MULTIFRACTAL CHARACTERIZATION FOR CLASSIFICATION OF NETWORK TRAFFIC

by

ROBERT L. BARRY

An M.Sc. Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba, Canada

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

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

MULTIFRACTAL CHARACTERIZATION FOR CLASSIFICATION OF NETWORK TRAFFIC

BY

Robert L. Barry

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

To my parents.

For their belief in me when mine was failing.

ABSTRACT

This thesis investigates the use of multifractal analysis to characterize network traffic and to facilitate reliable real-time traffic classification. In 1993, a seminal study by Leland *et al.* revealed the existence a self-affine structure within network traffic. However, despite this discovery, many researchers continued to use traditional techniques of traffic analysis and modelling that did not exploit this knowledge of self-affinity. To demonstrate the general versatility of multifractal techniques to characterize self-affine traffic, this thesis investigates the characterization and classification of a traffic recording from Pear's self-affine data sets which contains an unknown number of classes. To characterize the traffic, the variance fractal dimension trajectory (VFDT) is calculated using a carefully selected window size and window offset. The statistical mean, variance, skewness, and kurtosis are calculated for the VFDT, forming four new statistical trajectories. The histograms of these statistical trajectories are calculated for another appropriate window size, and their stationarity is modelled using the gamma distribution. The resulting eight parameters (two for each of the four gamma distributions) are further reduced to only four parameters using principal component analysis, and the K-means clustering algorithm and Kohonen's self-organizing feature map are used to cluster the data. A locally optimal spread parameter σ is determined for each probabilistic neural network (PNN) configuration, and a plot of PNN percentage classification accuracy as the number of classes increases reveals that there are most likely three classes in the traffic recording. Finally, an optimized PNN is trained with 50% of the multifractal signatures sampled at regular intervals from the trajectory, and achieves a representative correct classification accuracy of 95% when classifying previously unobserved self-affine traffic.

ACKNOWLEDGEMENTS

I would like to acknowledge and thank the support of my advisor, Dr. Witold Kinsner, for his guidance for the past three years, and for suggesting the topic of this thesis. I would also like to thank the Director of TR*Labs*, Dr. Jose A. Rueda, for welcoming me into the TR*Labs* family. A special thanks also goes out to Dr. Joseph Pear and Mr. Toby Martin in the Department of Psychology for sharing with our Delta research group their unique data sets of behavioural modification in Siamese Fighting Fish (*Betta splendens*).

I gratefully acknowledge the support of everybody in the Delta Research Group, past and present, including Kalen Brunham, Kevin Cannons, Jin Chen, Vincent Cheung, Dr. Richard Dansereau, Stephen Dueck, Hakim El Boustani, Dr. Ken Ferens, Neil Gadhok, Aram Faghfouri, Dr. Bin Huang, Jay Kraut, Michael Potter, Leila Safavian, Sharjeel Siddiqui, and Hong Zhang. This research would not have been possible without the support of TR*Labs* – a quiet haven where I could study and write this thesis. I would like to thank my colleagues there, including Dr. Attahiru Alfa, Dr. Jeff Diamond, Dr. Robert McLeod, Dr. James Schellenberg, Mr. George Ortega, Douglas Cornelsen, Blake Podaima, Paul Ratti, Alice Rueda, Glenda Stark, Julie Stewart, Clint Stuart, T. (Vasee) Vaseeharan, and Adam Zilkie. I also gratefully acknowledge the financial support of TR*Labs*.

This thesis is dedicated to my family. Their unwavering support over the years has helped me transform many of my dreams into realities.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS, ACRONYMS, AND TRANSLATIONS

2D	**2-d**imensional
4D	**4-d**imensional
8D	**8-d**imensional
Apr.	**Apr**il
a priori	(Latin) ; previously observed
ACM	The **A**ssociation for **C**omputing **M**achinery
ad infinitum	(Latin) ; to infinity
AIEE	**A**merican **I**nstitute of **E**lectrical **E**ngineers
ANN	**A**rtificial **N**eural **N**etwork
approx.	**approx**imately
ATM	**A**synchronous **T**ransfer **M**ode
B.Sc.	**B**achelor of **Sc**ience
bps	**b**its **p**er **s**econd
CCECE	**C**anadian **C**onference on **E**lectrical and **C**omputer **E**ngineering

CCS	**C**ommon **C**hannel **S**ignaling
CCSN	**C**ommon **C**hannel **S**ignaling **N**etwork
cm	**c**enti**m**etre(s)
Conf.	**Conf**erence
Dec.	**Dec**ember
DNA	**D**eoxyribo**n**ucleic **A**cid
DTMF	**D**ual **T**one **M**ulti-**F**requency
ed.	**ed**ition
Ed.	**Ed**itors
Eq(s).	**Eq**uation(**s**)
EMG	**E**lectro**m**yo**g**ram
et al.	**et al**ia (Latin) ; and others
Feb.	**Feb**ruary
Fig(s).	**Fig**ure(**s**)
FSK	**F**requency **S**hift **K**eying
FTP	**F**ile **T**ransfer **P**rotocol

hr(s)	**h**our(**s**)
Hz	**H**er**z**
i.e.	**i**d **e**st (Latin) ; that is
IEEE	The **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
i.i.d.	**i**ndependent and **i**dentically **d**istributed
Inc.	**Inc**orporated
INFOCOM	The Conference on **Com**puter **Com**munications
Intern.	**Intern**ational
IP	**I**nternet **P**rotocol
ISDN	**I**ntegrated **S**ervices **D**igital **N**etwork
J.	**J**ournal
Jan.	**Jan**uary
KB	**k**ilo**b**yte (or 1,024 bytes)
km	**k**ilo**m**etre(s) (or 1,000 metres)
LAN	**L**ocal **A**rea **N**etwork
LPF	**L**ow-**P**ass **F**ilter

LRD	**L**ong-**R**ange **D**ependence
Mar.	**Mar**ch
MATLAB	**Mat**rix **Lab**oratory
MB	**m**ega**b**yte (or 1,024 kilobytes)
min	**min**ute(s)
mm	**mi**lli**me**tre(s)
msec	**mi**lli**sec**ond(s)
M.Sc.	**M**aster of **Sc**ience
MSE	**M**ean **S**quare **E**rror
no.	**n**umber
Oct.	**Oct**ober
PCA	**P**rincipal **C**omponent **A**nalysis
PCM	**P**ulse **C**ode **M**odulation
PDF	**P**robability **D**ensity **F**unction
P.Eng.	**P**rofessional **Eng**ineer
Ph.D.	**Ph**ilosophiae **D**octor (Latin) ; **D**octor of **Ph**ilosophy

PNN	**P**robabilistic **N**eural **N**etwork
p.	**p**age
pp.	**p**ages
Proc.	**Proc**eedings
QoS	**Q**uality **o**f **S**ervice
Rep.	**Rep**ort
Rev.	**Rev**iew
sec	**sec**ond(s)
Sep.	**Sep**tember
SIGCOMM	**S**pecial **I**nterest **G**roup on Data **Comm**unications
SOFM	**S**elf-**O**rganizing **F**eature **M**ap
SPIE	The International **S**ociety for **O**p**t**ical **E**ngineering
SRD	**S**hort-**R**ange **D**ependence
SS7	**S**ignaling **S**ystem Number **7**
Tech.	**Tech**nical
TELNET	**Tel**etype **Net**work

Trans.	**Trans**actions
TR*Labs*	**T**elecommunications **R**esearch **Lab**oratorie**s**
UDP	**U**ser **D**atagram **P**rotocol
VBR	**V**ariable **B**it **R**ate
vel	**v**olume **el**ement
VoIP	**V**oice **o**ver **IP**
vol.	**vol**ume
VFDT	**V**ariance **F**ractal **D**imension **T**rajectory
WAN	**W**ide **A**rea **N**etwork
WWW	**W**orld **W**ide **W**eb

CHAPTER I

INTRODUCTION

1.1 Problem Definition

In 1993, a seminal study at Bell-Core by Leland, Taqqu, Willinger, and Wilson revealed an inherent self-similar (or more accurately, self-affine) nature in Ethernet traffic [LTWW94]. This discovery sparked a new wave of research around the world, and it soon became clear to researchers that self-affinity in network traffic was not merely a computer-induced artifact, but rather a fundamental property of the traffic itself.

Self-affine traffic possesses bursty structural similarities over a wide range of time scales. The importance of recognizing self-affinity in network traffic is realized in the problem of optimal resource allocation in dynamic operating conditions. A network engineer must satisfy as many network users as possible, given certain budgetary and practical constraints. A trade-off between network capacity and Quality of Service (QoS) requirements must be resolved by optimizing resource allocation algorithms to guarantee that the service provided to the end user meets the QoS constraints while maintaining maximum capacity [SaTe99].

Network control through optimal usage of resources is only possible through characterization and classification of the network traffic, and then by determining optimal buffer sizes, assignment of bandwidth and channels, and other resources for each class of traffic in order to achieve the desired QoS in terms of queuing delay, retransmission time, packet loss probability, and bit error rate [SaTe99]. Therefore, the three stages in this area

of study are (1) characterization, (2) classification, and (3) control. The first two stages, characterization and classification, will be the focus of this thesis.

To characterize and classify the network traffic, a real-time classifier must be developed to monitor the traffic. Traffic classification is not a new problem, and several classifiers have already been developed for this purpose. However, most products range greatly in sophistication and usefulness. Some classifiers can only distinguish between speech and non-speech [Benv93], whereas others can classify traffic into general categories of voice, data, and facsimile [Sard99]. The reliability of these classifiers is also of paramount importance, as some can correctly classify traffic over 90-95% of the time, and others have classification accuracies of only approximately 72% [Sewa96]. More general and highly reliable classifiers do exist that are able to recognize twelve signal classes with an accuracy of approximately 99% [Sard99].

However, none of the aforementioned classifiers utilize the knowledge that network traffic is self-affine in order to facilitate the implementation, or to improve the performance, of the classifier. Understanding the self-affine nature of network traffic is an important piece of knowledge that is used in this thesis to design and implement a new multifractal-based traffic classifier.

1.2 Thesis Statement and Objectives

The objectives of this thesis are: (1) to demonstrate the self-affine nature of Pear's data sets, (2) to characterize this traffic using multifractal analysis, (3) to cluster these characteristic features into natural classes, and (4) to train a neural network classifier with these features to classify previously unobserved traffic.

The variance fractal dimension [MaVa68], [Kins94a] is used to demonstrate the self-affine nature of the traffic, and the variance fractal dimension trajectory (VFDT) [Kins94a] of the traffic is calculated using a carefully selected window size and offset. The mean, variance, skewness, and kurtosis are calculated for each window of the VFDT, forming four new statistical trajectories. The histograms of these statistical trajectories are calculated for another appropriate window size, and their stationarity is modelled using the gamma distribution [Weis99a]. The resulting eight parameters (two for each of the four gamma distributions) are further reduced to only four parameters with principal component analysis [HyKO01, Ch. 6], and the K-means clustering algorithm [MacQ67], [Hart75, Ch. 4] is then used to determine the classes in the multifractal signatures. A self-organizing feature map [Koho88, Ch. 5], [Koho90] is also used to independently verify the results of the K-means clustering algorithm. A probabilistic neural network is trained with these signatures [Spec88], [Spec90a] and its performance on classifying unknown traffic is used to indicate the most likely number of classes in the data.

The primary data set used in this research is Record 11020219 from Pear's data sets, although other data sets are also discussed.

1.3 Organization of the Thesis

This thesis is organized into seven chapters. Chapter 1 presents the motivation behind this thesis and the formal definition of the thesis statement. Chapter 2 provides the necessary background on traffic, self-affine traffic, and the data sets studied in this thesis. Chapter 3 introduces fractals, fractal dimensions, multifractals, the calculation of the variance fractal dimension trajectory, and the relationship between multifractals and self-affine traffic. Chapter 4 highlights important issues in feature extraction and neural

network classification, including the use of higher-order statistics, histogram modelling, principal component analysis, K-means clustering, self-organizing feature maps, and probabilistic neural networks. Chapter 5 describes the design of the system and its verification, including the selection of the proper window sizes to be used for the calculation of the variance fractal dimension trajectory and the modelling of its statistical histograms. Chapter 6 discusses the characterization, verification, and classification experiments performed on Record 11020219, and the results of these experiments. Finally, Chapter 7 reflects upon the thesis statement by presenting the conclusions and contributions of this thesis, as well as recommendations for future work.

CHAPTER II

BACKGROUND ON NETWORK TRAFFIC

2.1 What is Traffic?

If you ask the question "What is traffic?" to ten people, you will most likely get ten different answers. A bus driver may say that traffic is the motion of automobiles on roads, and a network administrator may say that traffic is the number of "hits" on a web site. The first definition describes objects composed of metal and plastics burning hydrocarbons as they drive along roads of gravel, asphalt, or concrete. The second definition describes packets of binary information travelling across networks of copper or fiber optics cables. At first glance, these two definitions of traffic may seem totally dissimilar, but perhaps a more general definition of traffic exists that encompasses both of these definitions.

Traffic may be defined as *the movement of objects through a network.* In our example of the motion of automobiles on roads, the objects are automobiles and the network would be roads. Similarly, in our example of the number of "hits" on a web site, the objects are packets of binary information and the network is composed of interconnected computers.

If we take this analogy one step further, we may also say that both of these networks also have a measurable Quality of Service (QoS) associated with it. Drivers are happy when they are able to get from where they are to where they are going in a timely fashion, and with a minimal number of detours or inconveniences along the way. Similarly, web surfers are happy when they are able to access any web site and not have to

wait too long for its contents to load. Degradation in the QoS of road traffic occurs when drivers experience traffic jams during "rush hour", or when trains block the normal rhythm of traffic flow. Degradation in the QoS of network traffic happens when web sites take a long time to view, or files take a long time to download. QoS issues in road traffic are suited to the experience of city planners and politicians. Thankfully, QoS issues in network traffic are suited to network engineers. For this reason, this thesis will focus on traffic related to computer networks.

2.2 Previous Research in Traffic Classification

Characterization and classification of network traffic is an area of research that has progressed for several years under the supervision of Dr. Bruce Cockburn at the University of Alberta. Dr. Cockburn was the advisor for Deepak Sarda, a former TR*Labs* graduate student, who developed a real-time voiceband signal classifier for his M.Sc. thesis in 1999 that can classify 24 channels on a T1 line in a telephone network with about 99% accuracy [Sard99]. These twelve signal classes consist of four data modem classes, four facsimile classes, random binary, Frequency Shift Keying (FSK) signaling, ringback, and a class containing the twelve Dual Tone Multi-Frequency (DTMF) tones [Sard99].

Sarda's classifier is based on the algorithms developed by Jeremy Sewall, another former TR*Labs* graduate student, for his M.Sc. thesis in 1996. Sewall's algorithms involve the use of the first ten normalized autocorrelation sequence lags and the normalized second order-central moment with discriminant analysis to make a decision as to which of the twelve signal classes a segment of observed data most likely belongs [Sewa96]. Linear, quadratic, and hybrid discriminant functions were used to discriminate between the signal classes. The development of these algorithms by Sewall was made

possible by the prior research of Nevio Benvenuto, who was able to correctly classify between speech and non-speech with an accuracy of about 99% [Benv93].

Although Sarda's voiceband signal classifier does an excellent job of classifying traffic on a T1 line, its design imposes a severe limitation on how effectively it can be used for future applications. One of the most desirable features of any product is the ease with which it can be expanded or upgraded. An expansion of Sarda's classifier to include new signal classes would not be trivial, and may even require a new design. The proposed multifractal-based traffic classifier would resolve this design issue through a modular design where each class of traffic would possess a unique compressed multifractal signature. Removing outdated traffic classes or adding new traffic classes would simply involve the deletion or addition of the new traffic signatures and a re-training of the neural network.

2.3 Self-Affinity in Traffic

Self-affine traffic possesses bursty structural similarities over a wide range of time scales, such as milliseconds, seconds, minutes, hours, and even days or weeks [SaTe99]. A new wave of research was sparked that showed the existence of self-affinity in Local Area Network (LAN) traffic [LTWW94], Wide Area Network (WAN) traffic [PaFl95], World Wide Web (WWW) traffic [CrBe97], Teletype Network (TELNET) traffic [PaFl95], Integrated Services Digital Network (ISDN) traffic [LLTW94], File Transfer Protocol (FTP) traffic [PaFl95], Frame Relay traffic [FoWi98], Signaling System 7 (SS7) traffic [DuWi94], and Variable Bit Rate (VBR) video traffic [GaWi94], over Asynchronous Transfer Mode (ATM) networks [BSTW95]. These studies clearly showed that self-affinity is not a computer-induced artifact, but a fundamental property of network

traffic. In 1995, Willinger *et al.* showed that self-affine traffic may be generated through ON/OFF processes [WTSW95]. Earlier work in 1990 by McLeod, Schellenberg, and Hortensius at the University of Manitoba demonstrated the generation of self-affine traffic through various configurations of the network topology [McSH90].

2.3.1 Generation Through ON/OFF Processes

In 1995, a theory for the generation of self-affine traffic was proposed by Willinger *et al.* [WTSW95]. This theory extended an approach originally suggested by Mandelbrot in 1969 that the superposition of many ON/OFF processes with high variability produces self-affine traffic [Mand69]. This theory was later proved by Taqqu *et al.* in 1997 using a different mathematical approach that does not follow from the work by Mandelbrot [TaWS97]. A summary of this theory that follows Mandelbrot's work will now be presented [WTSW95].

