactory for class 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 14, 15, with an average classification rate (P_c) of 0.97. It is also observed that the classifi-

cation results of classes 0, 2, 4, 12, 13, 16, 17, 18, 19 are very poor. Further examination of transients from these radio transmitters and improvement is thus required.

5.4.2 Classification Based on Relative Triggering

To improve the classification, first we want to improve the consistency of noise segmentation, therefore we use the relative triggering scheme instead of the absolute triggering. The basic PNN is retrained and classification results are shown in Table 5.5 and Table 5.6.

Table 5.5: Classification results of the PNN with relative triggering scheme.

Class	0	1	2	3	4	5	6	7	8	9	10
0	**10**		1								
1		**11**									
2			**15**								
3				**19**	1						
4					**19**						
5				1		**79**					
6							**20**				
7								**20**			
8									**20**		
9										**20**	
10											**20**
11											
12											
13											
14											
15	1										
16											
17											

Table 5.5: Classification results of the PNN with relative triggering scheme.

Class	0	1	2	3	4	5	6	7	8	9	10
18											
19			2								
Unknown			2			1					
Total	11	11	20	20	20	80	20	20	20	20	20
P_c (before)	.64	1.0	.70	.95	.65	.99	1.0	1.0	1.0	1.0	1.0
P_c	**.90**	**1.0**	**.75**	**.95**	**.95**	**.99**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**

Table 5.6: Classification results of the PNN with relative triggering scheme (cont'd)

Class	11	12	13	14	15	16	17	18	19
0									1
1									
2									1
3		1		1					
4		3		1					
5									
6									
7									
8									
9									
10									
11	**20**								
12		**6**		1				2	
13		3	**11**			2	1	1	
14			1	**16**				1	
15				1	**20**				

Table 5.6: Classification results of the PNN with relative triggering scheme (cont'd)

Class	11	12	13	14	15	16	17	18	19
16			3			**11**	11	4	
17		1	3			2	**6**	1	
18		2	2			5	2	**11**	
19									**7**
Unknown		4							2
Total	20	20	20	20	20	20	20	20	11
P_c (before)	.95	.30	.40	.80	1.0	.55	.35	.55	.64
P_c	**1.0**	**.30**	**.55**	**.80**	**1.0**	**.55**	**.35**	**.35**	**.64**

We can see from tables above that there is overall improvement as the result of using relative triggering. However, the improvement on difficult radios is not significant. Further examination of the original recording shows that some of the recordings which cause failure in classification have abnormal waveforms due to inconsistent transient recording. Examples of such recordings are shown in Fig. 5.12. Such bad recordings are found in the training set as well. Therefore, the quality of the training set is another reason for poor classification performance. Since the transient recording system is not covered in this thesis, no further improvement on the classification performance can be achieved without changing the transient recording system. To correctly evaluate the classification ability of the PNN, radios having abnormal recordings in either the training or classification set are not considered in the evaluation.

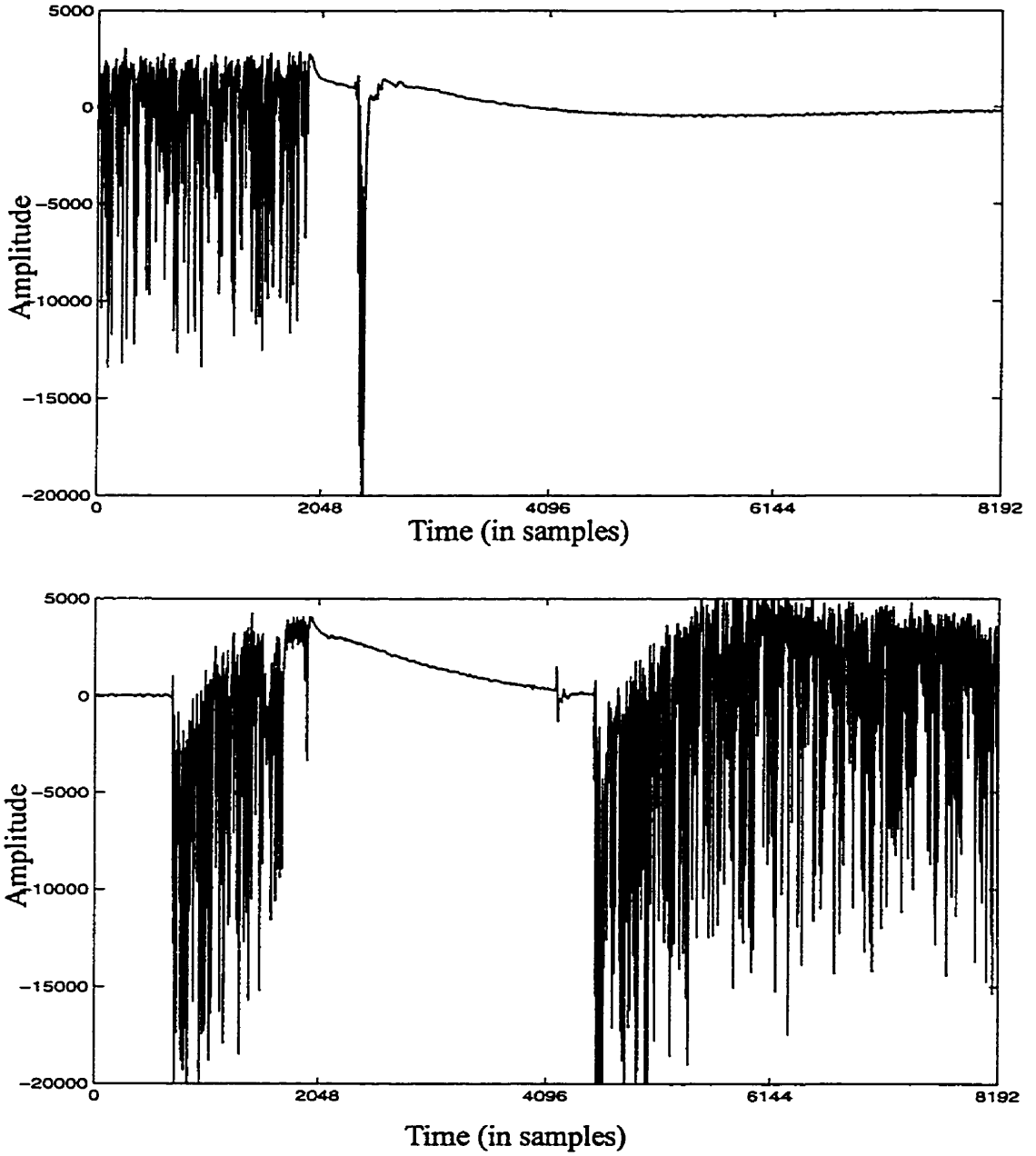


Fig. 5.12. Waveforms of transients that cause classification failure.
(a) Noise in the transient part. (b) Noise follows the transient.

5.4.3 PNN Preprocessing Using PCA

The PCA processing is performed before the basic PNN processing to speed up the classification procedure by reducing the dimensionality of each input vector. We set the

dimensionality of the input vectors to be reduced from 32 to 6, and the classification procedure is repeated. Results are provided in Table 5.7 to Table 5.9. The time is measured in millisecond.

Table 5.7: Classification results comparison: without PCA vs. with PCA.

Class	0	1	2	3	4	5	6	7	8	9
P_c (without PCA)	.64	1.0	.70	.95	.65	.99	1.0	1.0	1.0	1.0
P_c (with PCA)	**.45**	**1.0**	**.60**	**.70**	**.70**	**.98**	**1.0**	**.90**	**1.0**	**1.0**

Table 5.8: Classification results comparison: without PCA vs. with PCA (cont'd).

Classification Rate	10	11	12	13	14	15	16	17	18	19
P_c (without PCA)	1.0	.95	.30	.40	.80	1.0	.55	.35	.35	.64
P_c (with PCA)	**1.0**	**.80**	**.30**	**.35**	**.75**	**.90**	**.30**	**.35**	**.30**	**.64**

Table 5.9: Processing time comparison: without PCA vs. with PCA.

Preprocessing Option	Training	Classification (per recording)
Without PCA	241213	9
With PCA	**107313**	**3**

As expected, the training and classification speed is improved significantly as the result of dimensionality reduction. It is demonstrated in the tables that the training time is reduced to less than a half of the original training time, and the classification time is reduced to one third. It should be noted that because the complete classification consists of fractal segmentation and modelling procedures on an unknown recording, which take about 1450 milliseconds per recording, the improvement on classification achieved by the

PCA is not obvious. It is also observed that there is trade-off between the classification performance and the dimensionality of inputs. It is thus concluded that the PCA should only be performed when the speed is of critical concern,.

5.4.4 PNN Preprocessing Using SOFM

The SOFM performs clustering on training data, and increase the classification speed by reducing the size of the training set. In this thesis, the SOFM processing is triggered by a threshold set by the user. When there are more transients than the threshold value for a class, the SOFM is executed for the class and the number of transients in that class is reduced to the threshold value by selecting the most representative ones. The threshold for executing SOFM is set to be 10 samples for a class. The SOFM output layer is set to be a one-dimensional map with the number of nodes is fixed to be 5. The result comparison is shown in Table 5.10 to 5.12.

Table 5.10: Classification results comparison: without SOFM vs. with SOFM.

Class	0	1	2	3	4	5	6	7	8	9
P_c (without SOFM)	.64	1.0	.70	.95	.65	.99	1.0	1.0	1.0	1.0
P_c (with SOFM)	**.64**	**1.0**	**.70**	**.90**	**.65**	**.93**	**1.0**	**1.0**	**1.0**	**1.0**

Table 5.11: Classification results comparison: without SOFM vs. with SOFM (cont'd).

Class	10	11	12	13	14	15	16	17	18	19
P_c (without SOFM)	1.0	.95	.30	.40	.80	1.0	.55	.35	.35	.64
P_c (with SOFM)	**1.0**	**.95**	**.25**	**.45**	**.65**	**1.0**	**.65**	**.20**	**.20**	**.64**

Table 5.12: Processing time comparison: without vs. with SOFM.

Processing Time	Training	Classification (per recording)
Without SOFM	241213	9
With SOFM	**124808**	**4**

It is seen that the training time of the PNN is significantly reduced by the SOFM preprocessing. In the experiment, the training time is reduced to about a half. The classification speed is also improved noticeably. Trade-off again exists between the classification performance and the processing speed. This is due to the elimination of some training cases, since the PNN assumes a comprehensive training set to yield the best result. One possible solution to this problem is to eliminate only duplicate transients in the training set, and weight the contribution of each training case by the number of duplicates for the case in the training procedure.

5.5 Summary

In this chapter, we present the experimental results in a sequential order. Transient signals are first separated from channel noise according to their different fractality. With the variance fractal trajectory approach, the relative difference triggering scheme is found to produce more consistent segmentation than the absolute triggering scheme does, while it takes longer to process. The multifractal spectrum is studied as a better approach than the variance fractal trajectory in that it gives much more information about the underlying nonstationary signals.

The variance fractal trajectory method is also used for feature extraction. The effect of parameters for fractal calculation is discussed. Further study on second and third

runs of the trajectory suggests a more efficient way of transient representation. We also performed feature extraction on transient signals by strange attractor reconstruction and multifractal analysis on the attractors, which leads to good separability of transients from different radios.

A basic PNN is built for radios from each model or make, and the average classification rate is 97%, with the classification accuracy down to serial numbers. Both PCA and SOFM preprocessing schemes result in a significant improvement in the training and classification speed, with the trade-off on the classification performance.

CHAPTER VI
CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

In this thesis, a fast and reliable radio transmitter identification system is developed. The major tasks performed in the system are: channel noise segmentation, transient feature extraction and transient classification. The software package based on TAC-MM [Shaw97] is improved and updated accordingly.

The channel noise is segmented from the transient using variance fractal dimension trajectory analysis [Kins94a]. Relative difference triggering is found to provide more consistent segmentation results, while absolute difference triggering takes less time to process. Multifractal spectrum analysis is applied on the transient signals and it is concluded that it provides more information about the underlying signals than the variance dimension trajectory. As a result, multifractal spectrum analysis suggests a better way of separating noise from transient by choosing the largest transition from noise to transient on the multifractal surface.

Features contained in transients are extracted efficiently using a similar variance dimension trajectory analysis. A reduction ratio of 1536:32 (48:1) is found to provide sufficient representation of the transients. Second and third runs of the trajectory are performed and the results show further reduction can be achieved by considering the fractality of the trajectories, which gives more stable values of variance dimensions. An alternative approach is also employed to conduct feature extraction by multifractal charac-

terization of strange attractors of the transient signals. The results show the existence of strange attractors for transient signals and possible compact representation of the signals using the multifractal measures.

The PNN is used for classification and two techniques for preprocessing, PCA and SOFM, are employed to speed up the training and classification procedure. With these techniques, the training time and classification time are significantly reduced. The network consists of 20 radios from 10 different models, which has a complex decision boundary common in practice. The basic PNN has an average classification rate of 97% with the accuracy down to serial numbers. It is also observed that with preprocessing, there is trade-off on the classification performance. Therefore, when the speed is of critical concern, the preprocessing is desirable.

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

6.2 Contributions

This thesis had made the following contributions:

a) Multifractal spectrum analysis on nonstationary signals as the generalization of variance fractal dimension trajectory, which provides much more information about the underlying signals and suggests a better way of separating transient from noise;

b) Strange attractor reconstruction on transient signals and multifractal analysis on the attractors;

c) Preprocessing using PCA and SOFM to reduce the dimensionality of the PNN, thus achieving faster training and classification; and

d) Improvement on the software package TAC-MM to embed the new techniques.

6.3 Recommendations

The following recommendations are suggested for further research on this topic:

a) More experiments on feature extraction based on strange attractor characterization should be performed to test its ability to characterize radio transmitters from the same model uniquely and efficiently;

b) Second and third runs of variance dimension trajectories should be filtered to obtain constant fractal measures in these trajectories, thus achieving more compact representation of transients;

c) Exponential smoothing of the multifractal spectrum to obtain the Mandelbrot spectrum of the transient signals as an alternative to model the transients;

d) The bootstrapping technique should be applied to achieve confident and accurate statistical estimation (e.g. variance) of transients, which do not have a large set of data samples; and

e) Higher-order statistics to characterize transient signals while accounting for non-Gaussian characteristics and nonlinearities.

REFERENCES

[Addi97] P.S. Addison, *Fractals and Chaos: An Illustrated Course*, Institute of Physics Publishing, 1997.

[Ande95] D. Anderson, "Transient signal classification using wavelet packet bases," *B. Sc. Thesis*, Department of Electrical and Computer Engineering, University of Manitoba, 1995.

[BoJe70] G. Box and G.M. Jenkins, *Time Series Analysis, Forecasting and Control*, San Francisco: Holden-Day, 1970.

[BrDa87] P.J. Brockwell and R.A. Davis, *Time Series: Theory and Methods*, New York, NY: Springer-Verlag

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

[CPYS95] H.C. Choe, C.E. Poole, A.M. Yu and H.H. Szu, "Novel identification of intercepted signals from unknown radio transmitters," *Proc. of SPIE Wavelet Applications II*, vol. 1, pp. 504-517, April 1995.

[Diet94] J. Dietrich, "A wavelet analysis of transients in phase-locked loops," *B. Sc. Thesis*, Department of Electrical and Computer Engineering, University of Manitoba, 1994.

[Digg90] P.J. Diggle, *Time Series: A Biostatistical Introduction*, Oxford University Press, 1990.

[FrMe92] J. Froment and H. Messer, "The use of the wavelet transform in detection of an unknown transient signal," *IEEE Trans. Information Theory*, vol. 38, no. 2, pp. 892-897, 1992.

[FrSk91] J.S. Freeman and D.M. Skapura, *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison-Wesley Publishing Company, 1991.

[HiPa96] R.D. Hippenstiel and Y. Payal, "Wavelet based transmitter identification," *Proc. of International Symposium on Signal Processing and its Applications*, Gold Coast, Australia, pp. 740-743, August 1996.

[JoSm87] D.W. Jordine and P. Smith, *Nonlinear Ordinary Differential Equations*, New York, Oxford, 1987.

[Khan95] I. Khan, "Transient analysis in frequency synthesizers," *B. Sc. Thesis*, Department of Electrical and Computer Engineering, University of Manitoba, 1995.

[Kins94a] W. Kinsner, "Fractal dimensions: morphological, entropy, spectrum, and variance classes," *Technical Report DEL 94-4*, Department of Electrical and Computer Engineering, University of Manitoba, May 1994.

[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, June 1994.

[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.

[Koho82] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, pp. 59-69, 1982.

[Kola98] J. Kolakowski, "Evaluation of transient behaviour of radio communication transmitters using short-time Fourier transform," *Proc. of 14th International Wroclaw Symposium on Electromagnetic Compatibility*, pp. 162-165, Wroclaw, June 1998.

[Kwok95] R. Kwok, "High-speed capture of turn-on transients in transmitters," *B. Sc. Thesis*, Department of Electrical and Computer Engineering, University of Manitoba, 1995.

[Lang96] A. Langi, "Wavelet and fractal processing and compression of nonstationary signals," *Ph. D Thesis*, Department of Electrical and Computer Engineering, University of Manitoba, 1996.

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

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

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

[Meis72] W.S. Meisel, *Computer-oriented Approaches to Pattern Recognition*, New York: Academic Press, 1972.

[MoGr62] A.M. Mood and F.A. Graybill, Introduction to the theory of statistics, New York: Macmillan, 1962.

[PaCh87] T.S. Parker and L.O. Chua, "Chaos: A tutorial for engineers," *Proc. of the IEEE*, vol. 75, no. 8, pp. 982-1008, 1987.

[Papo84] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, New York: McGraw-Hill, 1984.

[Parz62] E. Parzen, "On estimation of a probability density function and mode," *Annals of Mathematical Statistics*, vol. 33, pp. 1065-1076, 1962.

[PaSc87] K. Pawelzik and H.G. Schuster, "Generalized dimensions and entropies from a measured time series," *Phys. Rev.*, vol. A 35, no. 1, pp. 481-484, 1987.

[Paya95] Y. Payal, "Identification of push-to-talk transmitters using wavelets," *Master's Thesis*, Department of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, California, 1995.

[PCFS80] N.H. Packard, J.P. Crutchfield, J.D. Farmer, and R.S. Shaw, "Geometry from a time series," *Phys. Rev. Lett.*, vol. 45, no. 9, pp. 712-715, 1980.

[PeJS92] H. Peitgen, H. Jügens, and D. Saupe, *Chaos and Fractals: New Frontiers of Science*, New York, NY: Springer Verlag, 1992.

[Prie81] M.B. Priestley, *Spectral Anlaysis and Time Series*, Vol. 2, Academic Press, 1981.

[Prie88] M.B. Priestley, *Non-linear and Non-stationary Time Series Analysis*, Academic Press, 1988.

[Rény55] A. Rényi, "On a new axiomatic theory of probability," *Acta Mathematica Hungarica 6*, pp. 285-335, 1955.

[Ruda94] T. Rudachek, "Phase-locked loops and their transients," *B. Sc. Thesis*, Department of Electrical and Computer Engineering, University of Manitoba, 1994.

[Scha97] R.J. Schalkoff, *Artificial Neural Networks*, McGraw-Hill, 1997.

[Shaw94] D.B. Shaw, "Identification of transients in phase-locked loops using neural networks," *B. Sc. Thesis*, Department of Electrical and Computer Engineering, University of Manitoba, 1994.

[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.

[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 Net-*

works, vol. 1, no. 1, pp. 111-121, 1990.

[SuKi98] L. Sun and W. Kinsner, "Fractal segmentation of signal from noise for radio transmitter fingerprinting," *Proc. of 1998 IEEE Can. Conf. Electrical & Computer Engineering*, vol. 2, pp. 561-564, May 1998.

[SuKS99] L. Sun, W. Kinsner and N. Serinken, "Characterization and feature extraction of transient signals using multifractal measures," *Proc. of 1999 IEEE Can. Conf. Electrical & Computer Engineering*, pp. 781-785, May 1999.

[Take81] F. Takens, *Lecture Notes in Mathematics*, Springer, vol. 898, pp. 266, 1981.

[Toon95] J. Toonstra, "Wavelet analysis and genetic modelling of transients in radio transmitters," *B. Sc. Thesis*, Department of Electrical and Computer Engineering, University of Manitoba, 1995.

[Toon97] J. Toonstra, "A radio transmitter fingerprinting system," *M. Sc. Thesis*, Department of Electrical and Computer Engineering, University of Manitoba, 1997.

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

[ZhLi93] X. Zhang and Y. Li, "Self-organizing map as a new method for clustering and data analysis," *Proc. of the 1993 Int. Joint Conf. on Neural Networks*, pp. 2448-2451, 1993.

APPENDIX A

OVERVIEW OF TAC-MM

A.1 Introduction

The software package TAC-MM used in the study of this thesis was developed by Kinsner and Shaw [Shaw97], and then evaluated and improved by CRC. It is a 32-bit, single document interface (SDI) Window 95 application, written in Visual C++ (version 5.0). Some modifications are implemented to include the additional user functions, as described in Chapter 4. An overview of the operation of TAC-MM is provided below.

A.2 The User Display Area

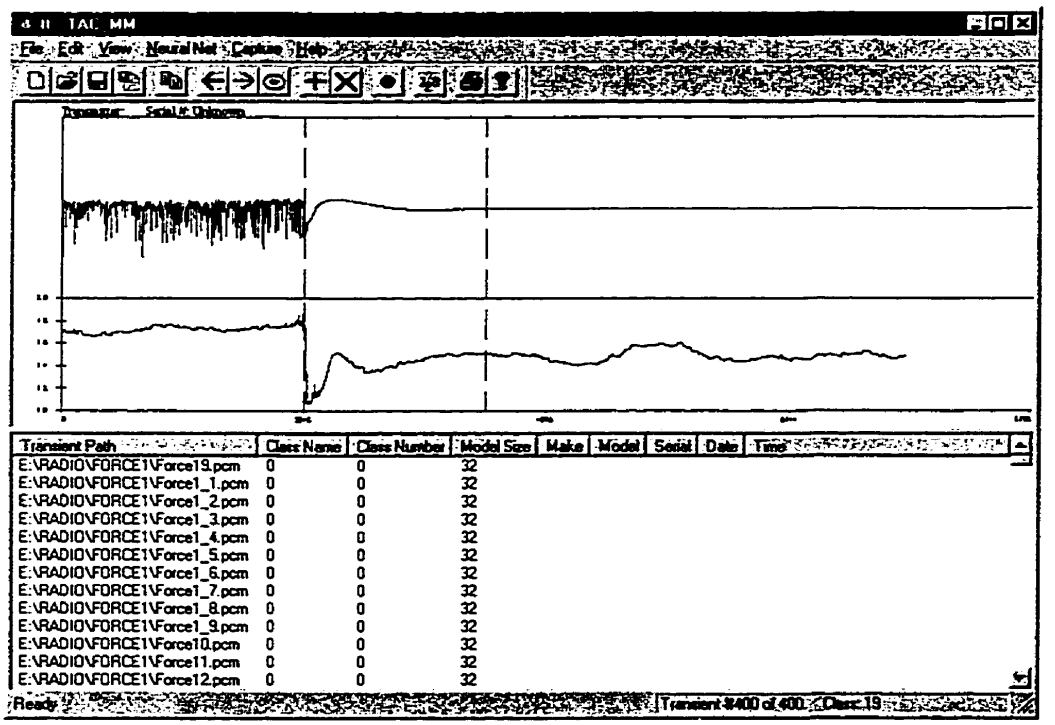


Fig. 1. The user display area.

The viewing area is divided into two windows. In the upper window, a raw signal and it variance fractal dimension trajectory is displayed. The user can choose to view the segmentation

as shown or to zoom in the transient only. The lower window can display the zoomed raw signal, the network details, and pertinent training and classification information with the options in the **View** menu.

A.3 The File Menu

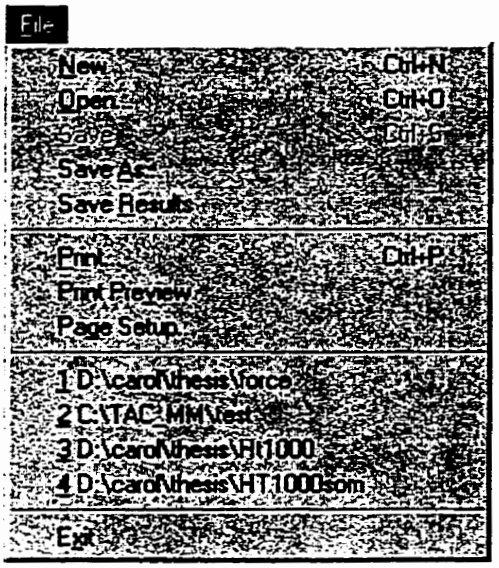


Fig. 2. The File pull-down menu.

The **File** menu has some common user functions, such as **Open, Save, Save as, Print, Print Preview, Page Setup** and **Exit**. It can store up to four recent files. The function **New** allows the user to start a new document (or database) of transient by bring up the dialog box shown in Fig. 3. It contains the complete parameter settings for noise segmentation and feature extraction using the variance fractal analysis described in Chapter 2., as well as PNN preprocessing options (PCA and SOFM) and related parameters. The suggested parameter settings are used as default.

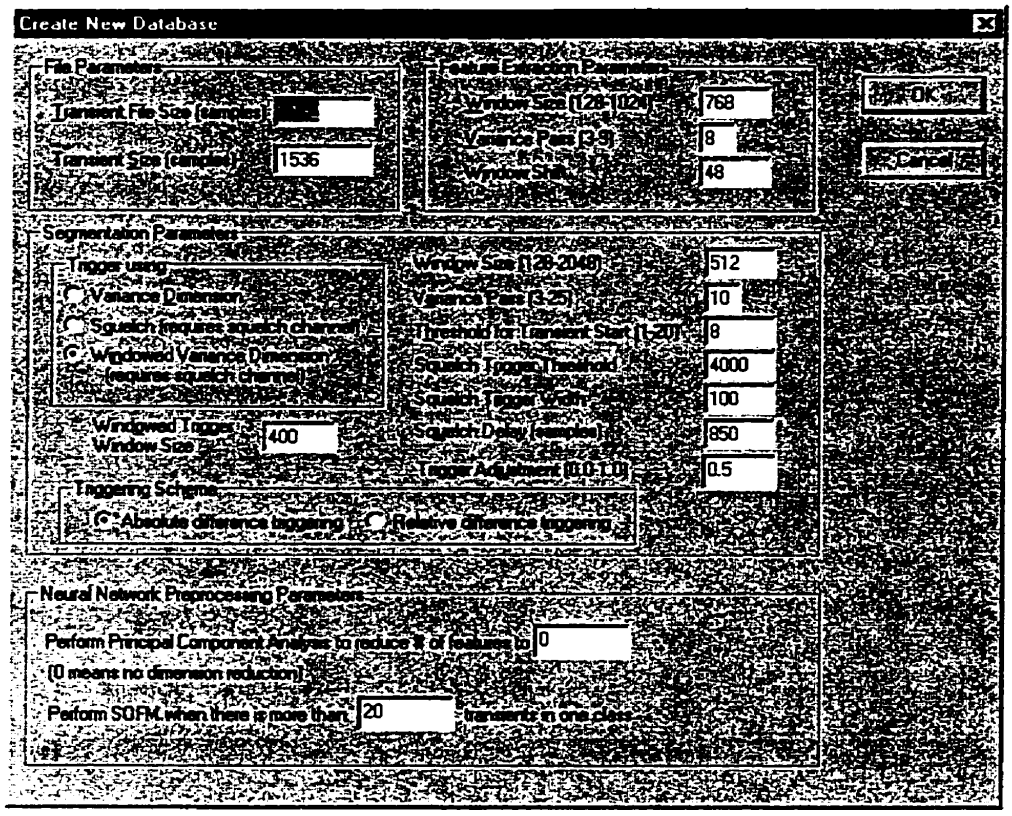


Fig. 3. The File: New dialog box.

A.4 The Edit Menu

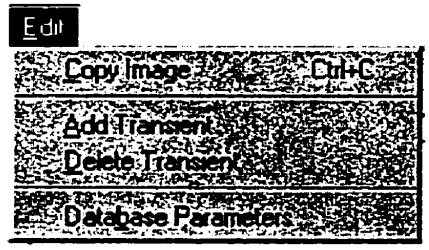


Fig. 4. The Edit pull-down menu.

The **Copy Image** function is used to save the image in the upper viewing window to the Windows clipboard as a device independent bitmap (DIB) [Shaw97]. The user can add a transient to the training database by selecting **Add Transient**. After the system loads the transient file suc-

cessfully, the noise segmentation and feature extraction are automatically performed using the parameters specified in the **File: New** dialog. The user can remove a transient using the **Delete Transient** function. The **Database Parameters** function is used to change the parameter settings for noise segmentation, feature extraction, and PNN preprocessing. It brings up the dialog box shown in Fig. 5.

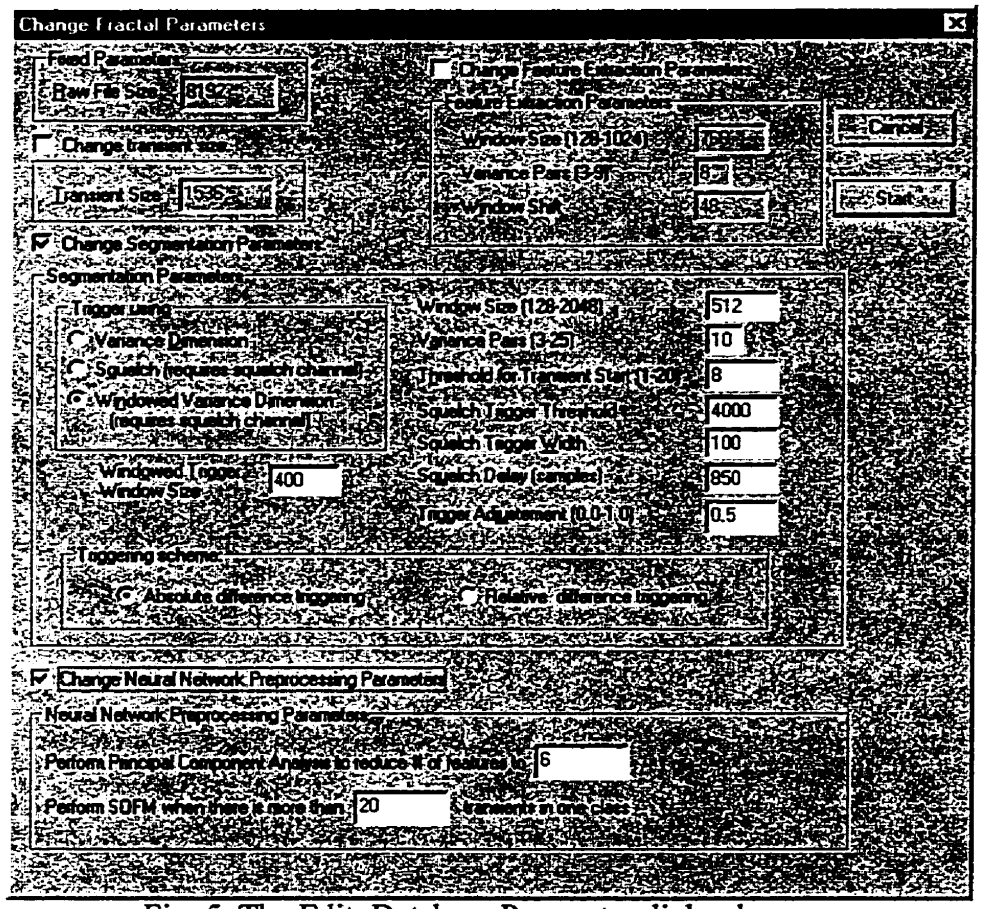


Fig. 5. The Edit: Database Parameter dialog box.

A.5 The View Menu

The first group of commands (**Toolbar, Status Bar**, and **Split**) are common Window commands to control the appearance of the program window. The second group is the display options for the upper viewing window and the next group contains the display options for the lower view-

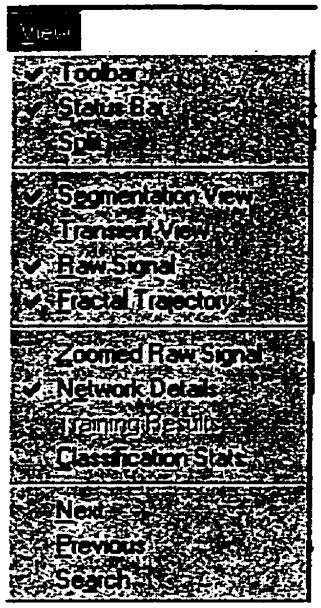


Fig. 6. The View pull-down menu.

ing window. They are self-explanatory. The last group serves to locate a transient stored in the database.

A.6 The Neural Net Menu

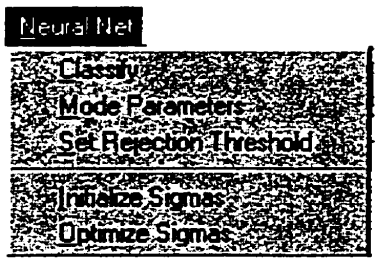


Fig. 7. The Neural Net pull-down menu.

The **Classify** command brings up a dialog box for the user to select files for classification, as shown in Fig. 8. The **Mode Parameters** function is described in [Shaw97] as another approach for consistent segmentation. The **Set Rejection Threshold** function allows the user to set a lower

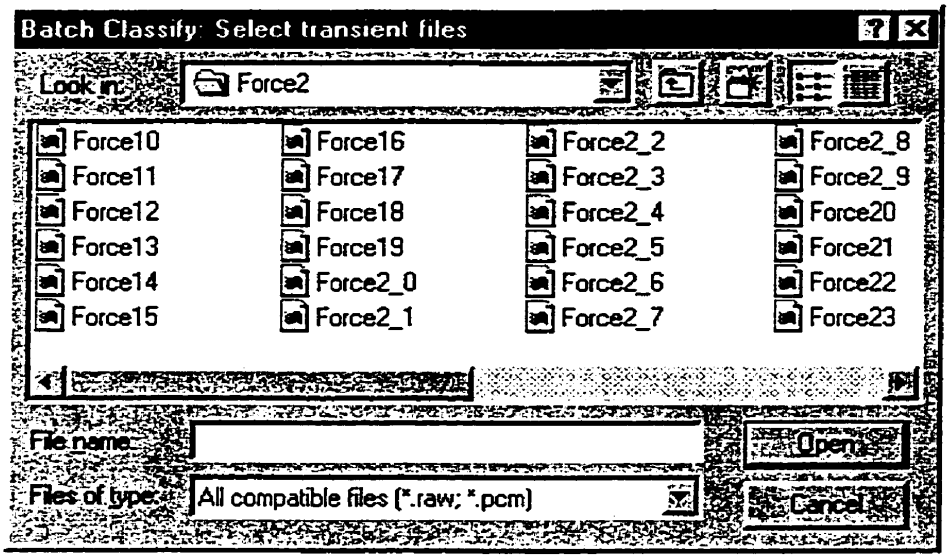


Fig. 8. The Neural Net: Classify dialog box.

limit for the summation neurons in the PNN to reject transients that are complete new. This function is very important in practice.

A.7 The Capture Menu

Fig. 9. The Capture menu.

This menu deals with the transient capture and recording module, which is integrated into the software by CRC. The Start function brings up another window shown in Fig. 10.

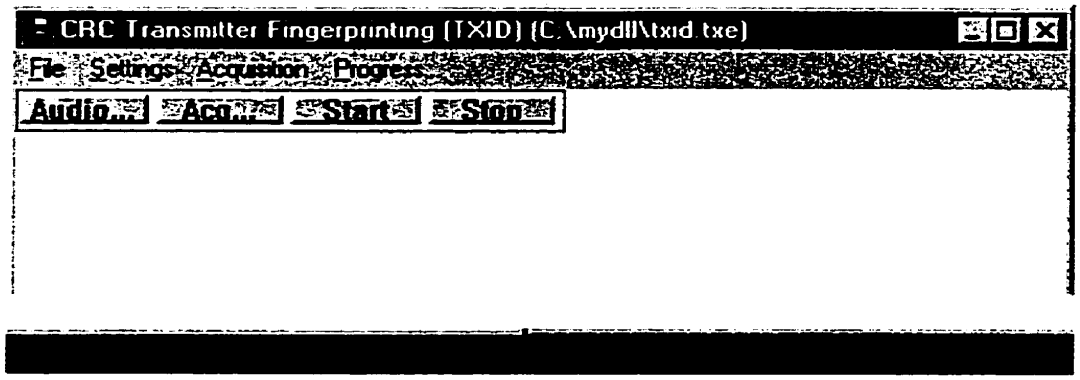


Fig. 10. The transient capture interface.

A.8 The Help Menu

Fig. 11. The Help menu.

Currently there is no other functions implemented here except the **About** function.

APPENDIX B

NETWORK PARAMETER SETTINGS

B.1 PNN with Absolute Triggering

Network Details

Network Name: tt
# of Classes: 20
# of Transients: 400

Transient File Parameters

Transient File Size: 8192
Transient Size: 1536

Segmentation Parameters

Trigger Type: Windowed Variance
Variance Window Size: 512
Variance Pairs: 10
Variance Threshold: 8.00
Squelch Threshold: 4000
Squelch Hysteresis Window: 100
Squelch Delay: 850
Trigger Window Size: 400
Trigger Scheme:Absolute Difference Triggering

Feature Extraction Parameters

Window Size: 768
Variance Pairs: 8
Window Shift: 48

Neural Network Preprocessing Parameters

Number of Principal Components: 0
SOFM Threshold: 20

Neural Network Sigma Optimized: No
Neural Network Rejection Threshold: 1e-20

B.2 PNN with Relative Triggering

Network Details

Network Name: tt_rel
# of Classes: 20
# of Transients: 400

Transient File Parameters

Transient File Size: 8192
Transient Size: 1536

Segmentation Parameters

Trigger Type: Windowed Variance
Variance Window Size: 512
Variance Pairs: 10
Variance Threshold: 8.00
Squelch Threshold: 4000
Squelch Hysteresis Window: 100
Squelch Delay: 850
Trigger Window Size: 400
Trigger Scheme:Relative Difference Triggering
Adjustment: 0.50

Feature Extraction Parameters

Window Size: 768
Variance Pairs: 8
Window Shift: 48

Neural Network Preprocessing Parameters

Number of Principal Components: 0
SOFM Threshold: 20

Neural Network Sigma Optimized: No
Neural Network Rejection Threshold: 1e-20

APPENDIX C

TAC-MM SOURCE CODE

Note: Unless stated as "implemented or modified by the author", the source files are written by D. Shaw [Shaw97] and CRC.

AddUnknownDialog.h

```
/*CAddUnknownDialog: defines and controls the dialog box displayed when the user chooses to add a recently classified transient to the database*/
// AddUnknownDialog.h : header file

// AddUnknownDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////
// CAddUnknownDialog dialog
#include "TAC_MMDoc.h"

class CAddUnknownDialog : public CDialog
{
// Construction
public:
	CAddUnknownDialog(CWnd* pParent = NULL);  // standard constructor
	CTAC_MMDoc* m_pDoc;

	// Dialog Data
	//{{AFX_DATA(CAddUnknownDialog)
	enum { IDD = IDD_ADD_UNKNOWN_DIALOG };
	CStringm_Date;
	int		m_Position;
	CStringm_Time;
	CStringm_TransmitterMake;
	CStringm_TransmitterModel;
	CStringm_TransmitterSerial;
	CStringm_strClassName;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAddUnknownDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CAddUnknownDialog)
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()

private:
	void AddClassNames();
};
```

AddUnknownDialog.cpp

```
// AddUnknownDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "AddUnknownDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAddUnknownDialog dialog


CAddUnknownDialog::CAddUnknownDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CAddUnknownDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CAddUnknownDialog)
	m_Date = _T("");
	m_Position = 0;
	m_Time = _T("");
	m_TransmitterMake = _T("");
	m_TransmitterModel = _T("");
	m_TransmitterSerial = _T("");
	m_strClassName = _T("");
	//}}AFX_DATA_INIT
}


void CAddUnknownDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAddUnknownDialog)
	DDX_Text(pDX, IDC_DATE, m_Date);
	DDX_Text(pDX, IDC_POSITION, m_Position);
	DDV_MinMaxInt(pDX, m_Position, 0, 10000000);
	DDX_Text(pDX, IDC_TIME, m_Time);
	DDX_Text(pDX, IDC_TXMAKE, m_TransmitterMake);
	DDX_Text(pDX, IDC_TXMODEL, m_TransmitterModel);
	DDX_Text(pDX, IDC_TXSERIAL, m_TransmitterSerial);
	DDX_Text(pDX, IDC_CLASS, m_strClassName);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CAddUnknownDialog, CDialog)
	//{{AFX_MSG_MAP(CAddUnknownDialog)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAddUnknownDialog message handlers

void CAddUnknownDialog::AddClassNames()
{
	CComboBox* pCB = (CComboBox*) GetDlgItem(IDC_CLASS); // Get a pointer to the combo box.

	for (int i = 0; i < (m_pDoc -> GetArray() -> FindNumClasses()); i++)
```

```
    {
        pCB -> AddString(m_pDoc -> GetArray() -> GetClassName(i)); // Add the class names to the combo box.
    }

    return;

}

BOOL CAddUnknownDialog::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    // Fill in the drop down combo box.
    AddClassNames();

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

BatchProgressDialog.h

```
/*CBatchProgressDialog: defines and controls the dialog box displayed to update the user on the progress of a batch classification*/
// BatchProgressDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CBatchProgressDialog dialog

class CBatchProgressDialog : public CDialog
{
// Construction
public:
	CBatchProgressDialog(CWnd* pParent = NULL);  // standard constructor
	int m_ProgressBarLimit;
	BOOL Create();

	void OnUpdateProgress(int CurrentProgress) ;

	// Dialog Data
	//{{AFX_DATA(CBatchProgressDialog)
	enum { IDD = IDD_BATCHPROGRESS_DIALOG };
	CProgressCtrlm_UpdateProgress;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CBatchProgressDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CBatchProgressDialog)
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

BatchProgressDialog.cpp

```
// BatchProgressDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "BatchProgressDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CBatchProgressDialog dialog


CBatchProgressDialog::CBatchProgressDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CBatchProgressDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CBatchProgressDialog)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}


void CBatchProgressDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CBatchProgressDialog)
	DDX_Control(pDX, IDC_UPDATEPROGRESS, m_UpdateProgress);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CBatchProgressDialog, CDialog)
	//{{AFX_MSG_MAP(CBatchProgressDialog)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBatchProgressDialog message handlers

BOOL CBatchProgressDialog::Create()
{
	return CDialog::Create(CBatchProgressDialog::IDD);
}

BOOL CBatchProgressDialog::OnInitDialog()
{
	CDialog::OnInitDialog();

	CProgressCtrl* pProg=(CProgressCtrl*)GetDlgItem(IDC_UPDATEPROGRESS);
	pProg->SetRange(0,m_ProgressBarLimit);
	pProg->SetPos(0);
	CString strText;
	strText.Format ("%d",m_ProgressBarLimit);
	SetDlgItemText(IDC_BARLIMIT, strText);
	strText.Format("0");
	SetDlgItemText(IDC_CURRENT, strText);
```

```
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CBatchProgressDialog::OnUpdateProgress(int CurrentProgress)
{
    CString strText;
    strText.Format ("%d",CurrentProgress);
    SetDlgItemText(IDC_CURRENT, strText);

    MSG message;
    if (::PeekMessage(&message, NULL, 0, 0, PM_REMOVE)) {
        ::TranslateMessage(&message);
        ::DispatchMessage(&message);
    }
}
```

ClassifyDialog.h

```
//CClassifyDialog: defines and controls the dialog box displayed to show the results of a classification
// ClassifyDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CClassifyDialog dialog

class CClassifyDialog : public CDialog
{
// Construction
public:
	CClassifyDialog(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	//{{AFX_DATA(CClassifyDialog)
	enum { IDD = IDD_CLASSIFY_DIALOG };
	doublem_Confidence;
	doublem_Activation;
	CStringm_strPredictedClass;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CClassifyDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CClassifyDialog)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

ClassifyDialog.cpp

```
// ClassifyDialog.h : implementation file

//
#include "stdafx.h"
#include "TAC_MM.h"
#include "ClassifyDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CClassifyDialog dialog


CClassifyDialog::CClassifyDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CClassifyDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CClassifyDialog)
	m_Confidence = 0.0;
	m_Activation = 0.0;
	m_strPredictedClass = _T("");
	//}}AFX_DATA_INIT
}


void CClassifyDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CClassifyDialog)
	DDX_Text(pDX, IDC_CONFIDENCE, m_Confidence);
	DDV_MinMaxDouble(pDX, m_Confidence, 0., 100.);
	DDX_Text(pDX, IDC_ACTIVATION, m_Activation);
	DDV_MinMaxInt(pDX, m_Activation, 0.0, 1.0);
	DDX_Text(pDX, IDC_PREDICTEDCLASS, m_strPredictedClass);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CClassifyDialog, CDialog)
	//{{AFX_MSG_MAP(CClassifyDialog)
		// NOTE: the ClassWizard will add message map macros here
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CClassifyDialog message handlers
```

EditFPProgressDialog.h

```
//CEditFPProgressDialog, which defines and controls the dialog box displayed to update the user on the progress of a change in fractal parameters
// EditFPProgressDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CEditFPProgressDialog dialog

class CEditFPProgressDialog : public CDialog
{
// Construction
public:
	CString m_strOperation;
	CEditFPProgressDialog(CWnd* pParent = NULL);  // standard constructor
	int m_ProgressBarLimit;
	BOOL Create();

	void OnUpdateProgress(int CurrentProgress) ;

// Dialog Data
	//{{AFX_DATA(CEditFPProgressDialog)
	enum { IDD = IDD_EDITFPPROGRESS_DIALOG };
	CProgressCtrlm_UpdateProgress;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CEditFPProgressDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CEditFPProgressDialog)
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

EditFPProgressDialog.cpp

```
// EditFPProgressDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "EditFPProgressDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CEditFPProgressDialog dialog


CEditFPProgressDialog::CEditFPProgressDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CEditFPProgressDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CEditFPProgressDialog)
	//}}AFX_DATA_INIT
}


void CEditFPProgressDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CEditFPProgressDialog)
	DDX_Control(pDX, IDC_UPDATEPROGRESS, m_UpdateProgress);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CEditFPProgressDialog, CDialog)
	//{{AFX_MSG_MAP(CEditFPProgressDialog)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CEditFPProgressDialog message handlers

BOOL CEditFPProgressDialog::Create()
{
	return CDialog::Create(CEditFPProgressDialog::IDD);
}

BOOL CEditFPProgressDialog::OnInitDialog()
{
	CDialog::OnInitDialog();

	CProgressCtrl* pProg=(CProgressCtrl*)GetDlgItem(IDC_UPDATEPROGRESS);
	pProg->SetRange(0,m_ProgressBarLimit);
	pProg->SetPos(0);
	CString strText;
	strText.Format ("%d",m_ProgressBarLimit);
	SetDlgItemText(IDC_BARLIMIT, strText);
	strText.Format("0");
	SetDlgItemText(IDC_CURRENT, strText);
	SetDlgItemText(IDC_OPERATION, m_strOperation);
```

```
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CEditFPProgressDialog::OnUpdateProgress(int CurrentProgress)
{
    CString strText;
    strText.Format ("%d",CurrentProgress);
    SetDlgItemText(IDC_CURRENT, strText);
}
```

EnterClassDialog.h

```
//CEnterClassDialog: defines and controls the dialog box displayed to enter the transmitter class integer of a transient being added to the database
// EnterClassDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CEnterClassDialog dialog


#include "TAC_MMDoc.h"

class CEnterClassDialog : public CDialog
{
// Construction
public:
	CTAC_MMDoc* m_pDoc;
	void AddClassNames();
	int m_Num_Files_Selected;
	CEnterClassDialog(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	//{{AFX_DATA(CEnterClassDialog)
	enum { IDD = IDD_ENTER_CLASS_DIALOG };
	CStringm_strClassName;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CEnterClassDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CEnterClassDialog)
	afx_msg void OnDelete();
	afx_msg void OnSkip();
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

EnterClassDialog.cpp

```
// EnterClassDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "EnterClassDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CEnterClassDialog dialog


CEnterClassDialog::CEnterClassDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CEnterClassDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CEnterClassDialog)
	m_strClassName = _T("");
	//}}AFX_DATA_INIT
}


void CEnterClassDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CEnterClassDialog)
	DDX_CBString(pDX, IDC_CLASS_NAME, m_strClassName);
	//}}AFX_DATA_MAP

}


BEGIN_MESSAGE_MAP(CEnterClassDialog, CDialog)
	//{{AFX_MSG_MAP(CEnterClassDialog)
	ON_BN_CLICKED(IDC_DELETE, OnDelete)
	ON_BN_CLICKED(IDC_SKIP, OnSkip)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CEnterClassDialog message handlers

void CEnterClassDialog::OnDelete()
{
	// TODO: Add your control notification handler code here
	EndDialog(IDC_DELETE);
}

void CEnterClassDialog::OnSkip()
{
	// TODO: Add your control notification handler code here
	EndDialog(IDC_SKIP);
}

BOOL CEnterClassDialog::OnInitDialog()
{
	CDialog::OnInitDialog();
```

```
	// TODO: Add extra initialization here

	// Display # of files selected and current file.
	CString strText;
	strText.Format("File %d of %d", m_Num_Files_Selected, m_Num_Files_Selected);
	SetDlgItemText(IDC_FILE_NUMBER, strText);

	// Fill in the drop down combo box.
	AddClassNames();

	return TRUE;  // return TRUE unless you set the focus to a control
		// EXCEPTION: OCX Property Pages should return FALSE
}

void CEnterClassDialog::AddClassNames()
{
	CComboBox* pCB = (CComboBox*) GetDlgItem(IDC_CLASS_NAME); // Get a pointer to the combo box.

	for (int i = 0; i < (m_pDoc -> GetArray() -> FindNumClasses()); i++)
	{
		pCB -> AddString(m_pDoc -> GetArray() -> GetClassName(i));
	}

	return;
}
```

EnterClassMultiDialog.h

```
#if !defined(AFX_ENTERCLASSMULTIDIALOG_H__E81FDD03_F92A_11D1_8BA7_00A024423FB9__INCLUDED_)
#define AFX_ENTERCLASSMULTIDIALOG_H__E81FDD03_F92A_11D1_8BA7_00A024423FB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// EnterClassMultiDialog.h : header file
//


#include "TAC_MMDoc.h"


/////////////////////////////////////////////////////////////////////////////
// CEnterClassMultiDialog dialog

class CEnterClassMultiDialog : public CDialog
{
// Construction
public:
	CTAC_MMDoc* m_pDoc;
	void AddClassNames();
	int m_Num_Files_Selected; // The number of selected.
	int m_Transient_File_Number; // Current file in selected file to accept.