An *idealized* ON/OFF source model, also called a *packet train* model, is characterized by a reward sequence $\{W(l), l = 0, 1, \ldots\}$: a discrete time stochastic process with $\{W(l)\} = 0$ or 1, depending on whether or not there was a packet at time l. The reward sequence $\{W(l)\}$ consists of a sequence of 1's if packets are transmitted (an "ON- period") and 0's if no packets are being transmitted (an "OFF-period"). Let $p \equiv P$ (a given period is an ON-period) = ½ , and assume that the lengths of the ON-periods and OFF-periods are governed by independent and identically distributed (i.i.d.) random variables denoted by U_k , $k = 0, 1, \ldots$. U denotes an arbitrary U_k with finite expectation $\varepsilon(U)$. Let $S_k = S_0 + U_1 + U_2 + \ldots + U_k$, $k \geq 0$ be the corresponding renewal times. $\{S_k, k \geq 0\}$ is made stationary by choosing the distribution of S_0 in the following way:

$$(P(S_0 = u) = (\varepsilon(U))^{-1} P(U \geq u + 1)), u = 0, 1, 2, \ldots \quad (2.1)$$

To ensure the stationarity of the sequence $\{W(l), l \geq 0\}$, let $l = 0$ be an ON-period with probability of ½.

Suppose that there are M i.i.d. sources where the m^{th} source ($m = 1, ..., M$) has its own reward sequence $\{W^{(m)}(l), l \geq 0\}$. Then the cumulative reward (also called the *packet load*) at time l is

$$\sum_{m=1}^{M} W^{(m)}(l)$$

Aggregating this load through non-overlapping time blocks of size b, where j denotes the block number, we get

$$W^*_{M,b}(j) = \sum_{l=bj+1}^{b(j+1)} \sum_{m=1}^{M} W^{(m)}(l), j = 0, 1, 2, \ldots \quad (2.2)$$

The statistical behaviour of the sequence $\{W^*_{M,b}\}$ is interesting for large M and b. This behaviour can only depend on the distribution of U, which still needs to be specified.

Motivated by the empirically derived fractional Gaussian noise model for aggregate packet traffic [LTWW94], the distribution of U should be chosen in a way such that as $M \to \infty$ and $b \to \infty$, $\{W^*_{M,b}\}$ adequately normalized is *fractional Gaussian noise* $\{G_{H,\sigma}(t), t \geq 0\}$: the only Gaussian sequence which is self-affine at all scales with Hurst exponent H in the range $\frac{1}{2} \leq H < 1$. This means that the finite-dimensional

distributions of

$$b^{-H} \sum_{l=bj+1}^{b(j+1)} G_{H,\sigma}(l), j = 0, 1, 2, \ldots$$

are the same for any block aggregation size b.

To obtain fractional Gaussian noise, suppose that U has a hyperbolic tail distribution that satisfies

$$P(U > u) \sim cu^{-\alpha} \text{ as } u \to \infty, 1 < \alpha < 2 \tag{2.3}$$

where c is a positive finite constant that is independent of u.

Mandelbrot refers to Eq. 2.3 as the *infinite variance syndrome* or the *Noah Effect* [MaWa68], [Mand75, p. 105], [Mand83, p. 248]. A value of $\alpha < 2$ implies $\varepsilon(U^2) = \infty$, while the choice $\alpha > 1$ ensures that $\varepsilon(U) < \infty$ and S_0 is not infinite. Under these conditions, Theorem 2.1 is true.

Theorem 2.1: For large enough source number M and block aggregation size b, the cumulative load $\{W^*_{M,b}(j), j \geq 0\}$ behaves statistically as

$$bM\frac{1}{2} + b^H M^{1/2} G_{H,\sigma}(j)$$

where

$$H = \frac{3-\alpha}{2} \tag{2.4}$$

and

$$\sigma^2 = \frac{c}{4\varepsilon(U)2(\alpha-1)(2-\alpha)(3-\alpha)} \tag{2.5}$$

More precisely,

$$L \lim_{b\to\infty} L \lim_{M\to\infty} b^{-H} M^{-1/2}\left(W^*_{M,b}(j) - \frac{bM}{2}\right) = G_{H,\sigma}(j) \tag{2.6}$$

where "L lim" means convergence in the sense of the finite-dimensional distributions.

Theorem 2.1 states that the mean level $\frac{bM}{2}$ provides the main contribution for large M and b, and fluctuations from that level are given by the fractional Gaussian noise $G_{H,\sigma}(j)$ scaled by a lower order factor $b^H M^{1/2}$. Finally, it is important to note that a value of $1 < \alpha < 2$ implies $\frac{1}{2} < H < 1$.

Consider a discrete time stochastic process $X(t)$, $t \in Z$, where $X(t)$ is the traffic volume in bytes at time instance t. The definition of *strict* stationarity will be used in this research where $(X(t_1), X(t_2), \ldots, X(t_n))$ and $(X(t_1 + k), X(t_2 + k), \ldots, X(t_n + k))$ possess the same joint distribution for all $n \in Z_+$, $t_1, \ldots, t_n$, and $k \in Z$ [PaWi00, p. 17].

2.4 Impact on Network Performance

The importance of recognizing self-affinity in traffic is realized in the problem of optimal resource allocation in dynamic operating conditions. One of the jobs of network engineers is to provide consistent and reliable network service to as many users as possible, given certain budgetary and practical constraints. This means that the trade-off between network capacity and Quality of Service (QoS) requirements must be resolved by

optimizing resource allocation algorithms to guarantee that the service provided to the end user meets the QoS constraints while maintaining maximum capacity [SaTe99]. The optimal usage of resources is only possible by first characterizing and classifying the network traffic, and then determining optimal buffer sizes, assignment of bandwidth and channels, and other resources for each class of traffic in order to achieve the desired QoS in terms of queuing delay, retransmission time, packet loss probability, and bit error rate [SaTe99].

Network traffic has often been described by Markovian models which have limited memory of the past and reflect the short-range dependence (SRD) of the network traffic. The existence of scale-invariant "burstiness" (or self-affinity) introduces new complexities by directly implying long-range dependence (LRD) which is not accounted for in the Markovian and other traditional models [PaWi00, p. 21]. A process that exhibits SRD has an autocorrelation function that decays exponentially, whereas a process that exhibit LRD has an autocorrelation function that decays *hyperbolically* [KaWo99]. Neglecting the existence of self-affine traffic and LRD can result in overly optimistic predictions of network performance [ChBa97]. Self-affine network traffic can have a detrimental impact on network performance, including increased queuing delay and packet loss rate, because the buffers needed at switches and multiplexers must be much larger than those predicted by traditional queuing analyses and simulations [SaTe99]. In fact, an exponential trade-off is observed between queuing delay and packet loss rate [PaKC97]. However, these buffers cannot be made arbitrarily large because large buffer sizes create long queuing delays, and queues that are too long can deteriorate the QoS for real-time network applications such as video conferencing and multimedia traffic.

In response to the daunting task of maintaining QoS under severe self-affine traffic, research has pointed at ways of reducing the degree of self-affinity by shaping, or "smoothing", the traffic to make it less-bursty [ChBa97]. This is a difficult process because self-affine traffic is robust with respect to changes in network topology, interference from cross-traffic with dissimilar traffic characteristics, and changes in the distribution of file request interarrival times [PaKC96]. Self-affinity will also not be removed by any server with finite second-order moment of queue length [SoNT99].

Another important complexity in the problem of optimal resource allocation with self-affine network traffic is that the nature of the traffic changes constantly. Different types of traffic will flow across the network at different times during the day, and even different days during the week or month. Each type of traffic will have unique self-affine characteristics and will possess varying degrees of LRD. This continual change in traffic self-affinity means that the optimal buffer sizes and assignment of bandwidth and channels must be dynamic, and continually change to meet the network capacity and QoS requirements.

2.5 Self-Affine Data Sets

As stated in section 2.3, self-affinity exists in many types of network traffic, and possesses structural similarities at different time scales. This section presents and visually demonstrates the similarity between three classes of self-affine network traffic, and the same class of self-affine traffic at different time scales.

2.5.1 LAN Traffic

Leland *et al.* demonstrated the self-affine nature of Ethernet traffic through seminar publications in 1993 and 1994. The data used in this research is publicly available on the Internet [DMPS03a], and is presented in this section.

A recording of 1,000,000 packets of LAN traffic on an Ethernet was done by Leland and Wilson at Bellcore between 1125.00 hrs and 1217.23 hrs on 29 August 1989. The measurement techniques used in these recordings is found in [LeWi91], and a detailed discussion of the recordings are presented in [FoLe91] and [LTWW94]. Figure 2.1 shows the LAN traffic transmitted at 100 millisecond (msec) intervals for the entire 3142.82 seconds duration.

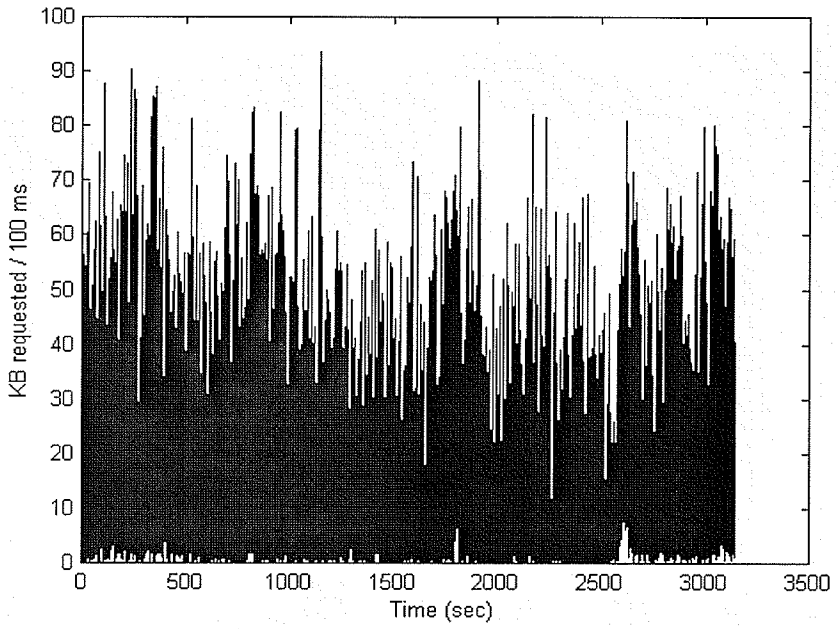

Fig. 2.1. Sampled LAN traffic (for approx. 3143 sec).

To visually demonstrate the self-affine nature of LAN traffic, Fig. 2.2 shows the same recording at 10 msec, 100 msec, and 1 sec time scales. Even though the traffic is averaged through time over three orders of magnitude, its bursty appearance does not vanish, but only varies, across different time scales.

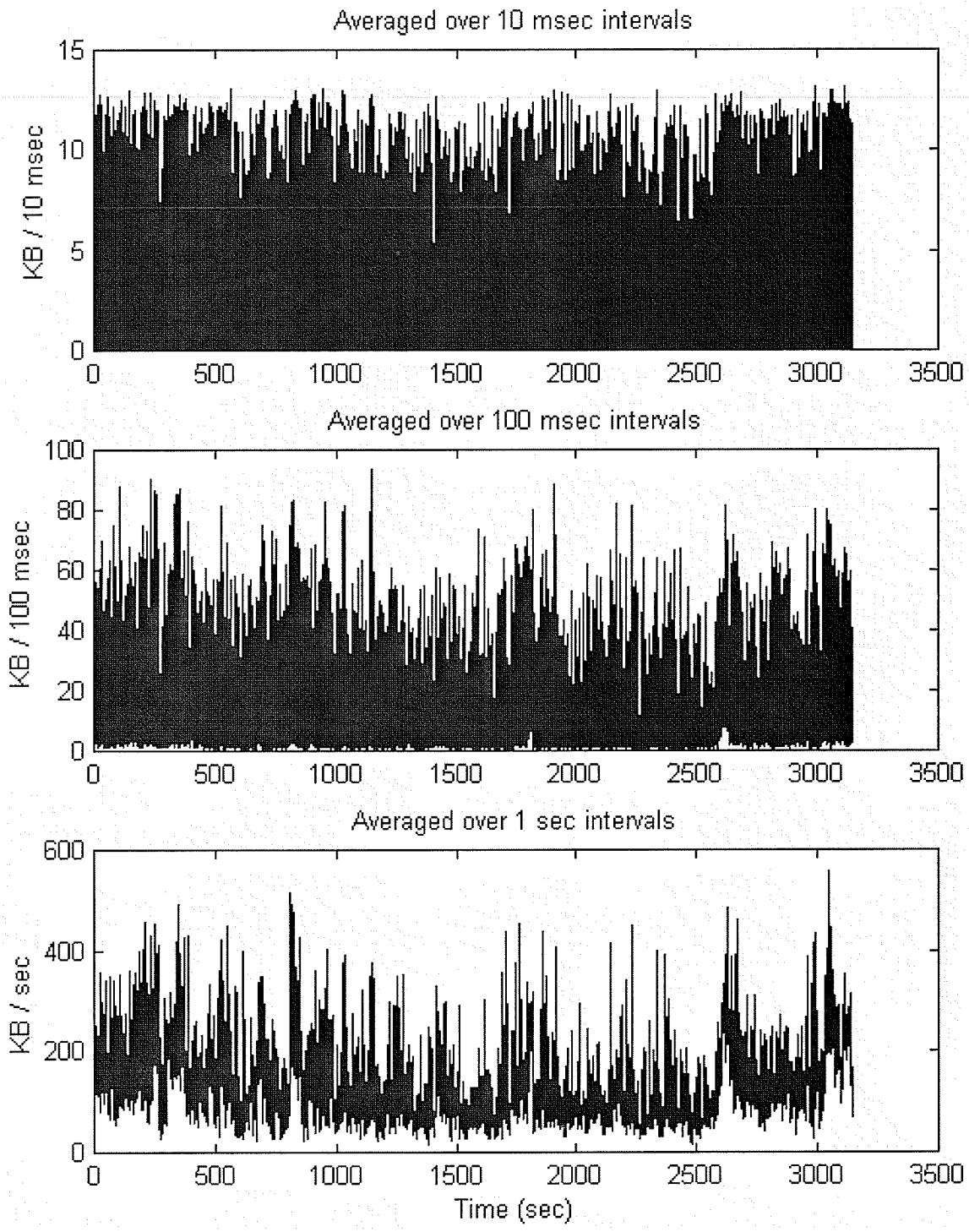

Fig. 2.2. LAN traffic averaged over (a) 10 msec, (b) 100 msec, and (c) 1 sec intervals.

2.5.2 WWW Traffic

In 1997, Crovella *et al.* demonstrated the self-affine nature of world wide web traffic [CrBe97]. The data used in this research is publicly available on the Internet [DMPS03b], and is presented in this section.

Between 01 February 1995 and 28 February 1995, a total of 100,669 unique Mosaic web requests were recorded on a network of 37 SparcStation 2 workstations by Cunha, Bestavros, and Crovella at Boston University [CuBC95]. Figure 2.3 shows the WWW traffic transmitted at 1 sec intervals for the entire 28 days, and Fig. 2.4 shows the same recording at 1 sec, 10 sec, and 100 sec time scales.

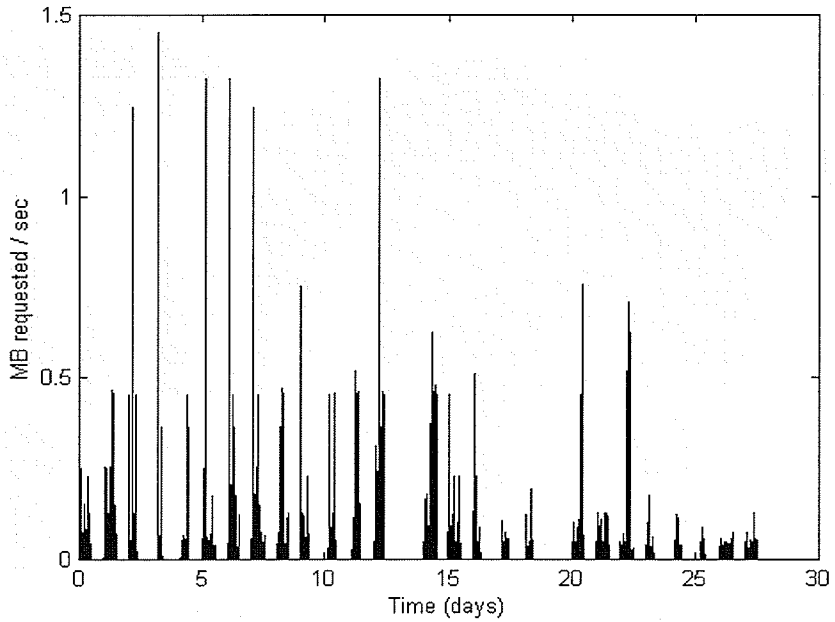

Fig. 2.3. Sampled WWW traffic (for approx. 28 days).

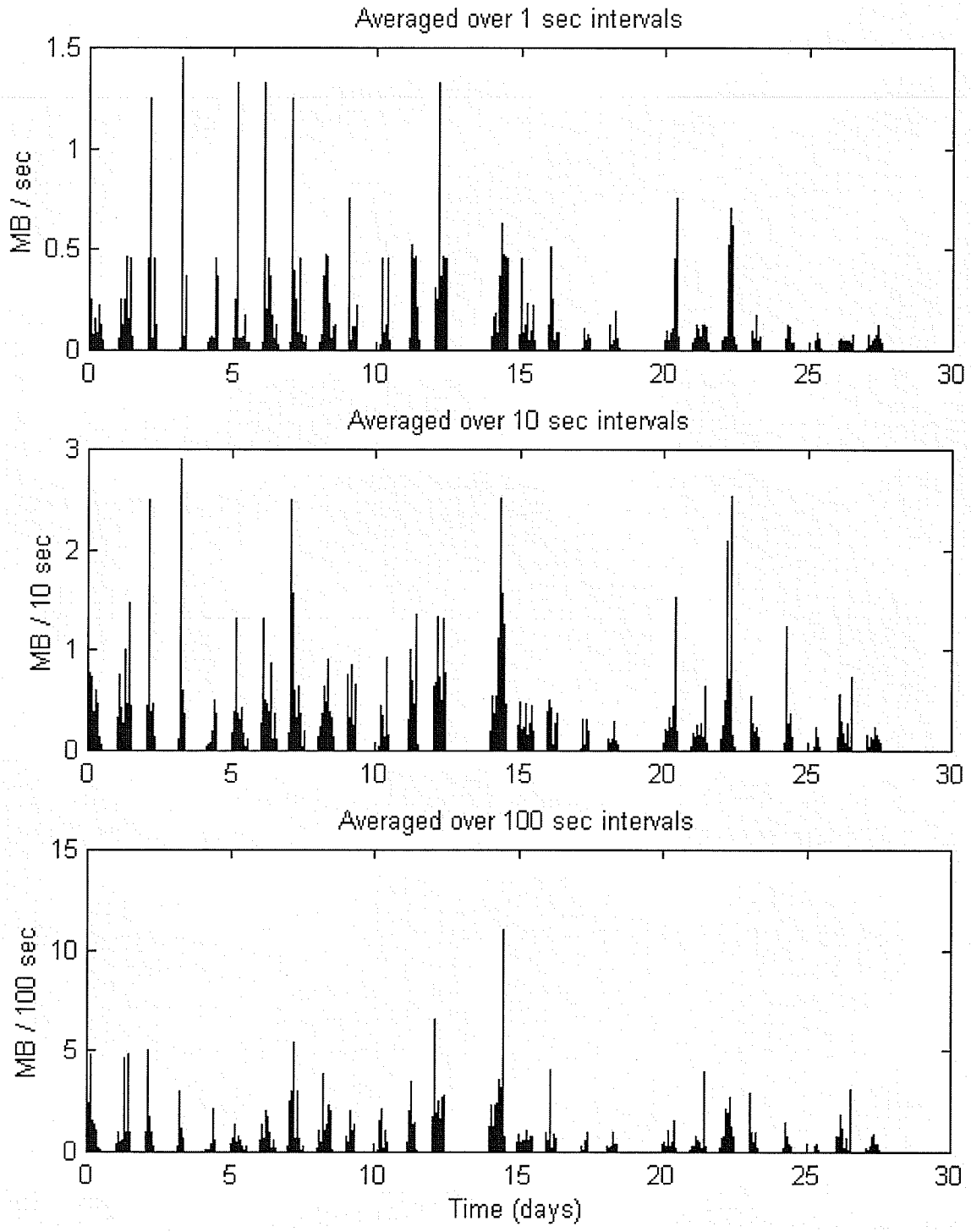

Fig. 2.4. WWW traffic averaged over (a) 1 sec, (b) 10 sec, and (c) 100 sec intervals.

2.5.3 VoIP Traffic

The previous examples of LAN and WWW traffic were captured by other researchers and acquired through publicly available archives. However, the importance of capturing one's own network traffic cannot be understated in the overall pedagogical process of studying network traffic. For this reason, Voice over IP (VoIP) traffic was captured at TR*Labs* (100-135 Innovation Drive, Winnipeg, MB, R3T 6A8, Canada) for 86400 seconds between 0800.00 hrs on 09 July 2003 and 0759.59 hrs on 10 July 2003. The software used to capture the traffic was Ethereal v.0.9.13 (publicly available at www.ethereal.com); a screen shot of this program is shown in Fig. 2.5. My sincere thanks goes out to Mr. Vasee Vaseeharan for his invaluable assistance in setting up the TR*Labs* network and a workstation to capture the VoIP traffic.

The resulting recording, or *trace*, consisted of 4,581,494 packets: 2,304,156 packets were User Datagram Protocol (UDP) packets used to transmit VoIP data, and were transmitted only when the phones were in use. The remaining packets were RX protocol packets used as network control signals, and were transmitted continuously regardless of whether or not the phones were in use. Therefore, to better understand the nature of voice data transmitted over a computer network, the study of VoIP traffic in this thesis will only consider the UDP packets.

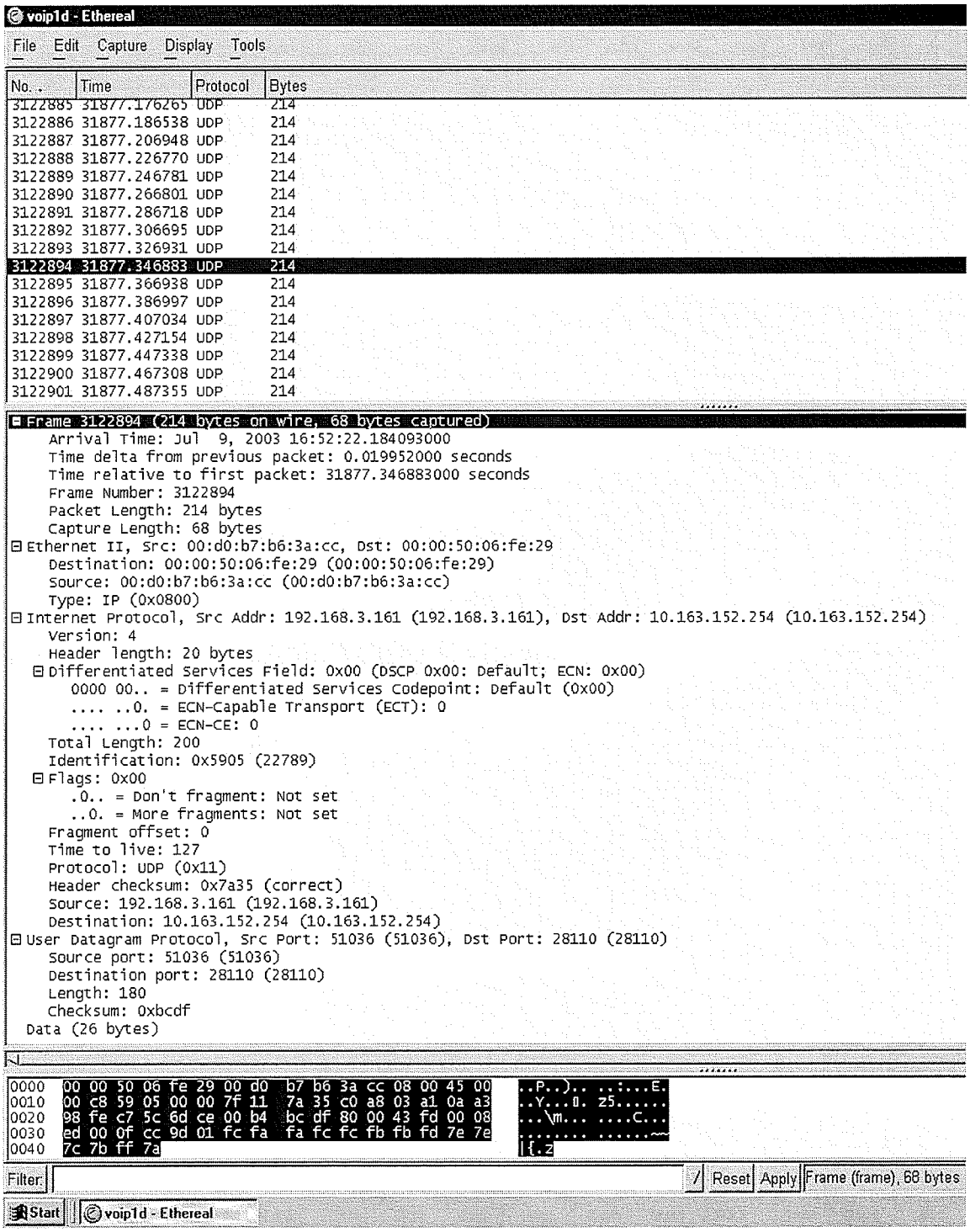

Fig. 2.5. Captured VoIP traffic using Ethereal.

Figure 2.6 shows the VoIP traffic for the entire 24 hours, expressed as kilobytes (KB) requested per 100 msec.

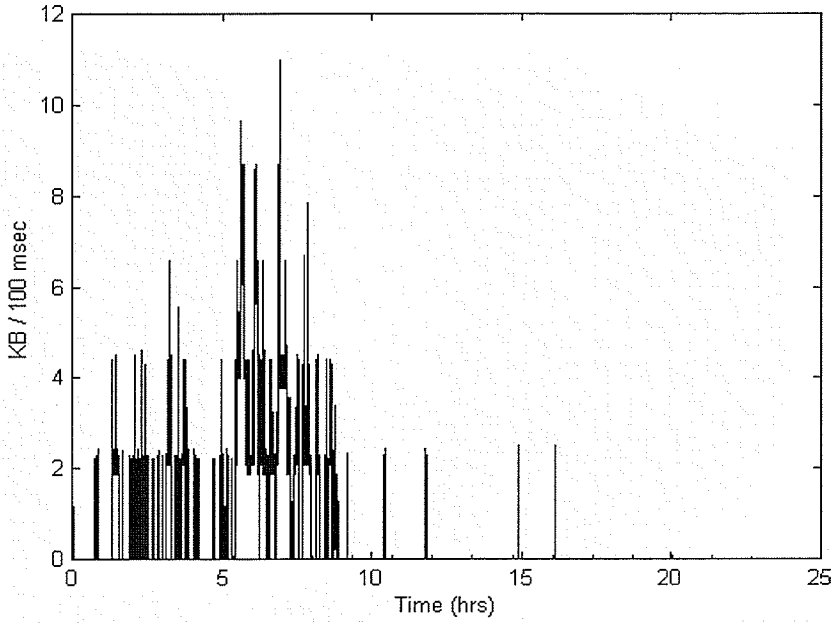

Fig. 2.6. Sampled VoIP traffic (for 24 hours).

At first glance, Fig. 2.6 was surprising because it looks quite different from Figs. 2.1 and 2.3, although some bursty similarities to WWW traffic do exist. However, when the nature of generation of VoIP traffic is considered, the graph makes sense. Firstly, there are long periods of no activity in the evening and early morning when the phones are generally not in use (although the traffic between 15 and 16 hours after the start of the trace, or around *midnight*, reflects the fact that some graduate students prefer quiet solitude in which to do their research). Secondly, the "step case" appearance of VoIP traffic reflects the fact that when a phone is in use, it transmits UDP packets of a fixed size at a fixed rate. Upon inspection of Fig. 2.7, which represents only the first two hours of

the traffic shown in Fig. 2.6, it seems that the transmitted data plateaus at intervals which are multiples of 1070 bytes / 100 msec. With UDP packets of size 214 bytes (as shown in Fig. 2.5), this rate corresponds to 5 UDP packets / 100 msec, or 50 UDP packets / sec.

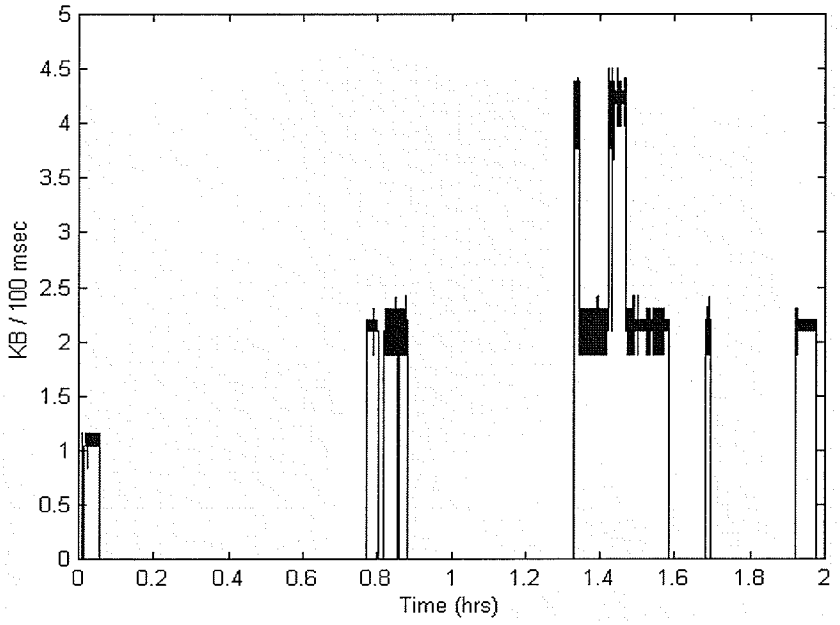

Fig. 2.7. Sampled VoIP traffic (first 2 hours only).

Finally, as in the examples of LAN and WWW traffic, VoIP traffic exhibits a very similar structure over many time scales. Fig. 2.8 shows the same VoIP trace averaged over four orders of magnitude of time, ranging from milliseconds to minutes. This self-affinity is *fractal* in nature, and will be discussed in detail in chapter 3.

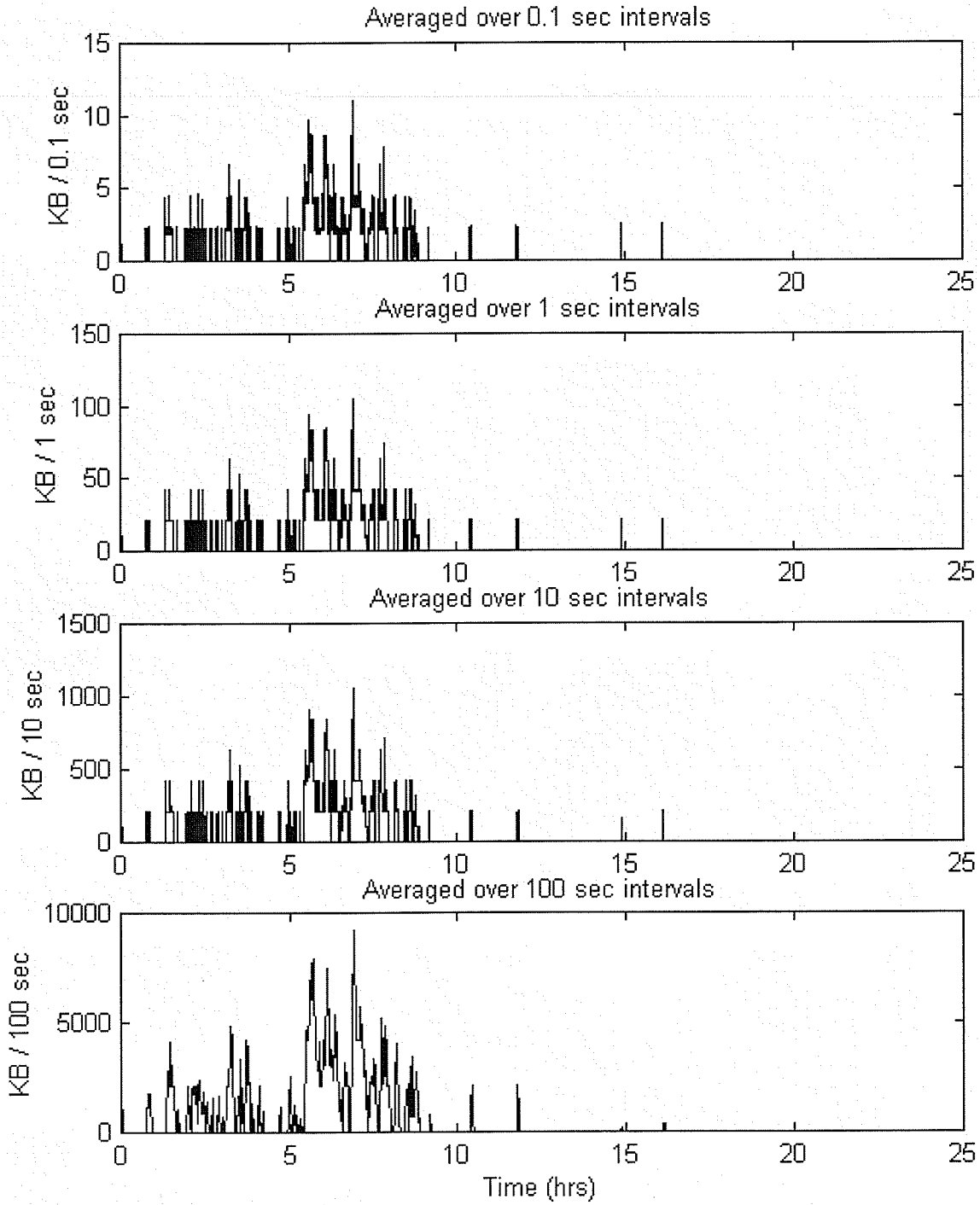

Fig. 2.8. VoIP traffic averaged over (a) 0.1 sec, (b) 1 sec, (c) 10 sec, and (d) 100 sec intervals.

2.6 Sarda's Data Sets

Deepak Sarda captured an extensive database of telecommunications traffic which he used in the design and refinement of his real-time voiceband signal classifier. The final voiceband signal classifier was able to accurately classify the following twelve signal classes [Sard99], [CoSa98]:

- V.22 and V.22*bis* forward channels
- V.22 and V.22*bis* reverse channels
- V.34 and V.90 uplink
- V.29 at all speeds
- V.32, V.32*bis*, and V.17 at speeds greater than 2400 bps
- V.27*ter* at 4800 bps
- V.27*ter* at 2400 bps
- Speech
- Random PCM samples and V.90 downlink
- FSK signalling
- Ringback
- DTMF tones for 0, 1, 2, ..., 9, *, and #

The original inspiration behind this thesis was to continue the work that had been done by Sarda, and to improve upon his research through the design of a third generation traffic classifier. After this new classifier was designed and built, it would have been tested using the same data sets as Sarda used, yielding an exact comparison between Sarda's classifier and the new classifier. It was anticipated that the classification accuracy of the new classifier would have been comparable to that of Sarda's classifier.

The primary advantage of the new classifier is that it would have implemented a modular design for each class of traffic. A modular design such as this would allow for each of these twelve classes of traffic to be represented by a unique multifractal signature

or "fingerprint". Future updates to the classifier's repertoire would have been relatively automated and efficient because the inclusion or removal of traffic classes would have been performed by adding or deleting the corresponding signatures of the traffic class of interest, and then re-training the traffic classifier (which will be discussed in chapter 4).

Unfortunately, while this research was already underway for more than a year, it was discovered that Sarda's data sets used were no longer available through Dr. Cockburn. Since this thesis had to be completed in a timely fashion, it was decided that another self-affine data set had to be acquired and used for the design and implementation of the new traffic classifier.