	CEnterClassMultiDialog(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	//{{AFX_DATA(CEnterClassMultiDialog)
	enum { IDD = IDD_ENTER_CLASS_MULTI_DIALOG };
	CStringm_strClassName;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CEnterClassMultiDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CEnterClassMultiDialog)
	afx_msg void OnDelete();
	afx_msg void OnSkip();
	afx_msg void OnAceptAll();
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_ENTERCLASSMULTIDIALOG_H__E81FDD03_F92A_11D1_8BA7_00A024423FB9__INCLUDED_)
```

EnterClassMultiDialog.cpp

```
// EnterClassMultiDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "EnterClassMultiDialog.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CEnterClassMultiDialog dialog


CEnterClassMultiDialog::CEnterClassMultiDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CEnterClassMultiDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CEnterClassMultiDialog)
	m_strClassName = _T("");
	//}}AFX_DATA_INIT
}


void CEnterClassMultiDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CEnterClassMultiDialog)
	DDX_CBString(pDX, IDC_CLASS_NAME, m_strClassName);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CEnterClassMultiDialog, CDialog)
	//{{AFX_MSG_MAP(CEnterClassMultiDialog)
	ON_BN_CLICKED(IDC_DELETE, OnDelete)
	ON_BN_CLICKED(IDC_SKIP, OnSkip)
	ON_BN_CLICKED(IDC_ACCEPT_ALL, OnAceptAll)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CEnterClassMultiDialog message handlers

void CEnterClassMultiDialog::OnDelete()
{
	// TODO: Add your control notification handler code here
	EndDialog(IDC_DELETE);
}

void CEnterClassMultiDialog::OnSkip()
{
	// TODO: Add your control notification handler code here
	EndDialog(IDC_SKIP);
}

void CEnterClassMultiDialog::OnAceptAll()
{
```

```
    // TODO: Add your control notification handler code here
    UpdateData(TRUE); // Retrieve info from dialog box.
    EndDialog(IDC_ACCEPT_ALL);
}

BOOL CEnterClassMultiDialog::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    // Display # of files selected and current file.
    CString strText;
    strText.Format("File %d of %d", m_Transient_File_Number, m_Num_Files_Selected);
    SetDlgItemText(IDC_FILE_NUMBER, strText);

    // Fill in the drop down combo box.
    AddClassNames();

    return TRUE;  // return TRUE unless you set the focus to a control
            // EXCEPTION: OCX Property Pages should return FALSE
}

void CEnterClassMultiDialog::AddClassNames()
{
    CComboBox* pCB = (CComboBox*) GetDlgItem(IDC_CLASS_NAME); // Get a pointer to the combo box.

    for (int i = 0; i < (m_pDoc -> GetArray() -> FindNumClasses()); i++)
    {
        pCB -> AddString(m_pDoc -> GetArray() -> GetClassName(i));
    }

    return;
}
```

EnterSigmaInitParamsDialog.h

```
//CEnterSigmaInitParams-Dialog: defines and controls the dialog box displayed to enter the parameters required for the sigma initialization
// EnterSigmaInitParamsDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CEnterSigmaInitParamsDialog dialog

class CEnterSigmaInitParamsDialog : public CDialog
{
// Construction
public:
	CEnterSigmaInitParamsDialog(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	//{{AFX_DATA(CEnterSigmaInitParamsDialog)
	enum { IDD = IDD_ENTER_SIGMAINIT_PARAMS };
	double	m_LowerSearchLimit;
	int		m_NumSearchPoints;
	double	m_UpperSearchLimit;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CEnterSigmaInitParamsDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CEnterSigmaInitParamsDialog)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

EnterSigmaInitParamsDialog.cpp

```
// EnterSigmaInitParamsDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "EnterSigmaInitParamsDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CEnterSigmaInitParamsDialog dialog


CEnterSigmaInitParamsDialog::CEnterSigmaInitParamsDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CEnterSigmaInitParamsDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CEnterSigmaInitParamsDialog)
	m_LowerSearchLimit = 0.0;
	m_NumSearchPoints = 0;
	m_UpperSearchLimit = 0.0;
	//}}AFX_DATA_INIT
}


void CEnterSigmaInitParamsDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CEnterSigmaInitParamsDialog)
	DDX_Text(pDX, IDC_LOWERLIM, m_LowerSearchLimit);
	DDV_MinMaxDouble(pDX, m_LowerSearchLimit, 1.e-004, 1000.);
	DDX_Text(pDX, IDC_NUMPOINTS, m_NumSearchPoints);
	DDV_MinMaxInt(pDX, m_NumSearchPoints, 3, 10000);
	DDX_Text(pDX, IDC_UPPERLIM, m_UpperSearchLimit);
	DDV_MinMaxDouble(pDX, m_UpperSearchLimit, 1.e-003, 1000.);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CEnterSigmaInitParamsDialog, CDialog)
	//{{AFX_MSG_MAP(CEnterSigmaInitParamsDialog)
		// NOTE: the ClassWizard will add message map macros here
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CEnterSigmaInitParamsDialog message handlers
```

Extract.h

```
//This file is modified by the author to implement the relative triggering scheme and second-run variance fractal trajectory
//CExtract: contains the segmentation and feature extraction routines
#include "GlobalStuff.h"

class CExtract:public CObject
{
    DECLARE_SERIAL(CExtract)
protected:
    int m_nRawFileSize;//size of file with raw data.
    int m_nTransientSize;//size of transient.
    int m_nSegmentationWindowSize;
    int m_nSegmentationVariancePairs;
    double m_Threshold;//abs diff from avg channel noise to trigger
                                        //the start of transient
    int m_nModelWindowSize;
    int m_nModelVariancePairs;
    int m_nWindowShift;//used for display purposes and data reduction
    int m_nTransientModelSize;

    int m_nVariancePairs;//number of pairs to send to lsr routine.
    int m_nDyadic;      //set to 0 if unit intervals used, else dyadic.
    int m_nWindowSize;//size of window for variance dimension calc.
    double *m_dSum1;
    double *m_dSum2;

    double *VarDimTrajSec;//the variance dimension of the original dimension trajctory

    int m_nTrigScheme;//0 is absolute triggering, 1 is relative triggering
    int m_nSquelchTrig; // Value of squelch channel which indicates a transient.
    int m_nSquelchDelay; // # of samples between the start of a transient and a squelch trigger.
    int m_nSquelchTrigWidth; // # of samples for which the squelch channel must stay above m_nSquelchTrig.
    int m_nTrigWinSize; // Window set by squelch trigger which the variance trigger must occur in.

    TriggerType m_TrigType;  // Indicates which trigger meathod to use.(see GlobalStuff.h)
    float  m_nAdjust;
public:
    void SetTransientModelSize(int TransientModelSize) {m_nTransientModelSize=
        TransientModelSize; }
    void SetTransientSize(int TransientSize) {m_nTransientSize = TransientSize;};
    TriggerType GetTriggerType();
    void SetTriggerType(TriggerType triggerType) {m_TrigType = triggerType;};
    int GetTrigWinSize();
    void SetTrigWinSize(int TrigWinSize) {m_nTrigWinSize = TrigWinSize;};
    int GetSquelchDelay();
    void SetSquelchDelay(int SquelchDelay) {m_nSquelchDelay = SquelchDelay;};
    int GetSquelchWidth();
    void SetSquelchWidth(int SquelchWidth) {m_nSquelchTrigWidth = SquelchWidth;};
    int GetSquelchTrigger();
    void SetSquelchTrigger(int SquelchTrigger) {m_nSquelchTrig = SquelchTrigger;};
    int GetTrigScheme();
    void SetTrigScheme(int TrigScheme) { m_nTrigScheme = TrigScheme;};
    float GetAdjust();
    void SetAdjust(float Adjust) {m_nAdjust = Adjust;};

    CExtract() {}//constructor can only be called once
    void Serialize(CArchive& ar);
    void SetParams(int RawFileSize, int TransientSize,
        int SegmentationWindowSize,int SegmentationVariancePairs,
        double Threshold, int ModelWindowSize,int ModelVariancePairs,
        int WindowShift, int SquelchTrig, int SquelchDelay,
        int SquelchTrigWidth, int TrigWinSize, TriggerType TrigType,
```

```
        int TrigScheme, float Adjust)
    {
        m_nRawFileSize=RawFileSize;
        m_nTransientSize=TransientSize;
        m_nSegmentationWindowSize=SegmentationWindowSize;
        m_nSegmentationVariancePairs=SegmentationVariancePairs;
        m_Threshold=Threshold;
        m_nModelWindowSize=ModelWindowSize;
        m_nModelVariancePairs=ModelVariancePairs;
        m_nWindowShift=WindowShift;
        m_nSquelchTrig = SquelchTrig;
        m_nSquelchDelay = SquelchDelay;
        m_nSquelchTrigWidth = SquelchTrigWidth;
        m_nTrigWinSize = TrigWinSize;
        m_TrigType = TrigType;
        m_nTrigScheme = TrigScheme;
        m_nAdjust = Adjust;
        m_nTransientModelSize = m_nTransientSize/m_nWindowShift;
    }

    void SetWindowSize(int WindowSize) { m_nSegmentationWindowSize=WindowSize; }
    void SetVariancePairs(int VarPairs) { m_nSegmentationVariancePairs=VarPairs; }
    void SetThreshold(double Threshold) { m_Threshold=Threshold; }
    void SetModelWindowSize(int WindowSize) { m_nModelWindowSize=WindowSize; }
    void SetModelWindowShift(int WindowShift) { m_nWindowShift=WindowShift; }
    void SetModelVariancePairs(int VarPairs) { m_nModelVariancePairs=VarPairs; }

    int GetRawFileSize() { return m_nRawFileSize; }
    int GetTransientSize() { return m_nTransientSize; }
    int GetVariancePairs() { return m_nSegmentationVariancePairs; }
    int GetWindowSize() { return m_nSegmentationWindowSize; }
    int GetWindowShift() { return m_nWindowShift; }
    int GetModelWindowSize() { return m_nModelWindowSize; }
    int GetModelVariancePairs() { return m_nModelVariancePairs; }
    int GetModelWindowShift() { return m_nWindowShift; }
    double GetThreshold() { return m_Threshold; }
    int GetTransientModelSize() { return m_nTransientModelSize; }

    int FindTransient(int *RawSignal, int *SquelchChannel, FileType filetype,
        double *VarDimTraj,double *TransientModel);
    void FillTransientModel(int *RawSignal, int TransientBeginsHere,
        double *TransientModel);
    //added function
    void FillTransientModel(double *RawSignal, int TransientBeginsHere,
        double *TransientModel);

    void FillVarDimArray(int *RawSignal,double *VarDimTraj,bool ViewSegmentation,
        int TransientBeginsHere, double *TransientModel);
    double CalcVarDim(int *RawSignal, int StartLoc);

    //added function
    double CalcVarDim(double *RawSignal, int StartLoc);
    void Interpolate(double *VarDimTraj, int StartPoint);
    void CExtract::FindMultiTransients(int *RawSignal, double *VarDimTraj,
                                       double **TransientModel, int NumModes,
                                       double *Threshholds, int MinSeperation,
                                       int *TransientBeginsHere);

#ifdef _DEBUG
    void Dump(CDumpContext& dc) const;
#endif
private:
    int FindVarianceTrigger(int *RawSignal, double *VarDimTraj);
    int FindSquelchTrigger(int* SquelchChannel);
};
```

Extract.cpp

```
//This file is modified by the author to implement the relative triggering scheme and second-run variance fractal trajectory
#include "StdAfx.h"
#include "Extract.h"
#include <math.h>


IMPLEMENT_SERIAL(CExtract, CObject,0)

#ifdef _DEBUG
void CExtract::Dump(CDumpContext& dc) const
{
	CObject::Dump(dc);
	dc << "\nm_nRawFileSize = " << m_nRawFileSize;
}
#endif

void CExtract::Serialize(CArchive& ar)
{
	int trigType;

	if(ar.IsStoring())
	{
		trigType = (int) m_TrigType; // Need to do this since the
					// << operators can't handle enumerated
					// types.

		ar<< (LONG)m_nRawFileSize << (LONG)m_nTransientSize
			<< (LONG)m_nSegmentationWindowSize
			<< (LONG)m_nSegmentationVariancePairs
			<< m_Threshold << (LONG)m_nModelWindowSize
			<< (LONG) m_nModelVariancePairs << (LONG)m_nWindowShift
			<< m_nSquelchTrig << m_nSquelchTrigWidth << m_nSquelchDelay
			<< m_nTrigWinSize << trigType << m_nTrigScheme << m_nAdjust
			<< m_nTransientModelSize;
	}
	else
	{
		ar>> (LONG&)m_nRawFileSize >> (LONG&)m_nTransientSize
			>> (LONG&)m_nSegmentationWindowSize
			>> (LONG&)m_nSegmentationVariancePairs
			>> m_Threshold >> (LONG&)m_nModelWindowSize
			>> (LONG&) m_nModelVariancePairs >> (LONG&)m_nWindowShift
			>> m_nSquelchTrig >> m_nSquelchTrigWidth >> m_nSquelchDelay
			>> m_nTrigWinSize >> trigType >> m_nTrigScheme >> m_nAdjust
			>> m_nTransientModelSize;

		m_TrigType = (TriggerType) trigType; // Need to do this since the
					// << operators can't handle enumerated
					// types.
	}

}

int CExtract::FindTransient(int *pRawSignal, int *pSquelchChannel, FileType filetype,
				double *pVarDimTraj,double *pTransientModel)
{
	int TransientBeginsHere;

	// Perform the correct trigger option.
	switch (m_TrigType)
	{
```

```
case SQUELCH: // Use the squelch channel to find the start of the transient.
        if (filetype != PCM)
        {
                AfxMessageBox("Must be a PCM file to use the squelch trigger option."
                                ,MB_OK);
                return -1;
        }
        TransientBeginsHere = FindSquelchTrigger(pSquelchChannel);
        break;

case VARIANCE: // Calculate variance dimension to find start of transient
        TransientBeginsHere = FindVarianceTrigger(pRawSignal,
                pVarDimTraj);
        break;

case WINDOWEDVARIANCE: // Calculates the variance dimensiona and makes sure it
                                // is within a window set by the squelch trigger.
        if (filetype != PCM)
        {
                AfxMessageBox("Must be a PCM file to use the windowed"
                        "variance trigger option.", MB_OK);
                return -1;
        }

        int SquelchTrigLocation;
        int VarianceTrigLocation;

        // Find the location of the squelch and variance triggers.
        SquelchTrigLocation = FindSquelchTrigger(pSquelchChannel);
        VarianceTrigLocation = FindVarianceTrigger(pRawSignal,
                pVarDimTraj);

        /*TRACE("Chris, The squelch window is from %d to %d. \n"
                "The variance trigger occurred at %d\n", (SquelchTrigLocation - m_nTrigWinSize/2),
                (SquelchTrigLocation + m_nTrigWinSize/2), VarianceTrigLocation);*/


        // If the variance trigger is outside the window set by
        // the squelch trigger, set the transient to begin at the
        // extrema of that window. If it is inside, set the transient
        // to begin at the location of the variance trigger.
        if (VarianceTrigLocation < SquelchTrigLocation - m_nTrigWinSize/2)
        {
                TransientBeginsHere = SquelchTrigLocation - m_nTrigWinSize/2;
        }
        else if (VarianceTrigLocation > SquelchTrigLocation + m_nTrigWinSize/2)
        {
                TransientBeginsHere = SquelchTrigLocation + m_nTrigWinSize/2;
        }
        else
        {
                TransientBeginsHere = VarianceTrigLocation;
        }

}

//TRACE("Transientbeginshere = %d\n", TransientBeginsHere);

// If a trigger was found, extract the features.
if (TransientBeginsHere!=-1)
{

        FillTransientModel(pRawSignal, TransientBeginsHere,
                pTransientModel); //feature extraction based on variance dimension traj
}
```

```
    return TransientBeginsHere;

}

//transient feature extraction procedure
void CExtract::FillTransientModel(int *RawSignal, int TransientBeginsHere, double *TransientModel)
{

    if(m_nModelVariancePairs>0)//transient model is fractal
    {
        m_nDyadic=1;
        m_nWindowSize=m_nModelWindowSize;
        m_nVariancePairs=m_nModelVariancePairs;
        int ctr3=-1;
        TRACE("Transient Model Vector:\n");
        TRACE("\n");
        for (int ctr=TransientBeginsHere;
            ctr<TransientBeginsHere+m_nTransientSize; ctr+=m_nWindowShift)
        {
            ctr3++;
            TransientModel[ctr3]=CalcVarDim(RawSignal, ctr);
            //TRACE("%d %f\n", ctr3, TransientModel[ctr3]);
            TRACE("%f ",TransientModel[ctr3]);
        }
            //TRACE("end of feature extraction\n");
    }

    else                                //transient model is simple moving average
    {
        int ctr2, ctr3=-1;
        double Average;
        for (int ctr=TransientBeginsHere;
                ctr<TransientBeginsHere+m_nTransientSize; ctr+=m_nWindowShift)
        {
            Average=0.0;
            for (ctr2=0; ctr2<m_nWindowShift; ctr2++)
            {
                Average+=RawSignal[ctr+ctr2];
            }
            ctr3++;
            TransientModel[ctr3]=Average/m_nWindowShift/65536.0 + 1.0;
        }
    }

}

void CExtract::FillTransientModel(double *RawSignal, int TransientBeginsHere,
                                                    double *TransientModel)
{

    m_nDyadic=1;
    m_nWindowSize=m_nModelWindowSize;
    m_nVariancePairs=m_nModelVariancePairs;
    int ctr3=-1;
    for (int ctr=TransientBeginsHere;
        ctr<TransientBeginsHere+m_nTransientSize; ctr+=m_nWindowShift)
    {
        ctr3++;
        TransientModel[ctr3]=CalcVarDim(RawSignal, ctr);
        //TRACE("TransientModel[%d]=%f\n", ctr3, TransientModel[ctr3]);
    }

}
```

```
void CExtract::FillVarDimArray(int *RawSignal,double *VarDimTraj,
                               bool ViewSegmentation, int TransientBeginsHere,
                               double *TransientModel)
{

    if(ViewSegmentation)
    {
        m_nDyadic=0;
        m_nWindowSize=m_nSegmentationWindowSize;
        m_nVariancePairs=m_nSegmentationVariancePairs;
        int GoodFileSize=m_nRawFileSize-m_nWindowSize;//don't perform calcs
                                                      //past the end of file
        /*int Traj2WindowSize = 512;
        int Traj3WindowSize = 512;
        int GoodFileSize2=GoodFileSize-Traj2WindowSize;
        int GoodFileSize3=GoodFileSize2-Traj2WindowSize;
        double *VarDimTrajSec, *VarDimTrajTrd;
        VarDimTrajSec = new double[GoodFileSize2];
        VarDimTrajTrd = new double[GoodFileSize3];*/

        m_dSum1=new double[m_nSegmentationVariancePairs+1];
        m_dSum2=new double[m_nSegmentationVariancePairs+1];
        for (int ctr=0; ctr<GoodFileSize; ctr+=1)
        {
            VarDimTraj[ctr]=CalcVarDim(RawSignal, ctr);
        }

        /*m_nWindowSize = Traj2WindowSize;//second order trajectory
        for (ctr=0; ctr<GoodFileSize2; ctr+=1)
        {
            VarDimTrajSec[ctr]=CalcVarDim(VarDimTraj,ctr);
        }

        m_nWindowSize = Traj3WindowSize;//third order trajectory
        for (ctr=0; ctr<GoodFileSize3; ctr+=1)
        {
            VarDimTrajTrd[ctr]=CalcVarDim(VarDimTrajSec,ctr);
        }

        for(ctr=0; ctr<GoodFileSize3; ctr+=1)
        {
            VarDimTraj[ctr] = VarDimTrajTrd[ctr];
        }


        delete []VarDimTrajSec;
        delete []VarDimTrajTrd*/

        delete [] m_dSum1;
        delete [] m_dSum2;

    }

    else//show feature extraction trajectory
    {   //pull VarDim from record... interpolate
        int ctr3=-1;
        for (int ctr=TransientBeginsHere;
            ctr<TransientBeginsHere+m_nTransientSize; ctr+=m_nWindowShift)
        {
            ctr3++;
            VarDimTraj[ctr]=TransientModel[ctr3];
        }
        VarDimTraj[ctr]=TransientModel[ctr3]; //fill last spot with same
                                              //vardim for interpolation
        Interpolate(VarDimTraj, TransientBeginsHere);
```

```
    }
    //FillTransientModel(VarDimTraj, TransientBeginsHere, TransientModel);
}

double CExtract::CalcVarDim(int *RawSignal, int StartLoc)
{
    double *LogOfVariance, *LogOfDeltaT;
    double Sum1,Sum2,DeltaB;
    double DimensionPower;
    double Var, VarDim;
    int k,t,NoOfDeltaBs,NoOfZeroes;

    LogOfVariance=new double[m_nVariancePairs+1];
    LogOfDeltaT=new double[m_nVariancePairs+1];

/*  MSG message;
    if (::PeekMessage(&message, NULL, 0, 0, PM_REMOVE)) {
        ::TranslateMessage(&message);
        ::DispatchMessage(&message);
    }
    AfxGetApp()->DoWaitCursor(1);
*/
    NoOfZeroes=0;
    if (m_nDyadic==0) {
        for (t=1; t<=m_nVariancePairs; t++) {
            NoOfDeltaBs=m_nWindowSize-t;
            if (StartLoc==0) {//must perform all calcs for first window
                m_dSum1[t]=0.0;
                m_dSum2[t]=0.0;
                for(k=StartLoc; k<(StartLoc+NoOfDeltaBs); k++) {
                    DeltaB=RawSignal[k+t]-RawSignal[k];
                    m_dSum1[t]+=DeltaB*DeltaB;
                    m_dSum2[t]+=DeltaB;
                }
            }
            else {
                DeltaB=RawSignal[StartLoc-1+t]-RawSignal[StartLoc-1];
                m_dSum1[t]-=DeltaB*DeltaB;
                m_dSum2[t]-=DeltaB;
                DeltaB=RawSignal[StartLoc+NoOfDeltaBs-1+t]
                    -RawSignal[StartLoc+NoOfDeltaBs-1];
                m_dSum1[t]+=DeltaB*DeltaB;
                m_dSum2[t]+=DeltaB;
            }
            Var=(m_dSum1[t]-(m_dSum2[t]*m_dSum2[t])/((double)NoOfDeltaBs))
                /((double)NoOfDeltaBs-1.0);
            if (Var==0.0) {
                NoOfZeroes++;
            }
            else {
                LogOfVariance[t-NoOfZeroes]=log(Var);
                LogOfDeltaT[t-NoOfZeroes]=log((double)t);
            }
        }
    }
    else {
        for (t=1; t<=m_nVariancePairs; t++) {
            Sum1=0.0;
            Sum2=0.0;
            DimensionPower=pow(2.0,(double)(t));
            NoOfDeltaBs=m_nWindowSize-(int)DimensionPower;
            for(k=StartLoc; k<(StartLoc+NoOfDeltaBs); k++) {
                DeltaB=RawSignal[k+(int)DimensionPower]-RawSignal[k];
                Sum1+=DeltaB*DeltaB;
                Sum2+=DeltaB;
```

```
                }
                if(NoOfDeltaBs!=0)//protect against divide by zero
                    Var=(Sum1-(Sum2*Sum2)/((double)NoOfDeltaBs))
                        /((double)NoOfDeltaBs-1.0);
                else
                    Var=0.0;
                if (Var==0.0) {
                    NoOfZeroes++;
                }
                else {
                    LogOfVariance[t-NoOfZeroes]=log(Var);
                    LogOfDeltaT[t-NoOfZeroes]=log(DimensionPower);
                }
            }
        }
        double sum1=0.0, sum2=0.0, sum3=0.0, sum4=0.0, sum5=0.0;

        for (int ctr=1; ctr<=(m_nVariancePairs-NoOfZeroes); ctr++)
            {   sum1+=LogOfDeltaT[ctr]*LogOfVariance[ctr];
                sum2+=LogOfDeltaT[ctr];
                sum3+=LogOfVariance[ctr];
                sum4+=LogOfDeltaT[ctr]*LogOfDeltaT[ctr];
                sum5+=LogOfDeltaT[ctr];
            }
        if (m_nDyadic==1) //these modifications (4 lines were made to implement the model
        m_nVariancePairs++; //transformation
        double Slope=(m_nVariancePairs*sum1-sum2*sum3)
            /(m_nVariancePairs*sum4-sum5*sum5);
        VarDim=2.0-0.5*Slope;
        if (m_nDyadic==1) // part of the transformation
        m_nVariancePairs--;  // last line of transformation.
        delete [] LogOfVariance;
        delete [] LogOfDeltaT;

        return VarDim;
}

double CExtract::CalcVarDim(double *RawSignal, int StartLoc)
{
        double *LogOfVariance, *LogOfDeltaT;
        double Sum1,Sum2,DeltaB;
        double DimensionPower;
        double Var, VarDim;
        int k,t,NoOfDeltaBs,NoOfZeroes;

        LogOfVariance=new double[m_nVariancePairs+1];
        LogOfDeltaT=new double[m_nVariancePairs+1];

/*      MSG message;
        if (::PeekMessage(&message, NULL, 0, 0, PM_REMOVE)) {
            ::TranslateMessage(&message);
            ::DispatchMessage(&message);
        }
        AfxGetApp()->DoWaitCursor(1);
*/
        NoOfZeroes=0;
        if (m_nDyadic==0) {
            for (t=1; t<=m_nVariancePairs; t++) {
                NoOfDeltaBs=m_nWindowSize-t;
                if (StartLoc==0) {//must perform all calcs for first window
                    m_dSum1[t]=0.0;
                    m_dSum2[t]=0.0;
                    for(k=StartLoc; k<(StartLoc+NoOfDeltaBs); k++) {
                        DeltaB=RawSignal[k+t]-RawSignal[k];
                        m_dSum1[t]+=DeltaB*DeltaB;
```

```
                    m_dSum2[t]+=DeltaB;
                }
            }
            else {
                DeltaB=RawSignal[StartLoc-1+t]-RawSignal[StartLoc-1];
                m_dSum1[t]-=DeltaB*DeltaB;
                m_dSum2[t]-=DeltaB;
                DeltaB=RawSignal[StartLoc+NoOfDeltaBs-1+t]
                    -RawSignal[StartLoc+NoOfDeltaBs-1];
                m_dSum1[t]+=DeltaB*DeltaB;
                m_dSum2[t]+=DeltaB;
            }
            Var=(m_dSum1[t]-(m_dSum2[t]*m_dSum2[t])/((double)NoOfDeltaBs))
                /((double)NoOfDeltaBs-1.0);
            if (Var==0.0) {
                NoOfZeroes++;
            }
            else {
                LogOfVariance[t-NoOfZeroes]=log(Var);
                LogOfDeltaT[t-NoOfZeroes]=log((double)t);
            }
        }
    }
    else {
        for (t=1; t<=m_nVariancePairs; t++) {
            Sum1=0.0;
            Sum2=0.0;
            DimensionPower=pow(2.0,(double)(t));
            NoOfDeltaBs=m_nWindowSize-(int)DimensionPower;
            for(k=StartLoc; k<(StartLoc+NoOfDeltaBs); k++) {
                DeltaB=RawSignal[k+(int)DimensionPower]-RawSignal[k];
                Sum1+=DeltaB*DeltaB;
                Sum2+=DeltaB;
            }
            if(NoOfDeltaBs!=0)//protect against divide by zero
                Var=(Sum1-(Sum2*Sum2)/((double)NoOfDeltaBs))
                    /((double)NoOfDeltaBs-1.0);
            else
                Var=0.0;
            if (Var==0.0) {
                NoOfZeroes++;
            }
            else {
                LogOfVariance[t-NoOfZeroes]=log(Var);
                LogOfDeltaT[t-NoOfZeroes]=log(DimensionPower);
            }
        }
    }
    double sum1=0.0, sum2=0.0, sum3=0.0, sum4=0.0, sum5=0.0;

    for (int ctr=1; ctr<=(m_nVariancePairs-NoOfZeroes); ctr++)
        {   sum1+=LogOfDeltaT[ctr]*LogOfVariance[ctr];
            sum2+=LogOfDeltaT[ctr];
            sum3+=LogOfVariance[ctr];
            sum4+=LogOfDeltaT[ctr]*LogOfDeltaT[ctr];
            sum5+=LogOfDeltaT[ctr];
        }
    if (m_nDyadic==1) //these modifications (4 lines were made to implement the model
    m_nVariancePairs++; //transformation
    double Slope=(m_nVariancePairs*sum1-sum2*sum3)
        /(m_nVariancePairs*sum4-sum5*sum5);
    VarDim=2.0-0.5*Slope;
    if (m_nDyadic==1) // part of the transformation
    m_nVariancePairs--;  // last line of transformation.
    delete [] LogOfVariance;
```

```
    delete [] LogOfDeltaT;

    return VarDim;
}

void CExtract::Interpolate(double *VarDimTraj, int StartPoint)//linear interpolation
{                                                              //for display purposes only
    int VarDimFileSize;
#ifdef _DEBUG
    int Count=0;
    double Mean=0.0, StdDev=0.0;
#endif
    if (StartPoint==0)
        VarDimFileSize=(m_nRawFileSize-m_nWindowSize);
    else
        VarDimFileSize=StartPoint+m_nTransientSize;

    for (int ctr=StartPoint;ctr<VarDimFileSize; ctr+=m_nWindowShift) {
#ifdef _DEBUG
        Count++;
        Mean+=VarDimTraj[ctr];
        StdDev+=VarDimTraj[ctr]*VarDimTraj[ctr];
#endif
        for (int ctr2=1; ctr2<m_nWindowShift; ctr2++) {
            VarDimTraj[ctr+ctr2]=VarDimTraj[ctr]+(float)((float)ctr2/(float)
                m_nWindowShift)*(VarDimTraj[ctr+m_nWindowShift]-VarDimTraj[ctr]);
        }
    }
#ifdef _DEBUG
    Mean/=(double)Count;
    StdDev=sqrt((StdDev/(double)(Count))-(Mean*Mean));
    TRACE("\nDon...Average from sample %d is %f", StartPoint, Mean);
    TRACE("\nDon...Standard Deviation is %f", StdDev);
#endif
}

void CExtract::FindMultiTransients(int *RawSignal, double *VarDimTraj,
                                   double **TransientModel, int NumModes,
                                   double *Threshholds, int MinSeperation,
                                   int *TransientBeginsHere)
{
    m_nDyadic=0;
    m_nWindowSize=m_nSegmentationWindowSize;
    m_nVariancePairs=m_nSegmentationVariancePairs;
    double Mean=0.0, StdDev=0.0, MeanDif=0.0;

    int ctr, ctr1, Start=1800;
    int GoodFileSize=m_nRawFileSize-m_nWindowSize;//don't perform calcs
                                                  //past the end of file
    m_dSum1=new double[m_nSegmentationVariancePairs+1];
    m_dSum2=new double[m_nSegmentationVariancePairs+1];

    // Calculate the mean and the Standard Deviation of the VarDimTraj from
    // the start of the file to 1/4 of the file size
    for (ctr=0; ctr<1800; ctr++) {//note windwoshift=1
        VarDimTraj[ctr]=CalcVarDim(RawSignal, ctr);
        Mean+=VarDimTraj[ctr];
        StdDev+=VarDimTraj[ctr]*VarDimTraj[ctr];

        if(ctr!=0)
            MeanDif+=fabs(VarDimTraj[ctr]-VarDimTraj[ctr-1]);

    }

    MeanDif/=(double)(1800-1);
```

```
        Mean/=(double)(1800);
        StdDev=sqrt((StdDev/(double)(1800))-(Mean*Mean));

        for(ctr1=0; ctr1<NumModes; ctr1++) {
            TransientBeginsHere[ctr1]=-1;

        // Adjust the threshhold based on the value of the standard deviation
            double Deviation=(Threshholds[ctr1]/100.0)*Mean+StdDev;


            switch(m_nTrigScheme) {
            case 0://absolute triggering
                for (ctr=Start; ctr<GoodFileSize; ctr++) {
                    VarDimTraj[ctr]=CalcVarDim(RawSignal, ctr);
                    if(fabs(VarDimTraj[ctr]-Mean)>Deviation) {
                        TRACE("Don, (multiply absolute triggering)Transient is at %d\n",ctr);
                        TransientBeginsHere[ctr1]=ctr;
                        break;
                    }
                }
                break;

            case 1: //relative triggering
                for (ctr=Start; ctr<GoodFileSize; ctr++) {
                    VarDimTraj[ctr]=CalcVarDim(RawSignal, ctr);
                    if(fabs(VarDimTraj[ctr]-VarDimTraj[ctr-1])>m_nAdjust*Deviation) {
                        TransientBeginsHere[ctr1]=ctr;
                    break;
                    }
                }
                break;
            default:
            }

            if (TransientBeginsHere[ctr1]
                >(m_nRawFileSize-m_nTransientSize-m_nModelWindowSize)) {
                TransientBeginsHere[ctr1]=-1;
            }

            if (TransientBeginsHere[ctr1]!=-1) {
                FillTransientModel(VarDimTraj, TransientBeginsHere[ctr1],
                    TransientModel[ctr1]);
                Start=TransientBeginsHere[ctr1]+MinSeperation;
            }
        }

        delete [] m_dSum1;
        delete [] m_dSum2;

        return;
}

int CExtract::FindSquelchTrigger(int *SquelchChannel)
{
        int TransientBeginsHere = -1; // -1 indicates no transient found.
        bool TriggerOccurred = FALSE;
        int i, j;

        // Step through the SquelchChannel array looking for it to
        // rise above m_nSquelchTrig.
        for (i = m_nSquelchDelay; i < (m_nRawFileSize - m_nSquelchTrigWidth); i++)
        {
            if (SquelchChannel[i] >= m_nSquelchTrig)
            {
                TriggerOccurred = TRUE;
```

```
                // Check if the squelch stays above m_nSquelchTrig
                // for m_nSquelchTrigWidth samples.
                for (j = 0; j < m_nSquelchTrigWidth; j++)
                {
                    if (SquelchChannel[i + j] <= m_nSquelchTrig)
                        TriggerOccurred = FALSE;

                }

                if (TriggerOccurred == TRUE)
                {
                    TransientBeginsHere = i - m_nSquelchDelay; // Take the squelch
                                                                // channel delay into account.
                    break;
                }

            }
        }

        return TransientBeginsHere;
}

int CExtract::FindVarianceTrigger(int *RawSignal, double * VarDimTraj)
{
            m_nDyadic=0;
            m_nWindowSize=m_nSegmentationWindowSize;
            m_nVariancePairs=m_nSegmentationVariancePairs;
            double Mean=0.0, StdDev=0.0;

            m_dSum1=new double[m_nSegmentationVariancePairs+1];
            m_dSum2=new double[m_nSegmentationVariancePairs+1];

            int TransientBeginsHere=-1;
            int ctr;
            int GoodFileSize=m_nRawFileSize-m_nWindowSize;//don't perform calcs
                                                          //past the end of file

            // Calculate the mean and the Standard Deviation of the VarDimTraj from
            // the start of the file up to 1/4 of the file size
            for (ctr=0; ctr<1800; ctr++) //note windwoshift=1
            {
                VarDimTraj[ctr]=CalcVarDim(RawSignal, ctr);
                Mean+=VarDimTraj[ctr];
                StdDev+=VarDimTraj[ctr]*VarDimTraj[ctr];

            }

            Mean/=(double)(1800);
            StdDev=sqrt((StdDev/(double)(1800))-(Mean*Mean));


            // Adjust the threshhold based on the value of the standard deviation
            double Deviation=(m_Threshold/100.0)*Mean+StdDev;

            //TRACE("Luotao,MeanDif is %lf\n", MeanDif);
            //TRACE("Luotao,Mean is %lf\n", Mean);
            //TRACE("Luotao,Deviation is %lf\n", Deviation);
            //TRACE("Luotao,StdDev is %lf\n", StdDev);
            double threshold;
            switch(m_nTrigScheme){

            case 0://absolute triggering
                threshold=Deviation;
                for (ctr=1800; ctr<GoodFileSize; ctr++)
```

```
        {
            VarDimTraj[ctr]=CalcVarDim(RawSignal, ctr);
            if(fabs(VarDimTraj[ctr]-Mean)>Deviation)
            {
                //TRACE("(Single Absolute triggering) Transient is at %d\n",ctr);
                TransientBeginsHere=ctr;
                break;
            }
        }
        break;
case 1://relative triggering
        threshold=m_nAdjust*Deviation;
        for (ctr=1800; ctr<GoodFileSize; ctr++)
        {
            VarDimTraj[ctr]=CalcVarDim(RawSignal, ctr);
            if(fabs(VarDimTraj[ctr]-VarDimTraj[ctr-6])>threshold)
            {
                //TRACE("(relative triggering)Transient is at %d\n",ctr);
                //TRACE("(Single) m_nAdjust = %f\n", m_nAdjust);
                TransientBeginsHere=ctr;
                break;
            }
            /*else if(fabs(VarDimTraj[ctr]-Mean)>Deviation)
            {
                TRACE("(return to absolute)Transient is at %d\n", ctr);
                TransientBeginsHere = ctr;
                break;
            }*/
        }
        if(TransientBeginsHere == -1) //if relative fails, go back to absolute triggering
        {
            threshold=Deviation;
            for (ctr=1800; ctr<GoodFileSize; ctr++)
            {
                VarDimTraj[ctr]=CalcVarDim(RawSignal, ctr);
                if(fabs(VarDimTraj[ctr]-Mean)>Deviation)
                {
                    //TRACE("(return to absolute) Transient is at %d\n",ctr);
                    TransientBeginsHere=ctr;
                    break;
                }
            }
        }
        break;
default:
        TRACE("default m_nAdjust = %f\n", m_nAdjust);
}

if (TransientBeginsHere>(m_nRawFileSize-m_nTransientSize-m_nModelWindowSize))
{
        TransientBeginsHere=-1;
}

if (m_Threshold==0.0)//used to circumvent segmentation procedure
        TransientBeginsHere=1800;

if (TransientBeginsHere!=-1)
{
        for (int i = TransientBeginsHere+1; i < GoodFileSize; i++)
        {
            VarDimTraj[i]=CalcVarDim(RawSignal, i);
        }

}
```

```
            delete [] m_dSum1;
            delete [] m_dSum2;


            return TransientBeginsHere;
}

int CExtract::GetSquelchTrigger()
{
      return m_nSquelchTrig;
}

int CExtract::GetSquelchWidth()
{
      return m_nSquelchTrigWidth;

}

int CExtract::GetSquelchDelay()
{
      return m_nSquelchDelay;
}

int CExtract::GetTrigScheme()
{
      return m_nTrigScheme;
}

float CExtract::GetAdjust()
{
      return m_nAdjust;
}
int CExtract::GetTrigWinSize()
{
      return m_nTrigWinSize;
}

TriggerType CExtract::GetTriggerType()
{
      return m_TrigType;
}
```

FractalParamsDialog.h

```
//This file is modified by the author to implement the relative triggering scheme
//CFractalParamDialog: defines and controls the dialog box displayed to change the fractal parameters
// FractalParamDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CFractalParamDialog dialog

#include "GlobalStuff.h" // Defines TriggerType

class CFractalParamDialog : public CDialog
{
// Construction
public:
	CFractalParamDialog(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	//{{AFX_DATA(CFractalParamDialog)
	enum { IDD = IDD_FRACTALPARAM_DIALOG };
	CEditm_nAdjustEdit;
	CEditm_editTrigWinSize;
	CEditm_editThreshold;
	CEditm_editSquelchWidth;
	CEditm_editSquelchTrigger;
	CEditm_editSquelchDelay;
	BOOLm_SetSegmentationParams;
	BOOLm_SetModelParams;
	CEditm_ModelWindowShiftEdit;
	CEditm_ModelWindowSizeEdit;
	CEditm_ModelVariancePairsEdit;
	CEditm_WindowSizeEdit;
	CEditm_VariancePairsEdit;
	CEditm_TransientSizeEdit;
	CEditm_RawFileSizeEdit;
	int		m_RawFileSize;
	int		m_TransientSize;
	int		m_WindowSize;
	int		m_VariancePairs;
	int		m_ModelVariancePairs;
	int		m_ModelWindowSize;
	int		m_ModelWindowShift;
	int		m_nSquelchDelay;
	int		m_nSquelchTrigger;
	int		m_nSquelchWidth;
	doublem_Threshold;
	int		m_nTrigWinSize;
	BOOLm_bChangeTransientSize;
	int		m_nTrigScheme;
	float	m_nAdjust;
	int		m_nReducedDim;
	int		m_nTransientNum;
	BOOLm_SetNNParams;
	//}}AFX_DATA

	TriggerType m_triggerType;
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CFractalParamDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL
```

```
// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CFractalParamDialog)
	virtual BOOL OnInitDialog();
	afx_msg void OnModelparams();
	afx_msg void OnSegmentationparams();
	afx_msg void OnTrigWinVar();
	afx_msg void OnTrigVar();
	afx_msg void OnTrigSquelch();
	afx_msg void OnChangeTransientSize();
	afx_msg void OnNnparams();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()

	void EnableTrigParams();
};
```

FractalParamsDialog.cpp

```
//This file is modified by the author to implement the relative triggering scheme

// FractalParamDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "FractalParamDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFractalParamDialog dialog


CFractalParamDialog::CFractalParamDialog(CWnd* pParent /*=NULL*/)
        : CDialog(CFractalParamDialog::IDD, pParent)
{
        //{{AFX_DATA_INIT(CFractalParamDialog)
    m_SetSegmentationParams = FALSE;
    m_SetModelParams = FALSE;
    m_nSquelchDelay = 0;
    m_nSquelchTrigger = 0;
    m_nSquelchWidth = 0;
    m_Threshold = 0;
    m_nTrigWinSize = 0;
    m_bChangeTransientSize = FALSE;
    m_nTrigScheme = -1;
    m_nAdjust = 0.0f;
    m_nReducedDim = 0;
    m_nTransientNum = 0;
    m_SetNNParams = FALSE;
    //}}AFX_DATA_INIT

    m_triggerType = VARIANCE;

}



void CFractalParamDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CFractalParamDialog)
    DDX_Control(pDX, IDC_ADJUST, m_nAdjustEdit);
    DDX_Control(pDX, IDC_TRIGWINSIZE, m_editTrigWinSize);
    DDX_Control(pDX, IDC_THRESHOLD, m_editThreshold);
    DDX_Control(pDX, IDC_SQUELCHWIDTH, m_editSquelchWidth);
    DDX_Control(pDX, IDC_SQUELCHTRIG, m_editSquelchTrigger);
    DDX_Control(pDX, IDC_SQUELCHDELAY, m_editSquelchDelay);
    DDX_Check(pDX, IDC_SEGMENTATIONPARAMS, m_SetSegmentationParams);
    DDX_Check(pDX, IDC_MODELPARAMS, m_SetModelParams);
    DDX_Control(pDX, IDC_MODELWINDOWSHIFT, m_ModelWindowShiftEdit);
    DDX_Control(pDX, IDC_WINDOWSIZE2, m_ModelWindowSizeEdit);
    DDX_Control(pDX, IDC_VARIANCEPAIRS2, m_ModelVariancePairsEdit);
    DDX_Control(pDX, IDC_WINDOWSIZE, m_WindowSizeEdit);
```

```
    DDX_Control(pDX, IDC_VARIANCEPAIRS, m_VariancePairsEdit);
    DDX_Control(pDX, IDC_TRANSIENTSIZE, m_TransientSizeEdit);
    DDX_Control(pDX, IDC_RAWFILESIZE, m_RawFileSizeEdit);
    DDX_Text(pDX, IDC_RAWFILESIZE, m_RawFileSize);
    DDV_MinMaxInt(pDX, m_RawFileSize, 1000, 25000);
    DDX_Text(pDX, IDC_TRANSIENTSIZE, m_TransientSize);
    DDV_MinMaxInt(pDX, m_TransientSize, 100, 25000);
    DDX_Text(pDX, IDC_WINDOWSIZE, m_WindowSize);
    DDV_MinMaxInt(pDX, m_WindowSize, 128, 2048);
    DDX_Text(pDX, IDC_VARIANCEPAIRS, m_VariancePairs);
    DDV_MinMaxInt(pDX, m_VariancePairs, 3, 25);
    DDX_Text(pDX, IDC_VARIANCEPAIRS2, m_ModelVariancePairs);
    DDV_MinMaxInt(pDX, m_ModelVariancePairs, -1, 9);
    DDX_Text(pDX, IDC_WINDOWSIZE2, m_ModelWindowSize);
    DDV_MinMaxInt(pDX, m_ModelWindowSize, 128, 1024);
    DDX_Text(pDX, IDC_MODELWINDOWSHIFT, m_ModelWindowShift);
    DDV_MinMaxInt(pDX, m_ModelWindowShift, 1, 512);
    DDX_Text(pDX, IDC_SQUELCHDELAY, m_nSquelchDelay);
    DDV_MinMaxInt(pDX, m_nSquelchDelay, 0, 32767);
    DDX_Text(pDX, IDC_SQUELCHTRIG, m_nSquelchTrigger);
    DDV_MinMaxInt(pDX, m_nSquelchTrigger, -32768, 32767);
    DDX_Text(pDX, IDC_SQUELCHWIDTH, m_nSquelchWidth);
    DDV_MinMaxInt(pDX, m_nSquelchWidth, 0, 32767);
    DDX_Text(pDX, IDC_THRESHOLD, m_Threshold);
    DDV_MinMaxDouble (pDX, m_Threshold, 1, 20);
    DDX_Text(pDX, IDC_TRIGWINSIZE, m_nTrigWinSize);
    DDV_MinMaxInt(pDX, m_nTrigWinSize, 0, 32767);
    DDX_Check(pDX, IDC_CHANGE_TRANSIENT_SIZE, m_bChangeTransientSize);
    DDX_Radio(pDX, IDC_TRIGSCHEME, m_nTrigScheme);
    DDX_Text(pDX, IDC_ADJUST, m_nAdjust);
    DDV_MinMaxFloat(pDX, m_nAdjust, 0.f, 1.f);
    DDX_Text(pDX, IDC_REDUCEDDIM, m_nReducedDim);
    DDV_MinMaxInt(pDX, m_nReducedDim, 0, 128);
    DDX_Text(pDX, IDC_TRANSIENTNUM, m_nTransientNum);
    DDV_MinMaxInt(pDX, m_nTransientNum, 0, 100);
    DDX_Check(pDX, IDC_NNPARAMS, m_SetNNParams);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CFractalParamDialog, CDialog)
        //{{AFX_MSG_MAP(CFractalParamDialog)
ON_BN_CLICKED(IDC_MODELPARAMS, OnModelparams)
ON_BN_CLICKED(IDC_SEGMENTATIONPARAMS, OnSegmentationparams)
ON_BN_CLICKED(IDC_TRIGWINVAR, OnTrigWinVar)
ON_BN_CLICKED(IDC_TRIGVAR, OnTrigVar)
ON_BN_CLICKED(IDC_TRIGSQUELCH, OnTrigSquelch)
ON_BN_CLICKED(IDC_CHANGE_TRANSIENT_SIZE, OnChangeTransientSize)
ON_BN_CLICKED(IDC_NNPARAMS, OnNnparams)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFractalParamDialog message handlers

BOOL CFractalParamDialog::OnInitDialog()
{
        CDialog::OnInitDialog();

        m_VariancePairsEdit.LimitText(2);
        m_ModelWindowShiftEdit.LimitText(3);
        m_ModelVariancePairsEdit.LimitText(2);
        m_ModelWindowSizeEdit.LimitText(4);
        m_WindowSizeEdit.LimitText(4);
        m_editThreshold.LimitText(6);
```

```
        m_nAdjustEdit.LimitText(4);//# of chars user can enter in edit box
        GetDlgItem(IDC_WINDOWSIZE)->EnableWindow(FALSE);
        GetDlgItem(IDC_VARIANCEPAIRS)->EnableWindow(FALSE);
        GetDlgItem(IDC_THRESHOLD)->EnableWindow(FALSE);
        GetDlgItem(IDC_WINDOWSIZE2)->EnableWindow(FALSE);
        GetDlgItem(IDC_VARIANCEPAIRS2)->EnableWindow(FALSE);
        GetDlgItem(IDC_MODELWINDOWSHIFT)->EnableWindow(FALSE);
        GetDlgItem(IDC_SQUELCHTRIG)->EnableWindow(FALSE);
        GetDlgItem(IDC_SQUELCHWIDTH)->EnableWindow(FALSE);
        GetDlgItem(IDC_SQUELCHDELAY)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRIGWINSIZE)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRIGVAR)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRIGSQUELCH)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRIGWINVAR)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRANSIENTSIZE)->EnableWindow(FALSE);
        GetDlgItem(IDC_ADJUST)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRIGSCHEME)->EnableWindow(FALSE);
        GetDlgItem(IDC_RADIO2)->EnableWindow(FALSE); //disable both radio buttons
        GetDlgItem(IDC_REDUCEDDIM)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRANSIENTNUM)->EnableWindow(FALSE);


        switch (m_triggerType)
        {
        case VARIANCE:
            CheckRadioButton(IDC_TRIGVAR, IDC_TRIGWINVAR, IDC_TRIGVAR);
            break;

        case SQUELCH:
            CheckRadioButton(IDC_TRIGVAR, IDC_TRIGWINVAR, IDC_TRIGSQUELCH);
            break;

        case WINDOWEDVARIANCE:
            CheckRadioButton(IDC_TRIGVAR, IDC_TRIGWINVAR, IDC_TRIGWINVAR);
            break;
        }


        m_SetModelParams=FALSE;
        m_SetSegmentationParams=FALSE;
        m_SetNNParams=FALSE;

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}

void CFractalParamDialog::OnModelparams()
{
    if(m_SetModelParams=IsDlgButtonChecked(IDC_MODELPARAMS))
    {
        GetDlgItem(IDC_WINDOWSIZE2)->EnableWindow(TRUE);
        GetDlgItem(IDC_VARIANCEPAIRS2)->EnableWindow(TRUE);
        GetDlgItem(IDC_MODELWINDOWSHIFT)->EnableWindow(TRUE);
    }
    else
    {
        GetDlgItem(IDC_WINDOWSIZE2)->EnableWindow(FALSE);
        GetDlgItem(IDC_VARIANCEPAIRS2)->EnableWindow(FALSE);
        GetDlgItem(IDC_MODELWINDOWSHIFT)->EnableWindow(FALSE);
    }

    return;
}

void CFractalParamDialog::OnSegmentationparams()
```

```
{
    if(m_SetSegmentationParams=IsDlgButtonChecked(IDC_SEGMENTATIONPARAMS))
    {
        GetDlgItem(IDC_TRIGVAR)->EnableWindow(TRUE);
        GetDlgItem(IDC_TRIGSQUELCH)->EnableWindow(TRUE);
        GetDlgItem(IDC_TRIGWINVAR)->EnableWindow(TRUE);
        GetDlgItem(IDC_TRIGSCHEME)->EnableWindow(TRUE);
        GetDlgItem(IDC_RADIO2)->EnableWindow(TRUE);
        EnableTrigParams();
    }
    else
    {
        GetDlgItem(IDC_WINDOWSIZE)->EnableWindow(FALSE);
        GetDlgItem(IDC_VARIANCEPAIRS)->EnableWindow(FALSE);
        GetDlgItem(IDC_THRESHOLD)->EnableWindow(FALSE);
        GetDlgItem(IDC_SQUELCHTRIG)->EnableWindow(FALSE);
        GetDlgItem(IDC_SQUELCHWIDTH)->EnableWindow(FALSE);
        GetDlgItem(IDC_SQUELCHDELAY)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRIGWINSIZE)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRIGVAR)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRIGSQUELCH)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRIGWINVAR)->EnableWindow(FALSE);
        GetDlgItem(IDC_ADJUST)->EnableWindow(FALSE);
    }

    return;
}


void CFractalParamDialog::OnTrigWinVar()
{
    // TODO: Add your control notification handler code here
    if (IsDlgButtonChecked(IDC_TRIGWINVAR))
    {
        m_triggerType = WINDOWEDVARIANCE;
        EnableTrigParams();
    }

    return;
}

void CFractalParamDialog::OnTrigVar()
{
    // TODO: Add your control notification handler code here
    if (IsDlgButtonChecked(IDC_TRIGVAR))
    {
        m_triggerType = VARIANCE;
        EnableTrigParams();
    }

    return;
}

void CFractalParamDialog::OnTrigSquelch()
{
    // TODO: Add your control notification handler code here
    if (IsDlgButtonChecked(IDC_TRIGSQUELCH))
    {
        m_triggerType = SQUELCH;
        EnableTrigParams();
    }

    return;
}
```

```
void CFractalParamDialog::EnableTrigParams()
{
	switch (m_triggerType)
	{
	case WINDOWEDVARIANCE:
		GetDlgItem(IDC_WINDOWSIZE)->EnableWindow(TRUE);
		GetDlgItem(IDC_VARIANCEPAIRS)->EnableWindow(TRUE);
		GetDlgItem(IDC_THRESHOLD)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHTRIG)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHWIDTH)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHDELAY)->EnableWindow(TRUE);
		GetDlgItem(IDC_TRIGWINSIZE)->EnableWindow(TRUE);
		GetDlgItem(IDC_ADJUST)->EnableWindow(TRUE);

		break;

	case SQUELCH:
		GetDlgItem(IDC_WINDOWSIZE)->EnableWindow(FALSE);
		GetDlgItem(IDC_VARIANCEPAIRS)->EnableWindow(FALSE);
		GetDlgItem(IDC_THRESHOLD)->EnableWindow(FALSE);
		GetDlgItem(IDC_SQUELCHTRIG)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHWIDTH)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHDELAY)->EnableWindow(TRUE);
		GetDlgItem(IDC_TRIGWINSIZE)->EnableWindow(FALSE);
		GetDlgItem(IDC_ADJUST)->EnableWindow(FALSE);

		break;

	case VARIANCE:
		GetDlgItem(IDC_WINDOWSIZE)->EnableWindow(TRUE);
		GetDlgItem(IDC_VARIANCEPAIRS)->EnableWindow(TRUE);
		GetDlgItem(IDC_THRESHOLD)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHTRIG)->EnableWindow(FALSE);
		GetDlgItem(IDC_SQUELCHWIDTH)->EnableWindow(FALSE);
		GetDlgItem(IDC_SQUELCHDELAY)->EnableWindow(FALSE);
		GetDlgItem(IDC_TRIGWINSIZE)->EnableWindow(FALSE);
		GetDlgItem(IDC_ADJUST)->EnableWindow(TRUE);

		break;
	}

	return;
}

void CFractalParamDialog::OnChangeTransientSize()
{
	// TODO: Add your control notification handler code here
	if(m_bChangeTransientSize=IsDlgButtonChecked(IDC_CHANGE_TRANSIENT_SIZE))
	{
		GetDlgItem(IDC_TRANSIENTSIZE)->EnableWindow(TRUE);
	}
	else
	{
		GetDlgItem(IDC_TRANSIENTSIZE)->EnableWindow(FALSE);
	}
}


void CFractalParamDialog::OnNnparams()
{
	// TODO: Add your control notification handler code here
	if(m_SetNNParams=IsDlgButtonChecked(IDC_NNPARAMS))
	{
		GetDlgItem(IDC_REDUCEDDIM)->EnableWindow(TRUE);
```

```
        GetDlgItem(IDC_TRANSIENTNUM)->EnableWindow(TRUE);
    }
    else
    {
        GetDlgItem(IDC_REDUCEDDIM)->EnableWindow(FALSE);
        GetDlgItem(IDC_TRANSIENTNUM)->EnableWindow(FALSE);
    }

    return;
}
```