2.7 Pear's Data Sets

To expedite the completion of the thesis, Dr. Kinsner approached Dr. Joseph Pear in the Department of Psychology at the University of Manitoba. Dr. Pear and his colleagues have performed extensive research in the area of behaviour modification [MaPe02], and have collected a wide variety of data on the biological processes of habituation and dishabituation. Dr. Pear and Mr. Toby Martin, a Ph.D. student in Pear's research group, graciously provided our research group with these recordings of the habituation and dishabituation processes in Siamese Fighting Fish.

2.7.1 Siamese Fighting Fish (*Betta splendens*)

Siamese Fighting Fish, or *Betta splendens*, are colourful fish that originated from the warm, shallow waters of Cambodia and Thailand (formerly known as *Siam*, which gave the fish its name) [Abou03]. The behaviour of *Betta splendens* has been observed and studied by psychologists for decades. Domesticated male *Betta splendens* exhibit

agonistic (aggressive and territorial) behaviour that is easily elicited and very similar from one fish to another [Simp68]. *Betta splendens* must be kept isolated from its own kind because of this behaviour, as a male will attack and most likely kill another male in defence of its territory. Some of the treat displays exhibited by *Betta splendens* include pectoral fin beating, pelvic fin flickering, tail beating, tail flashing, bites, and nips [Simp68]. Psychologists can observe and record the frequency and duration of these threat displays as a method of characterizing and classifying the behaviour of *Betta splendens*. A picture of *Spot*, the author's *Betta splendens*, is shown in Fig. 2.9.

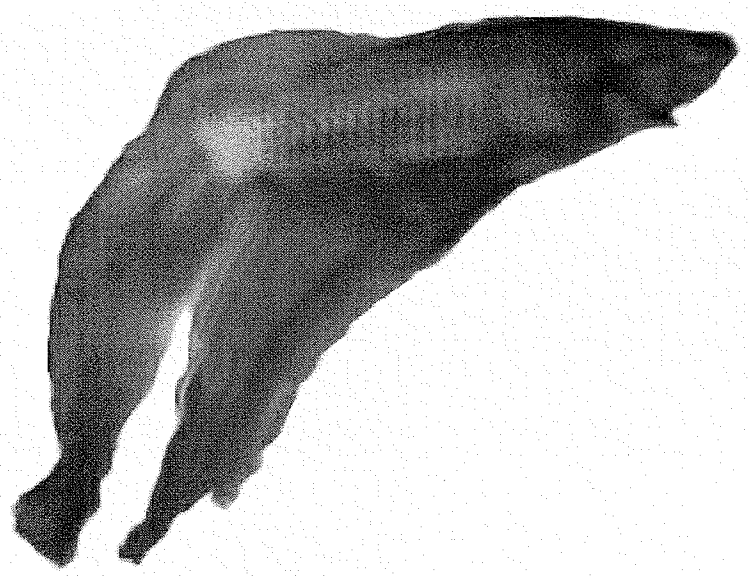

Fig. 2.9. *Spot* – a typical *Betta splendens*.

The agonistic behaviour of male *Betta splendens* may be elicited by the presence of another male (also called a *conspecific*), an artificial model of a conspecific, or its own reflection using a mirror [ShSh71]. This agonistic behaviour has been shown to habituate,

or decrease, over time as the stimulus is continuously presented [Bron94], [Pear01, pp. 15-17]. It has also been shown that the presence of a new eliciting stimulus may produce a response recovery [ThSp66] called *dishabituation* [Pear01, p. 17], [Mart02].

2.7.2 Video Recording System

Pear's research group recorded the activity of male *Betta splendens* in an aquarium using a stereoscopic video camera system [Mart02]. A photo of the camera system used is shown in Fig. 2.10. The complete system consists of three cameras: two cameras provide a stereoscopic view of the fish tank, and one camera is a regular video camera which video tapes the motion of the fish for later viewing and study by psychologists.

Figure 2.11 shows a diagram of the stereoscopic camera system which, not unlike the human visual system, is capable of perceiving, or *extrapolating*, the depth of an object based upon its position as seen by each camera. This camera system records the X-, Y-, and Z-coordinates of the *Betta splendens* ten times per second (a frequency of 10 Hz), with a maximum positional error of ±5 mm [PeMa02].

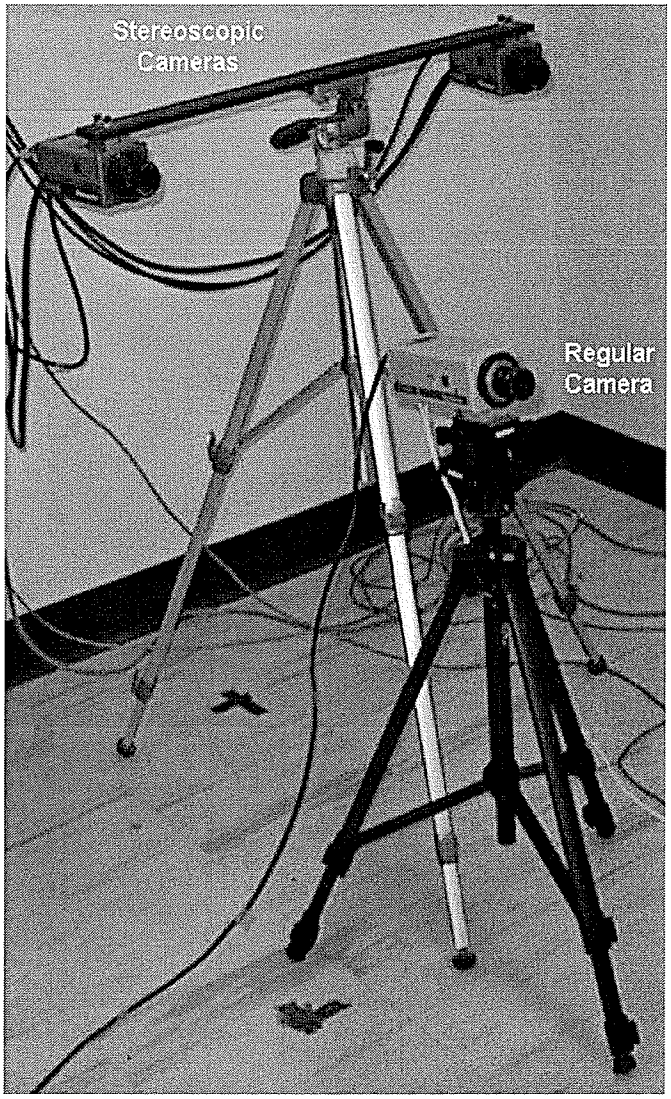

Fig. 2.10. Stereoscopic video camera system.

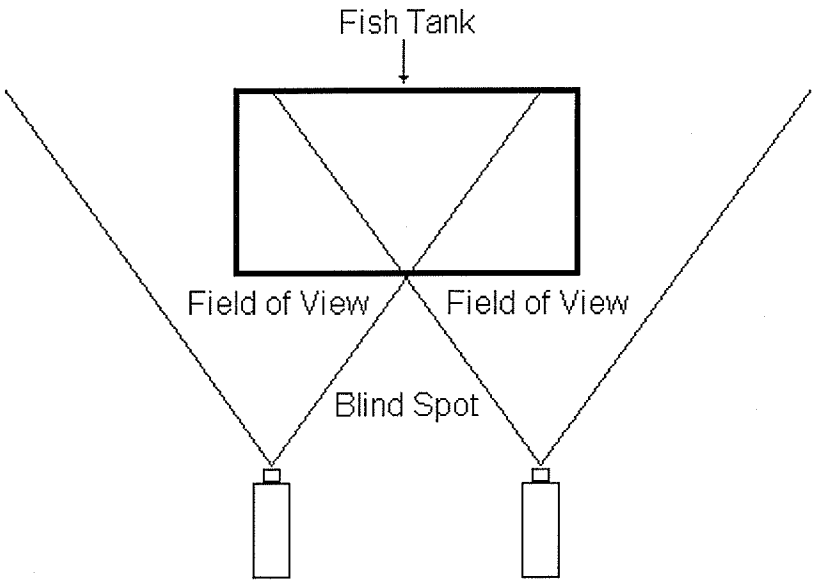

Fig. 2.11. Stereoscopic vision.

2.7.3 Pre-Processing

The raw recorded data of the motion of the *Betta splendens* need to be pre-processed in two ways. Firstly, due to the set-up of the stereoscopic cameras, the origin (0,0,0) is located somewhere in space between the two cameras. In this coordinate system, the centre of the fish tank is approximately at the coordinates (40,2000,45) [PeMa02], which can be confusing to interpret when discussing the position of the *Betta splendens* in the tank. Therefore, the coordinate system needs to be changed so that the origin (0,0,0) is in the left, front, bottom corner of the tank, as shown in Fig. 2.12. In this intuitive and equivalent coordinate system, the X-coordinate represents the distance from the fish to the mirror, the Y-coordinate represents the position of the fish along the mirror, and the Z-coordinate represents the height of the fish from the bottom of the aquarium.

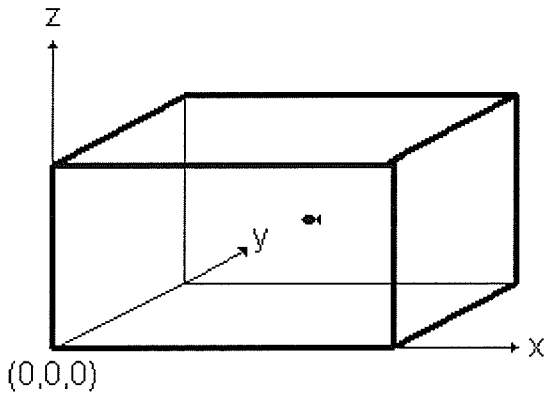

Fig. 2.12. A better coordinate system.

Secondly, there are occasional tracking errors when the stereoscopic cameras are unable to locate the position of the *Betta splendens*, which often happens when the fish is close to the mirror [PeMa02]. These tracking errors last for at most a few seconds, so consecutive tracking errors are replaced by data points calculated by linear interpolation between adjacent existing data points.

2.7.4 Dishabituation Experiments

The mature male *Betta splendens* used in these experiments had been used for previous experiments, but had not been subjected to experimental conditions for several months [Mart02]. The researchers attempted to demonstrate dishabituation by introducing two different stimuli in controlled experiments. All experiments lasted for 7 hours and 56 minutes, and started with the *Betta splendens* being placed in the aquarium. The one-way mirror at the one end had a incandescent light bulb placed behind it. When the light was off, the mirror was reflective; when the light was on, the mirror was non-reflective.

In dishabituation experiment #1, the light was turned off so the *Betta splendens* could see and habituate to its reflection. Seven hours into the experiment, the light was turned on for 60 seconds, making the mirror non-reflective. The light was then turned on again for the remaining 55 minutes of the experiment [Mart02]. This experimental set-up is shown in Fig. 2.13.

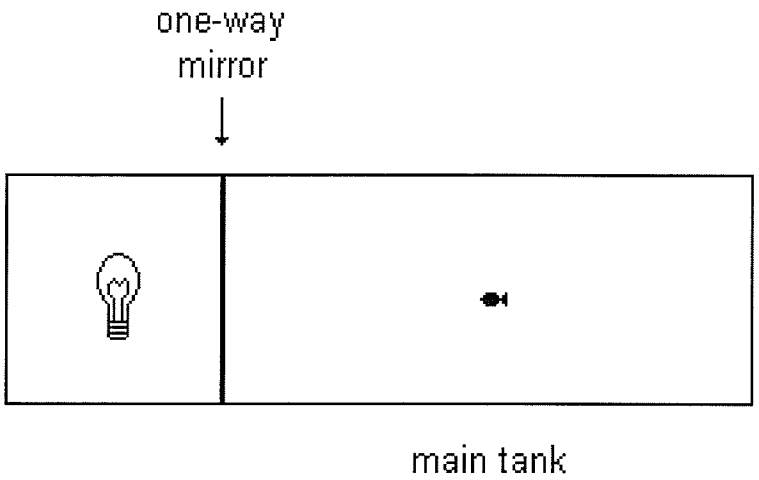

Fig. 2.13. Experiment #1 to attempt to produce dishabituation (after [Mart02]).

In dishabituation experiment #2, the light was once again turned off so the *Betta splendens* could see and habituate to its reflection. Seven hours into the experiment, the mirror was made non-reflective by turning on the light, while simultaneously removing a white plastic barrier on the opposite end of the aquarium to reveal the presence of a live conspecific. After 60 seconds, the plastic barrier was replaced and the light turned off, and it remained this way for the remaining 55 minutes of the experiment [Mart02]. This experimental set-up is shown in Fig. 2.14.

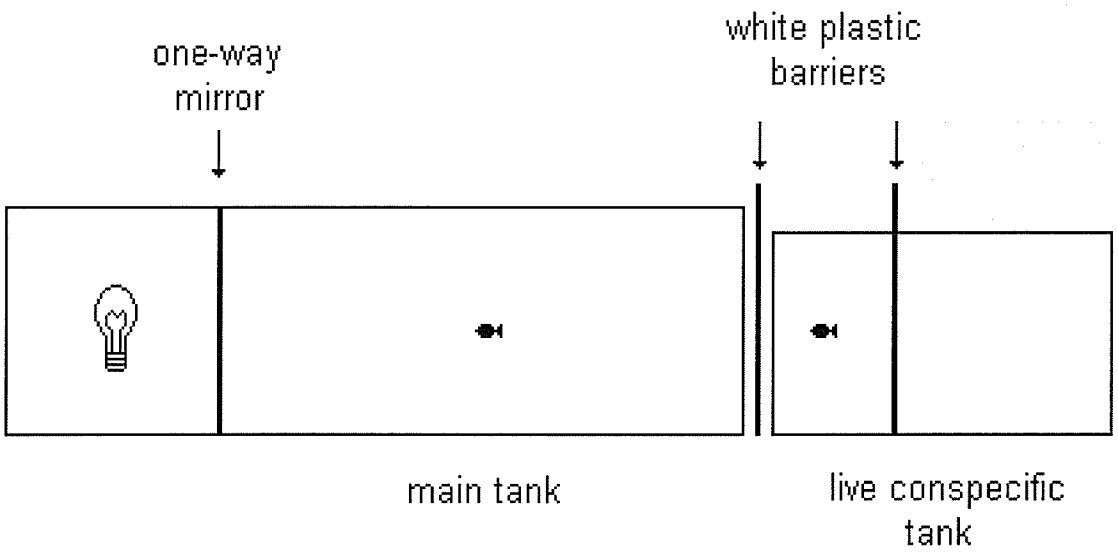

Fig. 2.14. Experiment #2 to attempt to produce dishabituation (after [Mart02]).

These experiments were repeated several times with two different *Betta splendens*. In general, the *Betta splendens* habituated to their reflection within the first couple hours of the experiment. The introduction of new stimuli 7 hours into each experiment were attempts to disrupt this habituation and produce dishabituation. Although several trails were performed for each experiment, one of the most interesting recordings was of experiment #1 for fish #11 that was performed on 19 February 2002. This session is referred to as Experiment 11020219.

2.7.5 Experiment 11020219

Figures 2.15, 2.16, and 2.17 show the X-, Y-, and Z-coordinates of Experiment 11020219 for Pear's data sets.

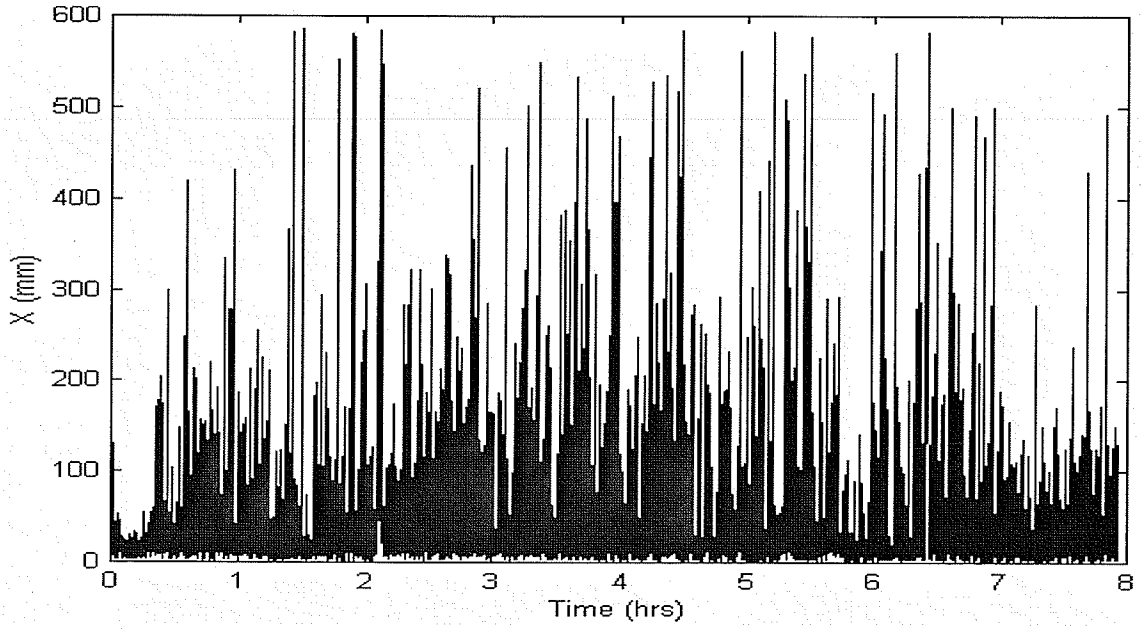

Fig. 2.15. X-coordinates of Experiment 11020219.