MainFrm.h

```
//CMainFrame, which controls all SDI frame features
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
	CMainFrame();
	DECLARE_DYNCREATE(CMainFrame)
// Attributes
protected:
	CSplitterWnd m_wndSplitter;
public:
// Operations
public:
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMainFrame)
	public:
	virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext);
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	//}}AFX_VIRTUAL

// Implementation
public:
	CSplitterWnd* GetSplitter();
	enum eView {TEXT = 9, LIST = 10, RESULTS = 11};
	void SwitchToView(int Row, int Col, eView nView);
	void ShowTrainingResults();
	virtual ~CMainFrame();

#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
	CStatusBar  m_wndStatusBar;

protected:  // control bar embedded members
	CToolBar    m_wndToolBar;

// Generated message map functions
protected:
	//{{AFX_MSG(CMainFrame)
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnViewSegmentation();
	afx_msg void OnViewFractaltrajectory();
	afx_msg void OnViewRawsignal();
	afx_msg void OnViewTransient();
	afx_msg void OnViewZoomedrawsignal();
	afx_msg void OnUpdateViewZoomedrawsignal(CCmdUI* pCmdUI);
	afx_msg void OnViewNetworkdetails();
	afx_msg void OnUpdateViewNetworkdetails(CCmdUI* pCmdUI);
	afx_msg void OnViewTrainingresults();
	afx_msg void OnUpdateViewTrainingresults(CCmdUI* pCmdUI);
	afx_msg void OnUpdateStartCapture(CCmdUI* pCmdUI);
	afx_msg void OnParentNotify(UINT message, LPARAM lParam);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
```

MainFrm.cpp

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "TAC_MM.h"

#include "MainFrm.h"
#include "TAC_MMDoc.h"
#include "TAC_MMView.h"
#include "RawView.h"
#include "ResultsView.h"
#include "TransientListView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_COMMAND(ID_VIEW_SEGMENTATION, OnViewSegmentation)
    ON_COMMAND(ID_VIEW_FRACTALTRAJECTORY, OnViewFractaltrajectory)
    ON_COMMAND(ID_VIEW_RAWSIGNAL, OnViewRawsignal)
    ON_COMMAND(ID_VIEW_TRANSIENT, OnViewTransient)
    ON_COMMAND(ID_VIEW_ZOOMEDRAWSIGNAL, OnViewZoomedrawsignal)
    ON_UPDATE_COMMAND_UI(ID_VIEW_ZOOMEDRAWSIGNAL, OnUpdateViewZoomedrawsignal)
    ON_COMMAND(ID_VIEW_NETWORKDETAILS, OnViewNetworkdetails)
    ON_UPDATE_COMMAND_UI(ID_VIEW_NETWORKDETAILS, OnUpdateViewNetworkdetails)
    ON_COMMAND(ID_VIEW_TRAININGRESULTS, OnViewTrainingresults)
    ON_UPDATE_COMMAND_UI(ID_VIEW_TRAININGRESULTS, OnUpdateViewTrainingresults)
    ON_UPDATE_COMMAND_UI(ID_START_CAPTURE, OnUpdateStartCapture)
    ON_WM_PARENTNOTIFY()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,       // status line indicator
    ID_SEPARATOR,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{


}

CMainFrame::~CMainFrame()
{
}
```

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // fail to create
    }

    // TODO: Remove this if you don't want tool tips or a resizeable toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::OnCreateClient( LPCREATESTRUCT /*lpcs*/,
    CCreateContext* pContext)
{
    CRect rectClient;
    GetClientRect(rectClient); // Retrieves the size of the client area
                               // to set the size of the splitter window.

    VERIFY(m_wndSplitter.CreateStatic(this, 2, 1));
    VERIFY(m_wndSplitter.CreateView(0, 0, RUNTIME_CLASS(CRawView),
            CSize(rectClient.Width(), rectClient.Height()/2), pContext));
    VERIFY(m_wndSplitter.CreateView(1, 0, RUNTIME_CLASS(CTransientListView),
            CSize(rectClient.Width(), rectClient.Height()/2), pContext));
    return TRUE;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}
```

```
void CMainFrame::Dump(CDumpContext& dc) const
{
	CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers



void CMainFrame::OnViewSegmentation()
{
	m_wndSplitter.SetActivePane(0, 0, NULL);
}

void CMainFrame::OnViewFractaltrajectory()
{
	m_wndSplitter.SetActivePane(0, 0, NULL);
}

void CMainFrame::OnViewRawsignal()
{
	m_wndSplitter.SetActivePane(0, 0, NULL);
}

void CMainFrame::OnViewTransient()
{
	m_wndSplitter.SetActivePane(0, 0, NULL);
}

void CMainFrame::OnViewZoomedrawsignal()
{
	// TODO: Add your command handler code here
	CView* pCurrentView = (CView*) m_wndSplitter.GetPane(1,0);
	CTAC_MMDoc* pDoc = (CTAC_MMDoc*) GetActiveDocument();

	// Set the view to display the raw signal not the training or classification results.
	pDoc -> m_bViewTrainingResults = FALSE;
	pDoc -> m_bScrollToTransient = TRUE;


	if (!(pCurrentView -> IsKindOf(RUNTIME_CLASS(CTAC_MMView))))
	{
		// Get the dimensions of the second pane.
		CRect rect;
		pCurrentView -> GetWindowRect(&rect);
		CSize paneSize(rect.Width(), rect.Height());

		// This is needed for CreateView. Some members might be optional?
		CCreateContext Context;
		Context.m_pNewViewClass = RUNTIME_CLASS(CTAC_MMView);
		Context.m_pCurrentDoc = GetActiveDocument();
		Context.m_pCurrentFrame = this;
		Context.m_pNewDocTemplate = GetActiveDocument() -> GetDocTemplate();
		Context.m_pLastView = (CView*) pCurrentView;

		// Delete the old view and add the new one.
		m_wndSplitter.DeleteView(1, 0);
		m_wndSplitter.CreateView(1, 0, RUNTIME_CLASS(CTAC_MMView), paneSize, &Context);
		CTAC_MMView* pNewView = (CTAC_MMView*) m_wndSplitter.GetPane(1,0);

		// Redraw the second view.
		pNewView -> GetParentFrame() -> RecalcLayout();
```

```
            m_wndSplitter.RecalcLayout();
            pNewView -> OnInitialUpdate(); // Needed to set the scroll ranges.
        }
        else
        {
            pDoc -> UpdateAllViews(NULL);
        }


        m_wndSplitter.SetActivePane(1,0);
}


void CMainFrame::OnUpdateViewZoomedrawsignal(CCmdUI* pCmdUI)
{
        // TODO: Add your command update UI handler code here
        CTAC_MMDoc* pDoc = (CTAC_MMDoc*) GetActiveDocument();

        pCmdUI->Enable((pDoc -> m_bFileInMemory)&&!(pDoc -> m_bInBatchProcess));
        pCmdUI -> SetCheck((m_wndSplitter.GetPane(1,0) ->
            IsKindOf(RUNTIME_CLASS(CTAC_MMView))) && (pDoc -> m_bViewTrainingResults ? 0 : 1));
}

void CMainFrame::OnViewNetworkdetails()
{

        // TODO: Add your command handler code here
        CView* pCurrentView = (CView*) m_wndSplitter.GetPane(1,0);
        CTAC_MMDoc* pDoc = (CTAC_MMDoc*) GetActiveDocument();

        if (!(pCurrentView -> IsKindOf(RUNTIME_CLASS(CTransientListView))))
        {
            // Get the dimensions of the second pane.
            CRect rect;
            pCurrentView -> GetWindowRect(&rect);
            CSize paneSize(rect.Width(), rect.Height());

            // This is needed for CreateView. Some members might be optional?
            CCreateContext Context;
            Context.m_pNewViewClass = RUNTIME_CLASS(CTransientListView);
            Context.m_pCurrentDoc = GetActiveDocument();
            Context.m_pCurrentFrame = this;
            Context.m_pNewDocTemplate = GetActiveDocument() -> GetDocTemplate();
            Context.m_pLastView = (CView*) pCurrentView;

            // Delete the old view and add the new one.
            m_wndSplitter.DeleteView(1, 0);
            m_wndSplitter.CreateView(1, 0, RUNTIME_CLASS(CTransientListView), paneSize, &Context);
            CView* pNewView = (CView*) m_wndSplitter.GetPane(1,0);

            // Redraw the second view.
            pNewView -> GetParentFrame() -> RecalcLayout();
            m_wndSplitter.RecalcLayout();
            pNewView -> OnInitialUpdate(); // Needed to set the scroll ranges.
        }
        else
        {
            pDoc -> UpdateAllViews(NULL);
        }


        m_wndSplitter.SetActivePane(1,0);
}

void CMainFrame::OnUpdateViewNetworkdetails(CCmdUI* pCmdUI)
```

```
{
    // TODO: Add your command update UI handler code here
    CTAC_MMDoc* pDoc = (CTAC_MMDoc*) GetActiveDocument();

    pCmdUI -> Enable((pDoc -> m_bFileInMemory)&&!(pDoc -> m_bInBatchProcess));
    pCmdUI -> SetCheck(m_wndSplitter.GetPane(1,0) -> IsKindOf(RUNTIME_CLASS(CTransientListView)));
}

void CMainFrame::OnViewTrainingresults()
{
    // TODO: Add your command handler code here
    CView* pCurrentView = (CView*) m_wndSplitter.GetPane(1,0);
    CTAC_MMDoc* pDoc = (CTAC_MMDoc*) GetActiveDocument();

    // Set the doc to display the training or classification results not the raw signal.
    pDoc -> m_bViewTrainingResults = TRUE;

    if (!(pCurrentView -> IsKindOf(RUNTIME_CLASS(CTAC_MMView))))
    {
        // Get the dimensions of the second pane.
        CRect rect;
        pCurrentView -> GetWindowRect(&rect);
        CSize paneSize(rect.Width(), rect.Height());

        // This is needed for CreateView. Some members might be optional?
        CCreateContext Context;
        Context.m_pNewViewClass = RUNTIME_CLASS(CTAC_MMView);
        Context.m_pCurrentDoc = GetActiveDocument();
        Context.m_pCurrentFrame = this;
        Context.m_pNewDocTemplate = GetActiveDocument() -> GetDocTemplate();
        Context.m_pLastView = (CView*) pCurrentView;

        // Delete the old view and add the new one.
        m_wndSplitter.DeleteView(1, 0);
        m_wndSplitter.CreateView(1, 0, RUNTIME_CLASS(CTAC_MMView), paneSize, &Context);
        CTAC_MMView* pNewView = (CTAC_MMView*) m_wndSplitter.GetPane(1,0);

        // Redraw the second view.
        pNewView -> GetParentFrame() -> RecalcLayout();
        m_wndSplitter.RecalcLayout();
        pNewView -> OnInitialUpdate(); // Needed to set the scroll ranges.
    }
    else
    {
        pDoc -> UpdateAllViews(NULL);
    }


    m_wndSplitter.SetActivePane(1,0);
}

void CMainFrame::OnUpdateViewTrainingresults(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
        // TODO: Add your command update UI handler code here
    CTAC_MMDoc* pDoc = (CTAC_MMDoc*) GetActiveDocument();

    pCmdUI->Enable((pDoc -> m_bFileInMemory)&&(pDoc -> m_bResultsExist)&&!(pDoc -> m_bInBatchProcess));
    pCmdUI -> SetCheck((m_wndSplitter.GetPane(1,0) ->
        IsKindOf(RUNTIME_CLASS(CTAC_MMView))) && !(pDoc -> m_bViewTrainingResults ? 0 : 1));

}


void CMainFrame::ShowTrainingResults()
```

```
{
    OnViewTrainingresults();
}


void CMainFrame::SwitchToView(int Row, int Col, eView nView)
{
    // Note: this only works for the bottom pane views. (Row=1, Col = 0)
    CView* pCurrentView = (CView*) m_wndSplitter.GetPane(Row, Col);
    CView* pNewView;

    // Check if the current view is the same as the new view.
    if (!((nView == TEXT && pCurrentView -> IsKindOf(RUNTIME_CLASS(CTAC_MMView)))
        || (nView == LIST && pCurrentView -> IsKindOf(RUNTIME_CLASS(CTransientListView)))
        || (nView == RESULTS && pCurrentView -> IsKindOf(RUNTIME_CLASS(CResultsView)))))
    {
        // No, the new view needs to be displayed.
        CRect rect;
        pCurrentView -> GetWindowRect(&rect);
        CSize paneSize(rect.Width(), rect.Height()); // Get the current view's    //size.

        // Change the window ID of the current view to remove it from the
        // the splitter window and hide it.
        // The new window ID can be from 8 -> 0xDFFF, the rest
        // are reserved by MFC.
        if (pCurrentView -> GetRuntimeClass() == RUNTIME_CLASS(CTAC_MMView))
        {
            pCurrentView -> SetDlgCtrlID(TEXT);
        }
        else if (pCurrentView -> GetRuntimeClass() == RUNTIME_CLASS(CTransientListView))
        {
            pCurrentView -> SetDlgCtrlID(LIST);
        }
        else if (pCurrentView -> GetRuntimeClass() == RUNTIME_CLASS(CResultsView))
        {
            pCurrentView -> SetDlgCtrlID(RESULTS);
        }
        // Add more cases here to select from more views.
        // The cases must also be added to the enumerated type eView.


        pCurrentView -> ShowWindow(SW_HIDE);

        // Does a view of this type already exist.
        if (m_wndSplitter.GetDlgItem(nView) == NULL)
        {
            // No, create the view.
            CCreateContext context;
            context.m_pCurrentDoc = pCurrentView -> GetDocument();

            switch (nView)
            {
            case TEXT:
                m_wndSplitter.CreateView(Row, Col,
                    RUNTIME_CLASS(CTAC_MMView), paneSize, &context);
                break;

            case LIST:
                m_wndSplitter.CreateView(Row, Col,
                    RUNTIME_CLASS(CTransientListView), paneSize, &context);
                break;

            case RESULTS:
                m_wndSplitter.CreateView(Row, Col,
```

```
                    RUNTIME_CLASS(CResultsView), paneSize, &context);
                break;

            // Add more cases here to select from more views.
            // The cases must also be added to the enumerated type eView.
            }

            pNewView = (CView*) m_wndSplitter.GetPane(Row, Col);
            pNewView -> OnInitialUpdate();
        }
        else
        {
            // Yes, set the window ID of the new view that already exists
            // to the window ID of the current view and show it.
            pNewView = (CView*) m_wndSplitter.GetDlgItem(nView);
            pNewView -> SetDlgCtrlID(m_wndSplitter.IdFromRowCol(Row, Col));
            pNewView -> ShowWindow(SW_SHOW);
        }

        RecalcLayout();
        m_wndSplitter.RecalcLayout();
    }


    m_wndSplitter.SetActivePane(Row, Col);
}


CSplitterWnd* CMainFrame::GetSplitter()
{
    return &m_wndSplitter;
}



void CMainFrame::OnUpdateStartCapture(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here

}

void CMainFrame::OnParentNotify(UINT message, LPARAM lParam)
{
    CFrameWnd::OnParentNotify(message, lParam);

    // TODO: Add your message handler code here
    if (message == WM_DESTROY)
    {
        TRACE("Chris, Parent Notify message: ChildId = %d, lParam = %d.\n",
            message, lParam);
    }


}
```

ModeParamsDialog.h

```
//CModeParamsDialog: defines and controls the dialog box displayed to change the segmentation mode parameters
// ModeParamsDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CModeParamsDialog dialog

class CModeParamsDialog : public CDialog
{
// Construction
public:
	CModeParamsDialog(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	//{{AFX_DATA(CModeParamsDialog)
	enum { IDD = IDD_MODEPARAMETERSDIALOG };
	int		m_Mode;
	int		m_NumModes;
	int		m_MinSeparation;
	doublem_LowThreshhold;
	doublem_UpperThreshhold;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CModeParamsDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CModeParamsDialog)
	afx_msg void OnMultimode();
	afx_msg void OnSinglemode();
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

ModeParamsDialog.cpp

```
// ModeParamsDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "ModeParamsDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CModeParamsDialog dialog


CModeParamsDialog::CModeParamsDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CModeParamsDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CModeParamsDialog)
	m_Mode = -1;
	m_NumModes = 0;
	m_MinSeparation = 0;
	m_LowThreshhold = 0.0;
	m_UpperThreshhold = 0.0;
	//}}AFX_DATA_INIT
}


void CModeParamsDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CModeParamsDialog)
	DDX_Radio(pDX, IDC_SINGLEMODE, m_Mode);
	DDX_Text(pDX, IDC_NUMMODES, m_NumModes);
	DDV_MinMaxInt(pDX, m_NumModes, 1, 1000);
	DDX_Text(pDX, IDC_MINSEPARATION, m_MinSeparation);
	DDV_MinMaxInt(pDX, m_MinSeparation, 1, 100000);
	DDX_Text(pDX, IDC_LOWTHRESHHOLD, m_LowThreshhold);
	DDV_MinMaxDouble(pDX, m_LowThreshhold, 0., 50.);
	DDX_Text(pDX, IDC_UPPERTHRESHHOLD, m_UpperThreshhold);
	DDV_MinMaxDouble(pDX, m_UpperThreshhold, 0., 50.);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CModeParamsDialog, CDialog)
	//{{AFX_MSG_MAP(CModeParamsDialog)
	ON_BN_CLICKED(IDC_MULTIMODE, OnMultimode)
	ON_BN_CLICKED(IDC_SINGLEMODE, OnSinglemode)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CModeParamsDialog message handlers

void CModeParamsDialog::OnMultimode()
{
	GetDlgItem(IDC_NUMMODES)->EnableWindow(TRUE);
	GetDlgItem(IDC_LOWTHRESHHOLD)->EnableWindow(TRUE);
```

```
    GetDlgItem(IDC_UPPERTHRESHHOLD)->EnableWindow(TRUE);
    GetDlgItem(IDC_MINSEPARATION)->EnableWindow(TRUE);
}

void CModeParamsDialog::OnSinglemode()
{
    GetDlgItem(IDC_NUMMODES)->EnableWindow(FALSE);
    GetDlgItem(IDC_LOWTHRESHHOLD)->EnableWindow(FALSE);
    GetDlgItem(IDC_UPPERTHRESHHOLD)->EnableWindow(FALSE);
    GetDlgItem(IDC_MINSEPARATION)->EnableWindow(FALSE);
}

BOOL CModeParamsDialog::OnInitDialog()
{
    CDialog::OnInitDialog();

    m_Mode=0;
    UpdateData(FALSE);

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

NewDatabase.h

```
//This file is modified by the author to implement the relative triggering scheme.
//CNewDataBase: defines and controls the dialog box displayed to select parameters for a new database

// NewDatabase.h : header file
//

/////////////////////////////////////////////////////////////////////////
// CNewDatabase dialog

#include "GlobalStuff.h" // Defines TriggerType


class CNewDatabase : public CDialog
{
// Construction
public:
	CNewDatabase(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	//{{AFX_DATA(CNewDatabase)
	enum { IDD = IDD_NEW_DATABASE_DIALOG };
	CEditm_nAdjustEdit;
	CEditm_TrigWinSizeEdit;
	CEditm_SquelchTrigWidthEdit;
	CEditm_SquelchDelayEdit;
	CEditm_SquelchTrigEdit;
	CEditm_ModelWindowShiftEdit;
	CEditm_ModelWindowSizeEdit;
	CEditm_ModelVariancePairsEdit;
	CEditm_ThresholdEdit;
	CEditm_WindowSizeEdit;
	CEditm_VariancePairsEdit;
	CEditm_TransientSizeEdit;
	CEditm_RawFileSizeEdit;
	int		m_RawFileSize;
	int		m_TransientSize;
	int		m_WindowSize;
	int		m_VariancePairs;
	doublem_Threshold;
	int		m_ModelVariancePairs;
	int		m_ModelWindowSize;
	int		m_ModelWindowShift;
	int		m_nSquelchTrig;
	int		m_nSquelchDelay;
	int		m_nSquelchTrigWidth;
	int		m_nTrigWinSize;
	int		m_nTrigScheme;
	float	m_nAdjust;
	int		m_nReducedDim;
	int		m_nTransientNum;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CNewDatabase)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
```

```
public:
    TriggerType m_TrigType;  // Indicates the trigger meathod selected.

protected:

    // Generated message map functions
    //{{AFX_MSG(CNewDatabase)
    virtual BOOL OnInitDialog();
    afx_msg void OnTrigSquelch();
    afx_msg void OnTrigVar();
    afx_msg void OnTrigWinVar();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

NewDatabase.cpp

```
//This file is modified by the author to implement the relative triggering scheme.
// NewDatabase.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "NewDatabase.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CNewDatabase dialog

CNewDatabase::CNewDatabase(CWnd* pParent /*=NULL*/)
	: CDialog(CNewDatabase::IDD, pParent)
{
	//{{AFX_DATA_INIT(CNewDatabase)
	m_RawFileSize = 0;
	m_TransientSize = 0;
	m_WindowSize = 0;
	m_VariancePairs = 0;
	m_Threshold = 0.0;
	m_ModelVariancePairs = 0;
	m_ModelWindowSize = 0;
	m_ModelWindowShift = 0;
	m_nSquelchTrig = 0;
	m_nSquelchDelay = 0;
	m_nSquelchTrigWidth = 0;
	m_nTrigWinSize = 0;
	m_nTrigScheme = -1;
	m_nAdjust = 0.0f;
	m_nReducedDim = 0;
	m_nTransientNum = 0;
	//}}AFX_DATA_INIT
}


void CNewDatabase::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CNewDatabase)
	DDX_Control(pDX, IDC_ADJUST, m_nAdjustEdit);
	DDX_Control(pDX, IDC_TRIGWINSIZE, m_TrigWinSizeEdit);
	DDX_Control(pDX, IDC_SQUELCHWIDTH, m_SquelchTrigWidthEdit);
	DDX_Control(pDX, IDC_SQUELCHDELAY, m_SquelchDelayEdit);
	DDX_Control(pDX, IDC_SQUELCHTRIG, m_SquelchTrigEdit);
	DDX_Control(pDX, IDC_MODELWINDOWSHIFT, m_ModelWindowShiftEdit);
	DDX_Control(pDX, IDC_WINDOWSIZE2, m_ModelWindowSizeEdit);
	DDX_Control(pDX, IDC_VARIANCEPAIRS2, m_ModelVariancePairsEdit);
	DDX_Control(pDX, IDC_THRESHOLD, m_ThresholdEdit);
	DDX_Control(pDX, IDC_WINDOWSIZE, m_WindowSizeEdit);
	DDX_Control(pDX, IDC_VARIANCEPAIRS, m_VariancePairsEdit);
	DDX_Control(pDX, IDC_TRANSIENTSIZE, m_TransientSizeEdit);
	DDX_Control(pDX, IDC_RAWFILESIZE, m_RawFileSizeEdit);
	DDX_Text(pDX, IDC_RAWFILESIZE, m_RawFileSize);
	DDV_MinMaxInt(pDX, m_RawFileSize, 1000, 25000);
	DDX_Text(pDX, IDC_TRANSIENTSIZE, m_TransientSize);
```

```
    DDV_MinMaxInt(pDX, m_TransientSize, 100, 25000);
    DDX_Text(pDX, IDC_WINDOWSIZE, m_WindowSize);
    DDV_MinMaxInt(pDX, m_WindowSize, 128, 2048);
    DDX_Text(pDX, IDC_VARIANCEPAIRS, m_VariancePairs);
    DDV_MinMaxInt(pDX, m_VariancePairs, 3, 25);
    DDX_Text(pDX, IDC_THRESHOLD, m_Threshold);
    DDV_MinMaxDouble(pDX, m_Threshold, 0., 20.);
    DDX_Text(pDX, IDC_VARIANCEPAIRS2, m_ModelVariancePairs);
    DDV_MinMaxInt(pDX, m_ModelVariancePairs, -1, 9);
    DDX_Text(pDX, IDC_WINDOWSIZE2, m_ModelWindowSize);
    DDV_MinMaxInt(pDX, m_ModelWindowSize, 128, 1024);
    DDX_Text(pDX, IDC_MODELWINDOWSHIFT, m_ModelWindowShift);
    DDV_MinMaxInt(pDX, m_ModelWindowShift, 1, 512);
    DDX_Text(pDX, IDC_SQUELCHTRIG, m_nSquelchTrig);
    DDV_MinMaxInt(pDX, m_nSquelchTrig, -32768, 32767);
    DDX_Text(pDX, IDC_SQUELCHDELAY, m_nSquelchDelay);
    DDV_MinMaxInt(pDX, m_nSquelchDelay, 0, 32767);
    DDX_Text(pDX, IDC_SQUELCHWIDTH, m_nSquelchTrigWidth);
    DDV_MinMaxInt(pDX, m_nSquelchTrigWidth, 0, 32767);
    DDX_Text(pDX, IDC_TRIGWINSIZE, m_nTrigWinSize);
    DDV_MinMaxInt(pDX, m_nTrigWinSize, 0, 32767);
    DDX_Radio(pDX, IDC_TRIGSCHEME, m_nTrigScheme);
    DDX_Text(pDX, IDC_ADJUST, m_nAdjust);
    DDV_MinMaxFloat(pDX, m_nAdjust, 0.f, 1.f);
    DDX_Text(pDX, IDC_REDUCEDDIM, m_nReducedDim);
    DDV_MinMaxInt(pDX, m_nReducedDim, 0, 128);
    DDX_Text(pDX, IDC_TRANSIENTNUM, m_nTransientNum);
    DDV_MinMaxInt(pDX, m_nTransientNum, 0, 100);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CNewDatabase, CDialog)
    //{{AFX_MSG_MAP(CNewDatabase)
    ON_BN_CLICKED(IDC_TRIGSQUELCH, OnTrigSquelch)
    ON_BN_CLICKED(IDC_TRIGVAR, OnTrigVar)
    ON_BN_CLICKED(IDC_TRIGWINVAR, OnTrigWinVar)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
// CNewDatabase message handlers

BOOL CNewDatabase::OnInitDialog()
{
    CDialog::OnInitDialog();

    //m_nTrigScheme = 1;
    UpdateData(FALSE);

    m_RawFileSizeEdit.LimitText(5);
    m_TransientSizeEdit.LimitText(4);
    m_VariancePairsEdit.LimitText(2);
    m_ModelWindowShiftEdit.LimitText(3);
    m_ModelVariancePairsEdit.LimitText(2);
    m_ModelWindowSizeEdit.LimitText(4);
    m_WindowSizeEdit.LimitText(4);
    m_ThresholdEdit.LimitText(6);
    m_SquelchDelayEdit.LimitText(6);
    m_SquelchTrigEdit.LimitText(6);//# of chars user can enter in edit box
    m_TrigWinSizeEdit.LimitText(6);
    m_SquelchTrigWidthEdit.LimitText(6);
    m_nAdjustEdit.LimitText(4);

    // Enable the correct controls depending on the trigger type.
```

```
	switch (m_TrigType)
	{
	case VARIANCE:
		CheckDlgButton(IDC_TRIGVAR, 1);
		GetDlgItem(IDC_SQUELCHDELAY)->EnableWindow(FALSE);
		GetDlgItem(IDC_SQUELCHTRIG)->EnableWindow(FALSE);
		GetDlgItem(IDC_SQUELCHWIDTH)->EnableWindow(FALSE);
		GetDlgItem(IDC_WINDOWSIZE)->EnableWindow(TRUE);
		GetDlgItem(IDC_VARIANCEPAIRS)->EnableWindow(TRUE);
		GetDlgItem(IDC_THRESHOLD)->EnableWindow(TRUE);
		GetDlgItem(IDC_TRIGWINSIZE)->EnableWindow(FALSE);

		break;

	case SQUELCH:
		CheckDlgButton(IDC_TRIGSQUELCH, 1);
		GetDlgItem(IDC_SQUELCHDELAY)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHTRIG)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHWIDTH)->EnableWindow(TRUE);
		GetDlgItem(IDC_WINDOWSIZE)->EnableWindow(FALSE);
		GetDlgItem(IDC_VARIANCEPAIRS)->EnableWindow(FALSE);
		GetDlgItem(IDC_THRESHOLD)->EnableWindow(FALSE);
		GetDlgItem(IDC_TRIGWINSIZE)->EnableWindow(FALSE);
		GetDlgItem(IDC_ADJUST)->EnableWindow(FALSE);

		break;

	case WINDOWEDVARIANCE:
		CheckDlgButton(IDC_TRIGWINVAR, 1);
		GetDlgItem(IDC_SQUELCHDELAY)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHTRIG)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHWIDTH)->EnableWindow(TRUE);
		GetDlgItem(IDC_WINDOWSIZE)->EnableWindow(TRUE);
		GetDlgItem(IDC_VARIANCEPAIRS)->EnableWindow(TRUE);
		GetDlgItem(IDC_THRESHOLD)->EnableWindow(TRUE);
		GetDlgItem(IDC_TRIGWINSIZE)->EnableWindow(TRUE);
		GetDlgItem(IDC_ADJUST)->EnableWindow(TRUE);
		break;

	}

	return TRUE;  // return TRUE unless you set the focus to a control
		// EXCEPTION: OCX Property Pages should return FALSE
}

void CNewDatabase::OnTrigSquelch()
{
	// TODO: Add your control notification handler code here
	if(IsDlgButtonChecked(IDC_TRIGSQUELCH))
	{
		m_TrigType = SQUELCH;
		GetDlgItem(IDC_SQUELCHDELAY)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHTRIG)->EnableWindow(TRUE);
		GetDlgItem(IDC_SQUELCHWIDTH)->EnableWindow(TRUE);
		GetDlgItem(IDC_WINDOWSIZE)->EnableWindow(FALSE);
		GetDlgItem(IDC_VARIANCEPAIRS)->EnableWindow(FALSE);
		GetDlgItem(IDC_THRESHOLD)->EnableWindow(FALSE);
		GetDlgItem(IDC_TRIGWINSIZE)->EnableWindow(FALSE);
		GetDlgItem(IDC_ADJUST)->EnableWindow(FALSE);
	}


}

void CNewDatabase::OnTrigVar()
```

```
{
    // TODO: Add your control notification handler code here
    if(IsDlgButtonChecked(IDC_TRIGVAR))
    {
        m_TrigType = VARIANCE;
        GetDlgItem(IDC_SQUELCHDELAY)->EnableWindow(FALSE);
        GetDlgItem(IDC_SQUELCHTRIG)->EnableWindow(FALSE);
        GetDlgItem(IDC_SQUELCHWIDTH)->EnableWindow(FALSE);
        GetDlgItem(IDC_WINDOWSIZE)->EnableWindow(TRUE);
        GetDlgItem(IDC_VARIANCEPAIRS)->EnableWindow(TRUE);
        GetDlgItem(IDC_THRESHOLD)->EnableWindow(TRUE);
        GetDlgItem(IDC_TRIGWINSIZE)->EnableWindow(FALSE);
        if(m_nTrigScheme==1)
            GetDlgItem(IDC_ADJUST)->EnableWindow(TRUE);
        else
            GetDlgItem(IDC_ADJUST)->EnableWindow(FALSE);
    }


}

void CNewDatabase::OnTrigWinVar()
{
    // TODO: Add your control notification handler code here
    if(IsDlgButtonChecked(IDC_TRIGWINVAR))
    {
        m_TrigType = WINDOWEDVARIANCE;
        GetDlgItem(IDC_SQUELCHDELAY)->EnableWindow(TRUE);
        GetDlgItem(IDC_SQUELCHTRIG)->EnableWindow(TRUE);
        GetDlgItem(IDC_SQUELCHWIDTH)->EnableWindow(TRUE);
        GetDlgItem(IDC_WINDOWSIZE)->EnableWindow(TRUE);
        GetDlgItem(IDC_VARIANCEPAIRS)->EnableWindow(TRUE);
        GetDlgItem(IDC_THRESHOLD)->EnableWindow(TRUE);
        GetDlgItem(IDC_TRIGWINSIZE)->EnableWindow(TRUE);
        GetDlgItem(IDC_ADJUST)->EnableWindow(TRUE);
    }

}
```

Pnn.h

```
//CPNN: contains all PNN processing routines

#include "TrainingResults.h"
#include "TransientArray.h"

class CPNN:public CObject
{
    DECLARE_SERIAL(CPNN)

protected:
    int m_nTransientModelSize;//init in constructor
    int m_nNumCases;
    int m_nNumClasses;
    double *m_Sigma;
    BOOL m_bSigmasInitialized;
    BOOL m_bSigmasOptimized;
    double m_dBestError;
    double m_LowerSearchLimit;
    double m_UpperSearchLimit;
    CTransientArray *m_TransientArray;

public:
    BOOL m_bStopNow; //these 4 vars must be accessable by
    double m_nUserDisplaySigma;//the dialog class that updates the user
    double m_dUserDisplayError;
    int m_nUserDisplayDiscreteError;
    double m_dUserDisplayImprovement;

    CPNN() { m_Sigma=new double[1]; } //in case never init
    ~CPNN() { delete [] m_Sigma; }
    void Initialize(int TransientModelSize);
    void Serialize(CArchive& ar);
    int Classify(double *Unknown, double *SummationNeuron, int Skip,
                 double RejectThreshold, double *Activation);
    BOOL TrainSigmasInit(int NumSearchPoints, BOOL GlobalDone,
        double *LowerSigma, double *LowerError, double *MiddleSigma,
        double *MiddleError, double *UpperSigma, double *UpperError);
    void SetNumCases(int NumCases) { m_nNumCases=NumCases; }
    void SetNumClasses(int NumClasses) { m_nNumClasses=NumClasses; }
    BOOL GetSigmasInitialized() { return m_bSigmasInitialized; }
    BOOL GetSigmasOptimized() { return m_bSigmasOptimized; }
    int GetNumClasses() { return m_nNumClasses; }
    double GetFirstSigma() { return m_Sigma[0]; }
    void SetLowerSearchLimit(double LowerSearchLimit)
        { m_LowerSearchLimit=LowerSearchLimit; }
    void SetUpperSearchLimit(double UpperSearchLimit)
        { m_UpperSearchLimit=UpperSearchLimit; }
    BOOL GlobMin(int NumSearchPoints, double *LowerSigma, double *LowerError,
        double *MiddleSigma, double *MiddleError, double *UpperSigma,
        double *UpperError);
    double FindError(double Sigma, int *DiscreteError);
    void SetTransientArrayPointer(CTransientArray *TransientArray)
        { m_TransientArray=TransientArray; }
    void SetSigmasInitialized(BOOL Init) { m_bSigmasInitialized=Init; }
    void SetSigmasOptimized(BOOL Opt) { m_bSigmasOptimized=Opt; }
    CString m_strUserMessage;
    void GoldMin(double *LowerSigma, double *LowerError, double *MiddleSigma,
        double *MiddleError, double *UpperSigma, double *UpperError);
    int FindDerivs(double *Unknown, double *SummationNeuron,
        int Skip, double *Deriv, double *Deriv2);
    double CumulateErrorAndDerivs(double *Sigma, double *Deriv,
```

```
        double *Deriv2, BOOL CumulateDerivs, int *DiscreteError);
    BOOL ConjGradientsMin(double ConvergenceTolerance);
    double FindGamma(double *g, double *Grad);
    void FindNewDir(double Gamma, double *g, double *h, double *Grad);
    double UniVarError(double Point, double *Sigma,double *Base, double *Deriv);
    BOOL CGGlobMin(int NumSearchPoints, double *LowerPoint,double *LowerError,
                   double *MiddlePoint, double *MiddleError, double *UpperPoint,
                   double *UpperError, double *Sigma, double *Base,
                   double *Direc);
    double BrentMin(int ItMax, double *LowerPoint, double *MiddlePoint,
                        double *UpperPoint, double MiddleError, double *Sigma,
                        double *Base, double *Direc);
    BOOL TrainSigmasOpt(double Tolerance);
    void FillTrainingResultsArray(CResultsArray *ResultsArray);

#ifdef _DEBUG
    void Dump(CDumpContext& dc) const;
#endif
};
```

Pnn.cpp

```
#include "StdAfx.h"
#include "PNN.h"
#include <math.h>

IMPLEMENT_SERIAL(CPNN, CObject,0)

#ifdef _DEBUG
void CPNN::Dump(CDumpContext& dc) const
{
	CObject::Dump(dc);
	dc << "Hi Don... CPNN Dump.";
}
#endif

void CPNN::Initialize(int TransientModelSize)
{
	m_nTransientModelSize=TransientModelSize;

	delete [] m_Sigma;//had to do dummy init in constructor...undo it now
	m_Sigma=new double[m_nTransientModelSize];
	m_Sigma[0]=1.0;
	m_dBestError=-999.0;
	m_dUserDisplayError=m_dBestError;
	m_nUserDisplayDiscreteError=-999;
	m_bSigmasInitialized=FALSE;
	m_bSigmasOptimized=FALSE;
}

void CPNN::Serialize(CArchive& ar)
{
	int ctr;

	if(ar.IsStoring()) {
		ar << m_bSigmasInitialized << m_bSigmasOptimized << m_dBestError
			<< m_nTransientModelSize << m_nNumCases << m_nNumClasses;
		for (ctr=0; ctr<m_nTransientModelSize; ctr++) {
			ar << m_Sigma[ctr];
		}
	}
	else {
		ar >> m_bSigmasInitialized >> m_bSigmasOptimized >> m_dBestError
			>> m_nTransientModelSize >> m_nNumCases >> m_nNumClasses;

		delete [] m_Sigma;//had to do dummy init in constructor...undo it now
		m_dUserDisplayError=m_dBestError;
		m_Sigma=new double[m_nTransientModelSize];
		for (ctr=0; ctr<m_nTransientModelSize; ctr++) {
			ar >> m_Sigma[ctr];
		}
	}
}

int CPNN::Classify(double *Unknown, double *SummationNeuron, int Skip,
				double RejectThreshold, double *Activation)
{
	//make sure m_nNumClasses and m_nNumCases are set prior to calling
	int TransientClass, TrgCase, UnknownClass, *NoSamplesInClass;
	double *TrgSetTransient, Distance, Diff, Best, psum;

	NoSamplesInClass=new int[m_nNumClasses];
```

```
    for (int ctr=0; ctr<m_nNumClasses; ctr++) {
        SummationNeuron[ctr]=0.0;// will sum kernels here
        NoSamplesInClass[ctr]=0;
    }

    for (TrgCase=0; TrgCase<m_nNumCases; TrgCase++) {
        if (TrgCase!=Skip) {//for training purposes, don't compare same transients
                                //skip will be -1 in classify mode...nothing skipped
            TrgSetTransient=m_TransientArray->
                GetAt(TrgCase)->GetTransientModel();
            TransientClass=m_TransientArray->
                GetAt(TrgCase)->GetTransientClass();

            Distance=0.0;                   // Will sum distance here
            for (ctr=0; ctr<m_nTransientModelSize; ctr++) {
                Diff = Unknown[ctr]-TrgSetTransient[ctr];
                Diff /= m_Sigma[ctr];// Scale per sigma
                Distance+=Diff*Diff;// Cumulate Euclidean distance
            }
            SummationNeuron[TransientClass]+=exp(-Distance);//Gaussian Kernal
            //SummationNeuron[TransientClass]+=1.0/(1.0+Distance*Distance);
            NoSamplesInClass[TransientClass]+=1;
        }
    }

    //decide if we want to account for prior probabilities in this loop
    psum=0.0;
    for(ctr=0; ctr<m_nNumClasses; ctr++) {
        if(NoSamplesInClass[ctr]!=0)
            SummationNeuron[ctr]/=NoSamplesInClass[ctr];//account for unequal trg
                                                        //sample repreesentation
        psum+=SummationNeuron[ctr];
    }

    Best=0.0;                       // Keep track of max across pops
    UnknownClass=-1;    //if all summation neurons equal 0, then give impossible answer
    for (ctr=0 ;ctr<m_nNumClasses; ctr++) {
        if (SummationNeuron[ctr]>Best) {       // find the highest activation
            Best=SummationNeuron[ctr] ;
            UnknownClass=ctr;
        }
    }

    *Activation=Best;//for multimodal classification

    if (Skip==-1&&Best<RejectThreshold)
        UnknownClass=-1;//reject cuz activation too low

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

    for (ctr=0; ctr<m_nNumClasses; ctr++) {
        SummationNeuron[ctr]/=psum;//normalize
    }

    delete [] NoSamplesInClass;
    return UnknownClass;
}

BOOL CPNN::TrainSigmasInit(int NumSearchPoints, BOOL GlobalDone,
                           double *LowerSigma, double *LowerError,
                           double *MiddleSigma, double *MiddleError,
                           double *UpperSigma, double *UpperError)
{
    BOOL IsNotSuccessful;
```

```
    m_bStopNow=FALSE;

    if(!GlobalDone) {
        m_strUserMessage.Format("In Global Minimization Algorithm.");
        m_nUserDisplaySigma=m_LowerSearchLimit;
        m_dUserDisplayError=m_dBestError;
        m_nUserDisplayDiscreteError=-999;
        IsNotSuccessful=GlobMin(NumSearchPoints, LowerSigma, LowerError,
            MiddleSigma, MiddleError, UpperSigma, UpperError);
        if (IsNotSuccessful)
            return FALSE;//GlobMin Unsuccessful
    }

    m_bSigmasInitialized=TRUE;

    m_strUserMessage.Format("In Golden Section Minimization Algorithm.");
    GoldMin(LowerSigma, LowerError,
        MiddleSigma, MiddleError, UpperSigma, UpperError);

    return TRUE;
}

BOOL CPNN::GlobMin(int NumSearchPoints, double *LowerSigma, double *LowerError,
                   double *MiddleSigma, double *MiddleError, double *UpperSigma,
                   double *UpperError)
{
    int ctr, ibest, CurrentDiscreteError, PreviousDiscreteError,
        LowerDiscreteError, MiddleDiscreteError, UpperDiscreteError;
    double CurrentSigma, CurrentError, Rate, Previous=0.0;
    BOOL Turned_Up=FALSE, UserQuit=FALSE;

    Rate=exp(log(m_UpperSearchLimit/m_LowerSearchLimit)/(NumSearchPoints-1));

    CurrentSigma=m_LowerSearchLimit;

    ibest = -1 ;    // For proper escape if error reaches 0

    for (ctr=0; ctr<NumSearchPoints; ctr++) {
        CurrentError=FindError(CurrentSigma, &CurrentDiscreteError);

        if ((ctr==0)||(CurrentError<*MiddleError)) {  // Keep track of best here
            ibest=ctr ;
            *MiddleSigma=CurrentSigma;
            *MiddleError=CurrentError;
            MiddleDiscreteError=CurrentDiscreteError;
            *LowerError=Previous;// Function value to its left
            LowerDiscreteError=PreviousDiscreteError;
            Turned_Up=FALSE;// Flag that min is not yet bounded
        }
        else if (ctr==(ibest+1)) {// Didn't improve so this point may
            *UpperError=CurrentError;// be the right neighbor of the best
            UpperDiscreteError=CurrentDiscreteError;
            Turned_Up=TRUE;   // Flag that min is bounded
        }

        Previous=CurrentError;// Keep track for left neighbor of best
        PreviousDiscreteError=CurrentDiscreteError;

        if ((MiddleDiscreteError==0)&&(ibest>0)&&Turned_Up)
            break; // Done if good enough and both neighbors found
        if (m_bStopNow) {
            UserQuit=TRUE;
            break;
        }
```

```
        CurrentSigma*=Rate;
    }

    *LowerSigma=*MiddleSigma/Rate;
    *UpperSigma=*MiddleSigma*Rate;

    if (!Turned_Up) {// Must extend to the right (larger x)
        for (;;) {   // Endless loop goes as long as necessary
            *UpperError=FindError(*UpperSigma, &UpperDiscreteError);

            if (*UpperError>*MiddleError)  // If function increased we are done
                break;
            if ((*LowerError==*MiddleError)&&(*MiddleError==*UpperError))
                break;                          // Give up if flat
            if (m_bStopNow) {
                UserQuit=TRUE;
                break;
            }
            *LowerSigma=*MiddleSigma;// Shift all points
            *LowerError=*MiddleError;
            LowerDiscreteError=MiddleDiscreteError;
            *MiddleSigma=*UpperSigma;
            *MiddleError=*UpperError;
            MiddleDiscreteError=UpperDiscreteError;
            Rate*=3.0;                          // Step further each time
            *UpperSigma*=Rate;
        }
    }
    else if(ibest==0) {// Must extend to the left (smaller x)
        for (;;) {   // Endless loop goes as long as necessary
            *LowerError=FindError(*LowerSigma, &LowerDiscreteError);

            if (*LowerError>*MiddleError)// If function increased we are done
                break;
            if ((*LowerError==*MiddleError)&&(*MiddleError==*UpperError))
                break;                          // Give up if flat
            if (m_bStopNow) {
                UserQuit=TRUE;
                break;
            }
            *UpperSigma=*MiddleSigma;// Shift all points
            *UpperError=*MiddleError;
            UpperDiscreteError=MiddleDiscreteError;
            *MiddleSigma=*LowerSigma;
            *MiddleError=*LowerError;
            MiddleDiscreteError=LowerDiscreteError;