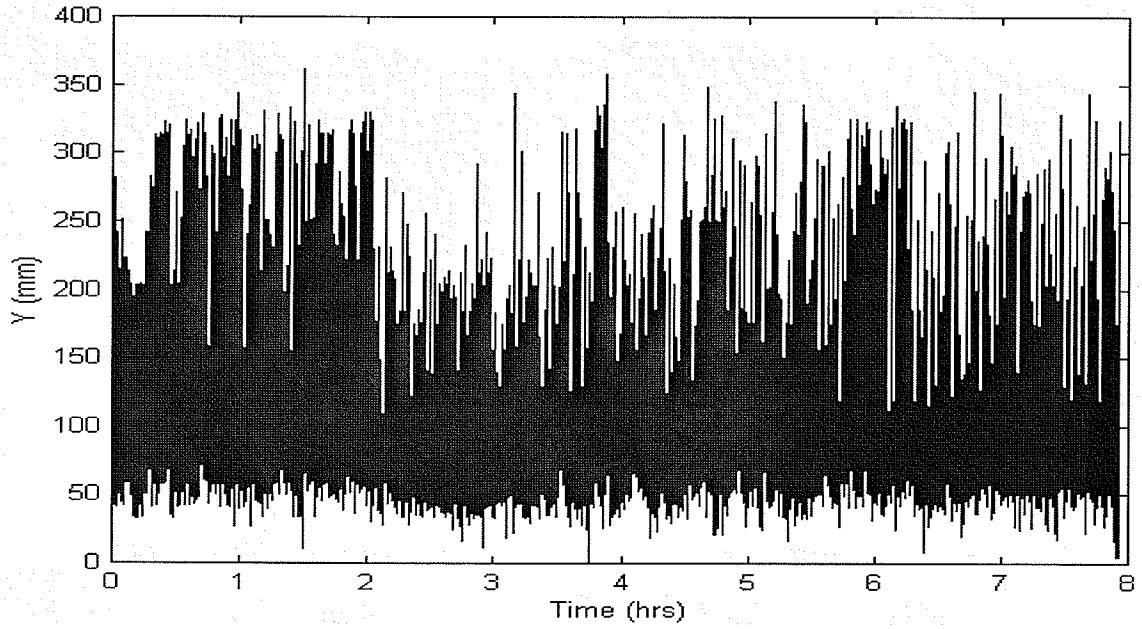

Fig. 2.16. Y-coordinates of Experiment 11020219.

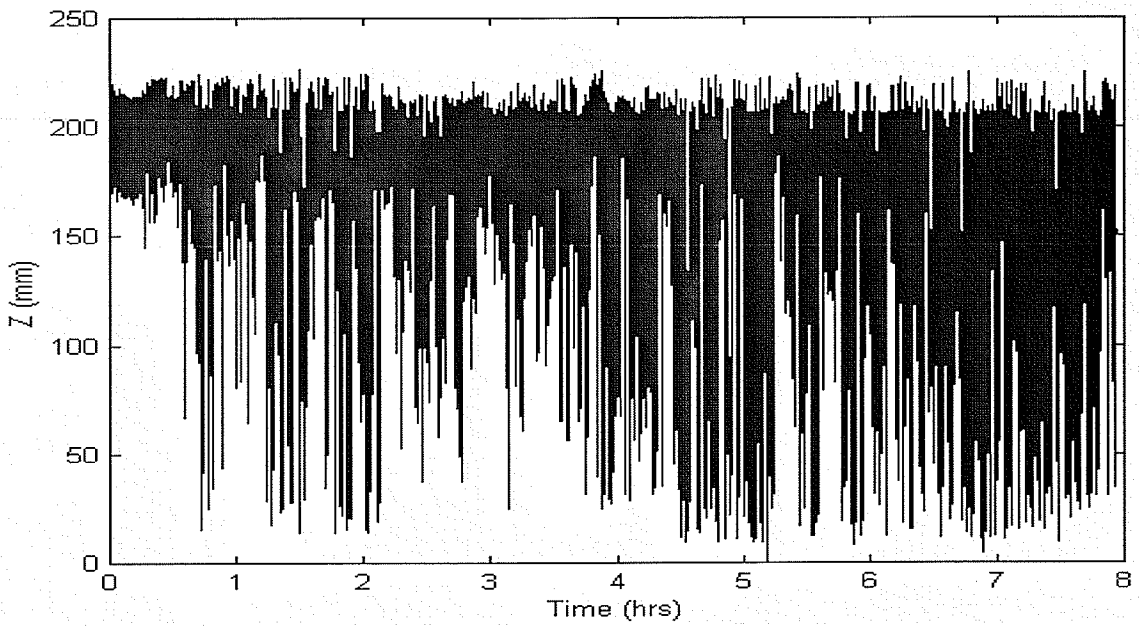

Fig. 2.17. Z-coordinates of Experiment 11020219.

These recordings are particularly interesting because the *Betta splendens* quickly habituates to the mirror after 15 minutes, but then exhibits behaviour that is characteristic of dishabituation when the new stimulus is introduced into the system after 7 hours. The dishabituation is most clearly seen by the X-coordinates in Fig. 2.15 when the *Betta splendens* spends the final 56 minutes much closer to the mirror than the previous 6 hours and 25 minutes of the experiment.

Therefore, since the X-coordinates of Experiment 11020219 seemed a good choice for an interesting and reliable data recording with which to substitute for Sarda's data sets, they henceforth became the primary data set used in this thesis, and will be referred to as Record 11020219.

2.7.6 Self-Affinity in Record 11020219

Previous examples of LAN, WWW, and VoIP traffic showed similar structures across multiple time scales. However, since this discussion has now shifted from network traffic to *biological behavioural modification* traffic, we need to re-address the notion of self-affinity in Pear's data sets and Record 11020219.

Self-affinity is a necessary requirement for the correct application of fractal analysis: only if the traffic is self-affine may fractal and multifractal techniques be applied to the traffic. To begin the analysis of the self-affine nature of Record 11020219, Fig. 2.18 displays this record at three different time scales. As anticipated, this figure reveals the structural similarities of Record 11020219 across multiple time scales. This analysis will continue in section 5.2 where the self-affine nature of Record 11020219 will be rigorously proven.

(While discussing the fractality of the motion of *Betta splendens*, it is both interesting and motivating to note that in 2001, Tikhonov *et al.* from Russia published a study on chaos and fractals in fish school motion [TEMM01].)

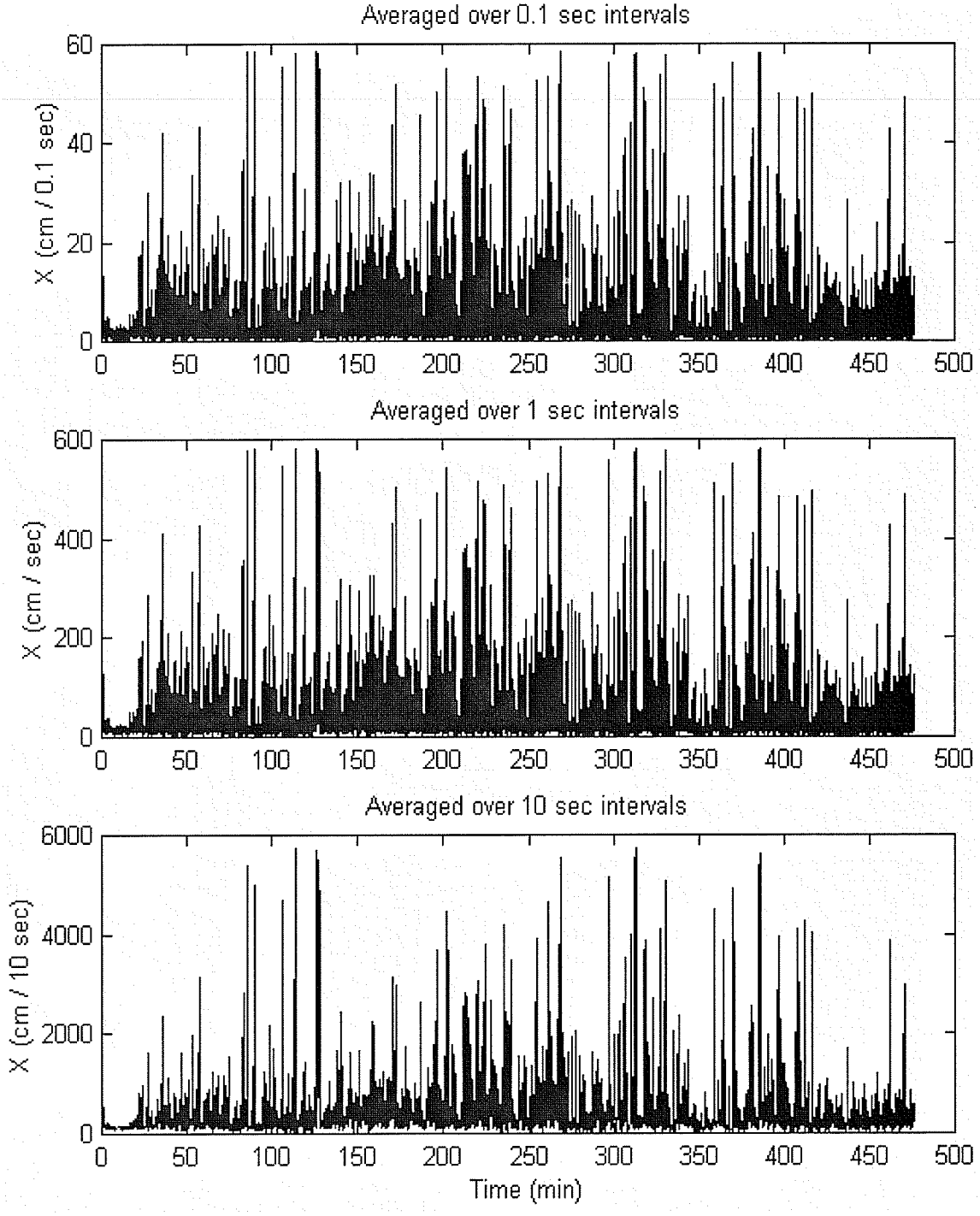

Fig. 2.18. *Betta splendens* traffic averaged over (a) 0.1 sec, (b) 1 sec, and (c) 10 sec intervals.

2.8 Summary

This chapter explores the nature of traffic, computer network traffic, and the significance of the existence and recognition of self-affine network traffic for its control in order to maintain acceptable QoS levels for the end-user. Examples of LAN, WWW, and VoIP traffic are presented as a demonstration of the visual characteristics of self-affine traffic. Some of the previous work that has been done by TR*Labs* in the area of telecommunications traffic characterization and classification has also been presented, and an improvement upon this work is the motivation behind this thesis.

The methodology used in this thesis is for the characterization and classification of *self-affine* traffic with applications specifically towards self-affine network traffic. Since Sarda's data sets became unavailable while the thesis was already well underway, Pear's data sets were acquired and studied so that this thesis might be completed in a timely manner. Although the primary data set studied, Record 11020219, is a recording of a different type of self-affine traffic, all of the analyses presented in the characterization and classification of Record 11020219 may be directly applied to other classes of self-affine traffic that are of interest to TR*Labs* and its sponsors.

CHAPTER III

BACKGROUND ON FRACTALS, MULTIFRACTALS, AND FRACTAL DIMENSIONS

Clouds are not spheres,
mountains are not cones,
coastlines are not circles,
and bark is not smooth,
nor does lightning travel in a straight line.
- B. B. Mandelbrot [Mand83, p.1]

3.1 What are Fractals?

The word "fractal" was coined by Benoit B. Mandelbrot from the Latin adjective *fractus*. The corresponding Latin verb is *frangere*, meaning "to break" [Mand83, p.4]. From this definition alone, one may intuitively feel that the study of fractals involves the study of objects which are broken or fragmented in some way. The word "fractal" is also commonly associated with images of intricate coloured patterns with infinite complexity and beauty. The beauty of fractals has been made popular in recent years through the printing of posters and calendars that vividly show the infinite range of patterns and colours that fractals can possess. A widely recognizable example of a fractal is the Mandelbrot set. Figures 3.1 and 3.2 show grayscale examples of the Mandelbrot set rendered in 2-dimensions and 3-dimensions (where the darkness of the fractal in 2D is mapped into the height of the object in 3D) using one of many readily available software packages for the generation of fractals – Fractal Explorer 2.00 with Fractal Landscape Library 1.05 [SiFe03]. These fractals possess both visual and mathematical beauty, and have infinite detail at any level of resolution.

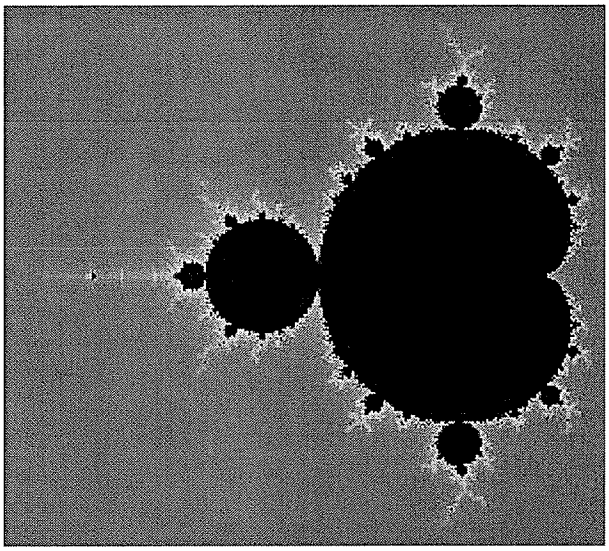

Fig. 3.1. Mandelbrot set in 2-dimensions.

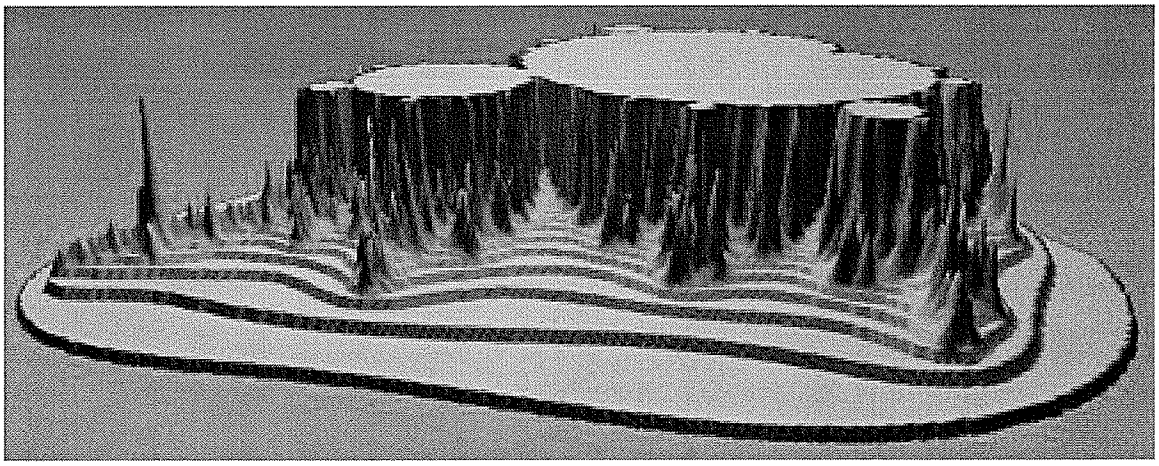

Fig. 3.2. Mandelbrot set in 3-dimensions.

There is no single, absolute definition for a fractal. In fact, Mandelbrot feels that "one would do better without a definition" [Mand83, p. 361]. In 1982, Mandelbrot proposed the definition that "a fractal is a set for which the Hausdorff-Besicovitch (fractal) dimension strictly exceeds the topological dimension" [Mand83, p. 15]. However, it was

later discovered that some fractals (ironically, including the Mandelbrot set) were excluded by the use of the word "strictly" in this definition. A revised definition proposed by Robert L. Devaney in 1992 defines a fractal as "a subset of $\Re^n$ which is self-similar and whose fractal dimension exceeds its topological dimension" [Deva92, p. 178].

In more simpler terms, fractals (and multifractals) are real constructs which have the following properties:

- they are self-affine, meaning that part of their structure is related to the whole structure through the property of scaling;
- their structure cannot be described using Euclidean geometry, and is often described as being "rough"; and
- their fractal dimensions can (and usually do) exist between integer dimensions.

The study of fractals involves the study of fractal geometry. Mandelbrot argues that the best possible definition of fractal geometry may be that it is *the study of scale-invariant roughness* [Mand02, p. 9].

The relationship between fractals and self-affinity is deep and profound, and in its most detailed form, the exploration of this relationship may require hundreds of pages. (Mandelbrot provides a 600-page discussion of the relationships between fractals and self-affinity in [Mand02].) For the sake of simplicity, an object that possesses scale-invariant roughness is said to be self-affine, and is also fractal or multifractal in nature. In other words, self-affinity implies fractality (and vice versa). Therefore, to study fractal geometry, one must also study affine geometry.

An affine plane is parameterized by two coordinates x and y [Mand02, p. 86]. An affine transformation is a one-to-one linear map of the form

$$x \to r_{xx}x + r_{xy}y + x_0 \tag{3.1}$$

$$y \to r_{xy}x + r_{yy}y + y_0 \tag{3.2}$$

The matrix of the transformation is

$$\begin{bmatrix} r_{xx} & r_{xy} \\ r_{yx} & r_{yy} \end{bmatrix}$$

For a linear map to be one-to-one, the necessary and sufficient condition is that the matrix is invertible. In other words,

$$r_{xx}r_{xy} - r_{xy}r_{yx} \neq 0 \tag{3.3}$$

An interesting quote provided by Snapper and Troyer provides an insight into the study of affine-geometry: "Roughly speaking, affine geometry is what remains after practically all ability to measure length, area, angles, etc., has been removed from Euclidean geometry. One might think that affine geometry is a poverty-stricken subject. On the contrary, affine geometry is quite rich. Even after almost all ability to measure has been removed from Euclidean geometry, there still remains the concept of parallelism. Consequently, the whole theory of homothetic figures lies within affine geometry. The notions of translation and magnification (these are the dilations) are in the domain of affine geometry and, more generally, as the name suggests, affine transformations can be characterized as one-to-one, onto functions which preserve parallelism; therefore, they constitute an affine notion" [SnTr71, p. 1].