            Rate*=3.0;                          // Step further each time
            *LowerSigma/=Rate;
        }
    }
    m_LowerSearchLimit=*LowerSigma;
    m_UpperSearchLimit=*UpperSigma;
    m_dBestError=*MiddleError;
    m_dUserDisplayError=*MiddleError;//for user update on timer 1
    m_nUserDisplayDiscreteError=MiddleDiscreteError;

    for(ctr=0; ctr<m_nTransientModelSize; ctr++) {
        m_Sigma[ctr]=*MiddleSigma;
    }
    return UserQuit;
}

double CPNN::FindError(double Sigma, int *DiscreteError)
```

```
{
    MSG message;
    int PredictedClass, CorrectClass, ctr1;
    double *CurrentTransient, *SummationNeuron, TotalError=0.0, Diff, Error, Act;
    *DiscreteError=0;

    SummationNeuron=new double[m_nNumClasses];
    for(int ctr=0; ctr<m_nTransientModelSize; ctr++) {
        m_Sigma[ctr]=Sigma;
    }

    for(ctr=0; ctr<m_nNumCases; ctr++) {
        CurrentTransient=m_TransientArray->
            GetAt(ctr)->GetTransientModel();
        PredictedClass=Classify(CurrentTransient, SummationNeuron, ctr,NULL,&Act);
        CorrectClass=m_TransientArray->GetAt(ctr)->GetTransientClass();
        if (PredictedClass!=CorrectClass) {
            *DiscreteError+=1;
        }
        Error=0.0;
        for (ctr1=0; ctr1<m_nNumClasses; ctr1++) {
            if (ctr1==CorrectClass) {
                Diff=1.0-SummationNeuron[ctr1];
                Error+=Diff*Diff;
            }
            else {
                Error+=SummationNeuron[ctr1]*SummationNeuron[ctr1];
            }
        }
        TotalError+=Error;
        if (::PeekMessage(&message, NULL, 0, 0, PM_REMOVE)) {
            ::TranslateMessage(&message);
            ::DispatchMessage(&message);
        }
        AfxGetApp()->DoWaitCursor(1);
    }
    TotalError/=m_nNumCases;

    m_nUserDisplayDiscreteError=*DiscreteError;//for user output on timer 1
    m_nUserDisplaySigma=Sigma;
    m_dUserDisplayError=TotalError;

    delete [] SummationNeuron;
    return TotalError;
}

void CPNN::GoldMin(double *LowerSigma, double *LowerError, double *MiddleSigma,
        double *MiddleError, double *UpperSigma, double *UpperError)
{
    int CurrentDiscreteError, MiddleDiscreteError;
    double LeftWidth, RightWidth, CurrentSigma, CurrentError;
    double Gold=2.0/(1.0+sqrt(5.0));//about 0.618 or (1.0 - 0.38197)

    for (;;) {    // Endless loop goes as long as necessary
        if (m_bStopNow||*MiddleSigma==0) {
            break;
        }
        if ((*UpperSigma-*LowerSigma)<(3.e-8**MiddleSigma)) {
            break;      //avoid refining beyond double precision
        }

        LeftWidth=*MiddleSigma/ *LowerSigma;
        RightWidth=*UpperSigma/ *MiddleSigma;

        if (LeftWidth>RightWidth) {//left interval larger so split it
```

```
                CurrentSigma=exp(log(*LowerSigma)+Gold*log(*MiddleSigma/ *LowerSigma));
                CurrentError=FindError(CurrentSigma, &CurrentDiscreteError);
                if ((CurrentError<*MiddleError)||((CurrentError==*MiddleError)
                    &&(*LowerError<*UpperError))) {//if we improved, or tied with left
                                                    //side favored, discard right point
                    *UpperSigma=*MiddleSigma;
                    *UpperError=*MiddleError;
                    *MiddleSigma=CurrentSigma;
                    *MiddleError=CurrentError;
                    MiddleDiscreteError=CurrentDiscreteError;
                }
                else {      //didn't improve so discard left point
                    *LowerSigma=CurrentSigma;
                    *LowerError=CurrentError;
                }
            }

            else {
                CurrentSigma=exp(log(*UpperSigma)+Gold*log(*MiddleSigma/ *UpperSigma));
                CurrentError=FindError(CurrentSigma, &CurrentDiscreteError);
                if ((CurrentError<*MiddleError)||((CurrentError==*MiddleError)
                    &&(*LowerError>*UpperError))) {//if we improved, or tied with right
                                                    //side favored, discard left point
                    *LowerSigma=*MiddleSigma;
                    *LowerError=*MiddleError;
                    *MiddleSigma=CurrentSigma;
                    *MiddleError=CurrentError;
                    MiddleDiscreteError=CurrentDiscreteError;
                }
                else {      //didn't improve so discard right point
                    *UpperSigma=CurrentSigma;
                    *UpperError=CurrentError;
                }
            }
        }
        m_dBestError=*MiddleError;
        m_dUserDisplayError=*MiddleError;//for user update on timer 1
        m_nUserDisplayDiscreteError=MiddleDiscreteError;
}

int CPNN::FindDerivs(double *Unknown, double *SummationNeuron,
                     int Skip, double *Deriv, double *Deriv2)
{
        double *VPtr, *WPtr, VSum, WSum;
        double Temp, Der1, Der2, psum;

        int TransientClass, TrgCase, UnknownClass, *NoSamplesInClass;
        double *TrgSetTransient, *v, *w, *DiffSqr, Distance, TrueDist, Diff, Best;

        v=new double[m_nTransientModelSize*m_nNumClasses];
        w=new double[m_nTransientModelSize*m_nNumClasses];
        DiffSqr=new double[m_nTransientModelSize];
        NoSamplesInClass=new int[m_nNumClasses];

        for (int ctr=0; ctr<m_nNumClasses; ctr++) {
            SummationNeuron[ctr]=0.0;// will sum kernels here
            NoSamplesInClass[ctr]=0;
            for (int ivar=0; ivar<m_nTransientModelSize; ivar++) {
                v[ctr*m_nTransientModelSize+ivar]=0.0;//Scratch for deriv calc
                w[ctr*m_nTransientModelSize+ivar]=0.0;//Ditto
            }
        }

        for (TrgCase=0; TrgCase<m_nNumCases; TrgCase++) {
            if (TrgCase!=Skip) {//for trg purposes, don't compare same transients
```

```
                                        //skip will be -1 in classify mode...nothing skipped
                TrgSetTransient=m_TransientArray->
                    GetAt(TrgCase)->GetTransientModel();
                TransientClass=m_TransientArray->
                    GetAt(TrgCase)->GetTransientClass();

                Distance=0.0;                   // Will sum distance here
                for (ctr=0; ctr<m_nTransientModelSize; ctr++) {
                    Diff=Unknown[ctr]-TrgSetTransient[ctr];
                    Diff/=m_Sigma[ctr];  // Scale per sigma
                    DiffSqr[ctr]=Diff*Diff;// Squared weighted distance
                    Distance+=DiffSqr[ctr];// Cumulate for all vars
                }
                Distance=exp(-Distance);
                //Distance=1.0/(1.0+Distance*Distance);
                TrueDist=Distance;
//              if(Distance<1.e-40)
//                  Distance=1.e-40;

                SummationNeuron[TransientClass]+=Distance;//Gaussian Kernal
                NoSamplesInClass[TransientClass]+=1;

                VPtr=v+TransientClass*m_nTransientModelSize;//Point to this row in v
                WPtr=w+TransientClass*m_nTransientModelSize;//And w
                for (ctr=0; ctr<m_nTransientModelSize; ctr++) {
                    Temp=TrueDist*DiffSqr[ctr];
                    VPtr[ctr]+=Temp;
                    WPtr[ctr]+=Temp*(2.0*DiffSqr[ctr]-3.0);
                }
            }
        }

        //decide if we want to account for prior probabilities in this loop
        psum=0.0;
        for(ctr=0; ctr<m_nNumClasses; ctr++) {
            if(NoSamplesInClass[ctr]!=0)
                SummationNeuron[ctr]/=NoSamplesInClass[ctr];//account for unequal trg
                                                                        //sample repreesentation
            psum+=SummationNeuron[ctr];
        }

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

        for (ctr=0; ctr<m_nNumClasses; ctr++) {
            SummationNeuron[ctr]/=psum;
        }

        //Compute the derivatives. VSum and WSum are the simple sums of v and w.

        for (ctr=0; ctr<m_nTransientModelSize; ctr++) {
            VSum=WSum=0.0;

            for(int OutVar=0; OutVar<m_nNumClasses; OutVar++) {//Cumulate VSum and WSum
                v[OutVar*m_nTransientModelSize+ctr]*=
                    2.0/(psum*m_Sigma[ctr]*NoSamplesInClass[OutVar]);
                w[OutVar*m_nTransientModelSize+ctr]*=
                    2.0/(psum*m_Sigma[ctr]*m_Sigma[ctr]*NoSamplesInClass[OutVar]);
                VSum+=v[OutVar*m_nTransientModelSize+ctr];
          WSum+=w[OutVar*m_nTransientModelSize+ctr];
            }

            for (OutVar=0; OutVar<m_nNumClasses; OutVar++) {
                Der1=v[OutVar*m_nTransientModelSize+ctr]
                    -SummationNeuron[OutVar]*VSum;
```

```
                Der2=w[OutVar*m_nTransientModelSize+ctr]
                    +2.0*SummationNeuron[OutVar]*VSum*VSum
                    -2.0*v[OutVar*m_nTransientModelSize+ctr]*VSum
                    -SummationNeuron[OutVar]*WSum;

                if (OutVar==m_TransientArray->GetAt(Skip)->GetTransientClass()) {
                    Temp=2.0*(SummationNeuron[OutVar]-1.0);
                }
                else {
                    Temp=2.0*SummationNeuron[OutVar];
                }

                Deriv[ctr]+=Temp*Der1;
                Deriv2[ctr]+=Temp*Der2+2.0*Der1*Der1;
            }
        }

        Best=0.0;                       // Keep track of max across pops
        UnknownClass=-1;   //if all summation neurons equal 0, then give impossible answer
        for (ctr=0 ;ctr<m_nNumClasses; ctr++) {
            if (SummationNeuron[ctr]>Best) {       // find the highest activation
                Best=SummationNeuron[ctr] ;
                UnknownClass=ctr;
            }
        }

        delete [] v;
        delete [] w;
        delete [] DiffSqr;
        delete [] NoSamplesInClass;

        return UnknownClass;
}


double CPNN::CumulateErrorAndDerivs(double *Sigma, double *Deriv,
                                    double *Deriv2, BOOL CumulateDerivs,
                                    int *DiscreteError)
{
        MSG message;
        int PredictedClass, CorrectClass, ctr1;
        double *CurrentTransient, *SummationNeuron, TotalError=0.0, Diff, Error,Act;
        *DiscreteError=0,

        SummationNeuron=new double[m_nNumClasses];
        for(int ctr=0; ctr<m_nTransientModelSize; ctr++) {
            m_Sigma[ctr]=Sigma[ctr];
        }

        if(CumulateDerivs) {
            for(ctr=0; ctr<m_nTransientModelSize; ctr++) {
                Deriv[ctr]=0.0;
                Deriv2[ctr]=0.0;
            }
        }

        for(ctr=0; ctr<m_nNumCases; ctr++) {
            CurrentTransient=m_TransientArray->
                GetAt(ctr)->GetTransientModel();
            if(CumulateDerivs) {
                PredictedClass=FindDerivs(CurrentTransient, SummationNeuron, ctr,
                Deriv, Deriv2);
            }
            else {
                PredictedClass=Classify(CurrentTransient,SummationNeuron,ctr,NULL,&Act);
```

```
            }
            CorrectClass=m_TransientArray->GetAt(ctr)->GetTransientClass();
            if (PredictedClass!=CorrectClass) {
                *DiscreteError+=1;
            }
            Error=0.0;
            for (ctr1=0; ctr1<m_nNumClasses; ctr1++) {
                if (ctr1==CorrectClass) {
                    Diff=1.0-SummationNeuron[ctr1];
                    Error+=Diff*Diff;
                }
                else {
                    Error+=SummationNeuron[ctr1]*SummationNeuron[ctr1];
                }
            }
            TotalError+=Error;
            if (::PeekMessage(&message, NULL, 0, 0, PM_REMOVE)) {
                ::TranslateMessage(&message);
                ::DispatchMessage(&message);
            }
        }

        TotalError/=m_nNumCases;
        if(CumulateDerivs) {
            for (ctr=0; ctr<m_nTransientModelSize; ctr++) {
                Deriv[ctr]/=m_nNumCases;
                Deriv2[ctr]/=m_nNumCases;
            }
        }
/*      m_nUserDisplayDiscreteError=*DiscreteError;//for user output on timer 1
        m_dUserDisplayError=TotalError;
*/
        delete [] SummationNeuron;

        return TotalError;
}

BOOL CPNN::ConjGradientsMin(double ImprovementTolerance)
{
        int ctr, ctr1, ItMax=32767, ConvergenceCtr, PoorCJCtr;
        double CurrentBest, CurrentValue, PreviousBest, Tolerance, Improvement;
        double Dot1, Dot2, DLen, High, Scale, Gamma;
        double LowerPoint, MiddlePoint, UpperPoint,
                LowerError, MiddleError, UpperError;
        double *Sigma, *Base, *Deriv, *Direc, *g, *h, *Deriv2;
        double ConvergenceTolerance=0.0000000001;
        int CurrentDiscreteError;
        BOOL UserQuit, ImprovementToleranceReached=FALSE;

        m_dUserDisplayError=m_dBestError;
        m_nUserDisplayDiscreteError=-999;
        m_dUserDisplayImprovement=0.0;
        Sigma=new double[m_nTransientModelSize];
        for(ctr=0; ctr<m_nTransientModelSize; ctr++) {
                Sigma[ctr]=m_Sigma[ctr];
        }
        Base=new double[m_nTransientModelSize];
        Deriv=new double[m_nTransientModelSize];
        Direc=new double[m_nTransientModelSize];
        g=new double[m_nTransientModelSize];
        h=new double[m_nTransientModelSize];
        Deriv2=new double[m_nTransientModelSize];

    CurrentBest=CumulateErrorAndDerivs(Sigma, Deriv, Deriv2, TRUE,
                &CurrentDiscreteError);
```

```
m_dUserDisplayError=CurrentBest;//should already equal m_dBestError
m_nUserDisplayDiscreteError=CurrentDiscreteError;

PreviousBest=1.e30;
for (ctr=0; ctr<m_nTransientModelSize; ctr++) {
      Direc[ctr]=-Deriv[ctr];
}
memcpy(g, Direc, m_nTransientModelSize*sizeof(double));
memcpy(h, Direc, m_nTransientModelSize*sizeof(double));

ConvergenceCtr=0;
PoorCJCtr=0;

for(ctr=0; ctr<ItMax; ctr++) {
      if (PreviousBest <= 1.0)// If the function is small
            Tolerance=ConvergenceTolerance;// Work on absolutes
      else                                          // But if it is large
            Tolerance=ConvergenceTolerance*PreviousBest;// Keep things relative

      if((PreviousBest-CurrentBest)<=Tolerance) {// If little improvement
            if(++ConvergenceCtr>=3)  // Then count how many
            m_dBestError=CurrentBest;
            break;                                          // And quit if too many
      }
      else                                                  // But a good iteration
            ConvergenceCtr=0;               // Resets this counter

      if (m_bStopNow) {
            m_dBestError=CurrentBest;
            break;//user has quit
      }
      Dot1=Dot2=DLen=0.0;// For finding directional derivs
      High=1.e-4;                       // For scaling glob_min
      for(ctr1=0; ctr1<m_nTransientModelSize; ctr1++) {
            Base[ctr1]=Sigma[ctr1];// We step out from here
            if(Deriv2[ctr1]>High)// Keep track of second derivatives
                  High=Deriv2[ctr1];// For linear search via glob_min
            Dot1+=Direc[ctr1]*g[ctr1];// Directional first derivative
            Dot2+=Direc[ctr1]*Direc[ctr1]*Deriv2[ctr1]; // and second
            DLen+=Direc[ctr1]*Direc[ctr1];// Length of search vector
      }

      DLen=sqrt(DLen);          // Actual length

      Scale=Dot1/Dot2;// Newton's ideal but unstable scale
      High=1.5/High;        // Less ideal but more stable heuristic
      if(High<1.e-4)        // Subjectively keep it realistic
            High=1.e-4;

      if(Scale<0.0)         // This is truly pathological
            Scale=High;             // So stick with old reliable
      else if(Scale<0.1*High)// Bound the Newton scale
            Scale=0.1*High;// To be close to the stable scale
      else if(Scale>10.0*High)// Bound it both above and below
            Scale=10.0*High;

      m_LowerSearchLimit=0.0;
      m_UpperSearchLimit=2.0*Scale;
      MiddleError=PreviousBest=CurrentBest;

      UserQuit=CGGlobMin(-3, &LowerPoint, &LowerError, &MiddlePoint, &MiddleError,
            &UpperPoint, &UpperError, Sigma, Base, Direc);

      if(UserQuit) {
            if(MiddleError<CurrentBest) {// If global caused improvement
```

```
            for(ctr1=0; ctr1<m_nTransientModelSize; ctr1++) {
                Sigma[ctr1]=Base[ctr1]+MiddlePoint*Direc[ctr1];
                if(Sigma[ctr1]<1.e-10)// Limit it away from zero
                    Sigma[ctr1]=1.e-10;// Fairly arbitrary constant
            }
            CurrentBest=MiddleError;
        }
        else {                                          // Else revert to starting point
            for(ctr1=0; ctr1<m_nTransientModelSize; ctr1++)
                Sigma[ctr1]=Base[ctr1];
        }
        m_dBestError=CurrentBest;
        break;      // user has quit
    }

    if(ConvergenceCtr)
        CurrentBest=BrentMin(20, &LowerPoint, &MiddlePoint, &UpperPoint,
        MiddleError, Sigma, Base, Direc);
    else
        CurrentBest=BrentMin(10, &LowerPoint, &MiddlePoint, &UpperPoint,
        MiddleError, Sigma, Base, Direc);

    for(ctr1=0; ctr1<m_nTransientModelSize; ctr1++) {
        Sigma[ctr1]=Base[ctr1]+MiddlePoint*Direc[ctr1];
        if(Sigma[ctr1]<1.e-10)// Limit it away from zero
            Sigma[ctr1]=1.e-10;// Fairly arbitrary constant
    }

    if(CurrentBest<0.0) {// If user quit during BrentMin
        m_dBestError=-CurrentBest;
        break;      //user has quit
    }

    Improvement=(PreviousBest-CurrentBest)/PreviousBest;
    m_dBestError=CurrentBest;
    CurrentValue=CumulateErrorAndDerivs(Sigma, Deriv, Deriv2, TRUE,
        &CurrentDiscreteError);//calc derivatives
    ASSERT(CurrentValue==CurrentBest);//make sure this occurs...

    for(ctr1=0; ctr1<m_nTransientModelSize; ctr1++)// Flip sign to get
        Direc[ctr1]=-Deriv[ctr1];   // negative gradient

    m_dUserDisplayError=CurrentBest;
    m_nUserDisplayDiscreteError=CurrentDiscreteError;
    m_dUserDisplayImprovement=Improvement*100.0;
    if ((Improvement*100.0)<=ImprovementTolerance) {
        ImprovementToleranceReached=TRUE;
        break;
    }

    Gamma=FindGamma(g, Direc);
    if(Gamma<0.0)
        Gamma=0.0;
    if(Gamma>10.0)
        Gamma=10.0;

    if(Improvement<0.001)// Count how many times we
        ++PoorCJCtr;    // got poor improvement
    else                                    // in a row
        PoorCJCtr=0;

    if(PoorCJCtr>=2) {// If several times
        if(Gamma>1.0) // limit gamma
            Gamma=1.0;
    }
```

```
            if (PoorCJCtr>=6) {// If too many times
                PoorCJCtr=0;    // set gamma to 0
                Gamma=0.0;          // to use steepest descent (gradient)
            }

            FindNewDir(Gamma, g, h, Direc);// Compute search direction
        }

        delete [] Sigma;
        delete [] Base;
        delete [] Deriv;
        delete [] Direc;
        delete [] g;
        delete [] h;
        delete [] Deriv2;

        return ImprovementToleranceReached;//FALSE if user quit or poor performance
}

double CPNN::FindGamma(double *g, double *Grad)
{
        double Denominator=0.0, Numerator=0.0;

        for (int ctr=0; ctr<m_nTransientModelSize; ctr++) {
            Denominator+=g[ctr]*g[ctr];
            Numerator+=(Grad[ctr]-g[ctr])*Grad[ctr];// Grad is neg gradient
        }

        if (Denominator==0.0)// Should never happen (means gradient is zero!)
            return 0.0;
        else
            return Numerator/Denominator;
}

void CPNN::FindNewDir(double Gamma, double *g, double *h, double *Grad)
{
        for(int ctr=0; ctr<m_nTransientModelSize; ctr++) {
            g[ctr]=Grad[ctr];
            Grad[ctr]=h[ctr]=g[ctr]+Gamma*h[ctr];
        }
}

double CPNN::UniVarError(double Point, double *Sigma,
                                double *Base, double *Direc)
{
        int DummyError;//not used here, but used in CumulateErrorAndDerivs()
        for(int ctr=0; ctr<m_nTransientModelSize; ctr++) {
            Sigma[ctr]=Base[ctr]+Point*Direc[ctr];
            if(Sigma[ctr]<1.e-10) {
                TRACE("Don, Sigma too small.");
                Sigma[ctr]=1.e-10;
            }
        }
        return CumulateErrorAndDerivs(Sigma, (double *)NULL, (double *)NULL,
            FALSE, &DummyError);
}

BOOL CPNN::CGGlobMin(int NumSearchPoints, double *LowerPoint,double *LowerError,
                        double *MiddlePoint, double *MiddleError, double *UpperPoint,
                        double *UpperError, double *Sigma, double *Base,
                        double *Direc)
{
        int ctr, ibest;
        double CurrentPoint, CurrentError, Rate, Previous=0.0;
```

```
BOOL Turned_Up=FALSE, KnowFirstPoint=FALSE, UserQuit=FALSE;

if (NumSearchPoints<0) {
      NumSearchPoints=-NumSearchPoints;
      KnowFirstPoint=TRUE;
}

Rate=(m_UpperSearchLimit-m_LowerSearchLimit)/(NumSearchPoints-1);

CurrentPoint=m_LowerSearchLimit;

ibest=-1 ;    // For proper escape if error reaches 0

for (ctr=0; ctr<NumSearchPoints; ctr++) {
      if(ctr||!KnowFirstPoint) {
            CurrentError=UniVarError(CurrentPoint, Sigma, Base, Direc);
      }
      else {
            CurrentError=*MiddleError;
      }

      if ((ctr==0)||(CurrentError<*MiddleError)) {  // Keep track of best here
            ibest=ctr ;
            *MiddlePoint=CurrentPoint;
            *MiddleError=CurrentError;
            *LowerError=Previous;// Function value to its left
            Turned_Up=FALSE;// Flag that min is not yet bounded
      }
      else if (ctr==(ibest+1)) {// Didn't improve so this point may
            *UpperError=CurrentError;// be the right neighbor of the best
            Turned_Up=TRUE;   // Flag that min is bounded
      }

      Previous=CurrentError;// Keep track for left neighbor of best

      if ((CurrentError==0)&&(ibest>0)&&Turned_Up)
            break; // Done if good enough and both neighbors found
      if (m_bStopNow) {
            UserQuit=TRUE;
            break;
      }

      CurrentPoint+=Rate;
}

*LowerPoint=*MiddlePoint-Rate;
*UpperPoint=*MiddlePoint+Rate;

if (!Turned_Up) {// Must extend to the right (larger x)
      for (;;) {    // Endless loop goes as long as necessary
            *UpperError=UniVarError(*UpperPoint, Sigma, Base, Direc);

            if (*UpperError>*MiddleError)  // If function increased we are done
                  break;
            if ((*LowerError==*MiddleError)&&(*MiddleError==*UpperError))
                  break;                              // Give up if flat
            if (m_bStopNow) {
                  UserQuit=TRUE;
                  break;
            }
            *LowerPoint=*MiddlePoint;// Shift all points
            *LowerError=*MiddleError;
            *MiddlePoint=*UpperPoint;
            *MiddleError=*UpperError;
            Rate*=3.0;                          // Step further each time
```

```
                *UpperPoint+=Rate;
            }
        }
        else if(ibest==0) {// Must extend to the left (smaller x)
            for (;;) {    // Endless loop goes as long as necessary
                *LowerError=UniVarError(*LowerPoint, Sigma, Base, Direc);

                if (*LowerError>*MiddleError)// If function increased we are done
                    break;
                if ((*LowerError==*MiddleError)&&(*MiddleError==*UpperError))
                    break;                              // Give up if flat
                if (m_bStopNow) {
                    UserQuit=TRUE;
                    break;
                }
                *UpperPoint=*MiddlePoint;// Shift all points
                *UpperError=*MiddleError;
                *MiddlePoint=*LowerPoint;
                *MiddleError=*LowerError;

                Rate*=3.0;                          // Step further each time
                *LowerPoint-=Rate;
            }
        }
        m_LowerSearchLimit=*LowerPoint;
        m_UpperSearchLimit=*UpperPoint;

        return UserQuit;
}

double CPNN::BrentMin(int ItMax, double *LowerPoint, double *MiddlePoint,
                      double *UpperPoint, double MiddleError, double *Sigma,
                      double *Base, double *Direc)
{
        int ctr;
        double PreviousDistance=0.0, Step=0.0, Tol=1.e-6, Tol1, Tol2, eps=1.e-7;
        double BestPoint, SecondBestPoint, ThirdBestPoint;
        double BestError, SecondBestError, ThirdBestError;
        double LowPoint, MidPoint, HighPoint, t1, t2;
        double Numerator, Denominator, TestDistance, RecentPoint, RecentError;

        BestPoint=SecondBestPoint=ThirdBestPoint=*MiddlePoint;
        LowPoint=*LowerPoint;
        HighPoint=*UpperPoint;
        BestError=SecondBestError=ThirdBestError=MiddleError;

        for(ctr=0; ctr<ItMax; ctr++) {
            if (m_bStopNow) {
                return -BestError;
            }
            MidPoint=0.5*(LowPoint+HighPoint);
            Tol1=Tol*(fabs(BestPoint)+eps);
            Tol2=2.*Tol1;

            if(fabs(BestPoint-MidPoint)<=(Tol2-0.5*(HighPoint-LowPoint)))
                break;

            if((ctr>=2)&&((ThirdBestError-BestError)<eps))
                break;

            if(fabs(PreviousDistance)>Tol1) {//If moved far enough try parabolic fit
                t1=(BestPoint-SecondBestPoint)*(BestError-ThirdBestError);
                t2=(BestPoint-ThirdBestPoint)*(BestError-SecondBestError);
                Numerator=(BestPoint-ThirdBestPoint)*
                    t2-(BestPoint-SecondBestPoint)*t1;
```

```
        Denominator=2.*(t1-t2);// Estimate will be numer / denom
        TestDistance=PreviousDistance;// Will verify interval is shrinking
        PreviousDistance=Step;// Save for next iteration
        if(Denominator!=0.0)// Avoid dividing by zero
            Step=Numerator/Denominator; // the parabolic estimate to min
        else
            Step=1.e30;     // Assures failure of next test

        if ((fabs(Step)<fabs(0.5*TestDistance))// If shrinking
            &&(Step+BestPoint>LowPoint)// and within known bounds
            &&(Step+BestPoint<HighPoint)) {// then we can use the
            RecentPoint=BestPoint+Step;// parabolic estimate
            if((RecentPoint-LowPoint<Tol2)||// If we are very close
                (HighPoint-RecentPoint<Tol2)) {// to known bounds
                if(BestPoint<MidPoint)// then stabilize
                    Step=Tol1;
                else
                    Step=-Tol1;
            }
        }
        else {      // Parabolic estimate poor, so use golden section
            PreviousDistance=(BestPoint>=MidPoint)?
                LowPoint-BestPoint:HighPoint-BestPoint;
            Step=.3819660*PreviousDistance;
        }
    }
    else {
        PreviousDistance=(BestPoint>=MidPoint)?
            LowPoint-BestPoint:HighPoint-BestPoint;
        Step=.3819660*PreviousDistance;
    }

    if(fabs(Step)>=Tol1)
        RecentPoint=BestPoint+Step;
    else {
        if(Step>0.0)
            RecentPoint=BestPoint+Tol1;
        else
            RecentPoint=BestPoint-Tol1;
    }

    RecentError=UniVarError(RecentPoint, Sigma, Base, Direc);

    if(RecentError<=BestError) {// If we improved...
        if(RecentPoint>=BestPoint)// Shrink the (lowpt,highpt) interval by
            LowPoint=BestPoint;// replacing the appropriate endpoint
        else
            HighPoint=BestPoint;
        ThirdBestPoint=SecondBestPoint;// Update x and f values for best,
        SecondBestPoint=BestPoint;// second and third best
        BestPoint=RecentPoint;
        ThirdBestError=SecondBestError;
        SecondBestError=BestError;
        BestError=RecentError;
    }
    else {                                  // We did not improve
        if(RecentPoint<BestPoint)// Shrink the (xlow,xhigh) interval by
            LowPoint=RecentPoint;// replacing the appropriate endpoint
        else
            HighPoint=RecentPoint;

        if((RecentError<=SecondBestError)// If we beat the second best
            ||(SecondBestPoint==BestPoint)) {  // or we had a duplication
            ThirdBestPoint=SecondBestPoint;// update the second and third
            SecondBestPoint=RecentPoint;// best, though not the best.
```

```
                    ThirdBestError=SecondBestError;
                    SecondBestError=RecentError;
                }
                else if((RecentError<=ThirdBestError)// Maybe at least we can
                    ||(ThirdBestPoint==BestPoint)// beat the third best or get
                    ||(ThirdBestPoint==SecondBestPoint)) { // rid of a duplication
                    ThirdBestPoint=RecentPoint;
                    ThirdBestError=RecentError;
                }
            }
        }
        *LowerPoint=LowPoint;
        *MiddlePoint=BestPoint;
        *UpperPoint=HighPoint;

        return BestError;
}

BOOL CPNN::TrainSigmasOpt(double Tolerance)
{
        BOOL ImprovementToleranceReached=FALSE;
        m_bStopNow=FALSE;

        m_strUserMessage.Format("In Conjugate Gradients Algorithm.");
        ImprovementToleranceReached=ConjGradientsMin(Tolerance);
        m_bSigmasOptimized=TRUE;
        return ImprovementToleranceReached;
}

void CPNN::FillTrainingResultsArray(CResultsArray *ResultsArray)
{
        int PredictedClass, CorrectClass, ctr1;
        double *CurrentTransient, *SummationNeuron, TotalError=0.0, Diff, Error, Act;

        SummationNeuron=new double[m_nNumClasses];

        for(int ctr=0; ctr<m_nNumCases; ctr++) {

                CurrentTransient=m_TransientArray->
                        GetAt(ctr)->GetTransientModel();
                PredictedClass=Classify(CurrentTransient, SummationNeuron, ctr, NULL, &Act);
                CorrectClass=m_TransientArray->GetAt(ctr)->GetTransientClass();
                Error=0.0;
                for (ctr1=0; ctr1<m_nNumClasses; ctr1++) {
                        if (ctr1==CorrectClass) {
                                Diff=1.0-SummationNeuron[ctr1];
                                Error+=Diff*Diff;
                        }
                        else {
                                Error+=SummationNeuron[ctr1]*SummationNeuron[ctr1];
                        }
                }
                CTrainingResults *TrainingResults=new CTrainingResults(1);
                                                                //only 1 Segmentation mode
                TrainingResults->SetRawFilePath(m_TransientArray->
                        GetAt(ctr)->GetRawFilePath());
                TrainingResults->SetCorrectClass(CorrectClass);
                TrainingResults->SetPredictedClass(PredictedClass,0);
                TrainingResults->SetError(Error,0);
                ResultsArray->InsertAt(ctr, TrainingResults);
        }

        delete [] SummationNeuron;
        return;
}
```

Princomp.h

```
//This file is written by the author based on the source code from [Mast95] to implement the PCA technique

//principal component analysis

#include "TransientArray.h"

class CPrincoData:public CObject
{
    DECLARE_SERIAL(CPrincoData)
public:
    CPrincoData();
    //void Initialize(int TransientModelSize, int NumCases, int ReducedDim);
    void PCAProcess();
    void Serialize(CArchive& ar);
    ~CPrincoData();
    inline void rotate(double *mat, int i, int j, int k, int l, double sine,
                       double tau, int n);
    void jacobi( int n,double *mat,double *evals,double *evect,
                 double *work1,double *work2);

    void factors(CTransientArray *TransientArray);
    void PCATransform(double *TransientModel);

    void SetTransientModelSize(int TransientModelSize) { m_nTransientModelSize=TransientModelSize; }
    void SetReducedDim(int ReducedDim) {m_nReducedDim = ReducedDim;}
    void SetNumCases(int NumCases) { m_nNumCases=NumCases;}
    int GetReducedDim() { return m_nReducedDim; }

    void SetTransientArrayPointer(CTransientArray *TransientArray)
        { m_TransientArray=TransientArray; }

protected:
    int m_nNumCases,m_nTransientModelSize,m_nReducedDim;
    int m_nStdize;//standardize inputs to equal variance
    double m_dFrac;//fraction(0-1) of variance to retain
    double *means, *evals, *evect, *work1;
    CTransientArray *m_TransientArray;

#ifdef _DEBUG
    void Dump(CDumpContext& dc) const;
#endif
};
```

Princomp.cpp

```
/* Princomp.cpp : implementation of CPrincoData*/

#include "StdAfx.h"
#include "math.h"
#include "princomp.h"

IMPLEMENT_SERIAL(CPrincoData, CObject,0)

#ifdef _DEBUG
void CPrincoData::Dump(CDumpContext& dc) const
{
      CObject::Dump(dc);
      dc << "Luotao.. CPrinco Dump.";
}
#endif

void CPrincoData::Serialize(CArchive& ar)
{
      if(ar.IsStoring()) {
            ar << m_nNumCases << m_nTransientModelSize
                  << m_nReducedDim << m_nStdize << m_dFrac;
      }
      else {
            ar >> m_nNumCases >> m_nTransientModelSize
                  >> m_nReducedDim >> m_nStdize >> m_dFrac;
      }
}

inline void CPrincoData::rotate(double *mat, int i, int j, int k, int l, double sine,
                        double tau, int n)
{
      double t1, t2;

      t1=mat[i*n+j];
      t2=mat[k*n+l];
      mat[i*n+j] = t1-sine*(t2+t1*tau);
      mat[k*n+l] = t2+sine*(t1-t2*tau);
}

//subroutine "jacobi" computes the eigenstructure of a real symmetric matrix
void CPrincoData::jacobi( int n,//size of matrix
                  double *mat,//square symmetric real matrix
                  double *evals,//output of n eigenvalues
                  double *evect,//output of n by n eigenvectors (each is a column)
                  double *work1,//work vector n long
                  double *work2//work vector n long
                  )
{
      int i,j,k,ibig, sweep;
      double err, *dptr, threshold, test, diff, theta;
      double corr, sine, cosine, tangent, tau, big, mat_ij;

      /*The rotations will be cumulated in the evect matrix, each of whose columns
      will be a normalized eigenvector. Initialize this to the identity. The output
      vector evals will always contain the current diagonal of mat, so initialize it now.
      The correlations to the diagonal as a result of rotation will be maintained in
      work1 and work2, so also initialize them.*/

      for(i=0;i<n;i++) {
            dptr = evect +i*n;//point to row i
            for(j=0;j<n;j++) //set entire row to 0
```

```
            dptr[j] = 0.0;
        dptr[i] = 1.0;              //set diagonal to 1
        evals[i] = work1[i]=mat[i*n+i]; //matrix diagonal
        //TRACE("eval0[%i]=%lf ",i,evals[i]);
        work2[i] = 0.0;
}

/* This is the main loop which does a single sweep through the matrix.
it is rare that more than a dozen sweeps will be needed to zero the
above-diagonal region. However, just for safety, we impose a limit here*/

for(sweep=0;sweep<200;sweep++) {
        /*start by checking for convergence. We simply sum the magnitude of
        the elements above the diagonal. when they become tiny we are done.*/

        err = 0.0;
        for(i=0;i<n-1;i++){
                dptr = mat+i*n;//point to row i
                for(j=i+1;j<n;j++)//above diagonal area
                        err += fabs(dptr[j]);//will be all zero when done
        }
        if(err<1.e-60)    //safety for portability than using zero
                break;                  //although numerically slightly inferior

        if(sweep>4)
                threshold = 0.0;
        else
                threshold = 0.15*err/(n*n);

        for(i=0;i<n-1;i++) {//row i
                for(j=i+1;j<n;j++) {//and column j
                        mat_ij = mat[i*n+j];//this is the elment to be zeroed

                        test = 128.0*fabs(mat_ij);

                        if((sweep>5)&&
                                (fabs(evals[i]) == (fabs(evals[i])+test)) &&
                                (fabs(evals[j]) == (fabs(evals[j])+test))) {
                                mat[i*n+j] = 0.0;
                                continue;
                        }

                        if(fabs(mat_ij)<threshold)
                                continue;
        /*We must do the rotations to zero the (i,j) above-diagonal element.
        Start by computing the rotatin angle, theta, then find our ultimate goal,
        its tangent. If the denominator of the strict formula wold be tiny, use
        an approximation that is essentially perfect in that instance. */

                        diff = evals[j] - evals[i];

                        if(fabs(diff) == (fabs(diff)+test)) {
                                if(diff!=0.0)
                                        tangent = mat_ij/diff;
                                else
                                        tangent = 0.0;
                        }
                        else{
                                theta = 0.5*diff/mat_ij;
                                tangent = 1.0/(fabs(theta)+sqrt(theta*theta+1.0));
                                if(theta<0.0)
                                        tangent = -tangent;
                        }

                        /*do rotations. we break it up into three steps,as this is an efficient
```

```
                    way to handle the requirement that we only do the elements above the diagonal.*/

                    cosine = 1.0/sqrt(tangent*tangent+1.0);
                    sine = cosine*tangent;
                    tau = sine/(cosine+1.0);

                    for(k=0;k<i;k++)//rows above(less than)i
                        rotate(mat, k,i,k,j,sine,tau,n);
                    for(k=i+1;k<j;k++)//rows between i and j
                        rotate(mat,i,k,k,j,sine, tau,n);
                    for(k=j+1;k<n;k++)//rows beyond j
                        rotate(mat,i,k,j,k,sine,tau,n);

                    for(k=0;k<n;k++)
                        rotate(evect,k,i,k,j,sine,tau,n);

                    corr = tangent*mat_ij;
                    //TRACE("corr=%lf ", corr);
                    evals[i] -= corr;
                    //TRACE("evals[%i]=%lf ",i, evals[i]);
                    work2[i] -= corr;
                    evals[j] += corr;
                    work2[j] += corr;
                    mat[i*n+j] = 0.0;
                }//j
            }//i

            //a sweep is completed. update the eigenvalues(diagonal) and work vectors

            for(k=0;k<n;k++) {
                work1[k] += work2[k];
                evals[k] = work1[k];
                //TRACE("evals[%i]=%lf ", k, evals[k]);
                work2[k] = 0.0;
            }
        }//sweep

        /*the final step is to sort the eigenvalues in descending order and simultaneously
        swap the vectors.*/

        for(i=0;i<n;i++) {//end of each pass gets next biggest
            big = evals[i];//keep track of biggest eval there
            ibig = i;           // and its location in array
            for(j=i+1;j<n;j++) { //check for any bigger below
                if(evals[j] >big) {
                    big = evals[j];
                    ibig = j;
                }
            }
            if(ibig ==i)
                continue;
            evals[ibig] = evals[i];
            evals[i] = big;
            for(j=0;j<n;j++) {
                test = evect[j*n+i];

                evect[j*n+i] = evect[j*n+ibig];
                //TRACE("evect[%i]=%lf ",j*n+i,evect[j*n+i]);
                evect[j*n+ibig] = test;
            }
        }
}

CPrincoData::CPrincoData()
{
```

```
    m_nStdize=1;//standarize inputs to equal variance
    m_dFrac=0.9;//retain 90% variance in the reduced transientmodel
}


void CPrincoData::PCAProcess()
{
    int n,icase,var, var2,fac,maxfacs;
    double *dptr,*std, diff1, diff2, sum, temp,*work2, *covar;

    maxfacs = m_nNumCases-1;

    if(m_nStdize) {
        std = new double[m_nTransientModelSize];
    }
    else
        std = NULL;

    means = new double[m_nTransientModelSize];
    evals = new double[m_nTransientModelSize];
    evect = new double[m_nTransientModelSize*m_nTransientModelSize];
    covar = new double[m_nTransientModelSize*m_nTransientModelSize];
    work1 = new double[m_nTransientModelSize];
    work2 = new double[m_nTransientModelSize];

    //compute means and covariance
    for(var=0; var<m_nTransientModelSize; var++)//zero the mean vector
        means[var] = 0.0;

    n = m_nTransientModelSize*m_nTransientModelSize;//zero the covaraince matrix
    while(n--)
        covar[n] = 0.0;

    for(icase=0;icase<m_nNumCases;icase++) { //cumulate means
        dptr = m_TransientArray->
                GetAt(icase)->GetTransientModel();//point to this case
        for(var=0; var<m_nTransientModelSize;var++)//all variables in this case
            means[var] += dptr[var];
    }

    for(var=0;var<m_nTransientModelSize;var++) {//mean is sum divided by n
        means[var] /= m_nNumCases;
        //TRACE("means[%i]=%lf ", var, means[var]);
    }

    for(icase=0;icase<m_nNumCases;icase++){//cumulate covariances
        dptr = m_TransientArray->GetAt(icase)->GetTransientModel();//point to this case
        for(var=0; var<m_nTransientModelSize; var++) {
            diff1 = dptr[var] - means[var];
            for(var2=var;var2<m_nTransientModelSize;var2++) {
                diff2=dptr[var2]-means[var2];
                covar[var*m_nTransientModelSize+var2]+=diff1*diff2;
            }
        }
    }

    for(var=0;var<m_nTransientModelSize;var++) {//divide sums by n
        for(var2=var;var2<m_nTransientModelSize;var2++) {//to get covariance matrix
            covar[var*m_nTransientModelSize+var2] /= m_nNumCases;
            if(var!=var2)           //symmetrically duplicate
                covar[var2*m_nTransientModelSize+var] = covar[var*m_nTransientModelSize+var2];
        }
    }

    /* if we are to standarize, save the standard deviations and adjust
```

```
the covariance matrix*/

if(std!=NULL) {
      for(var=0;var<m_nTransientModelSize;var++) {//each variable's variance
            std[var] = sqrt(covar[var*m_nTransientModelSize+var]);//is in the diagonal
            covar[var*m_nTransientModelSize+var] = 1.0;//which then becomes unity
      }
      for(var=0;var<m_nTransientModelSize-1;var++) {//adjust covaraicnes
            for(var2=var+1;var2<m_nTransientModelSize;var2++) {//for standarization
                  covar[var*m_nTransientModelSize+var2] /= (std[var]*std[var2]);
                  covar[var2*m_nTransientModelSize+var] = covar[var*m_nTransientModelSize+var2];
                  //TRACE("covar[%i]=%lf ",var2*m_nTransientModelSize+var, covar[var2*m_nTransientModelSize+var]);
            }
      }
}

//compute eigenstructure and number of factors to retain

jacobi(m_nTransientModelSize, covar, evals, evect, work1, work2);

/*sum = 0.0;
for(var=0;var<m_nTransientModelSize;var++)
      sum+=evals[var];//sum the eigenvalues
m_dFrac *= sum;      //user wants this much of sum retained

sum = 0.0;
for(int nfacs=0;nfacs<maxfacs;nfacs++){
      sum+=evals[nfacs];//we have retained this much so far
      if(sum>=m_dFrac) {//compare it to user's request
            ++nfacs;          //if there, break out with number of factors
            break;
      }
}
TRACE("Fraction of variance result is %d\n",nfacs);
TRACE("User set it to be %d\n",m_nReducedDim);*/

/*
We now do one or two things to avoid expensive operations in the 'factors' routine.
We must devide each factor by the squre root of the corrspoinding eigenvalue
in order to produce unit standard deviations for the factros.
If the user wants standarazation of the input variables, divide the factor weights
by the standard deviations now so we don't have to do it for each case later.
*/

for(fac=0;fac<m_nReducedDim;fac++) {
      if(evals[fac]>0.0){
            temp = 1.0/sqrt(evals[fac]);

      }
      else
            temp = 0.0;
      for(var=0;var<m_nTransientModelSize;var++)
            evect[var*m_nTransientModelSize+fac] *=temp;

}

if(std!=NULL) {
      for(var=0;var<m_nTransientModelSize;var++) {
            for(fac=0;fac<m_nReducedDim;fac++){
                  //TRACE("evect[%i]=%lf ",var*m_nTransientModelSize+fac,evect[var*m_nTransientModelSize+fac]);
                  evect[var*m_nTransientModelSize+fac] /= std[var];


            }
      }
```

```
    }

    if(std!=NULL)
        delete [] std;
    delete [] covar;
    delete [] work2;//we keep work1 for use by 'factors'
}

/*void CPrincoData::Initialize(int TransientModelSize,
                                int NumCases, int ReducedDim)
{
    m_nTransientModelSize=TransientModelSize;
    m_nNumCases=NumCases;
    m_nReducedDim=ReducedDim;
}*/

//destructor
CPrincoData::~CPrincoData()
{
    delete [] means;
    delete [] evals;
    delete [] evect;
    delete [] work1;
}


void CPrincoData::factors(CTransientArray *TransientArray)
{
    int icase, var, fac;
    double *dptr, sum, diff;
    for(icase=0;icase<m_nNumCases;icase++){//for each case
        //TRACE("\ncase %i:",icase);

        //TRACE("\n"); //will dot evect with case
        dptr = TransientArray->GetAt(icase)->GetTransientModel();//point to this case
        for(fac=0;fac<m_nReducedDim;fac++){//for each factor to be found
            sum = 0.0;

            for(var=0;var<m_nTransientModelSize;var++){//all variables in this case
                diff = dptr[var]-means[var];//must be centered
                sum += evect[var*m_nTransientModelSize+fac] * diff;//cumulate dot product

            }
            work1[fac] = sum;
            //replace original variables with principal factors
            //dptr[fac] = sum;

            //TRACE("%lf ",dptr[fac]);
        }

    }

}

void CPrincoData::PCATransform(double *TransientModel)
{
    int fac, var;
    double sum,diff;
    //TRACE("PCA transform\n");
    for(fac=0;fac<m_nReducedDim;fac++)
    {   //for each factor to be found
        sum = 0.0;

        for(var=0;var<m_nTransientModelSize;var++)
        {           //all variables in this case
```

```
                diff = TransientModel[var]-means[var];//must be centered
                sum += evect[var*m_nTransientModelSize+fac] * diff;//cumulate dot product

            }

            work1[fac] = sum;
            //replace original variables with principal factors
            //TransientModel[fac] = sum;

            TRACE("%lf ",TransientModel[fac]);
        }
    }
```

RawView.h

```
//CRawView: defines and manages the display in the upper viewing window
// RawView.h : interface of the CRawView class
//
/////////////////////////////////////////////////////////////////////////////

class CRawView : public CScrollView
{
private:
	CString GetTitleLine();
	int m_ViewFileSize;
	int m_VarDimHeight;
	int m_RawSignalHeight;
	int m_LeftTextMargin;
	int m_BottomTextMargin;
	int m_TopTextMargin;
	int m_WindowXDim;
	int m_WindowYDim;
	CRect m_rectClient;
	CPoint m_Origin;
	int m_ViewStart;//0 for entire raw signal, TransientBeginsHere for tran
	int m_ZoomYCoord;
	int    m_ClipYCoord;

protected: // create from serialization only
	CRawView();
	DECLARE_DYNCREATE(CRawView)

// Attributes
public:
	CTAC_MMDoc* GetDocument();
	void PrintTransmitterInfo(CDC* pDC);
	void PlotCommonBox(CDC* pDC);
	void PlotVarDimTraj(CDC* pDC);
	void PlotRawSignal(CDC* pDC);
	void AdjustForMargins(CDC* pDC);
	void SetViewFileSize();

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CRawView)
	public:
	virtual void OnDraw(CDC* pDC);  // overridden to draw this view
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	virtual void OnPrepareDC(CDC* pDC, CPrintInfo* pInfo = NULL);
	protected:
	virtual void OnInitialUpdate(); // called first time after construct
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
	virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CRawView();
#ifdef _DEBUG
	virtual void AssertValid() const;
```

```
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CRawView)
    afx_msg void OnViewSegmentation();
    afx_msg void OnUpdateViewSegmentation(CCmdUI* pCmdUI);
    afx_msg void OnViewTransient();
    afx_msg void OnUpdateViewTransient(CCmdUI* pCmdUI);
    afx_msg void OnEditCopy();
    afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in RawView.cpp
inline CTAC_MMDoc* CRawView::GetDocument()
   { return (CTAC_MMDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////
```

RawView.cpp

```
// RawView.cpp : implementation of the CRawView class
//

#include "stdafx.h"
#include "TAC_MM.h"

#include "TAC_MMDoc.h"
#include "RawView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CRawView

IMPLEMENT_DYNCREATE(CRawView, CScrollView)

BEGIN_MESSAGE_MAP(CRawView, CScrollView)
	//{{AFX_MSG_MAP(CRawView)
	ON_COMMAND(ID_VIEW_SEGMENTATION, OnViewSegmentation)
	ON_UPDATE_COMMAND_UI(ID_VIEW_SEGMENTATION, OnUpdateViewSegmentation)
	ON_COMMAND(ID_VIEW_TRANSIENT, OnViewTransient)
	ON_UPDATE_COMMAND_UI(ID_VIEW_TRANSIENT, OnUpdateViewTransient)
	ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
	ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CRawView construction/destruction

CRawView::CRawView()
{

}

CRawView::~CRawView()
{
}

BOOL CRawView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CScrollView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CRawView drawing

void CRawView::OnDraw(CDC* pDC)
{
	CTAC_MMDoc* pDoc = GetDocument();
```

```
    ASSERT_VALID(pDoc);

    if(!pDoc->m_bFileInMemory) {
        return;
    }

    pDC->SetBkMode(TRANSPARENT);

    if (pDoc->m_bNotClassifying) {
        PrintTransmitterInfo(pDC);
    }
    PlotCommonBox(pDC);

    if (pDoc->m_bViewVarDimTrajFlag) {
        PlotVarDimTraj(pDC);
    }

    if (pDoc->m_bViewRawSignalFlag) {
        PlotRawSignal(pDC);
    }
}

void CRawView::OnInitialUpdate()
{
    TRACE("Don, CRawView::OnInitialUpdate being executed.\n");
    CScrollView::OnInitialUpdate();

    CSize sizeTotal;
    // TODO: calculate the total size of this view
    sizeTotal.cx = sizeTotal.cy = 100;
    SetScrollSizes(MM_TEXT, sizeTotal);
}

/////////////////////////////////////////////////////////////////////////////
// CRawView printing

BOOL CRawView::OnPreparePrinting(CPrintInfo* pInfo)
{
    return DoPreparePrinting(pInfo);//displays the Print dialog box and creates
                                    //a printer device context. If you want to
                                    //initialize the Print dialog box with
                                    //values other than the defaults, assign
                                    //values to the members of pInfo before calling.

}

void CRawView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CRawView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CRawView diagnostics

#ifdef _DEBUG
void CRawView::AssertValid() const
{
    CScrollView::AssertValid();
}
```

```
void CRawView::Dump(CDumpContext& dc) const
{
	CScrollView::Dump(dc);
}

CTAC_MMDoc* CRawView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTAC_MMDoc)));
	return (CTAC_MMDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CRawView message handlers

void CRawView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
	CTAC_MMDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	if(!pDoc->m_bFileInMemory)
	{
		Invalidate();
		return;
	}

	m_VarDimHeight=0;
	m_RawSignalHeight=0;

	if (pDoc->m_bViewVarDimTrajFlag)
	{
		m_VarDimHeight=5000;
	}

	if (pDoc->m_bViewRawSignalFlag)
	{
		m_RawSignalHeight=8191;
	}

	Invalidate();
}

void CRawView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
	CScrollView::OnPrepareDC(pDC, pInfo);

	CTAC_MMDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	if(!pDoc->m_bFileInMemory)
	{
		return;
	}

	SetViewFileSize();
	m_LeftTextMargin=(int)(m_ViewFileSize/20.0);
	m_WindowXDim=m_ViewFileSize+m_LeftTextMargin;
	m_BottomTextMargin=(int)((m_RawSignalHeight+m_VarDimHeight)/20.0);
	m_TopTextMargin=(int)((m_RawSignalHeight+m_VarDimHeight)/20.0);
	m_WindowYDim=m_RawSignalHeight+m_BottomTextMargin+
			m_VarDimHeight+m_TopTextMargin;

	CSize DataSize(m_WindowXDim,m_WindowYDim);

	if(!pDC->IsPrinting())
```

```
    {
        GetClientRect(m_rectClient);
        pDC->SetMapMode(MM_ANISOTROPIC);
        pDC->SetWindowExt(DataSize);
        pDC->SetViewportExt(m_rectClient.right, -m_rectClient.bottom);
        m_Origin.x=(long)((double)m_LeftTextMargin/
            (double)m_WindowXDim*(double)m_rectClient.right);
        m_Origin.y=(long) (m_rectClient.bottom-
            (double)m_BottomTextMargin/(double)m_WindowYDim*(double)m_rectClient.bottom);
        pDC->SetViewportOrg(m_Origin);
    }
}


void CRawView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
{
    CTAC_MMDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    if(!pDoc->m_bFileInMemory) {
        return;
    }

    m_rectClient=pInfo->m_rectDraw;
    AdjustForMargins(pDC);

    CPen *pOldPen;
    CPen ThickBlackPen(PS_SOLID, m_WindowXDim/400, RGB(0,0,0));

    pOldPen=pDC->SelectObject(&ThickBlackPen);
    pDC->Rectangle(-m_LeftTextMargin,m_WindowYDim-m_BottomTextMargin,
        m_WindowXDim-m_LeftTextMargin,-m_BottomTextMargin);
    pDC->SelectObject(pOldPen);

    CScrollView::OnPrint(pDC, pInfo);
}

void CRawView::PrintTransmitterInfo(CDC* pDC)
{
    CTAC_MMDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    CString TitleLine=GetTitleLine();
    int TextWidth;
    if(TitleLine.GetLength()>(3*m_WindowXDim/4)) {
        TextWidth=(3*m_WindowXDim/4)/TitleLine.GetLength();
    }
    else {
        TextWidth=0;
    }
    CFont RomanFont;
    RomanFont.CreateFont(m_TopTextMargin, TextWidth, 0, 0, 400, FALSE, FALSE, 0,
                         ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                         CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                         DEFAULT_PITCH | FF_ROMAN, NULL);
    CFont *pOldFont=pDC->SelectObject(&RomanFont);
    pDC->TextOut(0, (m_WindowYDim-m_TopTextMargin),TitleLine);
    CString Time=pDoc -> m_Time;
    CString Date=pDoc -> m_Date;
    CFont HalfRomanFont;
    HalfRomanFont.CreateFont(m_TopTextMargin/2, 0, 0, 0, 400, FALSE, FALSE, 0,
                         ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                         CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                         DEFAULT_PITCH | FF_ROMAN, NULL);
    pDC->SelectObject(&HalfRomanFont);
```

```
        CSize TimeSize=pDC->GetTextExtent(Time);
        int TimeX=m_WindowXDim-TimeSize.cx-m_LeftTextMargin;
        CSize DateSize=pDC->GetTextExtent(Date);
        int DateX=m_WindowXDim-DateSize.cx-m_LeftTextMargin;
        pDC->TextOut(TimeX,(m_WindowYDim-m_TopTextMargin),Time);
        pDC->TextOut(DateX,(m_WindowYDim-m_TopTextMargin*1.5),Date);
        pDC->SelectObject(pOldFont);

        return;
}

void CRawView::PlotCommonBox(CDC* pDC)
{

        CPen *pOldPen;
        CFont *pOldFont;
        CPen ThickBlackPen(PS_SOLID, m_WindowXDim/500, RGB(0,0,0));

        pOldPen=pDC->SelectObject(&ThickBlackPen);
        pDC->MoveTo(m_WindowXDim-m_LeftTextMargin,
                (m_WindowYDim-(m_BottomTextMargin+m_TopTextMargin)));
        pDC->LineTo(0,(m_WindowYDim-(m_BottomTextMargin+m_TopTextMargin)));
        pDC->LineTo(0,-m_BottomTextMargin/5);
        pDC->MoveTo(0,0);
        pDC->LineTo(m_WindowXDim-m_LeftTextMargin,0);
        pDC->MoveTo(m_WindowXDim-m_LeftTextMargin-m_WindowXDim/400,0);
        pDC->LineTo(m_WindowXDim-m_LeftTextMargin-m_WindowXDim/400,-m_BottomTextMargin/5);
        pDC->MoveTo((m_WindowXDim-m_LeftTextMargin)/4,0);
        pDC->LineTo((m_WindowXDim-m_LeftTextMargin)/4,-m_BottomTextMargin/5);
        pDC->MoveTo((m_WindowXDim-m_LeftTextMargin)/2,0);
        pDC->LineTo((m_WindowXDim-m_LeftTextMargin)/2,-m_BottomTextMargin/5);
        pDC->MoveTo(3*(m_WindowXDim-m_LeftTextMargin)/4,0);
        pDC->LineTo(3*(m_WindowXDim-m_LeftTextMargin)/4,-m_BottomTextMargin/5);

        CFont SmallRomanFont;
        SmallRomanFont.CreateFont(m_BottomTextMargin/2, 0, 0, 0, 400, FALSE, FALSE, 0,
                                  ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                                  CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                                  DEFAULT_PITCH | FF_ROMAN, NULL);
        pOldFont=pDC->SelectObject(&SmallRomanFont);
        CString str;
        CSize NumberSize;
        str.Format("%d",0);
        int XShift=m_WindowXDim/500;
        pDC->TextOut(XShift,-m_BottomTextMargin/3,str);
        for(int ctr=m_ViewFileSize/4; ctr<m_ViewFileSize; ctr+=m_ViewFileSize/4) {
                str.Format("%d",ctr);
                NumberSize=pDC->GetTextExtent(str);
                XShift=ctr-NumberSize.cx/2;
                pDC->TextOut(XShift,-m_BottomTextMargin/3,str);
        }
        str.Format("%d",m_ViewFileSize);
        NumberSize=pDC->GetTextExtent(str);
        XShift=m_WindowXDim-m_LeftTextMargin-NumberSize.cx;
        pDC->TextOut(XShift,-m_BottomTextMargin/3,str);

        pDC->SelectObject(pOldFont);
        pDC->SelectObject(pOldPen);
}
void CRawView::PlotVarDimTraj(CDC* pDC)
{
        CTAC_MMDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        CPen *pOldPen;
```

```
CPen CyanSolidPen(PS_SOLID, 1, RGB(0,128,128));
CPen ThickBlackPen(PS_SOLID, m_WindowXDim/500, RGB(0,0,0));

CString YAxisTitle;
if (pDoc->m_bViewSegmentation)
{
    YAxisTitle="Variance Dimension";
}
else
{
    YAxisTitle="Multifractal Features";
}
int DivFactor=2;
if (pDoc->m_bViewRawSignalFlag) {
    DivFactor=3;
}
CFont RomanFont;
RomanFont.CreateFont(2*m_TopTextMargin/DivFactor, 0, 900, 900, 400, FALSE, FALSE, 0,
                     ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                     CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                     DEFAULT_PITCH | FF_ROMAN, NULL);
CFont* pOldFont=pDC->SelectObject(&RomanFont);
CSize TitleSize=pDC->GetTextExtent(YAxisTitle);
int TitleY=(m_VarDimHeight-TitleSize.cy)/2;
pDC->TextOut(-m_LeftTextMargin,TitleY,YAxisTitle);
pDC->SelectObject(pOldFont);

pOldPen=pDC->SelectObject(&ThickBlackPen);
pDC->MoveTo(m_WindowXDim-m_LeftTextMargin,m_VarDimHeight);
pDC->LineTo(0,m_VarDimHeight);

CFont SmallRomanFont;
SmallRomanFont.CreateFont(m_TopTextMargin/2, 0, 0, 0, 400, FALSE, FALSE, 0,
                     ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                     CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                     DEFAULT_PITCH | FF_ROMAN, NULL);
pOldFont=pDC->SelectObject(&SmallRomanFont);
CString str;
int YShift;
for(int ctr=0;ctr<=100; ctr+=20) {
    str.Format("%.1f",(1.0+(double)(ctr/100.0)));
    TitleSize=pDC->GetTextExtent(str);
    YShift=TitleSize.cy/2;
    pDC->TextOut(-m_LeftTextMargin/2,
        ((double)ctr/100.0)*(double)m_VarDimHeight+YShift,str);
    pDC->MoveTo(-m_LeftTextMargin/10,
        ((double)ctr/100.0)*(double)m_VarDimHeight);
    pDC->LineTo(m_LeftTextMargin/10,
        ((double)ctr/100.0)*(double)m_VarDimHeight);
}
pDC->SelectObject(pOldFont);

pDC->SelectObject(&CyanSolidPen);
double *VarDimTraj=pDoc->GetVarDimTraj();
//double *TransientModel=pDoc->GetTransientModel();
double hold;
if (pDoc->m_bViewSegmentation) {
    hold=(VarDimTraj[0]-1.0)*m_VarDimHeight;
    pDC->MoveTo(0,hold);
    for(ctr=1;ctr<(m_ViewFileSize-pDoc->GetExtract()->GetWindowShift()-2*
        pDoc->GetExtract()->GetWindowSize());ctr++) {
    //for(ctr=1;ctr<(m_ViewFileSize-pDoc->GetExtract()->GetWindowShift()-
        //pDoc->GetExtract()->GetWindowSize()-2*512);ctr++) {
        hold=(VarDimTraj[ctr]-1.0)*m_VarDimHeight;
        pDC->LineTo(ctr,hold);
```

```
            }
        }
        //view feature extraction
        else {
            hold=(VarDimTraj[m_ViewStart]-1.0)*m_VarDimHeight;
            pDC->MoveTo(0,hold);
            for(ctr=1; ctr<m_ViewFileSize; ctr++)
            {
                if(ctr==2010)
                {
                    hold=1;
                }
                hold=(VarDimTraj[ctr+m_ViewStart]-1.0)*m_VarDimHeight;
                pDC->LineTo(ctr,hold);
            }
        }
        /*else {
            hold=(TransientModel[0]-1.0)*m_VarDimHeight;
            pDC->MoveTo(0,hold);
            for(ctr=1; ctr<pDoc->GetExtract()->GetTransientModelSize(); ctr++)
            {
                if(ctr==2010)
                {
                    hold=1;
                }
                hold=(TransientModel[ctr]-1.0)*m_VarDimHeight;
                pDC->LineTo(ctr*pDoc->GetExtract()->GetWindowShift(),hold);
            }
        }*/

        CPen BlueDashPen(PS_DASH, 1, RGB(0,0,128));
            int TransientBeginsHere=pDoc->GetTransientBeginsHere();
            int TransientSize=pDoc->GetExtract()->GetTransientSize();
            pDC->SelectObject(&BlueDashPen);

            pDC->MoveTo(TransientBeginsHere,m_RawSignalHeight+m_VarDimHeight);
            pDC->LineTo(TransientBeginsHere,0);
            pDC->MoveTo(TransientBeginsHere+TransientSize,
                m_RawSignalHeight+m_VarDimHeight);
            pDC->LineTo(TransientBeginsHere+TransientSize,0);
            pDC->SelectObject(pOldPen);

        return;
}

void CRawView::PlotRawSignal(CDC* pDC)
{
        CTAC_MMDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        CPen *pOldPen;
        CPen RedSolidPen(PS_SOLID, 1, RGB(255,0,0));
        CPen ThickBlackPen(PS_SOLID, m_WindowXDim/500, RGB(0,0,0));

        int *RawSignal=pDoc->GetRawSignal();

        CString YAxisTitle="Raw Signal";
        int DivFactor=2;
        if(pDoc->m_bViewVarDimTrajFlag) {
            DivFactor=3;
        }
        CFont RomanFont;
        RomanFont.CreateFont(2*m_TopTextMargin/DivFactor, 0, 900, 900, 400, FALSE, FALSE, 0,
                                ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
```

```
                                    DEFAULT_PITCH | FF_ROMAN, NULL);
        CFont* pOldFont=pDC->SelectObject(&RomanFont);
        CSize TitleSize=pDC->GetTextExtent(YAxisTitle);
        int TitleY=(m_RawSignalHeight-TitleSize.cy)/2;
        pDC->TextOut(-m_LeftTextMargin,TitleY+m_VarDimHeight,YAxisTitle);
        pDC->SelectObject(pOldFont);

        pOldPen=pDC->SelectObject(&RedSolidPen);
        pDC->MoveTo(0,m_VarDimHeight+RawSignal[m_ViewStart]/m_ZoomYCoord-m_ClipYCoord);
        for(int ctr=1; ctr<m_ViewFileSize; ctr++) {
            pDC->LineTo(ctr,m_VarDimHeight+
                RawSignal[ctr+m_ViewStart]/m_ZoomYCoord-m_ClipYCoord);
        }

        if (pDoc->m_bViewSegmentation) {
            CPen BlueDashPen(PS_DASH, 1, RGB(0,0,128));
            int TransientBeginsHere=pDoc->GetTransientBeginsHere();
            int TransientSize=pDoc->GetExtract()->GetTransientSize();
            pDC->SelectObject(&BlueDashPen);

            pDC->MoveTo(TransientBeginsHere,m_RawSignalHeight+m_VarDimHeight);
            pDC->LineTo(TransientBeginsHere,0);
            pDC->MoveTo(TransientBeginsHere+TransientSize,
                m_RawSignalHeight+m_VarDimHeight);
            pDC->LineTo(TransientBeginsHere+TransientSize,0);
        }

        pDC->SelectObject(pOldPen);
        return;
}

void CRawView::AdjustForMargins(CDC* pDC)
{
        CTAC_MMDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        CPageSetupDialog* PSD=pDoc->GetPageSetupDlg();

        int PageWidth=PSD->m_psd.ptPaperSize.x-PSD->m_psd.rtMinMargin.left-
            PSD->m_psd.rtMinMargin.right;
        int PageLength=PSD->m_psd.ptPaperSize.y-PSD->m_psd.rtMinMargin.top
            -PSD->m_psd.rtMinMargin.bottom;
        int UsablePageWidth=PSD->m_psd.ptPaperSize.x-PSD->m_psd.rtMargin.left-
            PSD->m_psd.rtMargin.right;
        int UsablePageLength=PSD->m_psd.ptPaperSize.y-PSD->m_psd.rtMargin.top
            -PSD->m_psd.rtMargin.bottom;

        int NewWindowXDim=(double)m_WindowXDim*(double)PageWidth/
                                                (double)UsablePageWidth;
        int NewWindowYDim=(double)m_WindowYDim*(double)PageLength/
                                                (double)UsablePageLength;

        int LeftMarginDiff=PSD->m_psd.rtMargin.left-PSD->m_psd.rtMinMargin.left;
        int RightMarginDiff=PSD->m_psd.rtMargin.right-PSD->m_psd.rtMinMargin.right;
        int TopMarginDiff=PSD->m_psd.rtMargin.top-PSD->m_psd.rtMinMargin.top;
        int BottomMarginDiff=PSD->m_psd.rtMargin.bottom-
            PSD->m_psd.rtMinMargin.bottom;

        int LeftMarginLP=(double)(NewWindowXDim-m_WindowXDim)*(double)
            ((double)LeftMarginDiff/(double)(LeftMarginDiff+RightMarginDiff));
        int BottomMarginLP=(double)(NewWindowYDim-m_WindowYDim)*(double)
            ((double)BottomMarginDiff/(double)(BottomMarginDiff+TopMarginDiff));

        m_Origin.x=(double)(m_LeftTextMargin+LeftMarginLP)/
            (double)NewWindowXDim*(double)m_rectClient.right;
```

```
        m_Origin.y=m_rectClient.bottom-(double)(m_BottomTextMargin+BottomMarginLP)/
            (double)NewWindowYDim*(double)m_rectClient.bottom;

        CSize DataSize(NewWindowXDim,NewWindowYDim);

        pDC->SetMapMode(MM_ANISOTROPIC);
        pDC->SetWindowExt(DataSize);
        pDC->SetViewportExt(m_rectClient.right, -m_rectClient.bottom);
        pDC->SetViewportOrg(m_Origin);
}

void CRawView::SetViewFileSize()
{
        CTAC_MMDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        if(pDoc->m_bViewSegmentation) {
                m_ViewFileSize=pDoc->GetExtract()->GetRawFileSize();
                m_ViewStart=0;
                m_ZoomYCoord=8;
                m_ClipYCoord=0;
        }
        else {
                m_ViewFileSize=pDoc->GetExtract()->GetTransientSize();
                m_ViewStart=pDoc->GetTransientBeginsHere();
                m_ZoomYCoord=1+(pDoc->GetHighestValue()-pDoc->GetLowestValue())/8191;
                m_ClipYCoord=pDoc->GetLowestValue()/m_ZoomYCoord;
        }
}

void CRawView::OnViewSegmentation()
{
        CTAC_MMDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        pDoc->m_bViewSegmentation=TRUE;
        pDoc->ViewNeedsUpdating();
}

void CRawView::OnUpdateViewSegmentation(CCmdUI* pCmdUI)
{
        CTAC_MMDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        pCmdUI->Enable(pDoc->m_bFileInMemory&&!pDoc->m_bInBatchProcess);
        pCmdUI->SetCheck(pDoc->m_bViewSegmentation);
}

void CRawView::OnViewTransient()
{
        CTAC_MMDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        pDoc->m_bViewSegmentation = FALSE;
        pDoc->ViewNeedsUpdating();
}

void CRawView::OnUpdateViewTransient(CCmdUI* pCmdUI)
{
        CTAC_MMDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        pCmdUI->Enable(pDoc->m_bFileInMemory&&!pDoc->m_bInBatchProcess);
        pCmdUI->SetCheck(!(pDoc->m_bViewSegmentation));
}
```

```
void CRawView::OnEditCopy()
{
	CBitmap BitmapClip;
	CClientDC ClientDC (this);
	CDC MemDC;
	RECT Rect;

	GetClientRect (&Rect);
	BitmapClip.CreateCompatibleBitmap (&ClientDC, Rect.right-Rect.left,
                                                  Rect.bottom-Rect.top);
	MemDC.CreateCompatibleDC (&ClientDC);
	MemDC.SelectObject (&BitmapClip);
	MemDC.BitBlt(0, 0, Rect.right-Rect.left, Rect.bottom-Rect.top,
                        &ClientDC, 0, 0, SRCCOPY);
	if(!OpenClipboard())
		return;
	::EmptyClipboard();
	::SetClipboardData(CF_BITMAP, BitmapClip.m_hObject);
	BitmapClip.Detach();
	::CloseClipboard();
}

void CRawView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
	CTAC_MMDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	pCmdUI->Enable(pDoc->m_bFileInMemory);
}

CString CRawView::GetTitleLine()
{
	CTAC_MMDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	CString TitleLine;
	TitleLine="Transmitter: " + (pDoc -> m_TransmitterMake);
	TitleLine+=" " + (pDoc -> m_TransmitterModel);
	if(pDoc -> m_TransmitterSerial.IsEmpty()) {
		TitleLine+="  Serial #: Unknown";
	}
	else {
		TitleLine+="  Serial #: " + (pDoc -> m_TransmitterSerial);
	}

	return TitleLine;
}
```

ResultView.h

```
//CResultsView: defines and manages the display in the lower viewing window to display classification results
#if !defined(AFX_RESULTSVIEW_H__66C35B02_506A_11D2_8BA9_00A024423FB9__INCLUDED_)
#define AFX_RESULTSVIEW_H__66C35B02_506A_11D2_8BA9_00A024423FB9__INCLUDED_


#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ResultsView.h : header file
//
#include "TAC_MMDoc.h"
#include "TextTable.h"// Added by ClassView
/////////////////////////////////////////////////////////////////////////////
// CResultsView view

class CResultsView : public CScrollView
{
protected:
	CResultsView();		// protected constructor used by dynamic creation
	DECLARE_DYNCREATE(CResultsView)

// Attributes
public:

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CResultsView)
	public:
	virtual void OnPrepareDC(CDC* pDC, CPrintInfo* pInfo = NULL);
	protected:
	virtual void OnDraw(CDC* pDC);	// overridden to draw this view
	virtual void OnInitialUpdate();	// first time after construct
	virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	//}}AFX_VIRTUAL

// Implementation
protected:
	virtual ~CResultsView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

	CTAC_MMDoc* GetDocument() {return (CTAC_MMDoc*) m_pDocument;};



	// Generated message map functions
	//{{AFX_MSG(CResultsView)
		// NOTE - the ClassWizard will add and remove member functions here.
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	CTextTable m_SummaryTable;
```

```
    CTextTable m_ResultsTable;
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_RESULTSVIEW_H__66C35B02_506A_11D2_8BA9_00A024423FB9__INCLUDED_)
```

ResultView.cpp

```
// ResultsView.cpp : implementation file
//

#include "stdafx.h"
#include "tac_mm.h"
#include "ResultsView.h"
#include "TAC_MMDoc.h"
#include "TextTable.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CResultsView

IMPLEMENT_DYNCREATE(CResultsView, CScrollView)

CResultsView::CResultsView()
{
}

CResultsView::~CResultsView()
{
}


BEGIN_MESSAGE_MAP(CResultsView, CScrollView)
    //{{AFX_MSG_MAP(CResultsView)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CResultsView drawing

void CResultsView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    CSize sizeTotal;
    // TODO: calculate the total size of this view
    sizeTotal.cx = sizeTotal.cy = 100;
    SetScrollSizes(MM_TEXT, sizeTotal);

}

void CResultsView::OnDraw(CDC* pDC)
{
    CTAC_MMDoc* pDoc = GetDocument();

    // TODO: add draw code here
    CRect rectTotal(0,0,10,10); // CRect containing the entire painted results.
                                // Used to set the correct scroll sizes.
                                // 10, 10 are the starting coordinates.

    CPoint pointPos(rectTotal.Width(), rectTotal.Height());
                        // Holds the position where the next item will be
                        // painted.
```

```
    CFont ArialUnderlinedFont;
    ArialUnderlinedFont.CreateFont(20, 0, 0, 0, FW_SEMIBOLD, FALSE, TRUE, 0,
                                   DEFAULT_CHARSET, OUT_DEFAULT_PRECIS,
                                   CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                                   DEFAULT_PITCH | FF_ROMAN, "Arial");

    CFont *pOldFont=pDC->SelectObject(&ArialUnderlinedFont);
    pDC->TextOut(pointPos.x, pointPos.y, "Classification Results");

    // Inflate rectTotal accordingly and set the next position to print.
    CSize size = pDC->GetTextExtent("Classification Results");
    CRect rect(pointPos, size);
    rectTotal |= rect;
    pointPos.Offset(0, size.cy + 10);


    CFont ArialNormalFont;
    ArialNormalFont.CreateFont(16, 0, 0, 0, FW_NORMAL, FALSE, FALSE, 0,
                               DEFAULT_CHARSET, OUT_DEFAULT_PRECIS,
                               CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                               DEFAULT_PITCH | FF_ROMAN, "Arial");
    pDC->SelectObject(&ArialNormalFont);


    // Paint the results table and adjust rectTotal accordingly.
    rect = m_ResultsTable.PaintTable(pDC, pointPos.x, pointPos.y);
    rectTotal |= rect;
    pointPos.Offset(0, rect.Height() + 10);

    // Paint Classification Summary title;
    pDC->SelectObject(&ArialUnderlinedFont);
    pDC->TextOut(pointPos.x, pointPos.y, "Classification Summary");


    // Inflate rectTotal in the y direction accordingly and adjust pointPos.
    CSize size2 = pDC->GetTextExtent("Classification Summary");
    CRect rect2(pointPos, size2);
    rectTotal |= rect2;
    pointPos.Offset(0, size2.cy + 10);


    // Paint the summary table and adjust rectTotal accordingly.
    pDC->SelectObject(&ArialNormalFont);
    rect = m_SummaryTable.PaintTable(pDC, pointPos.x, pointPos.y);
    rectTotal |= rect;
    pointPos.Offset(0, rect.Height() + 10);


    // Set the scroll sizes for the view.
    CRect rectWnd;
    GetWindowRect(&rectWnd);
    SetScrollSizes(MM_TEXT, rectTotal.Size(), rectWnd.Size()),


    pDC->SelectObject(pOldFont);

}

/////////////////////////////////////////////////////////////////////////////
// CResultsView diagnostics

#ifdef _DEBUG
void CResultsView::AssertValid() const
{
```

```
	CScrollView::AssertValid();
}

void CResultsView::Dump(CDumpContext& dc) const
{
	CScrollView::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CResultsView message handlers

void CResultsView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
	// TODO: Add your specialized code here and/or call the base class

	CScrollView::OnPrepareDC(pDC, pInfo);



	// Set the mapping mode.
	pDC->SetMapMode(MM_TEXT);

}

void CResultsView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
{
	// TODO: Add your specialized code here and/or call the base class

	CScrollView::OnPrint(pDC, pInfo);

}

void CResultsView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
	// TODO: Add your specialized code here and/or call the base class

	CTAC_MMDoc* pDoc = GetDocument();

	// Adjust the size of the results table.
	m_ResultsTable.SetSize(5, (pDoc->GetResultsArray()->GetSize() + 1)); // +1 for headings.

	// Set the column headings.
	m_ResultsTable.SetText("Transient #", 0, 0);
	m_ResultsTable.SetText("Filename and Path", 1, 0);
	m_ResultsTable.SetText("Predicted Class", 2, 0);
	m_ResultsTable.SetText("Error (%)", 3, 0);
	m_ResultsTable.SetText("Activation", 4, 0);


	// Fill in the rest of the table.
	CString strTemp;
	int PredictedClass;

	// Also count the number of transients identified as each class
	// for the summary information.
	int *Summary = new int[pDoc->GetArray()->FindNumClasses()];
	for (int i = 0; i < pDoc->GetArray()->FindNumClasses(); i++)
	{
		Summary[i] = 0;
	}

	for (int row = 1; row < (m_ResultsTable.GetRows()); row++)
	{
		strTemp.Format("%d", row);
```

```
            m_ResultsTable.SetText(strTemp, 0, row);

            m_ResultsTable.SetText(pDoc->GetResultsArray()->GetAt(row - 1)
                ->GetRawFilePath(), 1, row);
            int LastClassNumber ;
            LastClassNumber = pDoc->GetArray()->FindNumClasses(); //Nur testing
            PredictedClass=pDoc->GetResultsArray()->GetAt(row - 1)->GetPredictedClass(0);
            strTemp = pDoc -> GetArray() -> GetClassName(PredictedClass);
            m_ResultsTable.SetText(strTemp, 2, row);
            if (PredictedClass + 1 >LastClassNumber ) //Nur Predicted Class start from 0.
                PredictedClass = -1 ; // Nur testing
            if (PredictedClass < -1)
                PredictedClass = -1 ; // Nur testing
            if (PredictedClass != -1)
                Summary[PredictedClass] += 1; // Increment the counter for the class.


            strTemp.Format("%.4lf", pDoc->GetResultsArray()->GetAt(row - 1)->GetError(0));
            m_ResultsTable.SetText(strTemp, 3, row);

            strTemp.Format("%.3e",pDoc->GetResultsArray()->GetAt(row - 1)->GetActivation(0));
            m_ResultsTable.SetText(strTemp, 4, row);
        }

        // Set the size of the summary table.
        m_SummaryTable.SetSize(2, (pDoc->GetArray()->FindNumClasses()+3));
        // +3 for header, Unknown and Total rows.

        // Add the headers
        m_SummaryTable.SetText("Class Name", 0, 0);
        m_SummaryTable.SetText("Number of Transients", 1, 0);

        // Add the totals for each class.
        int iKnown = 0; // Known classes
        for (i = 0; i < pDoc->GetArray()->FindNumClasses(); i++)
        {
            m_SummaryTable.SetText(pDoc->GetArray()->GetClassName(i), 0, (i+1));
            strTemp.Format("%d", Summary[i]);
            iKnown += Summary[i];
            m_SummaryTable.SetText(strTemp, 1, (i+1));
        }

        // Add the unknown and total rows.
        m_SummaryTable.SetText("Unknown", 0, (i+1));
        strTemp.Format("%d", (pDoc->GetResultsArray()->GetSize()-iKnown));
        m_SummaryTable.SetText(strTemp, 1, (i+1));
        i++;
        m_SummaryTable.SetText("Total", 0, (i+1));
        strTemp.Format("%d", pDoc->GetResultsArray()->GetSize());
        m_SummaryTable.SetText(strTemp, 1, (i+1));

}

void CResultsView::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
        // TODO: Add your specialized code here and/or call the base class

        CScrollView::OnBeginPrinting(pDC, pInfo);
}

void CResultsView::OnEndPrinting(CDC* pDC, CPrintInfo* pInfo)
{
        // TODO: Add your specialized code here and/or call the base class

        CScrollView::OnEndPrinting(pDC, pInfo);
```

```
}

BOOL CResultsView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // TODO: call DoPreparePrinting to invoke the Print dialog box

    return CScrollView::OnPreparePrinting(pInfo);
}
```

SearchDialog.h

```
//CSearchDialog: defines and controls the dialog box displayed to search a transient in the database
// SearchDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CSearchDialog dialog

class CSearchDialog : public CDialog
{
// Construction
public:
	CSearchDialog(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CSearchDialog)
	enum { IDD = IDD_SEARCH_DIALOG };
	int		m_FieldToSearch;
	int		m_Class;
	CStringm_Date;
	CStringm_Make;
	CStringm_Model;
	CStringm_Serial;
	CStringm_Time;
	CStringm_strClassName;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CSearchDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	void DisableAllEditBoxes();

	// Generated message map functions
	//{{AFX_MSG(CSearchDialog)
	virtual BOOL OnInitDialog();
	afx_msg void OnClassRadio();
	afx_msg void OnRadio1();
	afx_msg void OnRadio2();
	afx_msg void OnRadio3();
	afx_msg void OnRadio4();
	afx_msg void OnRadio5();
	afx_msg void OnRadio6();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

SearchDialog.cpp

```
// SearchDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "SearchDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSearchDialog dialog


CSearchDialog::CSearchDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CSearchDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSearchDialog)
	m_FieldToSearch = -1;
	m_Class = 0;
	m_Date = _T("");
	m_Make = _T("");
	m_Model = _T("");
	m_Serial = _T("");
	m_Time = _T("");
	m_strClassName = _T("");
	//}}AFX_DATA_INIT
}


void CSearchDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSearchDialog)
	DDX_Radio(pDX, IDC_CLASS_RADIO, m_FieldToSearch);
	DDX_Text(pDX, IDC_CLASS_EDIT, m_Class);
	DDV_MinMaxInt(pDX, m_Class, 0, 200000);
	DDX_Text(pDX, IDC_DATE_EDIT, m_Date);
	DDV_MaxChars(pDX, m_Date, 8);
	DDX_Text(pDX, IDC_MAKE_EDIT, m_Make);
	DDX_Text(pDX, IDC_MODEL_EDIT, m_Model);
	DDX_Text(pDX, IDC_SERIAL_EDIT, m_Serial);
	DDX_Text(pDX, IDC_TIME_EDIT, m_Time);
	DDV_MaxChars(pDX, m_Time, 8);
	DDX_Text(pDX, IDC_CLASS_NAME_EDIT, m_strClassName);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CSearchDialog, CDialog)
	//{{AFX_MSG_MAP(CSearchDialog)
	ON_BN_CLICKED(IDC_CLASS_RADIO, OnClassRadio)
	ON_BN_CLICKED(IDC_RADIO1, OnRadio1)
	ON_BN_CLICKED(IDC_RADIO2, OnRadio2)
	ON_BN_CLICKED(IDC_RADIO3, OnRadio3)
	ON_BN_CLICKED(IDC_RADIO4, OnRadio4)
	ON_BN_CLICKED(IDC_RADIO5, OnRadio5)
	ON_BN_CLICKED(IDC_RADIO6, OnRadio6)
```

```
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSearchDialog message handlers

BOOL CSearchDialog::OnInitDialog()
{
    CDialog::OnInitDialog();

//  m_FieldToSearch=0;
//  UpdateData(FALSE);

    DisableAllEditBoxes();

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CSearchDialog::OnClassRadio()
{
    DisableAllEditBoxes();
    GetDlgItem(IDC_CLASS_EDIT)->EnableWindow(TRUE);
}

void CSearchDialog::OnRadio1()
{
    DisableAllEditBoxes();
    GetDlgItem(IDC_MAKE_EDIT)->EnableWindow(TRUE);
}

void CSearchDialog::OnRadio2()
{
    DisableAllEditBoxes();
    GetDlgItem(IDC_MODEL_EDIT)->EnableWindow(TRUE);
}

void CSearchDialog::OnRadio3()
{
    DisableAllEditBoxes();
    GetDlgItem(IDC_SERIAL_EDIT)->EnableWindow(TRUE);
}

void CSearchDialog::OnRadio4()
{
    DisableAllEditBoxes();
    GetDlgItem(IDC_DATE_EDIT)->EnableWindow(TRUE);
}

void CSearchDialog::OnRadio5()
{
    DisableAllEditBoxes();
    GetDlgItem(IDC_TIME_EDIT)->EnableWindow(TRUE);
}

void CSearchDialog::DisableAllEditBoxes()
{
    CString strText;
    strText.Format ("%d",0);
    SetDlgItemText(IDC_CLASS_EDIT, strText);
    strText.Format("");
    SetDlgItemText(IDC_CLASS_NAME_EDIT, strText);
    SetDlgItemText(IDC_MAKE_EDIT, strText);
    SetDlgItemText(IDC_MODEL_EDIT, strText);
    SetDlgItemText(IDC_SERIAL_EDIT, strText);
```

```
        SetDlgItemText(IDC_DATE_EDIT, strText);
        SetDlgItemText(IDC_TIME_EDIT, strText);
        GetDlgItem(IDC_CLASS_NAME_EDIT)->EnableWindow(FALSE);
        GetDlgItem(IDC_CLASS_EDIT)->EnableWindow(FALSE);
        GetDlgItem(IDC_MODEL_EDIT)->EnableWindow(FALSE);
        GetDlgItem(IDC_MAKE_EDIT)->EnableWindow(FALSE);
        GetDlgItem(IDC_SERIAL_EDIT)->EnableWindow(FALSE);
        GetDlgItem(IDC_DATE_EDIT)->EnableWindow(FALSE);
        GetDlgItem(IDC_TIME_EDIT)->EnableWindow(FALSE);
}

void CSearchDialog::OnRadio6()
{
        // TODO: Add your control notification handler code here
        DisableAllEditBoxes();
        GetDlgItem(IDC_CLASS_NAME_EDIT)->EnableWindow(TRUE);
}
```

SetRejectionDialog.h

```
//CSetRejectionDialog: defines and controls the dialog box displayed to set the rejection threshold
#if !defined(AFX_SETREJECTIONDIALOG_H__C624B441_043F_11D1_BF55_444553540000__INCLUDED_)
#define AFX_SETREJECTIONDIALOG_H__C624B441_043F_11D1_BF55_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// SetRejectionDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CSetRejectionDialog dialog

class CSetRejectionDialog : public CDialog
{
// Construction
public:
	CSetRejectionDialog(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CSetRejectionDialog)
	enum { IDD = IDD_SETREJECTION_DIALOG };
	doublem_dRejectionThreshold;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CSetRejectionDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CSetRejectionDialog)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SETREJECTIONDIALOG_H__C624B441_043F_11D1_BF55_444553540000__INCLUDED_)
```

SetRejectionDialog.cpp

```
// SetRejectionDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "SetRejectionDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSetRejectionDialog dialog


CSetRejectionDialog::CSetRejectionDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CSetRejectionDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSetRejectionDialog)
	m_dRejectionThreshold = 0.0;
	//}}AFX_DATA_INIT
}


void CSetRejectionDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSetRejectionDialog)
	DDX_Text(pDX, IDC_REJECTIONTHRESHOLD, m_dRejectionThreshold);
	DDV_MinMaxDouble(pDX, m_dRejectionThreshold, 0., 1.);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CSetRejectionDialog, CDialog)
	//{{AFX_MSG_MAP(CSetRejectionDialog)
		// NOTE: the ClassWizard will add message map macros here
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSetRejectionDialog message handlers
```

SigmaInitDialog.h

```
//CSigmaInitDialog: defines and controls the dialog box displayed to update the user on the status of sigma initialization
// SigmaInitDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CSigmaInitDialog dialog

class CSigmaInitDialog : public CDialog
{
// Construction
public:
	CSigmaInitDialog(CWnd* pParent = NULL);   // standard constructor

	CTransientArray *m_TransientArray;
	CPNN *m_PNN;
	int m_nNumSearchPoints;
	BOOL m_bGlobalSearchDone;

// Dialog Data
	//{{AFX_DATA(CSigmaInitDialog)
	enum { IDD = IDD_TRAINSIGMASINIT_DIALOG };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CSigmaInitDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	 double m_dLowerSigma, m_dLowerError, m_dMiddleSigma, m_dMiddleError,
		m_dUpperSigma, m_dUpperError;
	// Generated message map functions
	//{{AFX_MSG(CSigmaInitDialog)
	afx_msg void OnStopbutton();
	afx_msg void OnTrainbutton();
	virtual BOOL OnInitDialog();
	afx_msg void OnTimer(UINT nIDEvent);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

SigmaInitDialog.cpp

```
// SigmaInitDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "PNN.h"

#include "SigmaInitDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSigmaInitDialog dialog


CSigmaInitDialog::CSigmaInitDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CSigmaInitDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSigmaInitDialog)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}


void CSigmaInitDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSigmaInitDialog)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CSigmaInitDialog, CDialog)
	//{{AFX_MSG_MAP(CSigmaInitDialog)
	ON_BN_CLICKED(IDC_STOPBUTTON, OnStopbutton)
	ON_BN_CLICKED(IDC_TRAINBUTTON, OnTrainbutton)
	ON_WM_TIMER()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSigmaInitDialog message handlers

void CSigmaInitDialog::OnStopbutton()
{
	CString strText;
	m_PNN->m_bStopNow=TRUE;
	strText.Format("Press Train to resume or OK to Exit.");
	SetDlgItemText(IDC_STATIC_CURRENT, strText);
	GetDlgItem(IDC_STOPBUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_TRAINBUTTON)->EnableWindow(TRUE);
	GetDlgItem(IDOK)->EnableWindow(TRUE);

}

void CSigmaInitDialog::OnTrainbutton()
```

```
{
	CString strText;

	GetDlgItem(IDC_STOPBUTTON)->EnableWindow(TRUE);
	GetDlgItem(IDC_TRAINBUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDOK)->EnableWindow(FALSE);

	int Timer=SetTimer(1,200,NULL); // ID #1,  1/5 second, NULL
	ASSERT(Timer!=0);
	if (m_bGlobalSearchDone) {
		m_PNN->TrainSigmasInit(m_nNumSearchPoints, TRUE, &m_dLowerSigma,
			&m_dLowerError,&m_dMiddleSigma, &m_dMiddleError, &m_dUpperSigma,
			&m_dUpperError);
	}
	else {
		m_bGlobalSearchDone=m_PNN->TrainSigmasInit(m_nNumSearchPoints, FALSE,
			&m_dLowerSigma, &m_dLowerError, &m_dMiddleSigma, &m_dMiddleError,
			&m_dUpperSigma, &m_dUpperError);
	}
	KillTimer(1);

	strText.Format("%.12lf",m_PNN->m_nUserDisplaySigma);
	SetDlgItemText(IDC_STATIC_SIGMA, strText);
	strText.Format("%.12lf",m_PNN->m_dUserDisplayError);
	SetDlgItemText(IDC_STATIC_ERROR, strText);
	CProgressCtrl* pProg=(CProgressCtrl*)GetDlgItem(IDC_INITPROGRESS);
	pProg->SetPos(m_PNN->m_nUserDisplayDiscreteError);
	strText.Format("%d",m_PNN->m_nUserDisplayDiscreteError);
	SetDlgItemText(IDC_STATIC_MISCLASS, strText);

	GetDlgItem(IDC_STOPBUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_TRAINBUTTON)->EnableWindow(TRUE);
	GetDlgItem(IDOK)->EnableWindow(TRUE);
}

BOOL CSigmaInitDialog::OnInitDialog()
{
	CDialog::OnInitDialog();

	CProgressCtrl* pProg=(CProgressCtrl*)GetDlgItem(IDC_INITPROGRESS);
	pProg->SetRange(0,m_TransientArray->GetSize());
	pProg->SetPos(0);
	CString strText;
	strText.Format("%d",m_TransientArray->GetSize());
	SetDlgItemText(IDC_STATIC_MAXERROR, strText);
	strText.Format("Press Train to begin.");
	SetDlgItemText(IDC_STATIC_CURRENT, strText);
	GetDlgItem(IDC_STOPBUTTON)->EnableWindow(FALSE);
	m_bGlobalSearchDone=FALSE;

	return TRUE;  // return TRUE unless you set the focus to a control
		// EXCEPTION: OCX Property Pages should return FALSE
}

void CSigmaInitDialog::OnTimer(UINT nIDEvent)
{
	if (nIDEvent==1) {
		CString strText;
		if (!m_PNN->m_bStopNow)
			SetDlgItemText(IDC_STATIC_CURRENT, m_PNN->m_strUserMessage);
		strText.Format("%.12lf",m_PNN->m_nUserDisplaySigma);
		SetDlgItemText(IDC_STATIC_SIGMA, strText);
		strText.Format("%.12lf",m_PNN->m_dUserDisplayError);
		SetDlgItemText(IDC_STATIC_ERROR, strText);
		CProgressCtrl* pProg=(CProgressCtrl*)GetDlgItem(IDC_INITPROGRESS);
```

```
            pProg->SetPos(m_PNN->m_nUserDisplayDiscreteError);
            strText.Format("%d",m_PNN->m_nUserDisplayDiscreteError);
            SetDlgItemText(IDC_STATIC_MISCLASS, strText);
        }

        CDialog::OnTimer(nIDEvent);
}
```

Sofm.h

```
//This file is written by the author to implement the SOFM technique

// Sofm.h : header file
//
static unsigned long seed;
#define RND_MAX 32767L

#include "TransientArray.h"

class CSOFM:public CObject
{
	DECLARE_SERIAL(CSOFM)

protected:
	int m_nTransientModelSize;//init in constructor
	int m_nNumClasses;
	int m_nTransientNum;//number of training cases to retain after SOFM
	int m_nMapSize;
	int m_nEpoch;//number of iterations
	double m_dRadius; //initial radius of neighbourhood
	double m_dAlpha; //initial learning rate
	double *m_Weight;//size is m_nMapSize*m_nModelSize

	CTransientArray *m_TransientArray; //return valu is the size-reduced matrix
	int CurrentClassIndex;
	int m_NoSamplesInClass;
	//CTransientArray *m_CurrentClassArray;
public:
	void Serialize(CArchive& ar);
	CSOFM();
	~CSOFM() {delete m_Weight;}
	void Initialize(int TransientModelSize);
	void osrand() {seed= 4000; }
	int orand() { return( (int) ((seed=(seed*23)%100000001)%RND_MAX) );}

	void RandInit();//random initialize weights
	void Normalize(double *vec, int vec_size);
	double DotProd(double *vec1, double *vec2,int vec_size);
	double Dist(double *vec1, double *vec2,int vec_size);
	void Adapt(double *vec, double *sample, int dim, double alpha);
	int FindWinner(double *m_CurrentTransientModel);
	void SOFMTrain();
	int Map_Loc(double *m_CurrentTransientModel);
	int Find_Centroid();
	void Cluster(int *DeleteTransientIndex);
	//void SetNumCases(int NumCases) { m_nNumCases=NumCases; }
	void SetNumClasses(int NumClasses) { m_nNumClasses=NumClasses; }
	int GetNumClasses() { return m_nNumClasses; }
	void SetTransientArrayPointer(CTransientArray *TransientArray)
		{ m_TransientArray=TransientArray; }
	//void FillTrainingResultsArray(CResultsArray *ResultsArray);
	void SetTransientNum(int TransientNum) { m_nTransientNum=TransientNum;}
	int GetTransientNum() {return m_nTransientNum;}
	void SetCurrentClassIndex (int index_init) {CurrentClassIndex=index_init;}
	int GetCurrentClassIndex() { return CurrentClassIndex;}
	void SetNoSamplesInClass (int Num) {m_NoSamplesInClass=Num;}

#ifdef _DEBUG
	void Dump(CDumpContext& dc) const;
#endif
};
```

Sofm.cpp

```
//Sofm.cpp : implementation of CSOFM

// This file is partially based on the program package 'som_pak'

#include "StdAfx.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "sofm.h"


IMPLEMENT_SERIAL(CSOFM, CObject,0)

#ifdef _DEBUG
void CSOFM::Dump(CDumpContext& dc) const
{
	CObject::Dump(dc);
	dc << "Luotao.. CSOFM Dump.";
}
#endif

CSOFM::CSOFM()
{
	m_nMapSize = 5; //map consists of 5 nodes
	m_nEpoch = 1000;
	m_dRadius = 5;
	m_dAlpha = 0.6;
	m_Weight = new double[1];
}

void CSOFM::Initialize(int TransientModelSize)
{
	m_nTransientModelSize=TransientModelSize;

	delete [] m_Weight;//had to do dummy init in constructor...undo it now
	m_Weight=new double[m_nMapSize*m_nTransientModelSize];
}


void CSOFM::Serialize(CArchive& ar)
{
	int ctr;

	if(ar.IsStoring()) {
		ar << m_nMapSize << m_nEpoch << m_dRadius << m_dAlpha
			<< m_nTransientModelSize << m_nTransientNum;
		for (ctr=0; ctr<m_nMapSize*m_nTransientModelSize; ctr++) {
			ar << m_Weight[ctr];
		}
	}
	else {
		ar >> m_nMapSize >> m_nEpoch >> m_dRadius >> m_dAlpha
			>> m_nTransientModelSize >> m_nTransientNum;
		for (ctr=0; ctr<m_nMapSize*m_nTransientModelSize; ctr++) {
			ar >> m_Weight[ctr];
		}
	}
}


//random initialize weight vectors
```

```
void CSOFM::RandInit()
{
 int i,j;
 double maval=2.0, mival=0.5;
 //range of input variable values (variance fractal dimension)
 // Randomize the weight vectors
      for (i = 0; i < m_nMapSize; i++)
      {
            for(j=0;j<m_nTransientModelSize;j++)
            {
                  m_Weight[i*m_nTransientModelSize+j] =
                        mival+(maval- mival) * ((double) orand() / 32768.0);
            }
 }

}



void CSOFM::Normalize(double *vec, int vec_size)
{
 double sum=0.0;
 int j;

 for (j=0; j<vec_size; j++)
       sum+=vec[j]*vec[j];

 sum=sqrt(sum);

 for (j=0; j<vec_size; j++)
       vec[j]/=sum;
}


double CSOFM::DotProd(double *vec1, double *vec2,int vec_size)
{
 double sum=0.0;
 int j;

 for (j=0; j<vec_size; j++)
       sum+=vec1[j]*vec2[j];

 return (sum);
}

double CSOFM::Dist(double *vec1, double *vec2,int vec_size)
{
      double dif,dist=0.0;
      for(int j=0;j<vec_size;j++)
      {
            dif=vec1[j]-vec2[j];
            dist +=dif*dif;
      }

      return sqrt(dist);
}


// move a weight vector towards another vector
void CSOFM::Adapt(double *vec, double *sample, int dim, double alpha)
{
      int i;
      for(i=0;i<dim;i++)
            vec[i] += alpha* (sample[i]-vec[i]);
}
```

```
/* Find_winner - finds the winning neuron using euclidean distance*/

int CSOFM::FindWinner(double *m_CurrentTransientModel)
{
      //double *m_dCurrentWeight;
   int i;
   double min_dist, diff, dist;
      int WinIndex;

   WinIndex = 0;
   //Diff = -1.0;

      ///Compute the distance between the first weight and input
      dist = 0.0;
      for (i = 0; i < m_nTransientModelSize; i++)
      {
            diff = m_Weight[i] - m_CurrentTransientModel[i];
            dist += diff * diff;

      }
      min_dist = dist; //store the first distance as currently the smallest
      WinIndex = 0; //current winner is the first neuron

   // Go through the rest of weight vectors
   for(int index=1; index<m_nMapSize; index++)
      {
            //m_dCurrentWeight=m_Weight[index*m_nTransientModelSize];
        dist = 0.0;

            // Compute the distance between weight and input entry
            for (i = 0; i < m_nTransientModelSize; i++)
            {
                  diff = m_Weight[index*m_nTransientModelSize+i]
                        - m_CurrentTransientModel[i];
                  dist += diff * diff;
                  if (dist > min_dist)
                        break;
            }

            // If distance is smaller than the smallest distance, update
            if (dist < min_dist)
            {
                  min_dist = dist;
                  WinIndex = index;
            }
      }

      return WinIndex;
}


/* som_training - train a SOM. Radius of the neighborhood decreases
   linearly from the initial value to one and the learning parameter
   decreases linearly from its initial value to zero. */

void CSOFM::SOFMTrain()
{

 int index,WinIndex;
 double trad, talp;
 double *m_CurrentTransientModel;

 m_CurrentTransientModel= new double[m_nTransientModelSize];
```

```
 RandInit(); //random initialize m_Weight

 index = CurrentClassIndex;

 //iteratively choose training sample from exsiting samples
 for (int iter = 0; iter < m_nEpoch; iter++,
       m_CurrentTransientModel = m_TransientArray->GetAt(index)->GetTransientModel())
 {

  // Radius decreases linearly to one
  trad = 1.0 + (m_dRadius - 1.0) * (double) (m_nEpoch-iter) / (double) m_nEpoch;
     //Alpha decrease linearly to zero
  talp = m_dAlpha* (double)(m_nEpoch-iter)/(double) m_nEpoch;

  // Find the best match for the current transient
  WinIndex= FindWinner(m_CurrentTransientModel);

  // Adapt all weights within neighbourhood area
     for(int i=0;i<m_nMapSize;i++)
     {
          if(fabs(i-WinIndex)<=trad)
               Adapt(&m_Weight[i*m_nTransientModelSize],
               m_CurrentTransientModel,m_nTransientModelSize,talp);
     }

     index++; //next training case in the class
     //assume all classes have the same number of training cases
     // if we are at the end of each class, go back to the start
     if(index >= (CurrentClassIndex+ m_TransientArray->GetSize()/m_nNumClasses) )
          index=CurrentClassIndex;
 }

 delete [] m_CurrentTransientModel;
}


// find the location on the map for a given input after training
int CSOFM::Map_Loc(double *m_CurrentTransientModel)
{
     int map_index;

     map_index = FindWinner(m_CurrentTransientModel);

     return map_index;
}

int CSOFM::Find_Centroid()
{
     int *node_hit, node_index, max_hit, max_node_index;
     double *m_CurrentTransientModel;
     m_CurrentTransientModel= new double[m_nTransientModelSize];

     node_hit = new int[m_nMapSize];
     //initialize the number of hits per node
     for(int i=0;i<m_nMapSize;i++)
     {
          node_hit[i] = 0;
     }

     TRACE("Transient per class is %i\n",m_NoSamplesInClass);
     for(i=0;i<m_NoSamplesInClass;i++) //go through all cases from a class
     {
          m_CurrentTransientModel=m_TransientArray->GetAt(i+CurrentClassIndex)
               ->GetTransientModel();
```

```
            node_index=Map_Loc(m_CurrentTransientModel);
            node_hit[node_index]++;
        }

        //find the highest hit-frequency node: the centroid of the map
        max_hit=node_hit[0];
        max_node_index = 0;
        for(i=1; i<m_nMapSize; i++)
        {
            if(node_hit[i]>max_hit)
            {
                max_hit=node_hit[i];
                max_node_index = i;
            }
        }
        delete [] node_hit;

        return max_node_index;
}

void CSOFM::Cluster(int *DeleteTransientIndex) //produce the reduced-size training cases for a class
{
        int centroid, count=0, node_index;
        double *m_CurrentTransientModel;

        centroid = Find_Centroid();
        TRACE("Centroid is node %i\n", centroid);
        for(int i=0; i<m_NoSamplesInClass;i++)
        {
            m_CurrentTransientModel=m_TransientArray->GetAt(i+CurrentClassIndex)
                ->GetTransientModel();
            node_index=Map_Loc(m_CurrentTransientModel);
            TRACE("Transient %i is mapped to node %i\n", i+CurrentClassIndex, node_index);
            if( (node_index != centroid)&&(count<(m_NoSamplesInClass-m_nTransientNum)) )
            {
                DeleteTransientIndex[count] = i+CurrentClassIndex;
                TRACE("Deleting transient %i\n", DeleteTransientIndex[count]);
                count++;
            }
        }
}
```

TAC_MM.h

```
// TAC_MM.h : main header file for the TAC_MM application
//

#if !defined(AFX_TAC_MM_H__778DC4E5_CDEC_11D0_BF54_444553540000__INCLUDED_)
#define AFX_TAC_MM_H__778DC4E5_CDEC_11D0_BF54_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMApp:
// See TAC_MM.cpp for the implementation of this class
//

class CTAC_MMApp : public CWinApp
{
public:
	CTAC_MMApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CTAC_MMApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CTAC_MMApp)
	afx_msg void OnAppAbout();
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_TAC_MM_H__778DC4E5_CDEC_11D0_BF54_444553540000__INCLUDED_)
```

TAC_MM.cpp

```
// TAC_MM.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "TAC_MM.h"

#include "MainFrm.h"
#include "TAC_MMDoc.h"
#include "TAC_MMView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMApp

BEGIN_MESSAGE_MAP(CTAC_MMApp, CWinApp)
    //{{AFX_MSG_MAP(CTAC_MMApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMApp construction

CTAC_MMApp::CTAC_MMApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CTAC_MMApp object

CTAC_MMApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMApp initialization

BOOL CTAC_MMApp::InitInstance()
{
    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();// Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();// Call this when linking to MFC statically
#endif
```

```
    LoadStdProfileSettings();  // Load standard INI file options (including MRU)

    // Register the application's document templates.  Document templates
    //  serve as the connection between documents, frame windows and views.