Self-similarity is a very special case of self-affinity where the matrix is of the form

$\alpha = r\|I\|$ where r is complex and $\|I\|$ is the unit matrix: the diagonal elements are one (1) and all other elements are zero (0) [Mand02, p. 87]. Therefore, although much of the published literature in the area of fractals and network traffic popularly discuss the issue of "self-similarity", it is more accurate to use the term "self-affinity" [Mand85]. For this reason, this thesis will use the terms "self-affine" and "self-affinity" in this context.

3.2 Generation of Fractals

There are two general classes of fractals: mathematical fractals and fractals that are found in nature (which will be described as "natural" fractals) [Vics92, pp. 11-13]. Mathematical fractals can be easily described, generated, and measured. Fractals in nature describe the world in which we live. Not surprisingly, the "real world" is more complex than a simple mathematical expression, so the fractals found in nature are usually more difficult to describe and measure.

3.2.1 Mathematical Fractals

"A picture is worth a thousand words" is a good phrase to keep in mind when exploring the world of fractals. The generation of mathematical fractals involves an iterative process that begins with an initiator and a generator [PeJS92, p. 15]. The creation of two commonly discussed fractals, the Koch curve [Koch04] and the Sierpinski carpet [Sier16], will serve as examples of the generation of mathematical fractals. (From an historical viewpoint, it is interesting to note that the Sierpinski *gasket* [Sier15], another fractal that is similar to the Sierpinski carpet, was created by Waclaw Sierpinski in 1915; however, the Sierpinski *carpet* was actually created by Sierpinski's former Ph.D. student Stefan Mazurkiewicz, also in 1915 [CiPo96].)

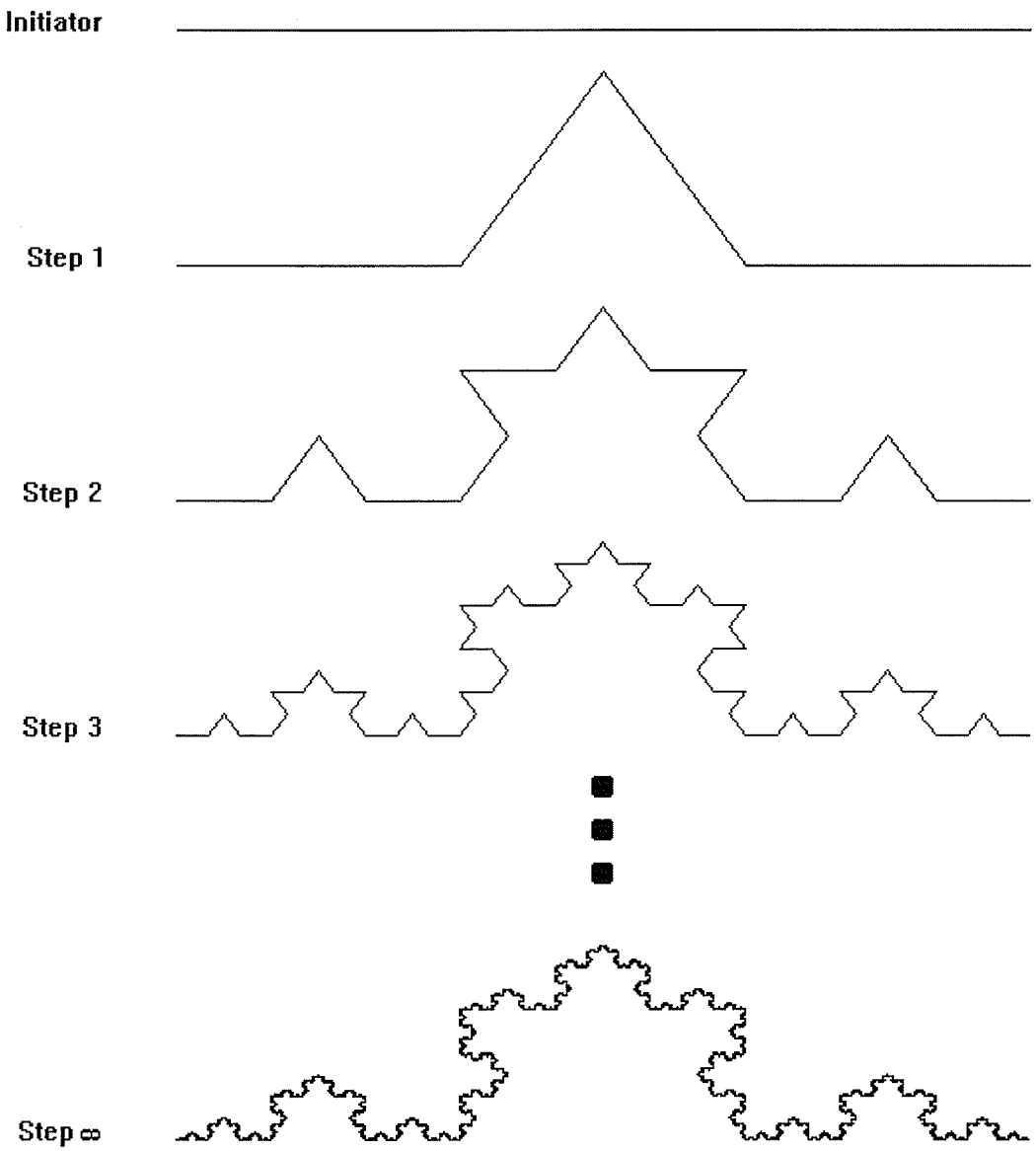

Fig. 3.3. Generation of the Koch curve fractal.

Figure 3.3 illustrates the iterative generation of the Koch curve fractal. The initiator is a single straight line segment. In step 1, the straight line is divided into three

equal line segments, and the middle third is removed and replaced by two line segments of the same length so they meet at a 60 degree angle. In step 2, each line segment is divided into three more equal line segments, with the middle segment removed and replaced by two more line segments in the same fashion as step 1. This process continues over and over again, and is repeated an infinite number of times.

There are several important things to note about the Koch curve:

- Since the iterative process continues *ad infinitum*, any segment of the curve looks like a smaller version of the entire Koch curve. This accounts for the self-similar nature of the fractal.
- The curve is non-intersection, meaning that as the generation process continues, two line segments will never occupy the same region of space.
- The length of the curve is infinite.
- The area of the curve is zero.

Figure 3.4 demonstrates the generation of the Sierpinski carpet fractal. The initiator is a solid square (instead of a straight line segment). In step 1, the square is divided into a 3-by-3 grid of nine equal sized squares, and the middle square is removed. In step 2, each of the remaining eight squares is once again divided into a 3-by-3 grid of nine equal sized squares, and the middle square is removed. As in the Koch curve, this process is repeated an infinite number of times, and any segment of the carpet looks like a smaller version of the whole carpet. The Sierpinski carpet also has an area of zero.

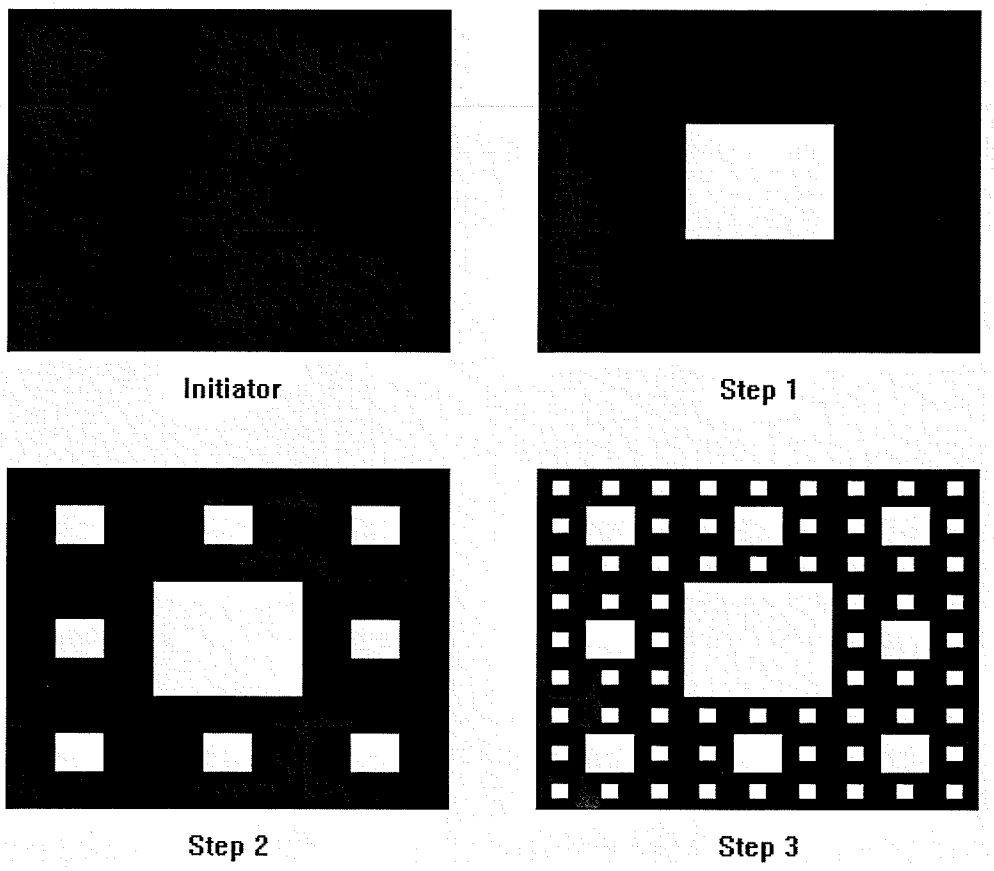

Fig. 3.4. Generation of the Sierpinski carpet fractal.

Figure 3.5 shows an image of the Menger sponge [Meng26] – the 3-dimensional version of the Sierpinski carpet. This fractal is especially interesting because it is an infinitely complex structure that has infinite surface area but zero volume [Glei87, p. 101].

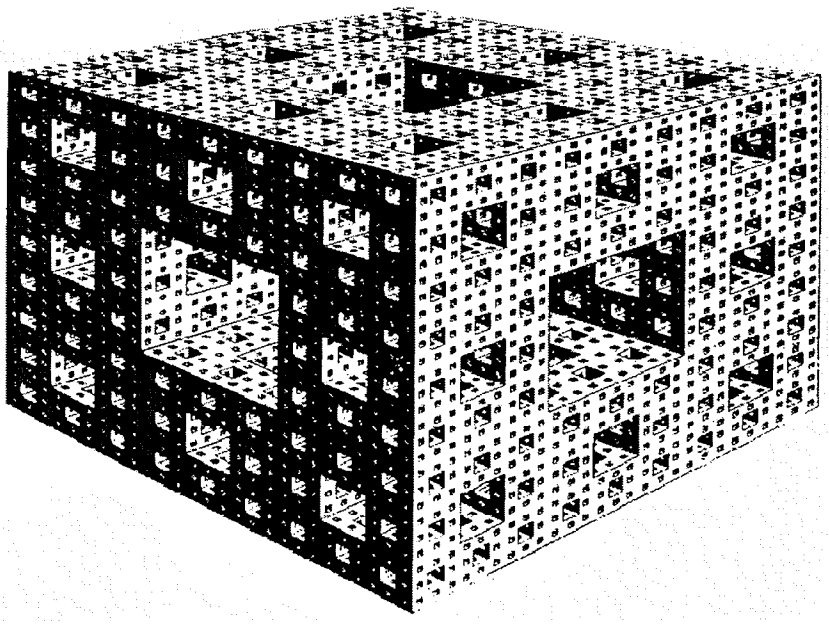

Fig. 3.5. The Menger sponge (after [BlMe70, p. 502]).

3.2.2 Natural Fractals

Mathematical fractals contain an infinite number of points, but can be described by a simple initiator and an iterative process. Fractals exist naturally in nature, but are much more complicated in their design and description. Fig. 3.6 show a picture of the coastline of Great Britain, and is an example of Mother Nature's fractals.

Fig. 3.6. Coastline of Great Britain.

In 1967, Mandelbrot posed the seemingly simple question: "How long is the coast of Britain?" [Mand67]. At first glance, many people may think that a coastline has a fixed length, and could only be changed through natural disasters such as earthquakes, or after countless centuries of erosion and tectonic movement. Table 3.1 shows the measured length of Great Britain's coastline using smaller and smaller divisions of measurement.

Table 3.1: Length of Great Britain's coastline (after [PeJS92, p. 192]).

Compass Length	Length of Coastline
500 km	2600 km
100 km	3800 km
54 km	5770 km
17 km	8640 km

This table clearly shows the discovery that the length of Great Britain's coastline is not fixed, but is very much dependent on the compass length that is used for the measurements. As the compass length decreased from 500 km to 17 km, the length of Great Britain's coastline increased over three times from 2600 km to 8640 km! Therefore, there is no exact length for this coastline of Great Britain, except that its length approaches infinity as the compass length approaches zero.

3.3 Fractal Dimensions

How can one measure a fractal that has infinite length, infinite surface area, or zero volume? These constructs exist, but cannot be described using Euclidean geometry. For this reason, fractal dimensions are used to describe and measure a fractal, and compare one fractal with another.

The measurement of a fractal dimension D begins with the assumption that similarities in the fractal exist at different scales. In other words, whereas some processes in engineering are time-invariant, fractal processes are *scale*-invariant and have a power law relationship. There are an infinite number of fractal dimensions that may be used to

describe a fractal [HePr83]. If a fractal is very simple (such as the Koch curve or Sierpinski carpet), then one dimension is often enough to adequately describe the fractal. When the nature of the fractal begins to change with space or time (and becomes multifractal), then more than one fractal dimension is usually required to adequately represent and characterize the fractal.

Fractal dimensions may be classified into the following four categories [Kins94a]:

- morphological-based dimensions
- entropy-based dimensions
- spectrum-based dimensions
- variance-based dimensions

Morphological-based dimensions consider the physical structure of a fractal, and may be used to determine the dimension of a fractal where the nature of the distribution is unknown [Mand83, p. 364]. An example of a morphological-based dimension is the Hausdorff-Besicovitch dimension, D_{HB} [Mand83, p. 364], [Mand85], [Kins94b].

3.3.1 Hausdorff-Besicovitch Dimension

A hypersphere, or *volume element* (vel) [Kins94a], is defined as an arbitrary region of N-dimensional space that covers part of a fractal. As the number of vels N_r covering the fractal [Edga90, Ch. 3] approaches infinity and the size of each vel, r, approaches zero, then the Hausdorff-Besicovitch dimension is defined as

$$D_{HB} = \lim_{r \to 0} \frac{\log(N_r)}{\log(1/r)} \tag{3.4}$$

Table 3.2 shows a list of the fractals that have been discussed so far in this chapter, and their approximate fractal dimensions.

Table 3.2: Fractal dimensions for various figures.

Fractal	Dimension	Reference
Koch curve	1.2618	[Mand83, p. 44]
Sierpinski carpet	1.8928	[Mand83, p. 144]
Menger sponge	2.7268	[Mand83, p. 144]
Great Britain's coastline	approx. 1.3	[PeJS92, p. 215]

Fractional dimensions such as 1.2618, 1.8928, or 2.7268 need some explanation to find meaning in a world that is typically described through Euclidean geometry. In Euclidean geometry, dimensions have integer values: $D_E = 0$ is a point, $D_E = 1$ is a line, $D_E = 2$ is a plane, and $D_E = 3$ is a space. A fractal dimension between $D = 0$ and $D = 1$ means that the fractal has characteristics that lie between those of a point and a line, a dimension between $D = 1$ and $D = 2$ means that the fractal has characteristics that lie between those of a line and a plane, and a dimension between $D = 2$ and $D = 3$ means that the fractal has characteristics that lie between those of a plane and a space. With its dimension of $D = 1.2618$, the Koch curve is a construct that is more complicated than a line, but not nearly as complicated as a plane. Similarly, the Menger sponge is a construct that is much more complicated than a plane, but not quite as complicated as a space, as is shown by its dimension of $D = 2.7268$.

3.3.2 Variance Fractal Dimension

The idea of a power-law relationship between the variance of the amplitude of a signal and its time increments dates back to the work done by Mandelbrot and Van Ness in 1968, and the introduction of fractional Brownian motion [MaVa68], [Mand71] as a generalization of Brownian motion [Brow28]. These ideas were further refined, and the first plots of this power-law relationship where shown by Mandelbrot and Wallis in 1969 [MaWa69a], [MaWa69b], [MaWa69c], [Mand83, p. 250]. However, the work presented in these papers is very mathematical, and recent work by Kinsner [Kins94a] and Grieder [Grie96] provide this thesis with a more palatable approach to the development of the variance dimension for the modelling of natural phenomena [MaSu93].

Given a discrete time signal $B(t)$, the variance σ^2 of its amplitude over a time increment is related to the time increment according to the power law [Kins94a]

$$Var[B(t_2) - B(t_1)] \propto |t_2 - t_1|^{2H} \tag{3.5}$$

where H is the Hurst exponent [Mand83, p. 396], [ZhBM90], [PeJS92, p. 493].

If we define

$$\Delta t = |t_2 - t_1| \tag{3.6}$$

and

$$(\Delta B)_{\Delta t} = B(t_2) - B(t_1) \tag{3.7}$$

then H can be calculated by

$$H = \lim_{\Delta t \to 0} \frac{1}{2} \frac{\log[Var(\Delta B)_{\Delta t}]}{\log(\Delta t)} \tag{3.8}$$

which is ½ times the slope of the line on the log-log plot, for decreasing values of Δt. For an embedding Euclidean dimension E [Vics92, p. 10], the variance dimension D_σ of a signal can be computed by

$$D_\sigma = E + 1 - H \tag{3.9}$$

The real-time coding procedure to calculate D_σ will now be shown. Given a window of a b-adic sequence ($b = 2$ is a dyadic sequence) containing N_T points,

$$K_{low} = 2 \tag{3.10}$$

$$K_{max} = \left\lfloor \frac{\log N_T}{\log b} \right\rfloor \tag{3.11}$$

$$K_{buf} = \left\lceil \frac{\log 30}{\log b} \right\rceil \tag{3.12}$$

and

$$K_{hi} = K_{max} - K_{buf} \tag{3.13}$$

For $K_{low} \le k \le K_{hi}$, the variance is calculated from

$$Var(\Delta B)_k = \frac{1}{N_k - 1} \left[\sum_{j=1}^{N_k} (\Delta B)_{jk}^2 - \frac{1}{N_k} \left(\sum_{j=1}^{N_k} (\Delta B)_{jk} \right)^2 \right] \tag{3.14}$$

where

$$n_k = b^k \tag{3.15}$$

and

$$N_k = \frac{N_T}{n_k} \tag{3.16}$$

The X and Y values for the log-log plot are then calculated by

$$X_k = \log[n_k] \tag{3.17}$$

and

$$Y_k = \log[Var(\Delta B)_k] \tag{3.18}$$

The slope s for the log-log plot is

$$s = \frac{K \sum_{i=K_{low}}^{K_{hi}} X_i Y_i - \sum_{i=K_{low}}^{K_{hi}} X_i \sum_{i=K_{low}}^{K_{hi}} Y_i}{K \sum_{i=K_{low}}^{K_{hi}} X_i^2 - \left(\sum_{i=K_{low}}^{K_{hi}} X_i \right)^2} \tag{3.19}$$

where

$$K = K_{hi} - K_{low} + 1 \tag{3.20}$$

Finally, as in the previous derivation,

$$H = \frac{1}{2}s \tag{3.21}$$

and with a 1-dimensional signal ($E = 1$),

$$D_\sigma = 2 - H \tag{3.22}$$

Another feature of the variance fractal dimension is that the values are normalized to lie between 1 and 2. $D_\sigma = 1$ would be the dimension of a straight line and $D_\sigma = 2$ would be the dimension of totally random and uncorrelated white noise.

3.4 Multifractals and Multifractal Dimensions

When fractals are combined either spatially or temporally, it is appropriate that the word "multifractal" be used to describe the new construct. The prefix "multi" means "more than one", so the term multifractal means that there is more than one fractal in time or space, or both [Vics92, p. 49]. Multifractal dimensions are an extension of regular fractal dimensions, and are better suited to the characterization of complex multifractal behaviour.

The trajectory of the variance fractal dimension will be discussed as a means of temporal multifractal characterization, and the Rényi dimension spectrum will be introduced as a test for multifractality and as a means of spatial multifractal characterization. The reliability and accuracy of these multifractal descriptions must also be considered, and is investigated by Chen [Chen97].

3.4.1 Variance Fractal Dimension Trajectory

As described in section 3.3.2, the variance fractal dimension is calculated from a portion, or window, of a 1-dimensional signal. Therefore, the numerical value of the variance fractal dimension is dependent on the size of the window. The selection of this window size is very important, and is discussed in detail in section 5.4. Furthermore, if the nature of the signal is expected to change over time, then the calculation of the variance fractal dimension is also dependent on the location of the window. When the window is shifted in time, it must also be decided if the window will either overlap, or not overlap, part of the previous window. If the window is non-overlapping, then any portion of the signal is only used once in the calculation of the variance fractal dimension; if the window is overlapping, then portions of the signal (excluding a few points at the beginning and end of a sequence equal to the offset of the window) are used more than once. Calculations of the variance fractal dimension for a sliding window in time results in a sequence, or trajectory, of values. This trajectory of dimensions may be called the variance fractal dimension trajectory (VFDT), and demonstrates a fractal dimension that changes with time. The calculation of the VFDT using an overlapping window is illustrated in Fig. 3.7, with the original signal shown on top and its VFDT shown on the bottom. Since a temporal window size of approximately 50 seconds was used to calculate the variance fractal dimensions, its VFDT is shifted backwards in time by the length of the window. Therefore, the spike located at 26.7 sec on the original plot is easily detected at about 27.6 sec on the VFDT.

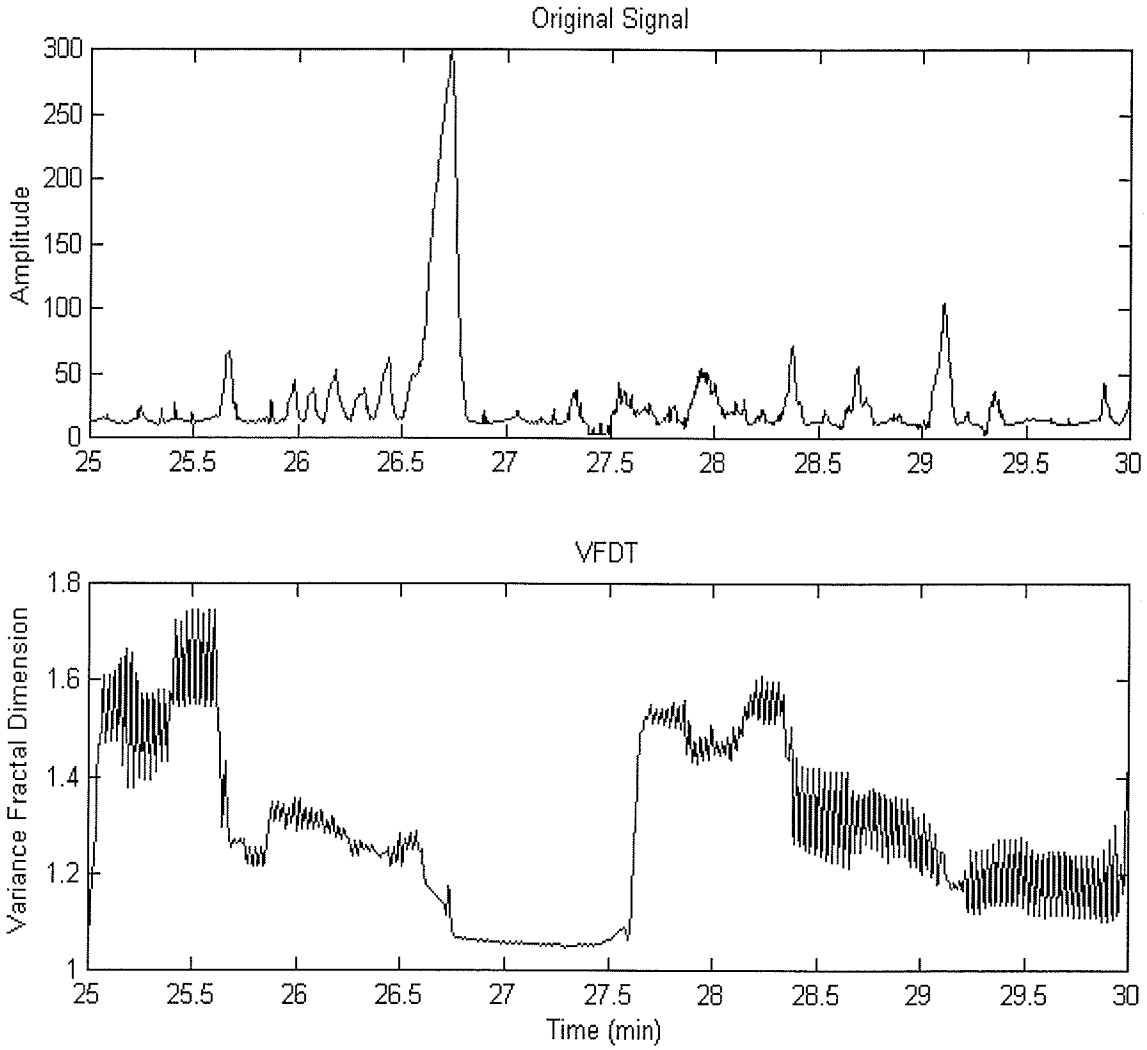

Fig. 3.7. Variance fractal dimension calculated through time.

Previous research has shown that the VFDT is useful in characterizing temporal signals such as speech [Grie96], radio transmitter transients [Toon97], [Shaw97], [Sun99], power systems transients [Chen02], and self-affine traffic [BaKi02], [BKPM03].

3.4.2 Rényi Dimension Spectrum

The Rényi (multifractal) dimension spectrum was introduced in 1983 by Hentschel and Procaccia [HePr83], and is an extension of the generalized entropy that was formalized by Alfréd Rényi in 1960 [Rény55], [Rény59], [Rény61]. A comprehensive discussion of the Rényi generalized entropy, the Rényi dimension spectrum, the generalized relative entropy, and the relative Rényi dimension spectrum is given by Dansereau [Dans01], and is summarized below.

Let X be a discrete random variable with finite alphabet χ and probability mass function $p(x) = \Pr\{X = x\}$, $\forall x \in \chi$.

Claude E. Shannon defined the Shannon entropy [Shan48a], [Shan48b] $H(X)$ of a discrete random variable X as

$$H(X) = -\sum_{x \in \chi} p(x) \log p(x) \tag{3.23}$$

The generalized form of Shannon entropy [Rény61] is

$$H(X) = \frac{-\sum_{x \in \chi} p(x) \log p(x)}{\sum_{x \in \chi} p(x)} \tag{3.24}$$

Eq. 3.23 implies that $p(x)$ be a normal probability distribution with the restriction that

$$\sum_{x \in \chi} p(x) = 1 \tag{3.25}$$

This restriction is relaxed in Eq. 3.24, and $p(x)$ can be an *incomplete* probability distribution [Dans01] where

$$0 < \sum_{x \in \chi} p(x) < 1 \tag{3.26}$$

The Rényi generalized entropy $H_q(X)$ of order q of a discrete random variable X is defined as

$$H_q(X) = \frac{1}{1-q} \log \frac{\sum_{x \in \chi} p^q(x)}{\sum_{x \in \chi} p(x)} \quad , -\infty \le q \le \infty \tag{3.27}$$

When $q = 1$, $H_q(X)$ becomes $H(X)$ and is the Shannon entropy [PeJS92, p. 737].

The Rényi dimension spectrum $D_q(X)$ (or simply D_q) of order q of a discrete random variable X with a probability distribution $p_r(x)$ at scale r is defined as

$$D_q(X) = \lim_{r \to 0} \frac{1}{1-q} \frac{\log \frac{\sum_{x \in \chi} p_r^q(x)}{\sum_{x \in \chi} p_r(x)}}{\log\left(\frac{1}{r}\right)} = \lim_{r \to 0} \frac{H_q(X)}{\log\left(\frac{1}{r}\right)} \tag{3.28}$$

The Rényi dimension spectrum is a monotonically non-increasing function of q [Dans01] for $-\infty \leq q \leq \infty$. If the object is a single fractal, then D_q is a constant for all values of q. If the object is multifractal, then the value of D_q varies with q. Fig. 3.7 gives an example of a Rényi dimension spectrum for a single fractal and a multifractal. The plot of the fractal object is a straight horizontal line whereas a plot of the multifractal object resembles a backwards-S curve.

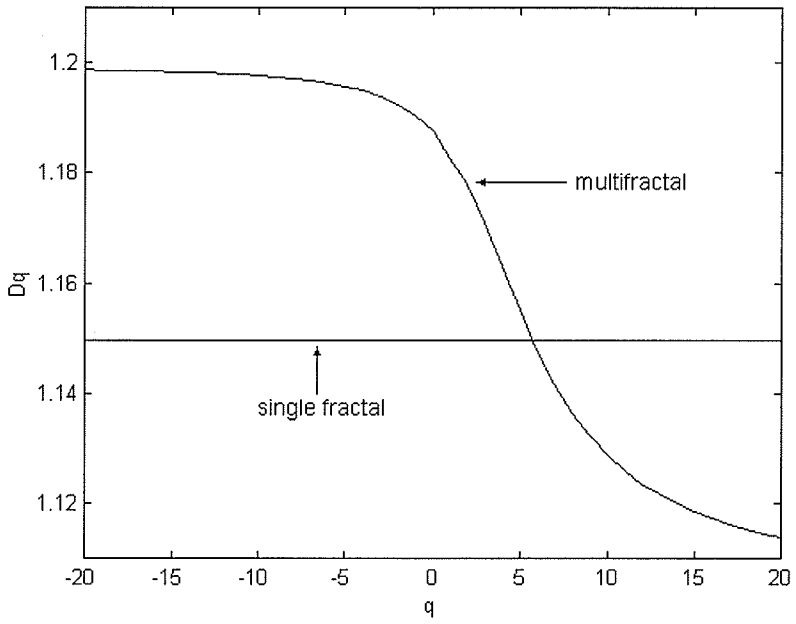

Fig. 3.8. A Rényi dimension spectrum for a single fractal and a multifractal.

A test to determine if an object is multifractal or not is to calculate the Rényi dimension spectrum for the object in question; if the spectrum is constant for all values of q (or, in practice, for a sufficiently large range of q), then the object is a single fractal. It should also be mentioned that in practice, care must be taken when calculating a small probability raised to a power q, as the precision needed to accurately represent extremely

small numbers may exceed the maximum precision of a computer. Previous research has also shown that the Rényi dimension spectrum is useful in characterizing more complicated signals such as EMG recordings [Ehti99] and DNA [Rifa98], [Zhan02].

If an object is a single fractal in space but varies through time, then it possesses temporal multifractality and may be described using the variance fractal dimension trajectory; if the object is multifractal in space and constant through time, then it possesses spatial multifractality and may be described using the Rényi dimension spectrum. More complicated phenomena do exist that exhibit multifractality in both space and time. These objects may be described through the changes in the Rényi dimension spectrum through the formation of the Rényi dimension spectrum *trajectory*, as demonstrated by Rifaat [Rifa98] and Sun [Sun99].

3.5 Summary

This chapter provides an introduction to fractals and multifractals in both space and time, and shows examples of the generation of mathematical fractals through a simple initiator and iterative process. An infinite number of fractal dimensions exist to characterize and describe fractals, but the Rényi dimension spectrum and the variance fractal dimension trajectory will be used in this thesis to demonstrate the multifractal nature of self-affine traffic, and to provide a multifractal signature for the traffic.

CHAPTER IV

BACKGROUND ON FEATURE EXTRACTION AND NEURAL NETWORK CLASSIFICATION

4.1 Basic Statistical Analysis

A sequence of numbers may be analyzed in many ways. In statistics, a starting point for analyzing a sequence of numbers is through the calculation of its mean and variance. (For the sake of clarity, it should be stated that the variance calculated in this chapter is the statistical variance of a sequence, and is not the same as the calculation of the variance *fractal dimension.*) The mean (or *expected value*) μ of a random variable X with probability distribution $f(x)$ [WaMM98, p. 85] is calculated by

$$\mu = \varepsilon(X) = \sum_x x f(x) \tag{4.1}$$

Similarly, the variance σ^2 of a random variable X with probability distribution $f(x)$ [WaMM98, pp. 93-94] is calculated by

$$\sigma^2 = \varepsilon[(X-\mu)^2] = \sum_x (x-\mu)^2 f(x) = \varepsilon(X^2) - \mu^2 \tag{4.2}$$

The statistical mean and variance of a sequence may be sufficient in its description, but if it is not, then the higher-order statistics of a sequence may also be calculated.

4.2 Higher-Order Statistics

The higher-order statistics used in addition to the mean and variance are called the *skewness* and *kurtosis*, and are described in sections 4.2.1 and 4.2.2, respectively.

4.2.1 Skewness

Skewness is the normalized *third* central moment of a distribution, and is a measure of the asymmetry of sampled data around the mean μ. If the data are perfectly symmetrical, then the skewness is zero. If the data are spread more to the left of μ, then the skewness is negative; if the data are spread more to the right of μ, then the skewness is positive.

The skewness s of a random variable X with probability distribution $f(x)$, mean μ, and variance σ^2 [Math03a] is calculated as

$$s = \frac{\varepsilon[(X-\mu)^3]}{\sigma^3} \tag{4.3}$$

4.2.2 Kurtosis

Kurtosis is the normalized *fourth* central moment of a distribution, and is a measure of the "peakedness" of a distribution. The kurtosis k a random variable X with probability distribution $f(x)$, mean μ, and variance σ^2 [Math03b] is calculated as

$$k = \frac{\varepsilon[(X-\mu)^4]}{\sigma^4} \tag{4.4}$$

The kurtosis of a Gaussian (or *normal*) distribution using Eq. 4.4 is 3. To make the kurtosis of a Gaussian distribution equal to zero, the following equation may also be used.

$$k = \frac{\varepsilon[(X-\mu)^4]}{\sigma^4} - 3 \tag{4.5}$$

4.3 Histogram Modelling

A histogram is a grouping of data into a finite number of intervals, or *bins*, based upon the frequency of occurrence of the individual data points within each bin. Figure 4.1 shows 1000 data points that have been selected from a Gaussian distribution.

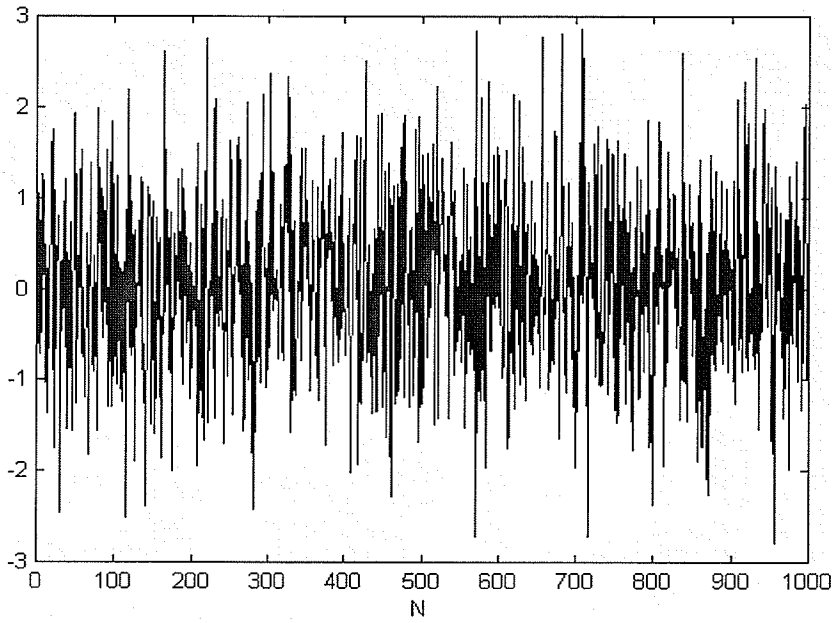

Fig. 4.1. 1000 data points selected from a Gaussian distribution.