    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CTAC_MMDoc),
        RUNTIME_CLASS(CMainFrame),      // main SDI frame window
        RUNTIME_CLASS(CTAC_MMView));
    AddDocTemplate(pDocTemplate);

    // Enable DDE Execute open
    EnableShellOpen();
    RegisterShellFileTypes(TRUE);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // Enable drag/drop open
    m_pMainWnd->DragAcceptFiles();

    m_pMainWnd->SetWindowText("TAC-MM");

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
```

```
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CTAC_MMApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMApp commands
```

TAC_MMDoc.h

```
//CTAC_MMDoc: defines and manages the data within the application
//This file is modified by the author
// TAC_MMDoc.h : interface of the CTAC_MMDoc class
//
/////////////////////////////////////////////////////////////////////////////

#ifndef TAC_MMDOC_H // Make sure this header only gets included once.
#define TAC_MMDOC_H

#include "Extract.h"
#include "PNN.h"
#include "princomp.h"
#include "sofm.h"

#include "EditFPProgressDialog.h"
#include "TrainingResults.h"

#include <afx.h> //for CTime class

#include "objmfc.h"// For Ben's stuff.
#include "objects.h"// For Ben's stuff.
#include "GlobalStuff.h"// Added by ClassView

class CTransientListView;

class CTAC_MMDoc : public CDocument
{
private:
	FileType m_CurrentTranFileType;
	CString m_strTransientClassName;
	int m_nTransientClass;
	CString m_strPathName;
	CTransientArray m_TransientArray;
	CExtract m_Extract;
	CPNN m_PNN;
	CPrincoData m_Princo;
	CSOFM m_SOFM;
	CResultsArray m_ResultsArray;

	int m_nCurrentTransient;
	int *m_nRawSignal;
	int *m_nSquelchChannel;
	int m_nHighestValue;
	int m_nLowestValue;
	int m_nTransientBeginsHere;
	double m_dRejectThreshold;
	int m_nNumClassifyModes;
	int m_nMinSeparation;
	double *m_dThreshholds;
	double *m_dVarDimTraj;
	double *m_dTransientModel;
	BOOL m_bDocCreated;
	CPageSetupDialog m_PSDdlg;

protected: // create from serialization only
	CTAC_MMDoc();
	DECLARE_DYNCREATE(CTAC_MMDoc)

// Attributes
public:
	BOOL m_bFileInMemory;
```

```
        BOOL m_bViewRawSignalFlag;
        BOOL m_bViewVarDimTrajFlag;
        BOOL m_bScrollToTransient;
        BOOL m_bViewSegmentation;
        BOOL m_bNotClassifying;
        BOOL m_bViewTrainingResults;
        BOOL m_bViewNetworkDetails;
        BOOL m_bResultsExist;
        BOOL m_bTrainingResults;
        BOOL m_bInBatchProcess;
        BOOL m_bDimReduction;
        CString m_TransmitterMake;
        CString m_TransmitterModel;
        CString m_TransmitterSerial;
        CString m_Date;//mm/dd/yy
        CString m_Time;
        CTransientListView* m_pTransientListView;

        int *GetRawSignal() { return m_nRawSignal; }
        double *GetVarDimTraj() { return m_dVarDimTraj; }

        int GetLowestValue() { return m_nLowestValue; }
        int GetHighestValue() { return m_nHighestValue; }
        int GetTransientBeginsHere() {return m_nTransientBeginsHere; }
        CTransientArray *GetArray() { return &m_TransientArray; }
        CExtract *GetExtract() { return &m_Extract; }
        CPageSetupDialog *GetPageSetupDlg() { return &m_PSDdlg; }
        CResultsArray *GetResultsArray() { return &m_ResultsArray; }
// Operations
public:
        BOOL ReadRawSignal(CString FilePath,BOOL IsAdding);//FALSE if file not good
        int GetCurrentTransient() { return m_nCurrentTransient; }
        void SetCurrentTransient(int Place) { m_nCurrentTransient=Place; }
        void IncrementCurrentTransient() {m_nCurrentTransient+=1;
                                                UpdateStatusBar(); }
        void DecrementCurrentTransient() {m_nCurrentTransient-=1;
                                                if(m_nCurrentTransient!=-1)
                                                        UpdateStatusBar();
                                                else
                                                        UpdateStatusBarNoTransients();}
        void UpdateStatusBar();
        void UpdateStatusBarNoTransients();
        void UpdateStatusBarUnknown();
        void ViewNeedsUpdating();
        void SegmentAllTransients(int NewModelSize,
                CEditFPProgressDialog *ProgressDlg);
        void ComputeAllModels(int NewModelSize, CEditFPProgressDialog *ProgressDlg);
        void DeleteCurrentTransient();
        void DeleteResultsArray();
        void CheckWindowsMessages(int NumChecks);
        double ComputeConfidence(double *SummationNeuron, int PredictedClass);
        int MultiModeClassify(double *HighestConfidence,
                double *Activation);


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CTAC_MMDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        virtual void DeleteContents();
        virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
        //}}AFX_VIRTUAL
```

```
// Implementation
public:
	void UpdateListView();
	void StoreTransientData(int index);
	void UpdateStatusBarNewTransient();
	virtual ~CTAC_MMDoc();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
	//{{AFX_MSG(CTAC_MMDoc)
	afx_msg void OnFileNew();
	afx_msg void OnDatabaseAddtransient();
	afx_msg void OnUpdateDatabaseAddtransient(CCmdUI* pCmdUI);
	afx_msg void OnDatabaseDeletetransient();
	afx_msg void OnUpdateDatabaseDeletetransient(CCmdUI* pCmdUI);
	afx_msg void OnViewPrev();
	afx_msg void OnUpdateViewPrev(CCmdUI* pCmdUI);
	afx_msg void OnViewNext();
	afx_msg void OnUpdateViewNext(CCmdUI* pCmdUI);
	afx_msg void OnViewRawsignal();
	afx_msg void OnViewFractaltrajectory();
	afx_msg void OnUpdateViewFractaltrajectory(CCmdUI* pCmdUI);
	afx_msg void OnUpdateViewRawsignal(CCmdUI* pCmdUI);
	afx_msg void OnFilePageSetup();
	afx_msg void OnUpdateFilePageSetup(CCmdUI* pCmdUI);
	afx_msg void OnUpdateFilePrint(CCmdUI* pCmdUI);
	afx_msg void OnUpdateFilePrintPreview(CCmdUI* pCmdUI);
	afx_msg void OnUpdateFileSave(CCmdUI* pCmdUI);
	afx_msg void OnNeuralnetClassify();
	afx_msg void OnUpdateNeuralnetClassify(CCmdUI* pCmdUI);
	afx_msg void OnNeuralnetInitializesigmas();
	afx_msg void OnUpdateNeuralnetInitializesigmas(CCmdUI* pCmdUI);
	afx_msg void OnEditFractalparameters();
	afx_msg void OnUpdateEditFractalparameters(CCmdUI* pCmdUI);
	afx_msg void OnViewSearch();
	afx_msg void OnUpdateViewSearch(CCmdUI* pCmdUI);
	afx_msg void OnNeuralnetOptimizesigmas();
	afx_msg void OnUpdateNeuralnetOptimizesigmas(CCmdUI* pCmdUI);
	afx_msg void OnNeuralnetBatchclassify();
	afx_msg void OnUpdateNeuralnetBatchclassify(CCmdUI* pCmdUI);
	afx_msg void OnNeuralnetModeparameters();
	afx_msg void OnUpdateNeuralnetModeparameters(CCmdUI* pCmdUI);
	afx_msg void OnUpdateFileNew(CCmdUI* pCmdUI);
	afx_msg void OnUpdateFileOpen(CCmdUI* pCmdUI);
	afx_msg void OnUpdateFileSaveAs(CCmdUI* pCmdUI);
	afx_msg void OnNeuralnetSetrejectionthreshold();
	afx_msg void OnUpdateNeuralnetSetrejectionthreshold(CCmdUI* pCmdUI);
	afx_msg void OnStartCapture();
	afx_msg void OnViewZoomedrawsignal();
	afx_msg void OnUpdateViewZoomedrawsignal(CCmdUI* pCmdUI);
	afx_msg void OnViewNetworkDetails();
	afx_msg void OnUpdateViewNetworkDetails(CCmdUI* pCmdUI);
	afx_msg void OnViewClassificationStats();
	afx_msg void OnUpdateViewClassificationStats(CCmdUI* pCmdUI);
	afx_msg void OnViewTrainingResults();
	afx_msg void OnUpdateViewTrainingResults(CCmdUI* pCmdUI);
	afx_msg void OnSaveResults();
	afx_msg void OnUpdateSaveResults(CCmdUI* pCmdUI);
	//afx_msg void OnNeuralnetDimreduction();
```

```
    //afx_msg void OnUpdateNeuralnetDimreduction(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
#define TAC_MMDOC_H
#endif
```

TAC_MMDoc.cpp

```
//This file is modified by the author
// TAC_MMDoc.cpp : implementation of the CTAC_MMDoc class
//

#include "stdafx.h"
#include <fstream.h>
#include <iomanip.h>
#include <process.h>
#include <mmsystem.h>

#include "TAC_MM.h"
#include "MainFrm.h"
#include "stdio.h"
#include "TAC_MMDoc.h"
#include "TAC_MMview.h"
#include "TransientListView.h"
#include "NewDatabase.h"
#include "EnterClassDialog.h"
#include "EnterClassMultiDialog.h"
#include "ClassifyDialog.h"
#include "AddUnknownDialog.h"
#include "SigmaInitDialog.h"
#include "EnterSigmaInitParamsDialog.h"
#include "TrainSigmasOptDialog.h"
#include "FractalParamDialog.h"
#include "SearchDialog.h"
#include "BatchProgressDialog.h"
#include "ModeParamsDialog.h"
#include "SetRejectionDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMDoc

IMPLEMENT_DYNCREATE(CTAC_MMDoc, CDocument)

BEGIN_MESSAGE_MAP(CTAC_MMDoc, CDocument)
    //{{AFX_MSG_MAP(CTAC_MMDoc)
    ON_COMMAND(ID_FILE_NEW, OnFileNew)
    ON_COMMAND(ID_DATABASE_ADDTRANSIENT, OnDatabaseAddtransient)
    ON_UPDATE_COMMAND_UI(ID_DATABASE_ADDTRANSIENT, OnUpdateDatabaseAddtransient)
    ON_COMMAND(ID_DATABASE_DELETETRANSIENT, OnDatabaseDeletetransient)
    ON_UPDATE_COMMAND_UI(ID_DATABASE_DELETETRANSIENT, OnUpdateDatabaseDeletetransient)
    ON_COMMAND(ID_VIEW_PREV, OnViewPrev)
    ON_UPDATE_COMMAND_UI(ID_VIEW_PREV, OnUpdateViewPrev)
    ON_COMMAND(ID_VIEW_NEXT, OnViewNext)
    ON_UPDATE_COMMAND_UI(ID_VIEW_NEXT, OnUpdateViewNext)
    ON_COMMAND(ID_VIEW_RAWSIGNAL, OnViewRawsignal)
    ON_COMMAND(ID_VIEW_FRACTALTRAJECTORY, OnViewFractaltrajectory)
    ON_UPDATE_COMMAND_UI(ID_VIEW_FRACTALTRAJECTORY, OnUpdateViewFractaltrajectory)
    ON_UPDATE_COMMAND_UI(ID_VIEW_RAWSIGNAL, OnUpdateViewRawsignal)
    ON_COMMAND(ID_FILE_PAGE_SETUP, OnFilePageSetup)
    ON_UPDATE_COMMAND_UI(ID_FILE_PAGE_SETUP, OnUpdateFilePageSetup)
    ON_UPDATE_COMMAND_UI(ID_FILE_PRINT, OnUpdateFilePrint)
    ON_UPDATE_COMMAND_UI(ID_FILE_PRINT_PREVIEW, OnUpdateFilePrintPreview)
    ON_UPDATE_COMMAND_UI(ID_FILE_SAVE, OnUpdateFileSave)
```

```
    ON_COMMAND(ID_NEURALNET_CLASSIFY, OnNeuralnetClassify)
    ON_UPDATE_COMMAND_UI(ID_NEURALNET_CLASSIFY, OnUpdateNeuralnetClassify)
    ON_COMMAND(ID_NEURALNET_INITIALIZESIGMAS, OnNeuralnetInitializesigmas)
    ON_UPDATE_COMMAND_UI(ID_NEURALNET_INITIALIZESIGMAS, OnUpdateNeuralnetInitializesigmas)
    ON_COMMAND(ID_EDIT_FRACTALPARAMETERS, OnEditFractalparameters)
    ON_UPDATE_COMMAND_UI(ID_EDIT_FRACTALPARAMETERS, OnUpdateEditFractalparameters)
    ON_COMMAND(ID_VIEW_SEARCH, OnViewSearch)
    ON_UPDATE_COMMAND_UI(ID_VIEW_SEARCH, OnUpdateViewSearch)
    ON_COMMAND(ID_NEURALNET_OPTIMIZESIGMAS, OnNeuralnetOptimizesigmas)
    ON_UPDATE_COMMAND_UI(ID_NEURALNET_OPTIMIZESIGMAS, OnUpdateNeuralnetOptimizesigmas)
    ON_COMMAND(ID_NEURALNET_BATCHCLASSIFY, OnNeuralnetBatchclassify)
    ON_UPDATE_COMMAND_UI(ID_NEURALNET_BATCHCLASSIFY, OnUpdateNeuralnetBatchclassify)
    ON_COMMAND(ID_NEURALNET_MODEPARAMETERS, OnNeuralnetModeparameters)
    ON_UPDATE_COMMAND_UI(ID_NEURALNET_MODEPARAMETERS, OnUpdateNeuralnetModeparameters)
    ON_UPDATE_COMMAND_UI(ID_FILE_NEW, OnUpdateFileNew)
    ON_UPDATE_COMMAND_UI(ID_FILE_OPEN, OnUpdateFileOpen)
    ON_UPDATE_COMMAND_UI(ID_FILE_SAVE_AS, OnUpdateFileSaveAs)
    ON_COMMAND(ID_NEURALNET_SETREJECTIONTHRESHOLD, OnNeuralnetSetrejectionthreshold)
    ON_UPDATE_COMMAND_UI(ID_NEURALNET_SETREJECTIONTHRESHOLD,
OnUpdateNeuralnetSetrejectionthreshold)
    ON_COMMAND(ID_START_CAPTURE, OnStartCapture)
    ON_COMMAND(ID_VIEW_ZOOMEDRAWSIGNAL, OnViewZoomedrawsignal)
    ON_UPDATE_COMMAND_UI(ID_VIEW_ZOOMEDRAWSIGNAL, OnUpdateViewZoomedrawsignal)
    ON_COMMAND(ID_VIEW_NETWORKDETAILS, OnViewNetworkDetails)
    ON_UPDATE_COMMAND_UI(ID_VIEW_NETWORKDETAILS, OnUpdateViewNetworkDetails)
    ON_COMMAND(ID_VIEW_CLASSIFICATION_STATS, OnViewClassificationStats)
    ON_UPDATE_COMMAND_UI(ID_VIEW_CLASSIFICATION_STATS, OnUpdateViewClassificationStats)
    ON_COMMAND(ID_VIEW_TRAINING_RESULTS, OnViewTrainingResults)
    ON_UPDATE_COMMAND_UI(ID_VIEW_TRAINING_RESULTS, OnUpdateViewTrainingResults)
    ON_COMMAND(ID_SAVE_RESULTS, OnSaveResults)
    ON_UPDATE_COMMAND_UI(ID_SAVE_RESULTS, OnUpdateSaveResults)
    //ON_COMMAND(ID_NEURALNET_DIMREDUCTION, OnNeuralnetDimreduction)
    //ON_UPDATE_COMMAND_UI(ID_NEURALNET_DIMREDUCTION, OnUpdateNeuralnetDimreduction)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////
// CTAC_MMDoc construction/destruction

CTAC_MMDoc::CTAC_MMDoc()
{
    m_bFileInMemory=FALSE;
    m_bDocCreated=FALSE;
    m_nCurrentTransient=-1;
    m_bViewRawSignalFlag=TRUE;
    m_bViewVarDimTrajFlag=TRUE;
    m_bScrollToTransient=FALSE;
    m_bViewSegmentation=TRUE;
    m_bNotClassifying=TRUE;
    m_bViewTrainingResults=FALSE;//View Raw signal by default
    m_bResultsExist=FALSE;
    m_bInBatchProcess=FALSE;
    m_PNN.SetTransientArrayPointer(&m_TransientArray);
    //m_Princo.SetTransientArrayPointer(&m_TransientArray);
    m_nNumClassifyModes=1;
    m_dRejectThreshold=1e-20;// Default setting.
    m_pTransientListView = NULL;   // Make sure the pointer is NULL until the view is created.

    //set default printer margins, paper size, orientation
    m_PSDdlg.m_psd.Flags=m_PSDdlg.m_psd.Flags|PSD_RETURNDEFAULT;
    m_PSDdlg.m_psd.rtMargin.top=1000;
    m_PSDdlg.m_psd.rtMargin.bottom=1000;
    m_PSDdlg.m_psd.rtMargin.left=1000;
    m_PSDdlg.m_psd.rtMargin.right=1000;
    m_PSDdlg.DoModal();//dlg not displayed cuz of PSD_RETURNDEFAULT
```

```
    (m_PSDdlg.GetDevMode())->dmOrientation=DMORIENT_LANDSCAPE;
    (m_PSDdlg.GetDevMode())->dmPaperSize=DMPAPER_LETTER;

    m_PSDdlg.DoModal();
    m_PSDdlg.m_psd.Flags=m_PSDdlg.m_psd.Flags^PSD_RETURNDEFAULT;//^is XOR
    AfxGetApp()->SelectPrinter(m_PSDdlg.m_psd.hDevNames,
        m_PSDdlg.m_psd.hDevMode, TRUE);



}

CTAC_MMDoc::~CTAC_MMDoc()
{
    CPageSetupDialog Temp;//select default printer before exiting
    AfxGetApp()->SelectPrinter(Temp.m_psd.hDevNames,
        Temp.m_psd.hDevMode, FALSE);
}

BOOL CTAC_MMDoc::OnNewDocument()
{
    TRACE("Don, OnNewDocument being executed.\n");

    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMDoc serialization

void CTAC_MMDoc::Serialize(CArchive& ar)
{
    m_Extract.Serialize(ar);
    m_TransientArray.Serialize(ar);
    m_PNN.Serialize(ar);
    m_Princo.Serialize(ar);
    m_SOFM.Serialize(ar);

    if (ar.IsStoring())
    {
        ar << m_nCurrentTransient;
    }
    else
    {
        ar >> m_nCurrentTransient;//position at transient at last save
        m_nRawSignal=new int[m_Extract.GetRawFileSize()];//mem for raw signal
        m_nSquelchChannel=new int[m_Extract.GetRawFileSize()];//Mem for squelch channel.
        m_dVarDimTraj=new double[m_Extract.GetRawFileSize()];
        m_dTransientModel=new double[m_Extract.GetTransientModelSize()];
        m_dThreshholds=new double[1];//dummy new so that delete [] m_dThreshholds
                                                    //doesn't fail when first called
        m_bDocCreated=TRUE;
        if(m_TransientArray.GetSize()>0) {
            m_bFileInMemory=TRUE;
//          UpdateStatusBar();
            ViewNeedsUpdating();
        }
        else {
            m_nCurrentTransient=-1;
```

```
                UpdateStatusBarNoTransients();
                UpdateAllViews(NULL);
            }
        }
}

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMDoc diagnostics

#ifdef _DEBUG
void CTAC_MMDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CTAC_MMDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMDoc commands


void CTAC_MMDoc::OnFileNew()
{
    TRACE("Don, OnFileNew being executed.\n");

    // Save the document if it has been modified
    // and needs to be saved.
    if (SaveModified() == FALSE)
        return;


    // Start a new document.
    // Ask for the new documents parameters.

    CNewDatabase NewDatabaseDlg;//declare dialog class object

    //set defaults
    NewDatabaseDlg.m_RawFileSize=8192;
    NewDatabaseDlg.m_TransientSize=1536;
    NewDatabaseDlg.m_WindowSize=512;//Segmentation window size
    NewDatabaseDlg.m_VariancePairs=10;//Segmentation variance pairs
    NewDatabaseDlg.m_Threshold=8.0;
    NewDatabaseDlg.m_ModelWindowSize=768;
    NewDatabaseDlg.m_ModelVariancePairs=8;
    NewDatabaseDlg.m_ModelWindowShift= 48;
    NewDatabaseDlg.m_nSquelchTrig = 4000;
    NewDatabaseDlg.m_nSquelchTrigWidth = 100;
    NewDatabaseDlg.m_nSquelchDelay = 850;
    NewDatabaseDlg.m_nTrigWinSize = 400;
    NewDatabaseDlg.m_TrigType = WINDOWEDVARIANCE;
    NewDatabaseDlg.m_nTrigScheme = 0; //0 is absolute triggering
    NewDatabaseDlg.m_nAdjust = 0.5;
    NewDatabaseDlg.m_nReducedDim=0; //0 means no pca processing
    NewDatabaseDlg.m_nTransientNum=20; //sofm processing when more than 20 transients per class


    if (NewDatabaseDlg.DoModal()==IDOK)
    {
        // Create the new document, deleting the old one.
        CTAC_MMDoc::OnNewDocument();
        m_Extract.SetParams(NewDatabaseDlg.m_RawFileSize,
```

```
                                        NewDatabaseDlg.m_TransientSize,
                                        NewDatabaseDlg.m_WindowSize,
                                        NewDatabaseDlg.m_VariancePairs,
                                        NewDatabaseDlg.m_Threshold,
                                        NewDatabaseDlg.m_ModelWindowSize,
                                        NewDatabaseDlg.m_ModelVariancePairs,
                                        NewDatabaseDlg.m_ModelWindowShift,
                                        NewDatabaseDlg.m_nSquelchTrig,
                                        NewDatabaseDlg.m_nSquelchDelay,
                                        NewDatabaseDlg.m_nSquelchTrigWidth,
                                        NewDatabaseDlg.m_nTrigWinSize,
                                        NewDatabaseDlg.m_TrigType,
                                        NewDatabaseDlg.m_nTrigScheme,
                                        NewDatabaseDlg.m_nAdjust);
            m_Princo.SetReducedDim(NewDatabaseDlg.m_nReducedDim);
            m_SOFM.SetTransientNum(NewDatabaseDlg.m_nTransientNum);

            m_nRawSignal = new int[NewDatabaseDlg.m_RawFileSize];//mem for raw siganl
            m_nSquelchChannel = new int[NewDatabaseDlg.m_RawFileSize];//Mem for squelch signal.
            m_dVarDimTraj=new double[NewDatabaseDlg.m_RawFileSize];
            m_dTransientModel=new double[m_Extract.GetTransientModelSize()];
            m_dThreshholds=new double[1];//dummy new so that delete [] m_dThreshholds
                                                    //doesn't fail when first called
            m_PNN.Initialize(m_Extract.GetTransientModelSize());
            //m_SOFM.Initialize(m_Extract.GetTransientModelSize());

            if((NewDatabaseDlg.m_nReducedDim!=0)
                &&(NewDatabaseDlg.m_nReducedDim!=m_Extract.GetTransientModelSize()) ) //pca needed
            {
                m_PNN.Initialize(NewDatabaseDlg.m_nReducedDim);
            }

            m_nCurrentTransient=-1;
            m_bDocCreated=TRUE;
            UpdateStatusBarNoTransients();

            // Clean up the list view if one exists.
            if (m_pTransientListView != NULL)
            {
                m_pTransientListView -> RemoveAllItems();
            }

            UpdateAllViews(NULL);
        }
}

void CTAC_MMDoc::DeleteContents()
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("Don, DeleteContents being executed.\n");

    if(m_bDocCreated) {
        delete [] m_nRawSignal;
        delete [] m_nSquelchChannel;
        delete [] m_dVarDimTraj;
        delete [] m_dTransientModel;
        delete [] m_dThreshholds;
        m_bFileInMemory=FALSE;
        m_bDocCreated=FALSE;
    }
    int ctr=m_TransientArray.GetSize();
    while (ctr--) {
        delete m_TransientArray.GetAt(ctr);
    }
    m_TransientArray.RemoveAll();
```

```
    DeleteResultsArray();

    CDocument::DeleteContents();
    UpdateAllViews(NULL);
}

void CTAC_MMDoc::OnDatabaseAddtransient()
{
    CString PathName;
    CString strFileNameBuff;  // Holds the list of pathnames for selected files.
    POSITION pos;             // Holds the position filenames in the filename buffer.
    int nFiles = 0;       // Number of files selected.
    int nFileToAdd = 0;  // Current File to be added.
    bool bAcceptAll = FALSE;  // TRUE if accepting all selected transients.
    bool bCancel = FALSE;     // TRUE to stop adding transients.
    int index;                // Index to the transient array.
    CString strLastClassName;  // Holds the class name of the last added trasient.
    int nLastClassNum;  // Holds the class number of the last added trasient.

    int TransientModelSize=m_Extract.GetTransientModelSize();
    //TRACE("TransientModelSize in Add transient is %d\n",TransientModelSize);

    CFileDialog FileDlg(TRUE,// TRUE means file open.
                                NULL,                       // Not used for file open.
                                NULL,                       // No initial filename.
                                OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT |
                                OFN_FILEMUSTEXIST | OFN_PATHMUSTEXIST,
                                "All compatible files (*.raw; *.pcm)| *.raw; *.pcm|"
                                "Raw Files (*.raw)|*.raw|PCM Files (*.pcm)|*.pcm|"
                                "All Files (*.*)|*.*||", // Selects which files to show.
                                NULL);

    //TRACE("Chris, adding a transient.\n");

    FileDlg.m_ofn.lpstrTitle="Add Transient: Select a File";
    FileDlg.m_ofn.Flags |= OFN_ALLOWMULTISELECT; // Allows multiple files to be
                                                 // selected at once.
    FileDlg.m_ofn.lpstrFile = strFileNameBuff.GetBuffer(20000); // Select the filename buffer.
    FileDlg.m_ofn.nMaxFile = 20000; // Sets the size of the buffer.


    if (FileDlg.DoModal()==IDOK)
    {
        HCURSOR Wait=AfxGetApp()->LoadStandardCursor(IDC_WAIT);
        ::SetCursor(Wait);

        // Find the number of files selected.
        pos = FileDlg.GetStartPosition( );// Gets the position of the first filename in the buffer.
        while (NULL != pos)        // Step through the filename buffer
        {
            PathName = FileDlg.GetNextPathName(pos); // Go to the next file name in the buffer.
            nFiles++;
        }


        // Add the file/s to the database.
        pos = FileDlg.GetStartPosition(); // Finds the position of the first filename in the buffer.
        while (nFileToAdd < nFiles)
        {
            PathName = FileDlg.GetNextPathName(pos); // Retrieve the filename.
            nFileToAdd++;                            // Next file.

            // Process each of the selected files.
            if (ReadRawSignal(PathName,TRUE)) //sees if file exists, is compatible, loads info for display
```

```
{
    m_bInBatchProcess=TRUE;// disables all menu functions
    m_nTransientBeginsHere=m_Extract.FindTransient(m_nRawSignal,
        m_nSquelchChannel, m_CurrentTranFileType, m_dVarDimTraj, m_dTransientModel);
    //Filltransientmodel on each transinet, but PCA processing must be for all transients.

    m_bInBatchProcess=FALSE;
    if (m_nTransientBeginsHere==-1)
    {
        AfxMessageBox("No transient found. Try adjusting settings."
            ,MB_OK);
    }
    else
    {
        m_bFileInMemory=TRUE;
        m_bScrollToTransient=TRUE;
        m_bViewSegmentation=TRUE;  // show tran Segmentation info when adding
        UpdateStatusBarNewTransient(); // Show "New transient in the status bar.
        UpdateAllViews(NULL);       // Display the transient.


        if (bAcceptAll == FALSE)
        {
            // Get the class name for this transient
            if (nFileToAdd != nFiles)
            {
                CEnterClassMultiDialog EnterClassMultiDlg;
                EnterClassMultiDlg.m_Num_Files_Selected = nFiles;
                EnterClassMultiDlg.m_Transient_File_Number = nFileToAdd;
                EnterClassMultiDlg.m_strClassName = m_TransmitterMake; // Default class name.
                EnterClassMultiDlg.m_pDoc = this; // Allow the dialog to access the documents data.


                switch (EnterClassMultiDlg.DoModal())
                {
                case IDC_ACCEPT_ALL: // Accept all button pressed.
                    bAcceptAll = TRUE;

                case IDOK: // Accept button pressed.

                    // Is this a new class?
                    if                                          ((index                                    =
m_TransientArray.SearchForClassName(EnterClassMultiDlg.m_strClassName)) != -1)
                    {
                        // Use the existing transient class #.
                        m_nTransientClass = m_TransientArray.GetAt(index) -> GetTransientClass();
                        nLastClassNum = m_nTransientClass;
                    }
                    else
                    {
                        // Use the next available class #.
                        m_nTransientClass = m_TransientArray.FindNumClasses();
                        nLastClassNum = m_nTransientClass;
                    }

                    // Use the class name from the dialog box.
                    m_strTransientClassName = EnterClassMultiDlg.m_strClassName;
                    strLastClassName = EnterClassMultiDlg.m_strClassName;

                    // Add the new transient to the storage array.
                    StoreTransientData(m_nCurrentTransient+1);

                    IncrementCurrentTransient();

                    SetModifiedFlag(TRUE);
```

```
                UpdateStatusBar();
                break;

            case IDC_SKIP: // Skip button pressed.
                break;

            case IDC_DELETE: // Delete button pressed. (Doesn't work)
                TRACE("Chris, attempting to delete the file.\n");
                if (AfxMessageBox("Are you sure you want to delete the transient file?",
                    MB_YESNO)==IDYES)
                {
                    LPCTSTR lpFilename;
                    lpFilename = PathName;
                    // Delete the transient file.
                    if (DeleteFile(lpFilename) != -1)
                        TRACE ("Chris, couldn't delete the file.\n");

                    // may want to bring up the dialog box
                    // for this transient again.
                }
                break;

            case IDCANCEL: // Cancel button pressed.
                // Display last transient in the array.
                bCancel = TRUE;
                break;

            }

        }
        else
        {
            CEnterClassDialog EnterClassDlg;
            EnterClassDlg.m_Num_Files_Selected = nFiles;
            EnterClassDlg.m_strClassName = m_TransmitterMake; // Default class name.
            EnterClassDlg.m_pDoc = this; // Allow the dialog to access the documents data.


            switch (EnterClassDlg.DoModal())
            {
            case IDOK: // Accept button pressed.
                // Is this a new class?
                if ((index = m_TransientArray.SearchForClassName(EnterClassDlg.m_strClassName)) !=
-1)

                {
                    // Use the existing transient class #.
                    m_nTransientClass = m_TransientArray.GetAt(index) -> GetTransientClass();
                }
                else
                {
                    // Use the next available class #.
                    m_nTransientClass = m_TransientArray.FindNumClasses();
                }

                // Use the name from the dialog box.
                m_strTransientClassName = EnterClassDlg.m_strClassName;

                // Add the new transient to the storage array.
                StoreTransientData(m_nCurrentTransient+1);

                IncrementCurrentTransient();

                SetModifiedFlag(TRUE);
                UpdateStatusBar();
                break;
```

```
                                case IDC_SKIP: // Skip button pressed.
                                    break;

                                case IDC_DELETE: // Delete button pressed. (Doesn't work)
                                    TRACE("Chris, attempting to delete the file.\n");
                                    if (AfxMessageBox("Are you sure you want to delete the transient file?",
                                        MB_YESNO)==IDYES)
                                    {
                                        // Delete the transient file.
                                        if (DeleteFile(PathName) != -1)
                                            TRACE ("Chris, couldn't delete the file.\n");

                                        // may want to bring up the dialog box
                                        // for this transient again.
                                    }
                                    break;

                                case IDCANCEL: // Cancel button pressed.
                                    bCancel = TRUE;
                                    break;
                                }
                            }
                        }
                        else
                        {
                            // Accepting every transient.
                            m_TransientArray.GetAt(m_nCurrentTransient)
                                ->SetTransientClass(nLastClassNum);
                            m_TransientArray.GetAt(m_nCurrentTransient)
                                -> SetTransientClassName(strLastClassName);

                            // Add the new transient to the storage array.
                            StoreTransientData(m_nCurrentTransient+1);

                            IncrementCurrentTransient();

                            SetModifiedFlag(TRUE);
                            UpdateStatusBar();
                        }
                        if (bCancel == TRUE)
                        {
                            UpdateStatusBar();
                            break; // Exit while loop, add no more transients.
                        }
                    }
                }
            }
            m_PNN.SetNumCases(m_TransientArray.GetSize());
            m_PNN.SetNumClasses(m_TransientArray.FindNumClasses());

            HCURSOR Arrow=AfxGetApp()->LoadStandardCursor(IDC_ARROW);
            ::SetCursor(Arrow);

            UpdateListView();
        }
}

BOOL CTAC_MMDoc::ReadRawSignal(CString FilePath, BOOL IsAdding)
{
    int RawFileSize=m_Extract.GetRawFileSize();
    int Temp=-1;

    m_strPathName = FilePath;
```

```
if (FilePath.Right(3).CompareNoCase("pcm") == 0)
{
	m_CurrentTranFileType = PCM;

	// Read the pcm file.
	CFile pcmFile;
	short int* Buffer; // Needs to be a short int because the pcm file
				// contains 16-bit values.
	Buffer = new short int[RawFileSize*2];


	CFileException fileException;
	if (pcmFile.Open(FilePath, CFile::modeRead | CFile::typeBinary,
		&fileException) == 0)
	{
		TRACE( "Chris, can't open file %s, error = %u\n",
			FilePath, fileException.m_cause );
		return FALSE;
	}


	// Place the PCM file in the buffer.
	if (pcmFile.Read(Buffer, (RawFileSize*4)) != ((UINT) RawFileSize*4))
	{
		AfxMessageBox("Error reading pcm file.",MB_OK);
		return FALSE;
	}
	pcmFile.Close();


	// Fill m_nRawSignal with the signal info.
	m_nHighestValue = 0;
	m_nLowestValue = 65535;
	for (int i = 0; i < RawFileSize; i++)
	{
		if ((Buffer[2*i] + 32768) < m_nHighestValue)
		{
			m_nHighestValue = Buffer[2*i] + 32768;
		}
		else if ((Buffer[2*i] + 32768) > m_nLowestValue)
		{
			m_nLowestValue = Buffer[2*i] + 32768;
		}
		m_nRawSignal[i] = Buffer[2*i] + 32768;
	}

	// Fill the squelch channel buffer;
	for (i = 0; i < RawFileSize; i++)
	{
		m_nSquelchChannel[i] = Buffer[2*i + 1];
	}

	delete Buffer;

	// Leave the transmitter info blank for now.
	m_TransmitterMake="";
	m_TransmitterModel="";
	m_TransmitterSerial="";
	m_Date="";
	m_Time="";

}
else
{
	m_CurrentTranFileType = RAW;
```

```
            // Read the raw file.
            ifstream tfile(FilePath, ios::nocreate);
            if (tfile.fail())
            {
                AfxMessageBox("Error opening file.",MB_OK);
                return FALSE;
            }
            m_nHighestValue=0;
            m_nLowestValue=65535;
            for(int ctr=0; ctr<RawFileSize; ctr++)
            {
                tfile >> Temp;
                if (Temp<0||Temp>65536||tfile.eof())
                {
                    AfxMessageBox("Incompatibe file format.",MB_OK);
                    return FALSE;
                }
                if(Temp>m_nHighestValue) {
                    m_nHighestValue=Temp;
                }
                else if(Temp<m_nLowestValue) {
                    m_nLowestValue=Temp;
                }
                m_nRawSignal[ctr]=Temp;
                Temp=-1;
            }
            if(IsAdding)
            {
                char temp[50];

            //  Temp=65000;
            //  while(Temp>=10000&&Temp<=65536) {
            //      tfile >> Temp;
            //  }

                tfile.getline(temp,50,'\n');
                tfile.getline(temp,50,'\n');
                m_TransmitterMake=temp;
                tfile.getline(temp,50,'\n');
                m_TransmitterModel=temp;
                tfile.getline(temp,50,'\n');
                m_TransmitterSerial=temp;
                tfile.getline(temp,50,'\n');
                tfile.getline(temp,50,'\n');
                m_Date=temp;
                tfile.getline(temp,50,'\n');
                m_Time=temp;
            }
            tfile.close();
        }

        return TRUE;
}

void CTAC_MMDoc::UpdateStatusBar()
{
        CString str;
        CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;
        CStatusBar *pStatus=&pFrame->m_wndStatusBar;

        if (pStatus)
        {
            if (m_TransientArray.GetSize() > 0)
            {
```

```
            // Display the current transients number and class.
            str.Format("Transient #%d of %d.   Class: %s",(m_nCurrentTransient+1),
                m_TransientArray.GetSize(),
                m_TransientArray.GetAt(m_nCurrentTransient)->GetClassName());
            pStatus->SetPaneText(1,str);
        }
        else
        {
            // There aren't any transients stored yet.
            str.Format(" No transients in memory.");
            pStatus->SetPaneText(1,str);
        }
    }
}

void CTAC_MMDoc::UpdateStatusBarNoTransients()
{
    CString str;
    CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;
    CStatusBar *pStatus=&pFrame->m_wndStatusBar;
    if (pStatus) {
        str.Format(" No transients in memory.");
        pStatus->SetPaneText(1,str);
    }
}

void CTAC_MMDoc::UpdateStatusBarUnknown()
{
    CString str;
    CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;
    CStatusBar *pStatus=&pFrame->m_wndStatusBar;
    if (pStatus) {
        str.Format(" Unknown Transient.");
        pStatus->SetPaneText(1,str);
    }
}

void CTAC_MMDoc::OnDatabaseDeletetransient()
{
    if (AfxMessageBox("Are you sure you want to remove the current transient from the database?",
        MB_YESNO)==IDYES)
    {
        DeleteCurrentTransient();
    }
}
void CTAC_MMDoc::ViewNeedsUpdating()
{
    // Display the wait cursor.
    CWaitCursor wait;

    if (m_bViewRawSignalFlag)
    {
        CString PathName=m_TransientArray.GetAt(m_nCurrentTransient)->
            GetRawFilePath();
        if (!ReadRawSignal(PathName,FALSE))
        {
            for(int ctr=0; ctr<m_Extract.GetRawFileSize(); ctr++)
            {
                m_nRawSignal[ctr]=0;
            }
        }
    }
    if (m_bViewVarDimTrajFlag)
    {
        m_Extract.FillVarDimArray(m_nRawSignal,m_dVarDimTraj,m_bViewSegmentation,
```

```
                m_TransientArray.GetAt(m_nCurrentTransient)->GetTransientBeginsHere(),
                m_TransientArray.GetAt(m_nCurrentTransient)->GetTransientModel());
    }

    m_CurrentTranFileType = m_TransientArray.GetAt(m_nCurrentTransient) -> GetFileType();
    m_nTransientBeginsHere=m_TransientArray.GetAt(m_nCurrentTransient) -> GetTransientBeginsHere();
    m_Date = m_TransientArray.GetAt(m_nCurrentTransient) -> GetDate();
    m_Time = m_TransientArray.GetAt(m_nCurrentTransient) -> GetTime();
    m_TransmitterMake = m_TransientArray.GetAt(m_nCurrentTransient) -> GetTransmitterMake();
    m_TransmitterModel = m_TransientArray.GetAt(m_nCurrentTransient) -> GetTransmitterModel();
    m_TransmitterSerial = m_TransientArray.GetAt(m_nCurrentTransient) -> GetTransmitterSerial();
    m_nTransientClass = m_TransientArray.GetAt(m_nCurrentTransient) -> GetTransientClass();
    m_strTransientClassName = m_TransientArray.GetAt(m_nCurrentTransient) -> GetClassName();


    m_bScrollToTransient=TRUE;

    UpdateAllViews(NULL);
}

void CTAC_MMDoc::OnViewPrev()
{
    if (m_nCurrentTransient>0)
    {
        DecrementCurrentTransient();
        UpdateListView();
        ViewNeedsUpdating();
    }
}

void CTAC_MMDoc::OnViewNext()
{
    if (m_nCurrentTransient<m_TransientArray.GetUpperBound())
    {
        IncrementCurrentTransient();
        UpdateListView();
        ViewNeedsUpdating();
    }
}

void CTAC_MMDoc::OnViewRawsignal()
{
    if(m_bViewRawSignalFlag)
        m_bViewRawSignalFlag=FALSE;
    else
        m_bViewRawSignalFlag=TRUE;
    UpdateAllViews(NULL);
}

void CTAC_MMDoc::OnViewFractaltrajectory()
{
    if(m_bViewVarDimTrajFlag)
        m_bViewVarDimTrajFlag=FALSE;
    else
        m_bViewVarDimTrajFlag=TRUE;
    UpdateAllViews(NULL);
}

void CTAC_MMDoc::OnFilePageSetup()
{
    if(m_PSDdlg.DoModal()==IDOK) {
        AfxGetApp()->SelectPrinter(m_PSDdlg.m_psd.hDevNames,
            m_PSDdlg.GetDevMode(), FALSE);
    }
}
```

```
void CTAC_MMDoc::OnNeuralnetClassify()
{
	CString PathName;
	int TransientModelSize=m_Extract.GetTransientModelSize();
	double *SummationNeuron, Activation;
	int nPredictedClass, index;
	TRACE("in OnNeuralnetClassify.\n");

	CFileDialog FileDlg(TRUE, "raw", "*.*");//TRUE means file open,

	FileDlg.m_ofn.lpstrTitle="Classify Transient: Select a File";

	if (FileDlg.DoModal()==IDOK)
	{
		HCURSOR Wait=AfxGetApp()->LoadStandardCursor(IDC_WAIT);
			::SetCursor(Wait);
		PathName=FileDlg.GetPathName();
		if (ReadRawSignal(PathName,TRUE)) //sees if file exists, is compatible
		{
			CClassifyDialog ClassifyDlg;
			if(m_nNumClassifyModes==1)
			{
				m_bInBatchProcess=TRUE; //disables menu functions
				m_nTransientBeginsHere=m_Extract.FindTransient(m_nRawSignal,
					m_nSquelchChannel, m_CurrentTranFileType, m_dVarDimTraj, m_dTransientModel);
				m_bInBatchProcess=FALSE;
				if (m_nTransientBeginsHere==-1)
				{
					AfxMessageBox("No transient found. Try adjusting settings."
						,MB_OK);
					return;
				}
				SummationNeuron=new double[m_PNN.GetNumClasses()];

				nPredictedClass = m_PNN.Classify(m_dTransientModel,
					SummationNeuron,-1, m_dRejectThreshold, &Activation);
				ClassifyDlg.m_strPredictedClass = m_TransientArray.GetClassName(nPredictedClass);
				ClassifyDlg.m_Confidence=100.0*ComputeConfidence
					(SummationNeuron, nPredictedClass);
				ClassifyDlg.m_Activation=Activation;
				delete [] SummationNeuron;
			}
			else
			{
				double Confidence;
				nPredictedClass=MultiModeClassify(&Confidence, &Activation);
				ClassifyDlg.m_strPredictedClass = m_TransientArray.GetClassName(nPredictedClass);
				ClassifyDlg.m_Confidence=Confidence;
				ClassifyDlg.m_Activation=Activation;
			}

			m_bScrollToTransient=TRUE;
			m_bViewSegmentation=TRUE;//show tran Segmentation info
			m_bNotClassifying=FALSE;
			UpdateAllViews(NULL);
			UpdateStatusBarUnknown();

			if(ClassifyDlg.DoModal()==IDOK) //Add to database selected
			{
				CAddUnknownDialog AddUnknownDlg;
				AddUnknownDlg.m_TransmitterMake=m_TransmitterMake;
				AddUnknownDlg.m_TransmitterModel=m_TransmitterModel;
				AddUnknownDlg.m_TransmitterSerial=m_TransmitterSerial;
				AddUnknownDlg.m_Date=m_Date;
```

```
                    AddUnknownDlg.m_Time=m_Time;
                    AddUnknownDlg.m_strClassName = m_TransientArray.GetClassName(nPredictedClass);
                    AddUnknownDlg.m_Position=m_TransientArray.GetSize()+1;
                    AddUnknownDlg.m_pDoc = this; // Allows the dialog to access the transient array and look up class names.

                    if(AddUnknownDlg.DoModal()==IDOK) // Add the class to the array.
                    {
                        // Is this a new class?
                        if ((index = m_TransientArray.SearchForClassName(AddUnknownDlg.m_strClassName)) != -1)
                        {
                            // Use the existing transient class #.
                            m_nTransientClass = m_TransientArray.GetAt(index) -> GetTransientClass();
                        }
                        else
                        {
                            // Use the next available class #.
                            m_nTransientClass = m_TransientArray.FindNumClasses();

                        }
                        // Use the transient data from the dialog box.
                        m_TransmitterMake = AddUnknownDlg.m_TransmitterMake;
                        m_TransmitterModel = AddUnknownDlg.m_TransmitterModel;
                        m_TransmitterSerial = AddUnknownDlg.m_TransmitterSerial;
                        m_Date = AddUnknownDlg.m_Date;
                        m_Time = AddUnknownDlg.m_Time;
                        m_strTransientClassName = AddUnknownDlg.m_strClassName;
                        // Store the transient in the transient array.
                        StoreTransientData(AddUnknownDlg.m_Position-1);


                        m_nCurrentTransient=AddUnknownDlg.m_Position-1;
                        UpdateStatusBar();
                        m_bScrollToTransient=TRUE;
                        m_bViewSegmentation=TRUE;//show tran Segmentation info
                        SetModifiedFlag(TRUE);
                        m_PNN.SetNumCases(m_TransientArray.GetSize());
                        m_PNN.SetNumClasses(m_TransientArray.FindNumClasses());

                    }
                }
                m_bNotClassifying=TRUE;
            }
            UpdateStatusBar();
            ViewNeedsUpdating();
        }
        HCURSOR Arrow=AfxGetApp()->LoadStandardCursor(IDC_ARROW);
        ::SetCursor(Arrow);
}

double CTAC_MMDoc::ComputeConfidence(double *SummationNeuron,
                                                    int PredictedClass)
{
        double Confidence, Sum=0.0;
        for(int ctr=0; ctr<m_PNN.GetNumClasses(); ctr++) {
            Sum+=SummationNeuron[ctr];
        }
        if(Sum==0.0||PredictedClass==-1)
            return 0.0;

        Confidence=SummationNeuron[PredictedClass]/Sum;
        return Confidence;
}


void CTAC_MMDoc::OnNeuralnetInitializesigmas()
```

```
{
    if (m_PNN.GetSigmasOptimized()==TRUE) {
        if (AfxMessageBox("Destroy Previous Sigma Optimization?"
            ,MB_YESNO)==IDNO) {
            return;
        }
    }

    if (m_bViewTrainingResults)
        OnViewZoomedrawsignal();
    //TRACE("m_nReducedDim=%i ",m_Princo.GetReducedDim());
    //TRACE("m_nTransientModelSize=%i ",m_Extract.GetTransientModelSize());

    //perform pca on training set here
    if((m_Princo.GetReducedDim()!=0)
        &&(m_Princo.GetReducedDim()!=m_Extract.GetTransientModelSize()))
    {
        unsigned int t1, t2, pca_time;
        t1=timeGetTime();

        //TRACE("PCA on training set\n");

        m_Princo.SetTransientArrayPointer(&m_TransientArray);
        m_Princo.SetNumCases(m_TransientArray.GetSize() );
        m_Princo.SetTransientModelSize( m_Extract.GetTransientModelSize() );
        m_Princo.PCAProcess();
        m_Princo.factors(&m_TransientArray);

        t2=timeGetTime();
        pca_time=t2-t1;
        TRACE("PCA processing on training set: %i (ms)\n", pca_time);
    }

    //perform SOFM to reduce the size of traning set here

    //first count number of cases for individual classes to prepare SOFM
    int IndCase,IndClass,m_nNumClasses, m_nNumCases, NoSamplesInClass;
    m_nNumClasses = m_PNN.GetNumClasses();
    m_nNumCases = m_PNN.GetNumCases();
    TRACE("Total number of classes = %i, number of cases =%i\n", m_nNumClasses,m_nNumCases);

    //assume all classes have the same number of training cases
    NoSamplesInClass = m_nNumCases/m_nNumClasses;


    m_SOFM.SetTransientArrayPointer(&m_TransientArray);
    m_SOFM.SetNumClasses(m_nNumClasses);
    m_SOFM.SetNoSamplesInClass(NoSamplesInClass);
    m_SOFM.SetCurrentClassIndex(0);
    m_SOFM.Initialize(m_Extract.GetTransientModelSize());

    for(IndClass=0; IndClass<m_nNumClasses;IndClass++)//for each class
    {

        int *DeleteTransientIndex;
        DeleteTransientIndex=new int[m_SOFM.GetTransientNum()]; //store the index of transients to be deleted

        if( (m_SOFM.GetTransientNum()>0)
            &&(m_SOFM.GetTransientNum()<=NoSamplesInClass) )//needs SOFM
        {
            TRACE("Performing SOFM for class %i\n",IndClass);

            m_SOFM.SOFMTrain();
            m_SOFM.Cluster(DeleteTransientIndex);
            //delete the training cases outside the cluster
```

```
                for(int i=0; i<(NoSamplesInClass-m_SOFM.GetTransientNum()); i++)
                {
                    m_nCurrentTransient = DeleteTransientIndex[i];
                    TRACE("Currently deleting transient #%i\n",m_nCurrentTransient);
                    DeleteCurrentTransient();
                    //delete m_TransientArray.GetAt(m_nCurrentTransient);
                    //m_TransientArray.RemoveAt(m_nCurrentTransient);
                }
                delete [] DeleteTransientIndex;
                m_SOFM.SetCurrentClassIndex( (IndClass+1)*m_SOFM.GetTransientNum());

            }
        }


        CEnterSigmaInitParamsDialog EnterSigmaInitParamsDlg;
        if(m_PNN.GetSigmasInitialized()==TRUE) {
            EnterSigmaInitParamsDlg.m_LowerSearchLimit=0.9*m_PNN.GetFirstSigma();
            EnterSigmaInitParamsDlg.m_UpperSearchLimit=1.1*m_PNN.GetFirstSigma();
        }
        else {
            EnterSigmaInitParamsDlg.m_LowerSearchLimit=0.01;
            EnterSigmaInitParamsDlg.m_UpperSearchLimit=3;
        }
        EnterSigmaInitParamsDlg.m_NumSearchPoints=30;

        if (EnterSigmaInitParamsDlg.DoModal()==IDOK) {

            CSigmaInitDialog SigmaInitDlg;
            SigmaInitDlg.m_TransientArray=&m_TransientArray;
            SigmaInitDlg.m_PNN=&m_PNN;
            m_PNN.SetLowerSearchLimit(EnterSigmaInitParamsDlg.m_LowerSearchLimit);
            m_PNN.SetUpperSearchLimit(EnterSigmaInitParamsDlg.m_UpperSearchLimit);
            SigmaInitDlg.m_nNumSearchPoints=
                EnterSigmaInitParamsDlg.m_NumSearchPoints;
            if(SigmaInitDlg.DoModal()==IDOK) {
                SetModifiedFlag(TRUE);
                DeleteResultsArray();
                m_PNN.FillTrainingResultsArray(&m_ResultsArray);
                m_bResultsExist=TRUE;
                m_bTrainingResults=TRUE;// false if results are from batch classify
            }

        }
}

void CTAC_MMDoc::OnNeuralnetOptimizesigmas()
{
        if (m_bViewTrainingResults)
            OnViewZoomedrawsignal();

        CTrainSigmasOptDialog SigmasOptDlg;
        SigmasOptDlg.m_TransientArray=&m_TransientArray;
        SigmasOptDlg.m_PNN=&m_PNN;
        SigmasOptDlg.m_dImprovementTolerance=1.0e-250;
        if(SigmasOptDlg.DoModal()==IDOK) {
            SetModifiedFlag(TRUE);
            DeleteResultsArray();
            m_PNN.FillTrainingResultsArray(&m_ResultsArray);
            m_bResultsExist=TRUE;
            m_bTrainingResults=TRUE;// false if results are from batch classify
        }
}
```

```
void CTAC_MMDoc::OnEditFractalparameters()
{
	int OldModelWindowShift;
	int NewModelSize=0;

	CFractalParamDialog FractalParamDlg;
	FractalParamDlg.m_RawFileSize=m_Extract.GetRawFileSize();
	FractalParamDlg.m_TransientSize=m_Extract.GetTransientSize();
	FractalParamDlg.m_WindowSize=m_Extract.GetWindowSize();//Segmentation window
	FractalParamDlg.m_VariancePairs=m_Extract.GetVariancePairs();//Segmentation
	FractalParamDlg.m_Threshold=m_Extract.GetThreshold();
	FractalParamDlg.m_ModelWindowSize=m_Extract.GetModelWindowSize();
	FractalParamDlg.m_ModelVariancePairs=m_Extract.GetModelVariancePairs();
	FractalParamDlg.m_ModelWindowShift=m_Extract.GetModelWindowShift();
	OldModelWindowShift = m_Extract.GetModelWindowShift();

	FractalParamDlg.m_nSquelchTrigger = m_Extract.GetSquelchTrigger();
	FractalParamDlg.m_nSquelchWidth = m_Extract.GetSquelchWidth();
	FractalParamDlg.m_nSquelchDelay = m_Extract.GetSquelchDelay();
	FractalParamDlg.m_nTrigWinSize = m_Extract.GetTrigWinSize();
	FractalParamDlg.m_triggerType = m_Extract.GetTriggerType();
	FractalParamDlg.m_nTrigScheme = m_Extract.GetTrigScheme();
	FractalParamDlg.m_nAdjust = m_Extract.GetAdjust();

	FractalParamDlg.m_nReducedDim = m_Princo.GetReducedDim();
	FractalParamDlg.m_nTransientNum = m_SOFM.GetTransientNum();

	if(FractalParamDlg.DoModal()==IDOK)
	{
		CEditFPProgressDialog EditFPProgressDlg;
		EditFPProgressDlg.m_ProgressBarLimit=m_TransientArray.GetSize();

		// Grab the new segmentation parameters from the
		// dialog box.
		if (FractalParamDlg.m_SetSegmentationParams)
		{
			m_Extract.SetWindowSize(FractalParamDlg.m_WindowSize);
			m_Extract.SetVariancePairs(FractalParamDlg.m_VariancePairs);
			m_Extract.SetThreshold(FractalParamDlg.m_Threshold);
			m_Extract.SetSquelchTrigger(FractalParamDlg.m_nSquelchTrigger);
			m_Extract.SetSquelchWidth(FractalParamDlg.m_nSquelchWidth);
			m_Extract.SetSquelchDelay(FractalParamDlg.m_nSquelchDelay);
			m_Extract.SetTrigWinSize(FractalParamDlg.m_nTrigWinSize);
			m_Extract.SetTriggerType(FractalParamDlg.m_triggerType);
			m_Extract.SetTrigScheme(FractalParamDlg.m_nTrigScheme);
			m_Extract.SetAdjust(FractalParamDlg.m_nAdjust);
		}


		// Grab the feature extraction parameters if they
		// are to be changed.
		if (FractalParamDlg.m_SetModelParams)
		{
			m_Extract.SetModelWindowSize(FractalParamDlg.m_ModelWindowSize);
			m_Extract.SetModelVariancePairs
				(FractalParamDlg.m_ModelVariancePairs);
			m_Extract.SetModelWindowShift(FractalParamDlg.m_ModelWindowShift);
		}

		//Grab the neural network preprocessing parameters
		if (FractalParamDlg.m_SetNNParams)
		{
			m_Princo.SetReducedDim(FractalParamDlg.m_nReducedDim);
			m_SOFM.SetTransientNum(FractalParamDlg.m_nTransientNum);
```

```
}

// Grab the transient size if it is to be changed.
if (FractalParamDlg.m_bChangeTransientSize)
{
	m_Extract.SetTransientSize(FractalParamDlg.m_TransientSize);
}



m_bInBatchProcess=TRUE;//disables menu functions


// See if the model size has changed
//if(OldModelWindowShift!=FractalParamDlg.m_ModelWindowShift)

NewModelSize=m_Extract.GetTransientModelSize();
//TRACE("TransientModelSize in EditFractalPara is %d\n",NewModelSize);

delete [] m_dTransientModel;
m_dTransientModel=new double
	[m_Extract.GetTransientModelSize()];

m_PNN.Initialize(m_Extract.GetTransientModelSize());
//m_SOFM.Initialize(m_Extract.GetTransientModelSize());

if((FractalParamDlg.m_nReducedDim!=0)
	&&(FractalParamDlg.m_nReducedDim!=NewModelSize)) //pca needed
	m_PNN.Initialize(FractalParamDlg.m_nReducedDim);


if (FractalParamDlg.m_SetSegmentationParams ||
	FractalParamDlg.m_bChangeTransientSize)
{
	// Display the progress box
	CEditFPProgressDialog EditFPProgressDlg;
	EditFPProgressDlg.m_ProgressBarLimit = m_TransientArray.GetSize();
	EditFPProgressDlg.m_strOperation = "Segmenting Transients";
	EditFPProgressDlg.Create();

	// Extract the transients
	SegmentAllTransients(m_Extract.GetTransientModelSize(),
		&EditFPProgressDlg);

	EditFPProgressDlg.DestroyWindow();
}

if (FractalParamDlg.m_SetModelParams ||
	FractalParamDlg.m_bChangeTransientSize)
{
	// Display the progress box
	CEditFPProgressDialog EditFPProgressDlg;
	EditFPProgressDlg.m_ProgressBarLimit = m_TransientArray.GetSize();
	EditFPProgressDlg.m_strOperation = "Extracting Features";
	EditFPProgressDlg.Create();

	// Calculate the model for each transient.
	ComputeAllModels(m_Extract.GetTransientModelSize(),
		&EditFPProgressDlg);

	EditFPProgressDlg.DestroyWindow();
}


if (FractalParamDlg.m_SetModelParams ||
```

```
            FractalParamDlg.m_bChangeTransientSize ||
            FractalParamDlg.m_SetSegmentationParams)
        {
            // Completely refresh the network details view
            // if it exists.
            if (m_pTransientListView != NULL)
            {
                m_pTransientListView -> RemoveAllItems();
                m_pTransientListView -> OnInitialUpdate();
            }

        }


        m_bInBatchProcess=FALSE;


    }

}

void CTAC_MMDoc::SegmentAllTransients(int NewModelSize,
                                      CEditFPProgressDialog *ProgressDlg)
{
    CString PathName;
    BOOL FileError=FALSE;
    int SegmentationErrors=0;
    int TransientBeginsHere;

    HCURSOR Wait=AfxGetApp()->LoadStandardCursor(IDC_WAIT);
    ::SetCursor(Wait);

    SetModifiedFlag(TRUE);
    for (int ctr=0; ctr<m_TransientArray.GetSize(); ctr++) {
        CheckWindowsMessages(5);
        ProgressDlg->OnUpdateProgress(ctr+1);
        ProgressDlg->m_UpdateProgress.SetPos(ctr+1);
        CheckWindowsMessages(5);
        PathName=m_TransientArray.GetAt(ctr)->GetRawFilePath();
        if (ReadRawSignal(PathName,FALSE)) {//checks to see if file exists
            TransientBeginsHere=m_Extract.FindTransient(m_nRawSignal,
                m_nSquelchChannel, m_CurrentTranFileType, m_dVarDimTraj, m_dTransientModel);
            //TRACE("feature extraction from SegmentAllTransients\n");
            if (TransientBeginsHere==-1) {
                SegmentationErrors+=1;
            }
            else {
                m_TransientArray.GetAt(ctr)->
                    SetTransientBeginsHere(TransientBeginsHere);
                if(NewModelSize>0) {
                    m_TransientArray.GetAt(ctr)->
                        UpdateTransientModelSize(NewModelSize);
                }
                m_TransientArray.GetAt(ctr)->
                    SetTransientModel(m_dTransientModel);
            }
        }
        else {
            AfxMessageBox("Raw file not found. Aborting",MB_OK);
            FileError=TRUE;
            break;
        }
    }

    if (SegmentationErrors&&!FileError) {
```

```
            CString strText;
            strText.Format
                ("There were %d segmentation errors.\nOriginal positions retained."
                ,SegmentationErrors);
            AfxMessageBox(strText,MB_OK);
        }

        if (FileError) {
            SetModifiedFlag(FALSE);
            UpdateStatusBarNoTransients();
            DeleteContents();//take no chance of user
                                                //killing original database
        }
        else{
            m_bViewSegmentation=TRUE;//show tran Segmentation info
            ViewNeedsUpdating();
            m_PNN.SetSigmasInitialized(FALSE);
            m_PNN.SetSigmasOptimized(FALSE);
        }
        HCURSOR Arrow=AfxGetApp()->LoadStandardCursor(IDC_ARROW);
        ::SetCursor(Arrow);
}

void CTAC_MMDoc::ComputeAllModels(int NewModelSize,
                                            CEditFPProgressDialog *ProgressDlg)
{
        CString PathName;
        BOOL FileError=FALSE;
        int TransientBeginsHere;

        HCURSOR Wait=AfxGetApp()->LoadStandardCursor(IDC_WAIT);
        ::SetCursor(Wait);

        SetModifiedFlag(TRUE);
        for (int ctr=0; ctr<m_TransientArray.GetSize(); ctr++) {
            CheckWindowsMessages(5);
            ProgressDlg->OnUpdateProgress(ctr+1);
            ProgressDlg->m_UpdateProgress.SetPos(ctr+1);
            CheckWindowsMessages(5);
            PathName=m_TransientArray.GetAt(ctr)->GetRawFilePath();
            if (ReadRawSignal(PathName,FALSE)) {//checks to see if file exists
                TransientBeginsHere=
                    m_TransientArray.GetAt(ctr)->GetTransientBeginsHere();
                m_Extract.FillTransientModel
                    (m_dVarDimTraj, TransientBeginsHere, m_dTransientModel);
                //TRACE("feature extraction from ComputeAllModels\n");
                if(NewModelSize>0) {
                    m_TransientArray.GetAt(ctr)->
                        UpdateTransientModelSize(NewModelSize);
                }
                m_TransientArray.GetAt(ctr)->
                    SetTransientModel(m_dTransientModel);
            }
            else {
                AfxMessageBox("Raw file not found. Aborting",MB_OK);
                FileError=TRUE;
                break;
            }
        }
        if (FileError) {
            SetModifiedFlag(FALSE);
            UpdateStatusBarNoTransients();
            DeleteContents();//take no chance of user
                                                //killing original database
        }
```

```
    else {
        m_bViewSegmentation=TRUE;//show tran Segmentation info
        ViewNeedsUpdating();
        m_PNN.SetSigmasInitialized(FALSE);
        m_PNN.SetSigmasOptimized(FALSE);
    }
    HCURSOR Arrow=AfxGetApp()->LoadStandardCursor(IDC_ARROW);
    ::SetCursor(Arrow);
}

void CTAC_MMDoc::OnViewSearch()
{
    CSearchDialog SearchDlg;
    if(SearchDlg.DoModal()!=IDOK) {
        return;
    }

    int Temp=-1;
    switch (SearchDlg.m_FieldToSearch)
    {
        case(0): {
            Temp=m_TransientArray.SearchForClass(SearchDlg.m_Class);
            break;
        }
        case(1): {
            Temp=m_TransientArray.SearchForMake(SearchDlg.m_Make);
            break;
        }
        case(2): {
            Temp=m_TransientArray.SearchForModel(SearchDlg.m_Model);
            break;
        }
        case(3): {
            Temp=m_TransientArray.SearchForSerial(SearchDlg.m_Serial);
            break;
        }
        case(4): {
            Temp=m_TransientArray.SearchForDate(SearchDlg.m_Date);
            break;
        }
        case(5): {
            Temp=m_TransientArray.SearchForTime(SearchDlg.m_Time);
            break;

        case(6):
            {
                Temp = m_TransientArray.SearchForClassName(SearchDlg.m_strClassName);
                break;
            }
        }
    }

    if(Temp!=-1) {
        m_nCurrentTransient=Temp;
        UpdateStatusBar();
        UpdateListView();
        ViewNeedsUpdating();
    }
    else {
        AfxMessageBox("Search field not found.",MB_OK);
    }
}
```

```
void CTAC_MMDoc::DeleteResultsArray()
{
	int	ctr=m_ResultsArray.GetSize();
	while (ctr--) {
		delete m_ResultsArray.GetAt(ctr);
	}
	m_ResultsArray.RemoveAll();
}

void CTAC_MMDoc::OnNeuralnetBatchclassify()
{
	unsigned int t1, t2, classify_time, seg_time;

	if (m_bViewTrainingResults)
		OnViewZoomedrawsignal();

	CString strFileNameBuff;  // Holds the list of pathnames for selected files.
	POSITION pos;		// Holds the position filenames in the filename buffer.
	int TransientModelSize=m_Extract.GetTransientModelSize();
	//TRACE("in CTAC_MMDoc::NeuralnetBatchClassify");

	//m_PNN.Initialize(m_Extract.GetTransientModelSize());
	int NumFiles = 0,
		NumClasses=m_PNN.GetNumClasses();
	double *SummationNeuron, Confidence, Activation;
	CString FileName;

	int PredictedClass;

	m_bNotClassifying=FALSE;  //so view isn't messed up by batch transients


	CFileDialog FileDlg(TRUE,// TRUE means file open.
				NULL,			// Not used for file open.
				NULL,			// No initial filename.
				OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT |
				OFN_FILEMUSTEXIST | OFN_PATHMUSTEXIST,
				"All compatible files (*.raw; *.pcm)| *.raw; *.pcm|"
				"Raw Files (*.raw)|*.raw|PCM Files (*.pcm)|*.pcm|"
				"All Files (*.*)|*.*||",  // Selects which files to show.
				NULL);

	FileDlg.m_ofn.lpstrTitle="Batch Classify: Select transient files.";
	FileDlg.m_ofn.Flags |= OFN_ALLOWMULTISELECT; // Allows multiple files to be
							// selected at once.
	FileDlg.m_ofn.lpstrFile = strFileNameBuff.GetBuffer(20000); // Select the filename buffer.
	FileDlg.m_ofn.nMaxFile = 20000; // Sets the size of the buffer.

	if (FileDlg.DoModal()!=IDOK) {
		m_bResultsExist=FALSE;
		return;
	}

	CCmdTarget::BeginWaitCursor();


	// Find the number of files selected.
	pos = FileDlg.GetStartPosition();// Gets the position of the first filename in the buffer.while (NULL != pos)
	while (NULL != pos)// Step through the filename buffer
	{
		FileName = FileDlg.GetNextPathName(pos); // Go to the next file name in the buffer.
		NumFiles++;
	}
```

```
// If there is only one file to classify show results in a dialog box.
// Note: This was originally a separate menu item.
if (NumFiles == 1)
{
    int nPredictedClass, index;


    HCURSOR Wait=AfxGetApp()->LoadStandardCursor(IDC_WAIT);
        ::SetCursor(Wait);


    if (ReadRawSignal(FileName,TRUE)) //sees if file exists, is compatible
    {

        CClassifyDialog ClassifyDlg;
        if(m_nNumClassifyModes==1)
        {
            m_bInBatchProcess=TRUE; //disables menu functions
            t1=timeGetTime();
            m_nTransientBeginsHere=m_Extract.FindTransient(m_nRawSignal,
                m_nSquelchChannel, m_CurrentTranFileType, m_dVarDimTraj, m_dTransientModel);
            t2=timeGetTime();
            seg_time=t2-t1;
            TRACE("Segmentation time for a single transient is %i (ms)\n", seg_time);
            //pca processing here, only multiply the existing transform matrix
            //overwrite the transientmodel vector
            if((m_Princo.GetReducedDim()!=0)
                &&(m_Princo.GetReducedDim()!=m_Extract.GetTransientModelSize()))
            {

                //TRACE("PCA on unknown single transient\n");
                m_Princo.PCATransform(m_dTransientModel);
            }

            m_bInBatchProcess=FALSE;
            if (m_nTransientBeginsHere==-1)
            {
                AfxMessageBox("No transient found. Try adjusting settings."
                    ,MB_OK);
                return;
            }
            SummationNeuron=new double[m_PNN.GetNumClasses()];

            t1=timeGetTime();

            nPredictedClass = m_PNN.Classify(m_dTransientModel,
                SummationNeuron,-1, m_dRejectThreshold, &Activation);

            t2=timeGetTime();
            classify_time=t2-t1;
            TRACE("Classification time for a single transient is %i (ms)\n", classify_time);

            ClassifyDlg.m_strPredictedClass = m_TransientArray.GetClassName(nPredictedClass);
            ClassifyDlg.m_Confidence=100.0*ComputeConfidence
                (SummationNeuron, nPredictedClass);
            ClassifyDlg.m_Activation=Activation;
            delete [] SummationNeuron;

        }
        else
        {
            double Confidence;
            nPredictedClass=MultiModeClassify(&Confidence, &Activation);
            ClassifyDlg.m_strPredictedClass = m_TransientArray.GetClassName(nPredictedClass);
```

```
            ClassifyDlg.m_Confidence=Confidence;
            ClassifyDlg.m_Activation=Activation;
        }

        m_bScrollToTransient=TRUE;
        m_bViewSegmentation=TRUE;//show tran Segmentation info
        m_bNotClassifying=FALSE;
        UpdateAllViews(NULL);
        UpdateStatusBarUnknown();

        if(ClassifyDlg.DoModal()==IDOK) //Add to database selected
        {
            CAddUnknownDialog AddUnknownDlg;
            AddUnknownDlg.m_TransmitterMake=m_TransmitterMake;
            AddUnknownDlg.m_TransmitterModel=m_TransmitterModel;
            AddUnknownDlg.m_TransmitterSerial=m_TransmitterSerial;
            AddUnknownDlg.m_Date=m_Date;
            AddUnknownDlg.m_Time=m_Time;
            AddUnknownDlg.m_strClassName = m_TransientArray.GetClassName(nPredictedClass);
            AddUnknownDlg.m_Position=m_TransientArray.GetSize()+1;
            AddUnknownDlg.m_pDoc = this; // Allows the dialog to access the transient array and look up class names.

            if(AddUnknownDlg.DoModal()==IDOK) // Add the class to the array.
            {
                // Is this a new class?
                if ((index = m_TransientArray.SearchForClassName(AddUnknownDlg.m_strClassName)) != -1)
                {
                    // Use the existing transient class #.
                    m_nTransientClass = m_TransientArray.GetAt(index) -> GetTransientClass();
                }
                else
                {
                    // Use the next available class #.
                    m_nTransientClass = m_TransientArray.FindNumClasses();

                }
                // Use the transient data from the dialog box.
                m_TransmitterMake = AddUnknownDlg.m_TransmitterMake;
                m_TransmitterModel = AddUnknownDlg.m_TransmitterModel;
                m_TransmitterSerial = AddUnknownDlg.m_TransmitterSerial;
                m_Date = AddUnknownDlg.m_Date;
                m_Time = AddUnknownDlg.m_Time;
                m_strTransientClassName = AddUnknownDlg.m_strClassName;
                // Store the transient in the transient array.
                StoreTransientData(AddUnknownDlg.m_Position-1);


                m_nCurrentTransient=AddUnknownDlg.m_Position-1;
                UpdateStatusBar();
                m_bScrollToTransient=TRUE;
                m_bViewSegmentation=TRUE;//show tran Segmentation info
                SetModifiedFlag(TRUE);
                m_PNN.SetNumCases(m_TransientArray.GetSize());
                m_PNN.SetNumClasses(m_TransientArray.FindNumClasses());

            }

        m_bNotClassifying=TRUE;
        }
        UpdateStatusBar();
        ViewNeedsUpdating();
}
HCURSOR Arrow=AfxGetApp()->LoadStandardCursor(IDC_ARROW);
::SetCursor(Arrow);
```

```
        // Switch to the network details view.
        CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;
        pFrame -> SwitchToView(1,0,pFrame -> LIST);

        return;
}

// Classsify multiple transients.

DeleteResultsArray();
m_bTrainingResults=FALSE;// TRUE if results are from training
m_bResultsExist=TRUE;


pos = FileDlg.GetStartPosition(); // Reset to the start of the filename buffer
for(int ctr=0; ctr<NumFiles; ctr++)
{
        CTrainingResults *TrainingResults=new CTrainingResults(1);
                                                        //only 1 Segmentation mode
        FileName = FileDlg.GetNextPathName(pos);
        TrainingResults->SetRawFilePath(FileName);
        TrainingResults->SetCorrectClass(-1);     // -1 for unknown.
        m_ResultsArray.InsertAt(ctr, TrainingResults);
}

CBatchProgressDialog BatchProgressDlg;
BatchProgressDlg.m_ProgressBarLimit=NumFiles;
m_bInBatchProcess=TRUE;// disables all menu functions
BatchProgressDlg.Create();

for(ctr=0; ctr<NumFiles; ctr++) {
        CheckWindowsMessages(20);
        BatchProgressDlg.OnUpdateProgress(ctr+1);
        BatchProgressDlg.m_UpdateProgress.SetPos(ctr+1);
        CheckWindowsMessages(20);
        if (ReadRawSignal(m_ResultsArray.GetAt(ctr)->GetRawFilePath(),FALSE)) {
                if(m_nNumClassifyModes==1) {
                        t1=timeGetTime();
                        m_nTransientBeginsHere=m_Extract.FindTransient(m_nRawSignal,
                                m_nSquelchChannel, m_CurrentTranFileType, m_dVarDimTraj, m_dTransientModel);

                        t2=timeGetTime();
                        seg_time=t2-t1;
                        TRACE("Segmentation time for a transient is %i (ms)\n", seg_time);

                        //perform PCA here if necessary
                        if((m_Princo.GetReducedDim()!=0)
                                &&(m_Princo.GetReducedDim()!=m_Extract.GetTransientModelSize()))
                        {
                                //m_Princo.Initialize(m_nTransientModelSize,m_nNumCases,m_nReducedDim);
                                //TRACE("PCA on unknown multiply transients\n");
                                m_Princo.PCATransform(m_dTransientModel);
                        }

                        if (m_nTransientBeginsHere!=-1) {
                                SummationNeuron=new double[NumClasses];

                                t1=timeGetTime();

                                PredictedClass=m_PNN.Classify(m_dTransientModel,
                                        SummationNeuron,-1,m_dRejectThreshold,&Activation);

                                m_ResultsArray.GetAt(ctr)
                                        ->SetPredictedClass(PredictedClass,0);
                                m_ResultsArray.GetAt(ctr)
```

```
                        ->SetActivation(Activation,0);
                    Confidence=100.0*ComputeConfidence
                        (SummationNeuron, PredictedClass);
                    m_ResultsArray.GetAt(ctr)->SetError(Confidence,0);

                    t2=timeGetTime();
                    classify_time=t2-t1;
                    TRACE("Classification time for transient%i is %i (ms)\n",ctr,classify_time);

                    delete [] SummationNeuron;
                }
                else {// no transient found in raw file
                    m_ResultsArray.GetAt(ctr)->SetPredictedClass(-999,0);
                    m_ResultsArray.GetAt(ctr)->SetError(0.0,0);
                    m_ResultsArray.GetAt(ctr)->SetActivation(0.0,0);
                }
            }
            else {
                double Confidence;
                PredictedClass=MultiModeClassify(&Confidence,&Activation);
                m_ResultsArray.GetAt(ctr)
                    ->SetPredictedClass(PredictedClass,0);
                m_ResultsArray.GetAt(ctr)->SetError(Confidence,0);
                m_ResultsArray.GetAt(ctr)->SetActivation(Activation,0);
            }
        }
    }

    BatchProgressDlg.DestroyWindow();
    m_bInBatchProcess=FALSE;
    HCURSOR Arrow=AfxGetApp()->LoadStandardCursor(IDC_ARROW);
    ::SetCursor(Arrow);
    m_bNotClassifying=TRUE;
    m_bViewTrainingResults = TRUE;
    OnViewClassificationStats();
    ViewNeedsUpdating();
}

void CTAC_MMDoc::CheckWindowsMessages(int NumChecks)
{
    MSG message;
    for(int ctr=0; ctr<NumChecks; ctr++) {
        if (::PeekMessage(&message, NULL, 0, 0, PM_REMOVE)) {
            ::TranslateMessage(&message);
            ::DispatchMessage(&message);
        }
    }
}

int CTAC_MMDoc::MultiModeClassify(double *WinningConfidence,
                                  double *Activation)
{
    int PredictedClass, PossibleClass;
    int *TransientBeginsHere;
    double **TransientModel, **SummationNeuron, HighestActivation;

    TransientBeginsHere=new int[m_nNumClassifyModes];
    TransientModel=new double*[m_nNumClassifyModes];
    SummationNeuron=new  double*[m_nNumClassifyModes];
    for(int ctr=0; ctr<m_nNumClassifyModes; ctr++) {
        TransientModel[ctr]=new double[m_Extract.GetTransientSize()];
        SummationNeuron[ctr]=new double[m_PNN.GetNumClasses()];
    }
    m_Extract.FindMultiTransients(m_nRawSignal, m_dVarDimTraj,
        TransientModel, m_nNumClassifyModes, m_dThreshholds,
```

```
            m_nMinSeparation, TransientBeginsHere);

        HighestActivation=0.0;
        for(ctr=0; ctr<m_nNumClassifyModes; ctr++) {
            if(TransientBeginsHere[ctr]!=-1) {
                PossibleClass=m_PNN.Classify(TransientModel[ctr],
                    SummationNeuron[ctr],-1, m_dRejectThreshold, Activation);
                if(*Activation>HighestActivation) {
                    *WinningConfidence=100.0*ComputeConfidence
                        (SummationNeuron[ctr], PossibleClass);
                    HighestActivation=*Activation;
                    PredictedClass=PossibleClass;
                    m_nTransientBeginsHere=TransientBeginsHere[ctr];
                    m_dTransientModel=TransientModel[ctr];
                }
            }
        }
        delete [] TransientBeginsHere;
        for(ctr=0; ctr<m_nNumClassifyModes; ctr++) {
            delete [] TransientModel[ctr];
            delete [] SummationNeuron[ctr];
        }
        delete [] TransientModel;
        delete [] SummationNeuron;

        *Activation=HighestActivation;

        if(*WinningConfidence>0.0)
            return PredictedClass;
        else
            return -999;// no valid transient found at all
}

void CTAC_MMDoc::OnNeuralnetModeparameters()
{
        CModeParamsDialog ModeParamsDlg;

        if(m_nNumClassifyModes==1)
            ModeParamsDlg.m_Mode=0;
        else {
            ModeParamsDlg.m_Mode=1;
            ModeParamsDlg.m_NumModes=m_nNumClassifyModes;
            ModeParamsDlg.m_LowThreshhold=m_dThreshholds[0];
            ModeParamsDlg.m_UpperThreshhold=m_dThreshholds[m_nNumClassifyModes-1];
            ModeParamsDlg.m_MinSeparation=m_nMinSeparation;
        }

        if(ModeParamsDlg.DoModal()==IDOK) {
            if(ModeParamsDlg.m_Mode==0||ModeParamsDlg.m_NumModes==1) {
                m_nNumClassifyModes=1;
                return;
            }
            else {
                m_nNumClassifyModes=ModeParamsDlg.m_NumModes;
                m_nMinSeparation=ModeParamsDlg.m_MinSeparation;
                delete [] m_dThreshholds;
                m_dThreshholds=new double[m_nNumClassifyModes];
                double Increment=(ModeParamsDlg.m_UpperThreshhold
                    -ModeParamsDlg.m_LowThreshhold)/(double)(m_nNumClassifyModes-1);
                for(int ctr=0; ctr<m_nNumClassifyModes; ctr++) {
                    m_dThreshholds[ctr]=ModeParamsDlg.m_LowThreshhold+ctr*Increment;
                }
            }
        }
}
```

```
void CTAC_MMDoc::OnNeuralnetSetrejectionthreshold()
{
    CSetRejectionDialog RejectionDialog;

    RejectionDialog.m_dRejectionThreshold=m_dRejectThreshold;
    if (RejectionDialog.DoModal()==IDOK)
        m_dRejectThreshold=RejectionDialog.m_dRejectionThreshold;
}

void CTAC_MMDoc::OnUpdateDatabaseAddtransient(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bDocCreated&&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateDatabaseDeletetransient(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bFileInMemory&&m_nCurrentTransient!=-1
        &&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateViewPrev(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bFileInMemory&&m_nCurrentTransient>0&&!m_bInBatchProcess);

}

void CTAC_MMDoc::OnUpdateViewNext(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bFileInMemory&&
        m_nCurrentTransient<m_TransientArray.GetUpperBound()
        &&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateViewFractaltrajectory(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bFileInMemory&&!m_bInBatchProcess);
    pCmdUI->SetCheck(m_bViewVarDimTrajFlag ? 1 : 0);
}

void CTAC_MMDoc::OnUpdateViewRawsignal(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bFileInMemory&&!m_bInBatchProcess);
    pCmdUI->SetCheck(m_bViewRawSignalFlag ? 1 : 0);
}

void CTAC_MMDoc::OnUpdateFilePageSetup(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bFileInMemory&&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateFilePrint(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bFileInMemory&&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateFilePrintPreview(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bFileInMemory&&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateFileSave(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(IsModified()&&!m_bInBatchProcess);
}
```

```
void CTAC_MMDoc::OnUpdateNeuralnetClassify(CCmdUI* pCmdUI)
{
      pCmdUI->Enable(m_bFileInMemory&&m_PNN.GetSigmasInitialized()
            &&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateNeuralnetInitializesigmas(CCmdUI* pCmdUI)
{
      pCmdUI->Enable(m_bFileInMemory&&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateEditFractalparameters(CCmdUI* pCmdUI)
{
      pCmdUI->Enable(m_bFileInMemory&&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateViewSearch(CCmdUI* pCmdUI)
{
      pCmdUI->Enable(m_bFileInMemory&&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateNeuralnetOptimizesigmas(CCmdUI* pCmdUI)
{
      pCmdUI->Enable(m_bFileInMemory&&m_PNN.GetSigmasInitialized()
            &&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateNeuralnetBatchclassify(CCmdUI* pCmdUI)
{
      pCmdUI->Enable(m_bFileInMemory&&m_PNN.GetSigmasInitialized()
            &&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateNeuralnetModeparameters(CCmdUI* pCmdUI)
{
      pCmdUI->Enable(m_bFileInMemory&&!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateFileNew(CCmdUI* pCmdUI)
{
      pCmdUI->Enable(!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateFileOpen(CCmdUI* pCmdUI)
{
      pCmdUI->Enable(!m_bInBatchProcess);}

void CTAC_MMDoc::OnUpdateFileSaveAs(CCmdUI* pCmdUI)
{
      pCmdUI->Enable(!m_bInBatchProcess);
}

void CTAC_MMDoc::OnUpdateNeuralnetSetrejectionthreshold(CCmdUI* pCmdUI)
{
            pCmdUI->Enable(m_bFileInMemory&&!m_bInBatchProcess);
}

void CTAC_MMDoc::DeleteCurrentTransient()
{
      // Remove the transient from the transient array.
      delete m_TransientArray.GetAt(m_nCurrentTransient);
      m_TransientArray.RemoveAt(m_nCurrentTransient);

      // Remove the data from the list view if it exists.
```

```
        if (m_pTransientListView != NULL)
        {
                m_pTransientListView -> RemoveTransientData(m_nCurrentTransient);
        }

        SetModifiedFlag(TRUE);
        if (m_nCurrentTransient>m_TransientArray.GetUpperBound()) {
                DecrementCurrentTransient();
        }
        else {
                UpdateStatusBar();//status bar is updated in decrement otherwise
        }
        if(m_nCurrentTransient>-1) {
                ViewNeedsUpdating();
        }
        else {
                m_bFileInMemory=FALSE;
                UpdateAllViews(NULL);
        }
        m_PNN.SetNumCases(m_TransientArray.GetSize());
        m_PNN.SetNumClasses(m_TransientArray.FindNumClasses());


}

void CTAC_MMDoc::UpdateStatusBarNewTransient()
{
        CString str;
        CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;
        CStatusBar *pStatus=&pFrame->m_wndStatusBar;
        if (pStatus) {
                str.Format(" New transient.");
                pStatus->SetPaneText(1,str);
        }
}

void CTAC_MMDoc::OnStartCapture()
{
        // TODO: Add your command handler code here


// This block is commented out until the DLL
// is working correctly.
// comment from here
        HINSTANCE hInst;
        char errp[250];
        unsigned short errc;
        unsigned long   hndl,fh;

        CWnd *pMainWnd= AfxGetApp()->m_pMainWnd;
        HWND hMainWnd = pMainWnd -> m_hWnd;

        hInst = AfxGetInstanceHandle();
        fh = NULL;
        hndl = NULL;
        errc = NO_ERR;
        char  temp[256];

        char *txidvar;   /* Get the value of the TXID environment variable. */
        txidvar= getenv( "TXID" );
        if( txidvar != NULL )
        {
                sprintf ( temp , "%s\\TXID.TXE", txidvar);
                if((errc = _file_o(&fh,errp,NEW_O,"/name/o/ro", temp)) != NO_ERR)
                {
```

```
            TRACE("Nur, open env file error: %s",errp);
        }
        else
        {
            TRACE("Nur , open env file it worked!!");
        }

        sprintf(temp,"/inst=%lX/pinst=0/phwnd=%lX",hInst,hMainWnd);
        if ((errc = _wtxid_o(&hndl, errp, LOD_O, temp, fh)) != NO_ERR)
        {
            TRACE("Chris, error: %s",errp);
        }
        else
        {
            TRACE("Chris, it worked!!");
        }
        _file_o(&fh,errp,DEL_O,"");
        }
    else
    {
    if ((errc = _wtxid_o(&hndl, errp, NEW_O, "", hInst, NULL, "", SW_SHOWNORMAL, hMainWnd)) != NO_ERR)
    {
        TRACE("Nur, without env txid error: %s",errp);
    }
    else
    {
        TRACE("Nur, without env TXID it worked!!");
    }
    }
    // Starts the TXID executable and returns when it is closed.
//  _spawnlp( _P_WAIT, "txid.exe", "C:\\mydll\\txid.txe", NULL );

}

void CTAC_MMDoc::StoreTransientData(int index)
{
    int TransientModelSize;
    TransientModelSize = m_Extract.GetTransientModelSize();
    //TRACE("TransientModelSize in Storedata is %d\n", TransientModelSize);

    CTransientData* pTransientData=
        new CTransientData(TransientModelSize);

    pTransientData->SetRawFilePath(m_strPathName);
    pTransientData->SetFileType(m_CurrentTranFileType);
    pTransientData->SetTransientBeginsHere(m_nTransientBeginsHere);
    pTransientData->SetTransmitterMake(m_TransmitterMake);
    pTransientData->SetTransmitterModel(m_TransmitterModel);
    pTransientData->SetTransmitterSerial(m_TransmitterSerial);
    pTransientData->SetTransmitDate(m_Date);
    pTransientData->SetTransmitTime(m_Time);
    pTransientData->SetTransientModelSize(TransientModelSize);
    pTransientData->SetTransientModel(m_dTransientModel);
    pTransientData->SetTransientClass(m_nTransientClass);
    pTransientData->SetTransientClassName(m_strTransientClassName);


    // Add the data to the transient array.
    m_TransientArray.InsertAt(index, pTransientData);


    // Add the data to the CTransientListView if it exists.
    if (m_pTransientListView != NULL)
    {
        m_pTransientListView -> AddTransientData(index, pTransientData);
```

```
    }
}

BOOL CTAC_MMDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
    if (!CDocument::OnOpenDocument(lpszPathName))
        return FALSE;

    // TODO: Add your specialized creation code here
    UpdateStatusBar();

    return TRUE;
}




void CTAC_MMDoc::OnViewZoomedrawsignal()
{
    // TODO: Add your command handler code here
    CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;

    // Set the view to display the raw signal not the training or classification results.
    m_bViewTrainingResults = FALSE;
    m_bScrollToTransient = TRUE;

    pFrame -> SwitchToView(1,0,pFrame -> TEXT);

    UpdateAllViews(NULL);
}

void CTAC_MMDoc::OnUpdateViewZoomedrawsignal(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // TODO: Add your command update UI handler code here
    CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;

    pCmdUI->Enable(m_bFileInMemory && !m_bInBatchProcess);
    pCmdUI -> SetCheck((pFrame -> GetSplitter() -> GetPane(1,0) ->
        IsKindOf(RUNTIME_CLASS(CTAC_MMView))) && !m_bViewTrainingResults);
}

void CTAC_MMDoc::OnViewNetworkDetails()
{
    // TODO: Add your command handler code here
    CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;
    pFrame -> SwitchToView(1,0,pFrame -> LIST);

    UpdateAllViews(NULL);

}

void CTAC_MMDoc::OnUpdateViewNetworkDetails(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;

    pCmdUI -> Enable(m_bFileInMemory && !m_bInBatchProcess);
    pCmdUI -> SetCheck(pFrame -> GetSplitter() -> GetPane(1,0) ->
        IsKindOf(RUNTIME_CLASS(CTransientListView)));
}

void CTAC_MMDoc::OnViewClassificationStats()
{
```

```
    // TODO: Add your command handler code here
    CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;

    // Set the doc to display the training or classification results not the raw signal.
    m_bViewTrainingResults = TRUE;
    pFrame -> SwitchToView(1,0,pFrame -> RESULTS);

    UpdateAllViews(NULL);
}

void CTAC_MMDoc::OnUpdateViewClassificationStats(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;

    pCmdUI->Enable(m_bFileInMemory && m_bResultsExist && !m_bInBatchProcess && !m_bTrainingResults);
    pCmdUI -> SetCheck((pFrame -> GetSplitter() -> GetPane(1,0) ->
        IsKindOf(RUNTIME_CLASS(CTAC_MMView))) && m_bViewTrainingResults
        && m_bFileInMemory && m_bResultsExist && !m_bInBatchProcess
        && !m_bTrainingResults);
}

void CTAC_MMDoc::OnViewTrainingResults()
{
    // TODO: Add your command handler code here
    CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;

    // Set the doc to display the training or classification results not the raw signal.
    m_bViewTrainingResults = TRUE;
    pFrame -> SwitchToView(1,0,pFrame -> TEXT);

    UpdateAllViews(NULL);
}

void CTAC_MMDoc::OnUpdateViewTrainingResults(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;

    pCmdUI->Enable(m_bFileInMemory && m_bResultsExist && !m_bInBatchProcess && m_bTrainingResults);
    pCmdUI -> SetCheck((pFrame -> GetSplitter() -> GetPane(1,0) ->
        IsKindOf(RUNTIME_CLASS(CTAC_MMView))) && m_bViewTrainingResults
        && m_bFileInMemory && m_bResultsExist && !m_bInBatchProcess
        && m_bTrainingResults);
}

void CTAC_MMDoc::UpdateListView()
{
    // Update the list view selection if the list view exists.
    if (m_pTransientListView != NULL)
    {
        m_pTransientListView -> SetCurrentSelection(m_nCurrentTransient);
    }

}

void CTAC_MMDoc::OnSaveResults()
{
    // TODO: Add your command handler code here
    CFileDialog FileDlg(FALSE,// FALSE means file save.
                        NULL,                   // No extension to append to the filename.
                        NULL,                   // No initial filename.
                        OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
                        "Text Files (*.*)|*.*||",  // Selects which files to show.
                        NULL);
```

```
FileDlg.m_ofn.lpstrTitle="Save results as";

// Ask for the output filename.
if (FileDlg.DoModal()==IDOK)
{
    CString PathName;

    PathName=FileDlg.GetPathName();

    // Open the output file.
    ofstream outfile;
    outfile.open(PathName);
    if (outfile.fail())
    {
        AfxMessageBox("Could not open the file!", MB_OK);
        return;
    }

    // Set the precision and flags for floating point output.
    outfile.precision(2);
    outfile.setf(ios::fixed | ios::showpoint);

    // Output the networks details.
    outfile << "Network Details" << endl;
    outfile << "Network Name: " << GetTitle() << endl;
    outfile << "# of Classes: " << m_TransientArray.FindNumClasses() << endl;
    outfile << "# of Transients: " << m_TransientArray.GetSize() << endl << endl;

    // Output the transient file parameters.
    outfile << "Transient File Parameters" << endl;
    outfile << "Transient File Size: " << m_Extract.GetRawFileSize() << endl;
    outfile << "Transient Size: " << m_Extract.GetTransientSize() << endl << endl;

    // Output the segmentation parameters.
    outfile << "Segmentation Parameters" << endl;
    switch (m_Extract.GetTriggerType())
    {
    case VARIANCE:
        outfile << "Trigger Type: Variance" << endl;
        outfile << "Window Size: " << m_Extract.GetWindowSize() << endl;
        outfile << "Variance Pairs: " << m_Extract.GetVariancePairs() << endl;
        outfile << "Threshold: " << m_Extract.GetThreshold() << endl << endl;
        break;

    case SQUELCH:
        outfile << "Trigger Type: Squelch" << endl;
        outfile << "Threshold: " << m_Extract.GetSquelchTrigger() << endl;
        outfile << "Hysteresis Window: " << m_Extract.GetSquelchWidth() << endl;
        outfile << "Delay: " << m_Extract.GetSquelchDelay() << endl << endl;
        break;

    case WINDOWEDVARIANCE:
        outfile << "Trigger Type: Windowed Variance" << endl;
        outfile << "Variance Window Size: " << m_Extract.GetWindowSize() << endl;
        outfile << "Variance Pairs: " << m_Extract.GetVariancePairs() << endl;
        outfile << "Variance Threshold: " << m_Extract.GetThreshold() << endl;
        outfile << "Squelch Threshold: " << m_Extract.GetSquelchTrigger() << endl;
        outfile << "Squelch Hysteresis Window: " << m_Extract.GetSquelchWidth() << endl;
        outfile << "Squelch Delay: " << m_Extract.GetSquelchDelay() << endl;
        outfile << "Trigger Window Size: " << m_Extract.GetTrigWinSize() << endl;
        break;
    }

    if(m_Extract.GetTrigScheme()==0)
```

```
{
    outfile << "Trigger Scheme:Absolute Difference Triggering" << endl;
}
else
{
    outfile << "Trigger Scheme:Relative Difference Triggering" << endl;
    outfile << "Adjustment: "<<m_Extract.GetAdjust() <<endl << endl;
}

// Output the feature extraction parameters.
outfile << "Feature Extraction Parameters" << endl;
outfile << "Window Size: " << m_Extract.GetModelWindowSize() << endl;
outfile << "Variance Pairs: " << m_Extract.GetModelVariancePairs() << endl;
outfile << "Window Shift: " << m_Extract.GetModelWindowShift() << endl << endl;

//Output the neural network preprocessing parameters
outfile << "Neual Network Preprocessing Parameters" << endl;
outfile << "Number of Principal Components: " << m_Princo.GetReducedDim() << endl;
//outfile << "SOFM Threshold: " << m_SOFM.GetTransientNum() << endl;

// Output the classification results.
outfile << endl << "Classification results" << endl << endl;

// Set the column headings.
CString column1 = "Transient",
        column2 = "File Name and Path",
        column3 = "Predicted Class",
        column4 = "Confidence(%)",
        column5 = "Activation";

// Find the minimum width for each of the fields.
int width1 = column1.GetLength(),
    width2 = column2.GetLength(),
    width3 = column3.GetLength(),
    width4 = column4.GetLength(),
    width5 = column5.GetLength();

// Also count the number of transients identified as each class
// for the summary information. (Summary[0] is the number of unknowns)
int *Summary = new int[m_TransientArray.FindNumClasses() + 1];
for (int i = 0; i < m_TransientArray.FindNumClasses() + 1; i++)
{
    Summary[i] = 0;
}

for (i = 0; i < m_ResultsArray.GetSize(); i++)
{
    // We're only really concerned with the width of the second
    // and third fields.
    int PredictedClassNum;

    if (m_ResultsArray.GetAt(i) -> GetRawFilePath().GetLength() > width2)
        width2 = m_TransientArray.GetAt(i) -> GetRawFilePath().GetLength();

    PredictedClassNum = m_ResultsArray.GetAt(i) -> GetPredictedClass(0);


    if (PredictedClassNum == -1)
    {
        Summary[0] += 1; // Increment # of unknowns
    }
    else
    {
        Summary[PredictedClassNum + 1] += 1; // Increment the counter for the class.
    }
```

```
        if (m_TransientArray.GetClassName(PredictedClassNum).GetLength() > width3)
            width3 = m_TransientArray.GetAt(i) -> GetClassName().GetLength();
}


// Output the column headings.
outfile.setf(ios::left);

outfile << " " << setw(width1) << column1 << " "
        << " " << setw(width2) << column2 << " "
        << " " << setw(width3) << column3 << " "
        << " " << setw(width4) << column4 << " "
        << " " << setw(width5) << column5 << endl;

for (i = 0; i < width1 + width2 + width3 + width4 + width5 + 10; i++)
{
    outfile << "-";
}

outfile << endl;

// Output the rest of the table.
for (i = 0; i < m_ResultsArray.GetSize(); i++)
{
    int PredictedClassNum;
    double Error, Activation;

    CString field1,
            field2,
            field3,
            field4,
            field5;

    // Format each of the fields.
    field1.Format("%i", i+1);
    field2.Format("%s", m_ResultsArray.GetAt(i) -> GetRawFilePath());

    PredictedClassNum = m_ResultsArray.GetAt(i) -> GetPredictedClass(0);
    if (PredictedClassNum != -1)
    field3.Format("%s", m_TransientArray.GetClassName(PredictedClassNum));
    Error = m_ResultsArray.GetAt(i) -> GetError(0);
    field4.Format("%.4f", Error);
    Activation = m_ResultsArray.GetAt(i) -> GetActivation(0);
    field5.Format("%.3e", Activation);

    // Output the fields.
    outfile << " " << setw(width1) << field1 << " "
            << " " << setw(width2) << field2 << " "
            << " " << setw(width3) << field3 << " "
            << " " << setw(width4) << field4 << " "
            << " " << setw(width5) << field5 << endl;

}

// Output a summary of the classification.
column1 = "Class Name";
column2 = "# of Transients";

width1 = column1.GetLength();
width2 = column2.GetLength();

// Find the maximum width for the first column.
for (i = 0; i < m_TransientArray.FindNumClasses(); i++)
{
```

```
            if (m_TransientArray.GetClassName(i).GetLength() > width1)
            {
                width1 = m_TransientArray.GetClassName(i).GetLength();
            }
        }

        outfile << endl << endl << "Classification Summary" << endl << endl;

        // Output the column headings.
        outfile << setw(width1) << column1 << " "
                << setw(width2) << column2 << endl;

        // Output the classes and the count for each one.
        for (i = 0; i < m_TransientArray.FindNumClasses(); i++)
        {
            outfile << setw(width1) << m_TransientArray.GetClassName(i) << " "
                    << setw(width2) << Summary[i + 1] << endl;
        }

        outfile << setw(width1) << "Unknown" << " "
                << setw(width2) << Summary[0] << endl;

        outfile << setw(width1) << "Total" << " "
                << setw(width2) << m_ResultsArray.GetSize() << endl;


        outfile.close();
    }
}

void CTAC_MMDoc::OnUpdateSaveResults(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    CMainFrame *pFrame=(CMainFrame*) AfxGetApp()->m_pMainWnd;

    pCmdUI->Enable(m_bFileInMemory && m_bResultsExist &&
        !m_bInBatchProcess && !m_bTrainingResults);

}
```

TAC_MMView.h

```
// CTAC_MMView: defines and manages the display in the lower viewing window
// TAC_MMView.h : interface of the CTAC_MMView class
//
/////////////////////////////////////////////////////////////////////////////

class CTAC_MMView : public CScrollView
{
protected: // create from serialization only
	CTAC_MMView();
	DECLARE_DYNCREATE(CTAC_MMView)
// Attributes
public:
	CTAC_MMDoc* GetDocument();
// Operations
public:
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CTAC_MMView)
	public:
	virtual void OnDraw(CDC* pDC);  // overridden to draw this view
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	virtual void OnPrepareDC(CDC* pDC, CPrintInfo* pInfo = NULL);
	virtual void OnInitialUpdate(); // called first time after construct
	protected:
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
	virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CTAC_MMView();
	void PrintTrainingResults(CDC* pDC);

#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:
	CRect m_rectClient;
	int m_TopMargin;
	int m_BottomMargin;
	int m_LeftMargin;
	int m_RightMargin;
	int m_FontHeight;
	int m_TextLinesPerPage;

// Generated message map functions
protected:
	//{{AFX_MSG(CTAC_MMView)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in TAC_MMView.cpp
inline CTAC_MMDoc* CTAC_MMView::GetDocument()
   { return (CTAC_MMDoc*)m_pDocument; }
#endif
```

TAC_MMView.cpp

```
// TAC_MMView.cpp : implementation of the CTAC_MMView class
//

#include "stdafx.h"
#include "TAC_MM.h"

#include "TAC_MMDoc.h"
#include "TAC_MMView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMView

IMPLEMENT_DYNCREATE(CTAC_MMView, CScrollView)

BEGIN_MESSAGE_MAP(CTAC_MMView, CScrollView)
	//{{AFX_MSG_MAP(CTAC_MMView)
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMView construction/destruction

CTAC_MMView::CTAC_MMView()
{

}

CTAC_MMView::~CTAC_MMView()
{
}

BOOL CTAC_MMView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CScrollView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMView drawing

void CTAC_MMView::OnDraw(CDC* pDC)
{
	CTAC_MMDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	OnPrepareDC(pDC);
	if(!pDoc->m_bFileInMemory) {
		return;
	}
```

```
    if (pDoc->m_bViewTrainingResults) {
        PrintTrainingResults(pDC);
        return;
    }

    int TransientBeginsHere=pDoc->GetTransientBeginsHere();

    int SigHeight=pDoc->GetHighestValue()-pDoc->GetLowestValue();
    int ClipYCoord=pDoc->GetLowestValue();

    CRect rectClient;
    GetClientRect(rectClient);

    int *RawSignal=pDoc->GetRawSignal();
    int TransientSize=pDoc->GetExtract()->GetTransientSize();
    int RawFileSize=pDoc->GetExtract()->GetRawFileSize();
    int LeftTextMargin=(int)(rectClient.right/20.0);

    CPen *pOldPen;
    CPen GreyDashPen(PS_DASH, 1, RGB(192,192,192));
    CPen ThickBlackPen(PS_SOLID, 3, RGB(0,0,0));

    pOldPen=pDC->SelectObject(&ThickBlackPen);
    pDC->MoveTo(LeftTextMargin,0);
    pDC->LineTo(LeftTextMargin,rectClient.bottom);
    pDC->LineTo(LeftTextMargin+RawFileSize,rectClient.bottom);
    pDC->SelectObject(pOldPen);
    //Draw Axis and other good stuff on screen

    //CSize GetViewportExt( ) const;

    CPoint Init(LeftTextMargin,rectClient.bottom- (int) ((float)(RawSignal[0]
        -ClipYCoord)*(float)((float)rectClient.bottom/(float)SigHeight)));
    pDC->MoveTo(Init);
    // Plot the signal.
    for(int ctr=1; ctr<RawFileSize; ctr++) {
        CPoint Next(LeftTextMargin+ctr,rectClient.bottom- (int)
            ((float)(RawSignal[ctr]-ClipYCoord)*
            (float)((float)rectClient.bottom/(float)SigHeight)));
        pDC->LineTo(Next);
    }

    pOldPen=pDC->SelectObject(&GreyDashPen);
    pDC->MoveTo(LeftTextMargin+TransientBeginsHere,0);
    pDC->LineTo(LeftTextMargin+TransientBeginsHere,rectClient.bottom);
    pDC->MoveTo(LeftTextMargin+TransientBeginsHere+TransientSize,0);
    pDC->LineTo(LeftTextMargin+TransientBeginsHere+TransientSize,
        rectClient.bottom);
    pDC->SelectObject(pOldPen);

    if(pDoc->m_bScrollToTransient) {
        CPoint Scroll(TransientBeginsHere,0);
        ScrollToPosition(Scroll);
        pDoc->m_bScrollToTransient=FALSE;
    }
}

void CTAC_MMView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    CSize sizeTotal;
    // TODO: calculate the total size of this view
    sizeTotal.cx = sizeTotal.cy = 100;
```

```
    SetScrollSizes(MM_TEXT, sizeTotal);
}

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMView printing

BOOL CTAC_MMView::OnPreparePrinting(CPrintInfo* pInfo)
{
    CTAC_MMDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    CPageSetupDialog* PSD=pDoc->GetPageSetupDlg();

    int PageWidth=PSD->m_psd.ptPaperSize.x-PSD->m_psd.rtMinMargin.left-
        PSD->m_psd.rtMinMargin.right;
    int PageLength=PSD->m_psd.ptPaperSize.y-PSD->m_psd.rtMinMargin.top
        -PSD->m_psd.rtMinMargin.bottom;
    int UsablePageWidth=PSD->m_psd.ptPaperSize.x-PSD->m_psd.rtMargin.left-
        PSD->m_psd.rtMargin.right;
    int UsablePageLength=PSD->m_psd.ptPaperSize.y-PSD->m_psd.rtMargin.top
        -PSD->m_psd.rtMargin.bottom;

    m_TopMargin=(PSD->m_psd.rtMargin.top-PSD->m_psd.rtMinMargin.top)/10;
    m_BottomMargin=(PSD->m_psd.rtMargin.bottom-PSD->m_psd.rtMinMargin.bottom)/10;
    m_LeftMargin=(PSD->m_psd.rtMargin.left-PSD->m_psd.rtMinMargin.left)/10;
    m_RightMargin=(PSD->m_psd.rtMargin.right-PSD->m_psd.rtMinMargin.right)/10;

    m_rectClient.left=0;
    m_rectClient.right=PageWidth/10;//convert to MM_LOENGLISH
    m_rectClient.top=0;
    m_rectClient.bottom=PageLength/10;//convert to MM_LOENGLISH

    m_FontHeight=16;

    m_TextLinesPerPage=UsablePageLength/10/m_FontHeight;
    int NumTextLines=(pDoc->GetResultsArray()->GetSize());
    int LinesPrinted=m_TextLinesPerPage-6, PrintPages=1;//-6 accts for title
    while (LinesPrinted<(NumTextLines+4)) {//+4 accts for totals
        PrintPages+=1;
        LinesPrinted+=m_TextLinesPerPage-2;//-2 accts for row headings
    }
    pInfo->SetMaxPage(PrintPages);

    return DoPreparePrinting(pInfo);
}

void CTAC_MMView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{

}

void CTAC_MMView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMView diagnostics

#ifdef _DEBUG
void CTAC_MMView::AssertValid() const
{
    CScrollView::AssertValid();
}
```

```
void CTAC_MMView::Dump(CDumpContext& dc) const
{
	CScrollView::Dump(dc);
}

CTAC_MMDoc* CTAC_MMView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTAC_MMDoc)));
	return (CTAC_MMDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CTAC_MMView message handlers


void CTAC_MMView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
	Invalidate();

}

void CTAC_MMView::PrintTrainingResults(CDC* pDC)
{
	CTAC_MMDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	CString Field1, Field2, Field3, Field4, Field5, Field6;

	int Spacing=m_rectClient.right/8;

	if(pDoc->m_bTrainingResults)
	{
		Field1.Format("  Training Results  ");
	}
	else
	{
		Field1.Format("Batch Classify Results");
	}

	CFont BigRomanFont;
	int Size=(m_rectClient.right/3)/(Field1.GetLength()-5);
	BigRomanFont.CreateFont(1.5*Size, Size, 0, 0, 600, FALSE, TRUE, 0,
				ANSI_CHARSET, OUT_DEFAULT_PRECIS,
				CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
				DEFAULT_PITCH | FF_ROMAN, NULL);
	CFont *pOldFont=pDC->SelectObject(&BigRomanFont);
	pDC->TextOut(m_rectClient.right/3, 0, Field1);

	Field1.Format("Transient #");
	Field2.Format("FileName and Path");
	Field3.Format("Class");
	Field4.Format("Predicted");

	if(pDoc->m_bTrainingResults)
		Field5.Format("Error");
	else
		Field5.Format("Conf(%%)");

	Field6.Format("Activation");

	CFont RomanFontUnder;
	RomanFontUnder.CreateFont(0, 0, 0, 0, 400, FALSE, TRUE, 0,
				ANSI_CHARSET, OUT_DEFAULT_PRECIS,
				CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
```

```
                                DEFAULT_PITCH | FF_ROMAN, NULL);
    pDC->SelectObject(&RomanFontUnder);
    pDC->TextOut(0, 3*Size, Field1);
    pDC->TextOut(Spacing, 3*Size, Field2);
    pDC->TextOut(4*Spacing, 3*Size, Field3);
    pDC->TextOut(5*Spacing, 3*Size, Field4);
    pDC->TextOut(6*Spacing, 3*Size, Field5);
    pDC->TextOut(7*Spacing, 3*Size, Field6);

    CFont RomanFont;
    RomanFont.CreateFont(0, 0, 0, 0, 400, FALSE, FALSE, 0,
                                ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                                DEFAULT_PITCH | FF_ROMAN, NULL);
    pDC->SelectObject(&RomanFont);

    int NumLines=(pDoc->GetResultsArray()->GetSize());
    TEXTMETRIC Metrics;
    pDC->GetTextMetrics(&Metrics);
    int Height= (int) 1.25*(Metrics.tmHeight);
    CSize DataSize(m_rectClient.right-14, 10*Size+Height*NumLines);
    CSize SizePage=(0,100*Height);
    CSize SizeLine=(0,Height);
    SetScrollSizes(MM_TEXT,DataSize,SizePage,SizeLine);

    double Error, TotalError=0.0;
    int PredictedClass, CorrectClass, NumCorrect=0;

    for (int ctr=0; ctr<NumLines; ctr++)
    {
        Field1.Format("%d",ctr+1);

        Field2=pDoc->GetResultsArray()->GetAt(ctr)->GetRawFilePath();
        CorrectClass=pDoc->GetResultsArray()->GetAt(ctr)->GetCorrectClass();
        Field3 = pDoc -> GetArray() -> GetClassName(CorrectClass);
        PredictedClass=pDoc->GetResultsArray()->GetAt(ctr)->GetPredictedClass(0);
        Field4 = pDoc -> GetArray() -> GetClassName(PredictedClass);


        if (PredictedClass==CorrectClass)
            NumCorrect++;
        TotalError+=Error=pDoc->GetResultsArray()->GetAt(ctr)->GetError(0);
        Field5.Format("%.4lf",Error);
        if(pDoc->m_bTrainingResults)
            Field6.Format("N/A");
        else
            Field6.Format("%.3e",pDoc->GetResultsArray()
                ->GetAt(ctr)->GetActivation(0));

        pDC->TextOut(0, 5*Size+ctr*Height, Field1);
        pDC->TextOut(Spacing, 5*Size+ctr*Height, Field2);
        pDC->TextOut(4*Spacing, 5*Size+ctr*Height, Field3);
        pDC->TextOut(5*Spacing, 5*Size+ctr*Height, Field4);
        pDC->TextOut(6*Spacing, 5*Size+ctr*Height, Field5);
        pDC->TextOut(7*Spacing, 5*Size+ctr*Height, Field6);
    }

/*  Field1.Format("Number of Correct Classifications: %d/%d.",
        NumCorrect, NumLines);
    pDC->TextOut(0, 6*Size+NumLines*Height, Field1); */

    if(pDoc->m_bTrainingResults)
    {
            Field1.Format("Average Error: %.12lf.", TotalError/NumLines);
            pDC->TextOut(0, 6*Size+(NumLines+1)*Height, Field1);
```

```
    }

    pDC->SelectObject(pOldFont);

    return;
}

void CTAC_MMView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    //this function is called immediately before OnDraw for screen
    //and immediately before OnPrint for the printer
    CScrollView::OnPrepareDC(pDC, pInfo);

    CTAC_MMDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    if(pDoc->m_bFileInMemory)
    {
        if(!pDoc->m_bViewTrainingResults)
        {
            GetClientRect(m_rectClient);
            int RawFileSize=pDoc->GetExtract()->GetRawFileSize();
            int LeftTextMargin=(int)(m_rectClient.right/20.0);
            CSize DataSize(RawFileSize+LeftTextMargin,m_rectClient.bottom-14);
                    //compensate for bottom scrollbar(14) which is about to appear
            int TransientSize=pDoc->GetExtract()->GetTransientSize();
            CSize SizePage=(TransientSize,0);//page scroll moves by transient size

            SetScrollSizes(MM_TEXT,DataSize,SizePage);
        }
        else
        {
            if (!pDC->IsPrinting())
            {
                pDC->SetMapMode(MM_TEXT);
                GetClientRect(m_rectClient);
            }
            else
                pDC->SetMapMode(MM_LOENGLISH);
        }
    }
}

static double TotalError;
static int NumCorrect;

void CTAC_MMView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
{
    CTAC_MMDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    if (!pDoc->m_bViewTrainingResults) {
        CScrollView::OnPrint(pDC, pInfo);
        return;
    }

    static
    CString Field1, Field2, Field3, Field4, Field5, Field6;
    int Spacing=(m_rectClient.right-m_LeftMargin-m_RightMargin)/8;
    int HeaderOffset;

    int Start, AvailableLines;
    UINT Page=pInfo->m_nCurPage;

    if(Page==1) {
```

```
        TotalError=0.0;
        NumCorrect=0;
        Start=0;
        if(pDoc->m_bTrainingResults) {
            Field1.Format("Training Results");
        }
        else {
            //Field1.Format("Batch Classify Results");
            Field1.Format("");
        }

        CFont BigRomanFont;
        BigRomanFont.CreateFont(2*m_FontHeight, 0, 0, 0, 600, FALSE, TRUE, 0,
                                ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                                DEFAULT_PITCH | FF_ROMAN, NULL);
        CFont *pOldFont=pDC->SelectObject(&BigRomanFont);
        int Length=Field1.GetLength();
        TEXTMETRIC Metrics;
        pDC->GetTextMetrics(&Metrics);
        Length*=Metrics.tmAveCharWidth;
        pDC->TextOut(m_LeftMargin+(m_rectClient.right-m_LeftMargin
            -m_RightMargin-Length)/2, -m_TopMargin, Field1);

        pDC->SelectObject(pOldFont);
        AvailableLines=m_TextLinesPerPage-6;
        HeaderOffset=4;
    }
    else {
        Start=(m_TextLinesPerPage-6)+(Page-2)*(m_TextLinesPerPage-2);
        AvailableLines=m_TextLinesPerPage-2;
        HeaderOffset=0;
    }

    Field1.Format("Transient #");
    Field2.Format("FileName and Path");
    Field3.Format("Class");
    Field4.Format("Predicted");
    if(pDoc->m_bTrainingResults)
        Field5.Format("Error");
    else
        Field5.Format("Conf(%%)");
    Field6.Format("Activation");

    CFont RomanFontUnder;
    RomanFontUnder.CreateFont(m_FontHeight, 0, 0, 0, 400, FALSE, TRUE, 0,
                              ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                              CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                              DEFAULT_PITCH | FF_ROMAN, NULL);
    CFont *pOldFont=pDC->SelectObject(&RomanFontUnder);
    pDC->TextOut(m_LeftMargin, -HeaderOffset*m_FontHeight-m_TopMargin, Field1);
    pDC->TextOut(m_LeftMargin+Spacing, -HeaderOffset*m_FontHeight-m_TopMargin,
        Field2);
    pDC->TextOut(m_LeftMargin+4*Spacing, -HeaderOffset*m_FontHeight-m_TopMargin,
        Field3);
    pDC->TextOut(m_LeftMargin+5*Spacing, -HeaderOffset*m_FontHeight-m_TopMargin,
        Field4);
    pDC->TextOut(m_LeftMargin+6*Spacing, -HeaderOffset*m_FontHeight-m_TopMargin,
        Field5);
    pDC->TextOut(m_LeftMargin+7*Spacing, -HeaderOffset*m_FontHeight-m_TopMargin,
        Field6);

    HeaderOffset+=2;
    CFont RomanFont;
    RomanFont.CreateFont(m_FontHeight, 0, 0, 0, 400, FALSE, FALSE, 0,
```

```
                                ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                                DEFAULT_PITCH | FF_ROMAN, NULL);
        pDC->SelectObject(&RomanFont);

        double Error;
        int PredictedClass, CorrectClass;

        for (int ctr=Start; (ctr<Start+AvailableLines)
                &&ctr<pDoc->GetResultsArray()->GetSize(); ctr++) {
            Field1.Format("%d",ctr+1);
            Field2=pDoc->GetResultsArray()->GetAt(ctr)->GetRawFilePath();
            CorrectClass=pDoc->GetResultsArray()->GetAt(ctr)->GetCorrectClass();
            Field3 = pDoc -> GetArray() -> GetClassName(CorrectClass);
            PredictedClass=pDoc->GetResultsArray()->GetAt(ctr)->GetPredictedClass(0);
            Field4 = pDoc -> GetArray() -> GetClassName(PredictedClass);
            if (PredictedClass==CorrectClass)
                NumCorrect++;

            TotalError+=Error=pDoc->GetResultsArray()->GetAt(ctr)->GetError(0);
            if(Error>99.9999)
                Field5.Format("%.3lf",Error);
            else if (Error==0.0)
                Field5.Format("%.5lf",Error);
            else
                Field5.Format("%.4lf",Error);

            if(pDoc->m_bTrainingResults)
                Field6.Format("N/A");
            else
                Field6.Format("%.3e",pDoc->GetResultsArray()
                    ->GetAt(ctr)->GetActivation(0));

            pDC->TextOut(m_LeftMargin,
                (-HeaderOffset-(ctr-Start))*m_FontHeight-m_TopMargin, Field1);
            pDC->TextOut(m_LeftMargin+Spacing,
                (-HeaderOffset-(ctr-Start))*m_FontHeight-m_TopMargin, Field2);
            pDC->TextOut(m_LeftMargin+4*Spacing,
                (-HeaderOffset-(ctr-Start))*m_FontHeight-m_TopMargin, Field3);
            pDC->TextOut(m_LeftMargin+5*Spacing,
                (-HeaderOffset-(ctr-Start))*m_FontHeight-m_TopMargin, Field4);
            pDC->TextOut(m_LeftMargin+6*Spacing,
                (-HeaderOffset-(ctr-Start))*m_FontHeight-m_TopMargin, Field5);
            pDC->TextOut(m_LeftMargin+7*Spacing,
                (-HeaderOffset-(ctr-Start))*m_FontHeight-m_TopMargin, Field6);
        }

        if(Page==pInfo->GetMaxPage()) {
            int NumLines=(pDoc->GetResultsArray()->GetSize());
                HeaderOffset+=2;
            Field1.Format("Number of Correct Classifications: %d/%d.",
                NumCorrect, NumLines);
            pDC->TextOut(m_LeftMargin,
                (-HeaderOffset-(ctr-Start))*m_FontHeight-m_TopMargin, Field1);
                if(pDoc->m_bTrainingResults) {
                    Field1.Format("Average Error: %.12lf.", TotalError/NumLines);
                    pDC->TextOut(m_LeftMargin, (-HeaderOffset-(ctr-Start-1))
                        *m_FontHeight-m_TopMargin, Field1);
                }
        }

        pDC->SelectObject(pOldFont);

        return;
}
```

TextTable.h

```
// TextTable.h: interface for the CTextTable class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_TEXTTABLE_H__66C35B05_506A_11D2_8BA9_00A024423FB9__INCLUDED_)
#define AFX_TEXTTABLE_H__66C35B05_506A_11D2_8BA9_00A024423FB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CTextTable
{
public:
	CTextTable();
	CTextTable(int col, int row);
	virtual ~CTextTable();
	void SetSize(int col, int row);
	int GetRows();
	CRect PaintTable(CDC* pDC, int x, int y);
	void SetText(CString strText, int col, int row);


private:
	CStringArray m_TextArray;
	int		m_nRow;
	int		m_nCol;
	CDC*	m_pDC;
};

#endif // !defined(AFX_TEXTTABLE_H__66C35B05_506A_11D2_8BA9_00A024423FB9__INCLUDED_)
```

TextTable.cpp

```
// TextTable.cpp: implementation of the CTextTable class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "tac_mm.h"
#include "TextTable.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CTextTable::CTextTable() : m_nCol (0), m_nRow (0)
{
}


CTextTable::CTextTable(int col, int row) : m_nCol (col), m_nRow (row)
{
	// Couldn't get a two dimensional array to work
	// so use a one dimensional one.
	m_TextArray.SetSize(m_nCol*m_nRow);

}

CTextTable::~CTextTable()
{

}

void CTextTable::SetText(CString strText, int col, int row)
{
	ASSERT(col < m_nCol);
	ASSERT(row < m_nRow);

	m_TextArray[(m_nCol * row) + col] = strText;
}


CRect CTextTable::PaintTable(CDC* pDC, int x, int y)
{
	// Retrieve the size of characters for the selected font.
	TEXTMETRIC tm;
	pDC->GetTextMetrics(&tm);
	int cxChar = tm.tmAveCharWidth;
	int cyChar = tm.tmHeight + tm.tmExternalLeading;

	CSize sizeText;
	int xMax; // The maximum x position in each column.
	int xCurrentPos = x;
	int yCurrentPos = y;

	// Retrieve the current font and make a bold,
	// underlined font from it.
	CFont *pFont = pDC->GetCurrentFont();
```

```
        LOGFONT logfont;
        pFont->GetLogFont(&logfont);
        logfont.lfWeight += 200;

        CFont FontBold;
        FontBold.CreateFontIndirect(&logfont);



        // Paint each column
        for (int col = 0; col < m_nCol; col++)
        {
            xMax = xCurrentPos;
            yCurrentPos = y;

            // Paint each element in the column.
            for (int row = 0; row < m_nRow; row++)
            {
                if (row == 0)
                    pDC->SelectObject(&FontBold);
                else if (row == 1)
                    pDC->SelectObject(pFont);

                pDC->TextOut(xCurrentPos, yCurrentPos, m_TextArray[(m_nCol * row) + col]);

                // Calculate the size of the painted item.
                sizeText = pDC->GetTextExtent(m_TextArray[(m_nCol * row) + col]);
                // Adjust xMax if necessary.
                if (xCurrentPos + sizeText.cx > xMax)
                    xMax = xCurrentPos + sizeText.cx;

                yCurrentPos += cyChar;
            }
            xCurrentPos = xMax + 2*cxChar;

        }

        // Calculate the rect that contains the table and return iot
        CRect rectTable(x, y, xMax, yCurrentPos);
        return (rectTable);
}

int CTextTable::GetRows()
{
        return m_nRow;
}

void CTextTable::SetSize(int col, int row)
{
        m_nCol = col;
        m_nRow = row;

        m_TextArray.SetSize(col*row);
}
```

TrainingResults.h

```
//CTrainingResults: the storage class for the results obtained during training or from batch classification
#ifndef _TRAINRESULTS
#define _TRAINRESULTS

class CTrainingResults:public CObject
{
    //DECLARE_SERIAL(CTrainingResults)
protected:
    CString m_strRawFilePath;
    int m_nCorrectClass;
    int *m_nPredictedClass;
    double *m_dError;
    double *m_dWinningActivation;

public:
    CTrainingResults() { }
    CTrainingResults(int NumClassifyModes)
        { m_nPredictedClass=new int[NumClassifyModes];
          m_dError=new double[NumClassifyModes];
          m_dWinningActivation=new double[NumClassifyModes];
        }
    ~CTrainingResults() { delete [] m_nPredictedClass;
                          delete [] m_dError;
                          delete [] m_dWinningActivation;}

    void SetRawFilePath(CString FilePath) { m_strRawFilePath=FilePath; }
    void SetCorrectClass(int CorrectClass) { m_nCorrectClass=CorrectClass; }
    void SetPredictedClass(int PredictedClass, int ModeIndex)
        { m_nPredictedClass[ModeIndex]=PredictedClass; }
    void SetError(double Error, int ModeIndex)
        { m_dError[ModeIndex]=Error; }
    void SetActivation(double Activation, int ModeIndex)
        { m_dWinningActivation[ModeIndex]=Activation; }

    CString GetRawFilePath() { return m_strRawFilePath; }
    int GetCorrectClass() { return m_nCorrectClass; }
    int GetPredictedClass(int ModeIndex) {return m_nPredictedClass[ModeIndex]; }
    double GetError(int ModeIndex) {return m_dError[ModeIndex]; }
    double GetActivation(int ModeIndex)
        {return m_dWinningActivation[ModeIndex]; }

#ifdef _DEBUG
    void Dump(CDumpContext& dc) const;
#endif
};

typedef CTypedPtrArray <CObArray, CTrainingResults*> CResultsArray;

#endif
```

TrainingResults.cpp

```
#include "StdAfx.h"
#include "TrainingResults.h"

//IMPLEMENT_SERIAL(CTrainingResults, CObject,0)

#ifdef _DEBUG
void CTrainingResults::Dump(CDumpContext& dc) const
{
	CObject::Dump(dc);
	dc << "\nm_strRawFilePath = " << m_strRawFilePath;
}
#endif
```

TrainSigmasOptDialog.h

```
//CTrainSigmasOptDialog: defines and controls the dialog box displayed to update the user on the status of sigma optimization
// TrainSigmasOptDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CTrainSigmasOptDialog dialog

class CTrainSigmasOptDialog : public CDialog
{
// Construction
public:
	CTrainSigmasOptDialog(CWnd* pParent = NULL);  // standard constructor

	CTransientArray *m_TransientArray;
	CPNN *m_PNN;
	double m_dImprovementTolerance;

// Dialog Data
	//{{AFX_DATA(CTrainSigmasOptDialog)
	enum { IDD = IDD_TRAINSIGMASOPT_DIALOG };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CTrainSigmasOptDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CTrainSigmasOptDialog)
	afx_msg void OnStopbutton();
	afx_msg void OnTrainbutton();
	virtual BOOL OnInitDialog();
	afx_msg void OnTimer(UINT nIDEvent);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

TrainSigmasOptDialog.cpp

```
// TrainSigmasOptDialog.cpp : implementation file
//

#include "stdafx.h"
#include "TAC_MM.h"
#include "PNN.h"
#include "TrainSigmasOptDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CTrainSigmasOptDialog dialog


CTrainSigmasOptDialog::CTrainSigmasOptDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CTrainSigmasOptDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CTrainSigmasOptDialog)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}


void CTrainSigmasOptDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CTrainSigmasOptDialog)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CTrainSigmasOptDialog, CDialog)
	//{{AFX_MSG_MAP(CTrainSigmasOptDialog)
	ON_BN_CLICKED(IDC_STOPBUTTON, OnStopbutton)
	ON_BN_CLICKED(IDC_TRAINBUTTON, OnTrainbutton)
	ON_WM_TIMER()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTrainSigmasOptDialog message handlers

void CTrainSigmasOptDialog::OnStopbutton()
{
	CString strText;
	m_PNN->m_bStopNow=TRUE;
	strText.Format("Press Train to resume or OK to Exit.");
	SetDlgItemText(IDC_STATIC_CURRENT, strText);
	GetDlgItem(IDC_STOPBUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_TRAINBUTTON)->EnableWindow(TRUE);
	GetDlgItem(IDOK)->EnableWindow(TRUE);
}

void CTrainSigmasOptDialog::OnTrainbutton()
{
	BOOL ToleranceReached=FALSE;
```

```
    CString strText;

    GetDlgItem(IDC_STOPBUTTON)->EnableWindow(TRUE);
    GetDlgItem(IDC_TRAINBUTTON)->EnableWindow(FALSE);
    GetDlgItem(IDOK)->EnableWindow(FALSE);

    int Timer=SetTimer(1,1000,NULL); // ID #1,  1 second, NULL
    ASSERT(Timer!=0);
    ToleranceReached=m_PNN->TrainSigmasOpt(m_dImprovementTolerance);
    KillTimer(1);

    strText.Format("%.12lf",m_PNN->m_dUserDisplayError);
    SetDlgItemText(IDC_STATIC_ERROR, strText);
    CProgressCtrl* pProg=(CProgressCtrl*)GetDlgItem(IDC_INITPROGRESS);
    pProg->SetPos(m_PNN->m_nUserDisplayDiscreteError);
    strText.Format("%d",m_PNN->m_nUserDisplayDiscreteError);
    SetDlgItemText(IDC_STATIC_MISCLASS, strText);

    GetDlgItem(IDC_STOPBUTTON)->EnableWindow(FALSE);
    if(!ToleranceReached)
        GetDlgItem(IDC_TRAINBUTTON)->EnableWindow(TRUE);
    else {
        strText.Format("Tolerance of %.10f reached. Hit OK to Exit."
            ,m_dImprovementTolerance);
        SetDlgItemText(IDC_STATIC_CURRENT, strText);
    }
    GetDlgItem(IDOK)->EnableWindow(TRUE);
}

BOOL CTrainSigmasOptDialog::OnInitDialog()
{
    CDialog::OnInitDialog();

    CProgressCtrl* pProg=(CProgressCtrl*)GetDlgItem(IDC_INITPROGRESS);
    pProg->SetRange(0,m_TransientArray->GetSize());
    pProg->SetPos(0);
    CString strText;
    strText.Format("%d",m_TransientArray->GetSize());
    SetDlgItemText(IDC_STATIC_MAXERROR, strText);
    strText.Format("Press Train to begin.");
    SetDlgItemText(IDC_STATIC_CURRENT, strText);
    GetDlgItem(IDC_STOPBUTTON)->EnableWindow(FALSE);

    return TRUE;  // return TRUE unless you set the focus to a control
            // EXCEPTION: OCX Property Pages should return FALSE
}

void CTrainSigmasOptDialog::OnTimer(UINT nIDEvent)
{
    if (nIDEvent==1) {
        CString strText;
        if (!m_PNN->m_bStopNow)
            SetDlgItemText(IDC_STATIC_CURRENT, m_PNN->m_strUserMessage);
        strText.Format("%.12lf",m_PNN->m_dUserDisplayImprovement);
        SetDlgItemText(IDC_STATIC_IMPROVEMENT, strText);
        strText.Format("%.12lf",m_PNN->m_dUserDisplayError);
        SetDlgItemText(IDC_STATIC_ERROR, strText);
        CProgressCtrl* pProg=(CProgressCtrl*)GetDlgItem(IDC_INITPROGRESS);
        pProg->SetPos(m_PNN->m_nUserDisplayDiscreteError);
        strText.Format("%d",m_PNN->m_nUserDisplayDiscreteError);
        SetDlgItemText(IDC_STATIC_MISCLASS, strText);
    }

    CDialog::OnTimer(nIDEvent);
}
```

Trandata.h

```
//CTransientData: the storage class for individual transmitter transients
#include "GlobalStuff.h"// Added by ClassView

class CTransientData:public CObject
{
    DECLARE_SERIAL(CTransientData)
protected:
    FileType m_FileType;
    CString m_strTransientClassName;
    CString m_strTransmitterMake;
    CString m_strTransmitterModel;
    CString m_strTransmitterSerial;
    CString m_Date;
    CString m_Time;
    CString m_strRawFilePath;
    int m_nTransientBeginsHere;
    int m_nTransientClass;
    int m_nTransientModelSize;
    double *m_TransientModel;
public:
    FileType GetFileType();
    void SetFileType (FileType filetype);
    int GetTransientModelSize();
    void SetTransientClassName(CString ClassName);
    CTransientData() { }
    CTransientData(int TransientModelSize)
        { m_TransientModel=new double[TransientModelSize]; }
    ~CTransientData() { delete [] m_TransientModel; }
    void UpdateTransientModelSize(int NewModelSize)
        { delete [] m_TransientModel;
          m_TransientModel=new double[NewModelSize];
          m_nTransientModelSize=NewModelSize; }
    void Serialize(CArchive& ar);
    void SetRawFilePath(CString FilePath) { m_strRawFilePath=FilePath; }
    void SetTransientBeginsHere(int Here) { m_nTransientBeginsHere=Here; }
    void SetTransmitterMake(CString Make) { m_strTransmitterMake=Make; }
    void SetTransmitterModel(CString Model) { m_strTransmitterModel=Model; }
    void SetTransmitterSerial(CString Serial) { m_strTransmitterSerial=Serial; }
    void SetTransmitDate(CString Date) { m_Date=Date; }
    void SetTransmitTime(CString Time) { m_Time=Time; }
    void SetTransientModelSize(int TransientModelSize) {m_nTransientModelSize=
        TransientModelSize; }
    void SetTransientModel(double *TransientModel);
    void SetTransientClass(int TranClass) { m_nTransientClass=TranClass; }


    int GetTransientBeginsHere() { return m_nTransientBeginsHere; }
    CString GetClassName();
    CString GetRawFilePath() { return m_strRawFilePath; }
    CString GetTransmitterMake() { return m_strTransmitterMake; }
    CString GetTransmitterModel() { return m_strTransmitterModel; }
    CString GetTransmitterSerial() { return m_strTransmitterSerial; }
    CString GetTime() { return m_Time; }
    CString GetDate() { return m_Date; }
    double* GetTransientModel() { return m_TransientModel; }
    int GetTransientClass() { return m_nTransientClass; }

#ifdef _DEBUG
    void Dump(CDumpContext& dc) const;
#endif
};
```

Trandata.cpp

```
#include "StdAfx.h"
#include "Trandata.h"

IMPLEMENT_SERIAL(CTransientData, CObject,0)

#ifdef _DEBUG
void CTransientData::Dump(CDumpContext& dc) const
{
	CObject::Dump(dc);
	dc << "\nm_strTransmitterMake = " << m_strTransmitterMake;
}
#endif

void CTransientData::Serialize(CArchive& ar)
{
	int ctr;

	if(ar.IsStoring())
	{
		ar << m_Date << (LONG)m_nTransientBeginsHere << (LONG)m_nTransientClass << m_strTransientClassName
			<< m_strRawFilePath << m_strTransmitterMake << m_strTransmitterModel
			<< m_strTransmitterSerial << m_Time << m_nTransientModelSize;

		for (ctr=0; ctr<m_nTransientModelSize; ctr++)
		{
			ar << m_TransientModel[ctr];
		}
	}
	else
	{
		ar >> m_Date >> (LONG&)m_nTransientBeginsHere >> (LONG&)m_nTransientClass >> m_strTransientClassName
			>> m_strRawFilePath >> m_strTransmitterMake >> m_strTransmitterModel
			>> m_strTransmitterSerial >> m_Time >> m_nTransientModelSize;


		if (m_strRawFilePath.Right(3).CompareNoCase("pcm") == 0)
		{
			m_FileType = PCM;
		}
		else
		{
			m_FileType = RAW;
		}


		m_TransientModel=new double[m_nTransientModelSize];
		for (ctr=0; ctr<m_nTransientModelSize; ctr++)
		{
			ar >> m_TransientModel[ctr];
		}
	}
}


void CTransientData::SetTransientModel(double *TransientModel)
{
	int ctr;

	for (ctr=0; ctr<m_nTransientModelSize; ctr++) {
		m_TransientModel[ctr]=TransientModel[ctr];
	}
```

```
}


CString CTransientData::GetClassName()
{
	return m_strTransientClassName;
}

void CTransientData::SetTransientClassName(CString ClassName)
{
	m_strTransientClassName = ClassName;
	return;
}

int CTransientData::GetTransientModelSize()
{
	return m_nTransientModelSize;
}

void CTransientData::SetFileType(FileType filetype)
{
	m_FileType = filetype;
}

FileType CTransientData::GetFileType()
{
	return m_FileType;
}
```

TransientArray.h

```
// TransientArray.h: interface for the CTransientArray class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_TRANSIENTARRAY_H__D641E1E6_05BF_11D2_8BA7_00A024423FB9__INCLUDED_)
#define AFX_TRANSIENTARRAY_H__D641E1E6_05BF_11D2_8BA7_00A024423FB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "TranData.h"

class CTransientArray : public CTypedPtrArray<CObArray, CTransientData*>
{
public:
	int SearchForClassName(CString ClassName);
	CString GetClassName(int nClassNum);
	CTransientArray();
	virtual ~CTransientArray();

	int FindNumClasses();
	int SearchForClass(int Class);
	int SearchForMake(CString Make);
	int SearchForModel(CString Model);
	int SearchForSerial(CString Serial);
	int SearchForDate(CString Date);
	int SearchForTime(CString TTime);

};

#endif // !defined(AFX_TRANSIENTARRAY_H__D641E1E6_05BF_11D2_8BA7_00A024423FB9__INCLUDED_)
```

TransientArray.cpp

```
// TransientArray.cpp: implementation of the CTransientArray class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "tac_mm.h"
#include "TransientArray.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CTransientArray::CTransientArray()
{

}

CTransientArray::~CTransientArray()
{

}


int CTransientArray::FindNumClasses()
{
	int Biggest=0;

	if (GetSize() == 0)
		return 0;  // No transients yet.

	for(int ctr=0; ctr < GetSize(); ctr++)
	{
		if (GetAt(ctr)->GetTransientClass() > Biggest)
		{
			Biggest = GetAt(ctr) -> GetTransientClass();
		}
	}
	return (Biggest + 1);  // +1 to account for the zero index.
}



int CTransientArray::SearchForClass(int Class)
{
	for(int ctr=0; ctr<GetSize(); ctr++)
	{
		if (GetAt(ctr)->GetTransientClass()==Class)
		{
			return ctr;
		}
	}
	return -1;//search unsuccessful
}

int CTransientArray::SearchForMake(CString Make)
```

```
{
	for(int ctr=0; ctr<GetSize(); ctr++)
	{
		if (GetAt(ctr) -> GetTransmitterMake()==Make)
		{
			return ctr;
		}
	}
	return -1;//search unsuccessful
}

int CTransientArray::SearchForModel(CString Model)
{
	for(int ctr=0; ctr < GetSize(); ctr++)
	{
		if (GetAt(ctr) -> GetTransmitterModel()==Model)
		{
			return ctr;
		}
	}
	return -1;//search unsuccessful
}

int CTransientArray::SearchForSerial(CString Serial)
{
	for(int ctr=0; ctr < GetSize(); ctr++)
	{
		if (GetAt(ctr) -> GetTransmitterSerial()==Serial)
		{
			return ctr;
		}
	}
	return -1;//search unsuccessful
}

int CTransientArray::SearchForDate(CString Date)
{
	for(int ctr=0; ctr < GetSize(); ctr++)
	{
		if (GetAt(ctr) -> GetDate()==Date)
		{
			return ctr;
		}
	}
	return -1;//search unsuccessful
}

int CTransientArray::SearchForTime(CString TTime)
{
	for(int ctr=0; ctr < GetSize(); ctr++)
	{
		if (GetAt(ctr) -> GetTime()==TTime)
		{
			return ctr;
		}
	}
	return -1;//search unsuccessful
}

CString CTransientArray::GetClassName(int nClassNum)
{
	CString strClassName;

	for (int i = 0; i < GetSize(); i++)
	{
```

```
            if ((GetAt(i) -> GetTransientClass()) == nClassNum)
            {
                strClassName = GetAt(i) -> GetClassName();
                return strClassName;
            }
        }

        return "Unknown"; // Couldn't find the class.
}

int CTransientArray::SearchForClassName(CString ClassName)
{
        for(int ctr=0; ctr < GetSize(); ctr++)
        {
            if (ClassName.CompareNoCase(GetAt(ctr) -> GetClassName()) == 0)
            {
                // Strings are identical (non-case sensitive).
                return ctr;
            }
        }
        return -1;// No match found.
}
```

TransientListView.h

```
#if !defined(AFX_TRANSIENTLISTVIEW_H__5C294EE3_0C25_11D2_8BA8_00A024423FB9__INCLUDED_)
#define AFX_TRANSIENTLISTVIEW_H__5C294EE3_0C25_11D2_8BA8_00A024423FB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// TransientListView.h : header file
//
#include "ListVwEx.h"


class CTAC_MMDoc;
class CTransientData;

/////////////////////////////////////////////////////////////////////////////
// CTransientListView view

class CTransientListView : protected CListViewEx
{
protected:
	CTransientListView(); // protected constructor used by dynamic creation
	DECLARE_DYNCREATE(CTransientListView)

// Attributes
public:

// Operations
public:
	void SetCurrentSelection(int row);
	void RemoveAllItems();
	int m_nPage;
	void RemoveTransientData(int index);
	void ShowNetworkDetails();
	void AddTransientData(int nItem, CTransientData* TranData);



// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CTransientListView)
	public:
	virtual void OnInitialUpdate();
	protected:
	virtual void OnDraw(CDC* pDC);	// overridden to draw this view
	//}}AFX_VIRTUAL

// Implementation
protected:
	virtual ~CTransientListView();
	CTAC_MMDoc* GetDocument();

#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

	// Generated message map functions
protected:
	int m_nColumns;
	CImageList m_ImageList;
	//{{AFX_MSG(CTransientListView)
```

```
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnItemchanged(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnHeaderItemdblclick(NMHDR* pNMHDR, LRESULT* pResult);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_TRANSIENTLISTVIEW_H__5C294EE3_0C25_11D2_8BA8_00A024423FB9__INCLUDED_)
```

TransientListView.cpp

```
// TransientListView.cpp : implementation file
//

#include "stdafx.h"
#include "tac_mm.h"
#include "TAC_MMDoc.h"
#include "TransientListView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////////////////////////////////
// CTransientListView

IMPLEMENT_DYNCREATE(CTransientListView, CListViewEx)

CTransientListView::CTransientListView()
{
}

CTransientListView::~CTransientListView()
{
}


BEGIN_MESSAGE_MAP(CTransientListView, CListViewEx)
    //{{AFX_MSG_MAP(CTransientListView)
    ON_WM_CREATE()
    ON_NOTIFY_REFLECT(LVN_ITEMCHANGED, OnItemchanged)
    ON_NOTIFY(HDN_ITEMDBLCLICK, 0, OnHeaderItemdblclick)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTransientListView drawing

void CTransientListView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}

/////////////////////////////////////////////////////////////////////////////
// CTransientListView diagnostics

#ifdef _DEBUG
void CTransientListView::AssertValid() const
{
    CListView::AssertValid();
}

void CTransientListView::Dump(CDumpContext& dc) const
{
    CListView::Dump(dc);
}
#endif //_DEBUG
```

```
/////////////////////////////////////////////////////////////////////////////
// CTransientListView message handlers

CTAC_MMDoc* CTransientListView::GetDocument()
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTAC_MMDoc)));
	return (CTAC_MMDoc*)m_pDocument;
}

int CTransientListView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	lpCreateStruct->style |= LVS_REPORT | LVS_SINGLESEL;
	if (CListView::OnCreate(lpCreateStruct) == -1)
		return -1;

	// TODO: Add your specialized creation code here
	CTAC_MMDoc* pDoc = GetDocument();
	pDoc -> m_pTransientListView = this; //Give the document a poiter to this view.

	return 0;
}



void CTransientListView::ShowNetworkDetails()
{
	// Desplay the column headings.
	DisplayColumnHeadings(IDS_COL_TRANDATA);

	CTAC_MMDoc* pDoc = GetDocument();
	int nTransients = pDoc -> GetArray() -> GetSize();
	CTransientData* pTranData;

	SetRedraw(FALSE); // Don't update the list view until it is filled.
	for (int i = 0; i < nTransients; i++)
	{
		pTranData = pDoc -> GetArray() -> GetAt(i);
		AddTransientData(i, pTranData);
	}
	SetRedraw(TRUE); // Show changes now.

}

void CTransientListView::AddTransientData(int nItem, CTransientData* TranData)
{
	CString Index,
		ClassNumber,
		ModelSize;


	Index.Format("%d",nItem);
	ClassNumber.Format("%d",TranData -> GetTransientClass());
	ModelSize.Format("%d",TranData -> GetTransientModelSize());

	AddItem(nItem,0,TranData -> GetRawFilePath());
	AddItem(nItem,1,TranData -> GetClassName());
	AddItem(nItem,2,ClassNumber);
	AddItem(nItem,3,ModelSize);
	AddItem(nItem,4,TranData -> GetTransmitterMake());
	AddItem(nItem,5,TranData -> GetTransmitterModel());
	AddItem(nItem,6,TranData -> GetTransmitterSerial());
	AddItem(nItem,7,TranData -> GetDate());
	AddItem(nItem,8,TranData -> GetTime());
}
```

```
void CTransientListView::OnInitialUpdate()
{
	CListView::OnInitialUpdate();

	// TODO: Add your specialized code here and/or call the base class
	ShowNetworkDetails();
	AutoSizeColumns();
}

void CTransientListView::OnItemchanged(NMHDR* pNMHDR, LRESULT* pResult)
{
	NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;

	// TODO: Add your control notification handler code here
	int nItemSelected = pNMListView -> iItem;
	CTAC_MMDoc* pDoc = GetDocument();

	if (nItemSelected >= 0)
	{
		RepaintSelectedItems();
		CTAC_MMDoc* pDoc = GetDocument();
		pDoc -> SetCurrentTransient(nItemSelected);
		pDoc -> ViewNeedsUpdating();
		pDoc -> UpdateStatusBar();
	}

	*pResult = 0;
}

void CTransientListView::OnHeaderItemdblclick(NMHDR* pNMHDR, LRESULT* pResult)
{
	HD_NOTIFY *phdn = (HD_NOTIFY *) pNMHDR;
	// TODO: Add your control notification handler code here

	AutoSizeColumns(phdn -> iItem);

	*pResult = 0;
}

void CTransientListView::RemoveTransientData(int index)
{
	CListCtrl& ListCtrl = GetListCtrl();
	ListCtrl.DeleteItem(index);

}



void CTransientListView::RemoveAllItems()
{
	CListCtrl& ctlList = GetListCtrl();
	ctlList.DeleteAllItems();
	AutoSizeColumns();
}

void CTransientListView::SetCurrentSelection(int row)
{
	CListCtrl& ctlList = GetListCtrl();
	ctlList.SetItemState( row, LVIS_SELECTED | LVIS_FOCUSED , LVIS_SELECTED | LVIS_FOCUSED);
}
```