If we did not know that the data were generated using a Gaussian distribution, we could figure this out by dividing the range of data into equal sized bins and counting the

number of data points in each bin [Meis72, pp. 41-42]. The number of bins used is also important: if too few (5) or too many (99) bins are used (as shown in Figs. 4.2 and 4.3, respectively), then the underlying distribution may not be so easily identified. Figure 4.4 shows a good histogram formed by choosing 19 bins.

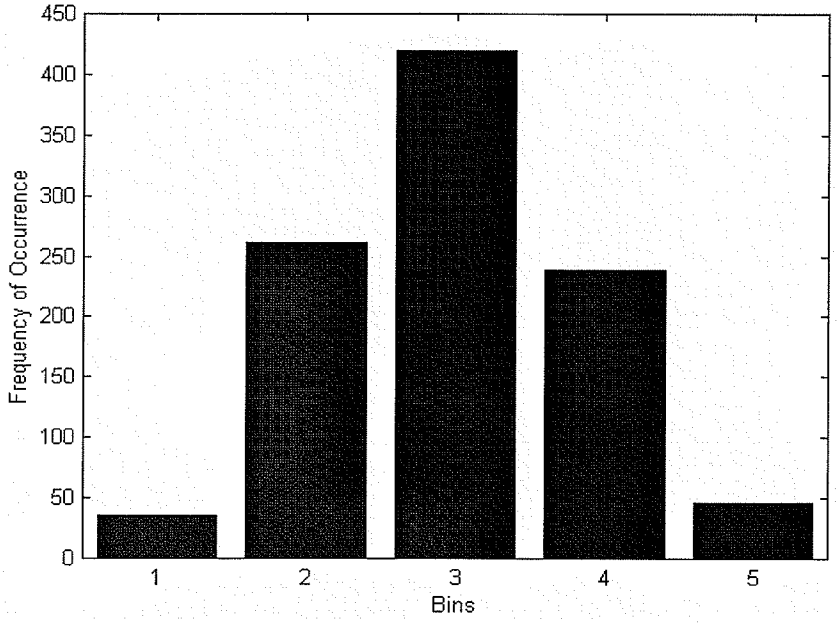

Fig. 4.2. Too few bins used to construct a histogram.

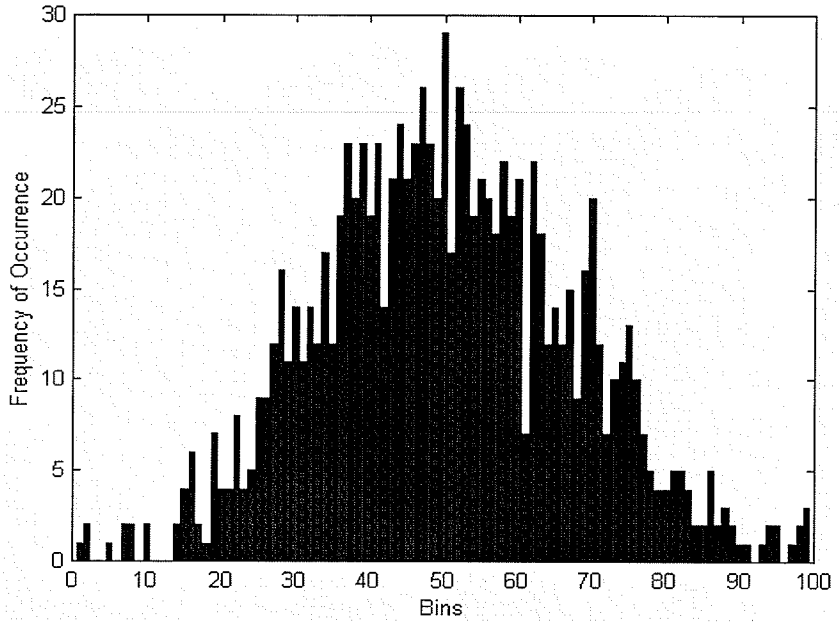

Fig. 4.3. Too many bins used to construct a histogram.

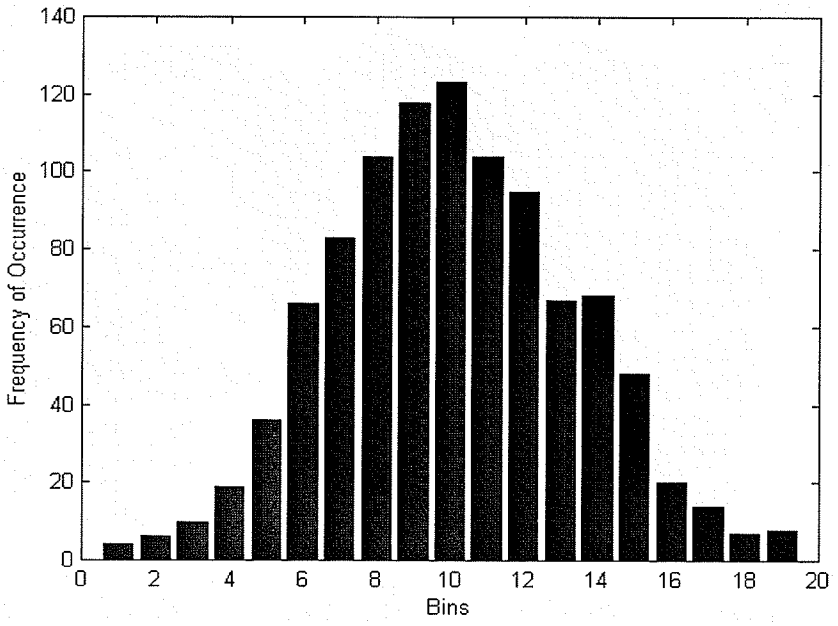

Fig. 4.4. A good number of bins used to construct a histogram.

The Gaussian distribution $N(x)$ is characterized by two parameters: the mean μ and variance σ^2 (or standard deviation σ) [WaMM98, p.145], and is expressed as

$$N(x; u, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} \tag{4.6}$$

With these two parameters, a bar graph of a particular Gaussian distribution can be constructed which closely resembles, or *models*, Fig. 4.4. Using Eqs. 4.1 and 4.2, the data used to generated Fig. 4.1 has $\mu = 0.514$ and $\sigma^2 = 0.9717$. A bar graph of a Gaussian distribution with these two parameters is shown in Fig. 4.5.

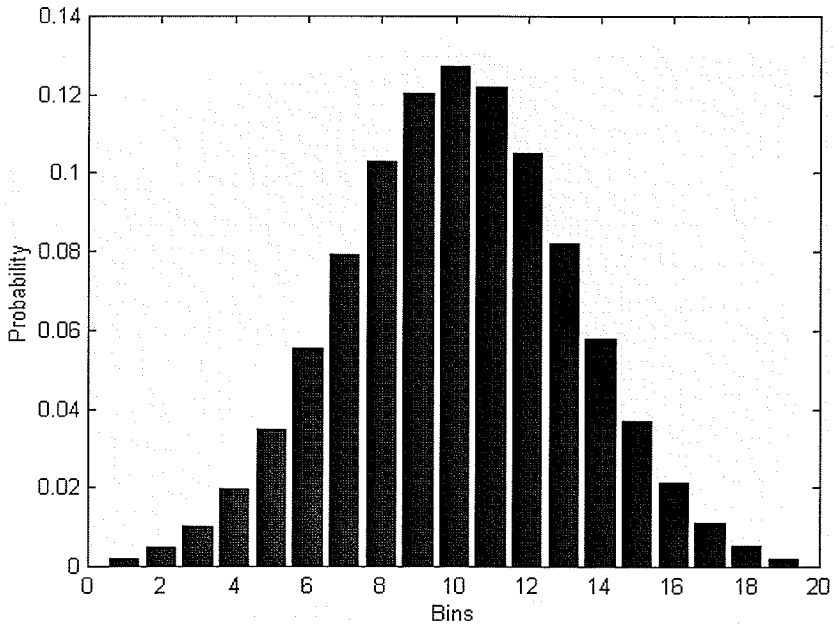

Fig. 4.5. Histogram modelling using the Gaussian distribution.

A visual comparison between the histogram in Fig. 4.4 with the Gaussian distribution in Fig. 4.5 shows us that our model fits the original histogram quite well. The vertical axes on these graphs are different, but equivalent. In Fig. 4.4, the vertical axis is the frequency of occurrence with a maximum of approximately 120; since there are 1000 data points used to construct the histogram, this number represents 12% (or 0.12) of the data. This normalized representation is the probability, and is the vertical axis on Fig. 4.5.

4.3.1 Gamma Distribution

The gamma distribution $G(x)$ is a generalized statistical distribution represented by the two parameters alpha α and beta β [Weis99a], and is expressed as

$$G(x;\alpha,\beta) = \frac{x^{\alpha-1}e^{-\frac{x}{\beta}}}{\beta^{\alpha}\,\Gamma(\alpha)} \tag{4.7}$$

where $\Gamma(\alpha)$ is the gamma function [Weis99b]

$$\Gamma(\alpha) = \int_0^{\infty} t^{\alpha-1}e^{-t}\,dt \tag{4.8}$$

Unlike the Gaussian distribution which always maintains its symmetric bell-shape form when the parameters μ and σ are varied, the gamma distribution is very flexible and can take on a wide range of shapes as the parameters α and β are varied. This flexibility makes the gamma distribution an ideal choice for modelling rapidly changing probability distributions. Figure 4.6 shows the gamma distribution with $\alpha = 1$ and varying β and Fig. 4.7 shows the gamma distribution with $\beta = 1$ and varying α.

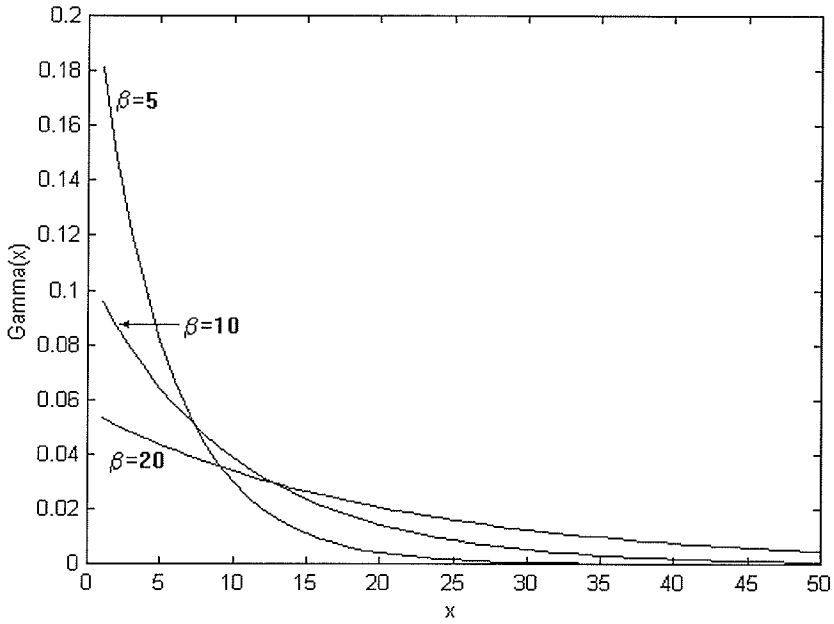

Fig. 4.6. Gamma distribution with $\alpha = 1$ and varying β.

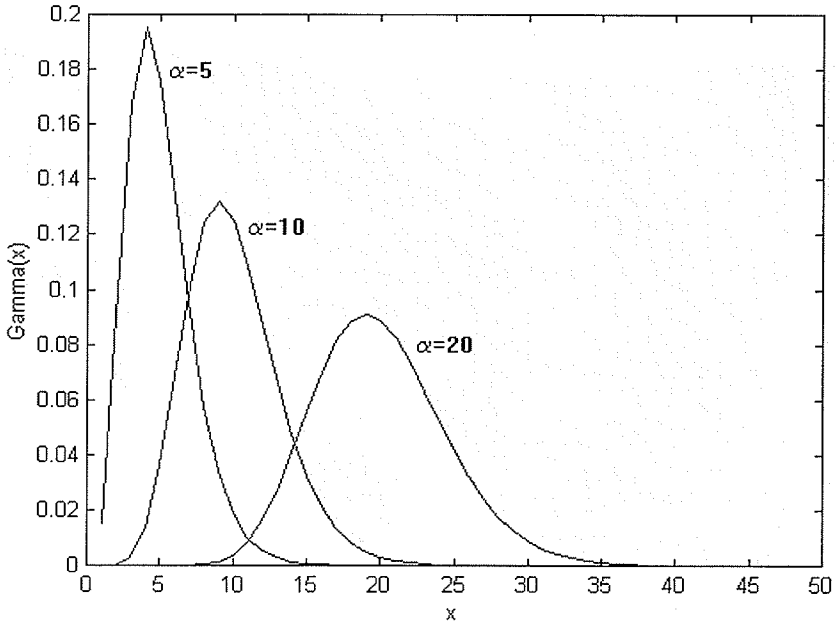

Fig. 4.7. Gamma distribution with $\beta = 1$ and varying α.

Figure 4.6 reveals that when $\alpha = 1$, the gamma distribution becomes the exponential distribution

$$G(x;\alpha = 1, \beta) = \frac{1}{\beta} e^{-\frac{x}{\beta}} \tag{4.9}$$

Figure 4.7 shows that the gamma distribution can also take on a form that is similar to a log-normal or Gaussian distribution. When both parameters α and β are varied simultaneously, the resulting distribution can even look like a "squished" Gaussian distribution or reversed exponential distribution, as shown in Fig. 4.8.

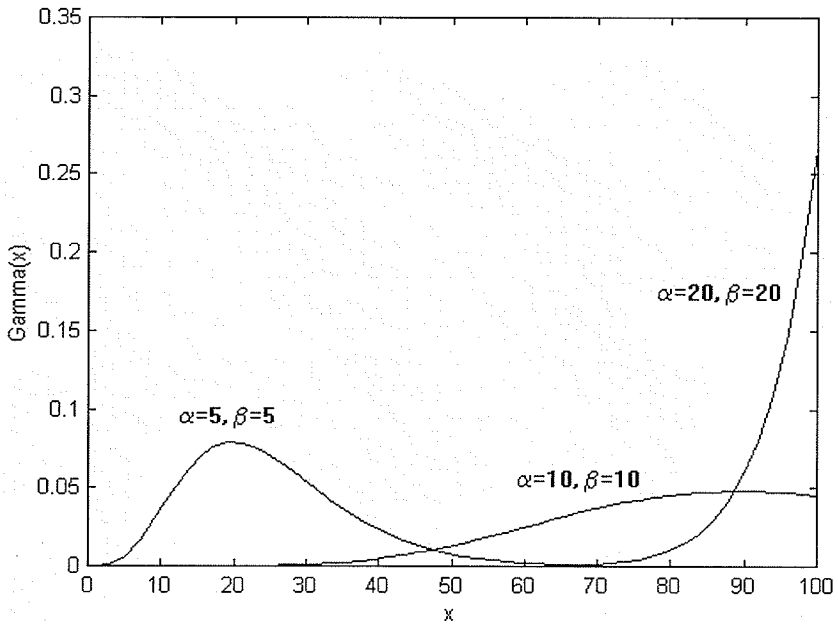

Fig. 4.8. Gamma distribution with varying α and β.

4.4 Principal Component Analysis

Principal component analysis (PCA) is a linear transformation used to isolate and extract the most important features from a signal through the removal of redundant information. In a system represented by several variables, it is most likely that some degree of correlation exists between variables; for example, if variable A always increases when variable B decreases, then there is correlation between variables A and B. If these variables can be transformed into new variables A^* and B^*, where A^* contains most of the information between A and B, and B^* contains the remaining information, then it may be possible to retain A^* and discard B^* and still preserve enough of the information to properly describe the relationship between the original variables A and B.

The following derivation of PCA is provided by Bishop [Bish00, p. 310-313]. PCA maps vectors $\boldsymbol{x}^n$ in d-dimensional space $(x_1, ..., x_d)$ onto vectors $\boldsymbol{z}^n$ in M-dimensional space $(z_1, ..., z_M)$ where $M < d$. The vector $\boldsymbol{x}$ can be represented as a linear combination of orthonormal vectors $\boldsymbol{u}_i$, as shown by

$$\boldsymbol{x} = \sum_{i=1}^{d} z_i \boldsymbol{u}_i \tag{4.10}$$

The vectors $\boldsymbol{u}_i$ satisfy the orthonormality relation

$$\boldsymbol{u}_i^T \boldsymbol{u}_j = \delta_{ij} \tag{4.11}$$

where δ_{ij} is the Kronecker delta symbol where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise.

Using Eq. 4.11, an expression for the coefficients z_i is given by

$$z_i = \boldsymbol{u}_i^T \boldsymbol{x} \tag{4.12}$$

If a subset $M < d$ of the basis vectors $\boldsymbol{u}_i$ are retained so that only M coefficients z_i are used, then the remaining coefficients can be replaced by constants b_i and each vector x approximated by

$$\tilde{\boldsymbol{x}} = \sum_{i=1}^{M} z_i \boldsymbol{u}_i + \sum_{i=M+1}^{d} b_i \boldsymbol{u}_i \tag{4.13}$$

Suppose we have a data set of N vectors $\boldsymbol{x}^n$ where $n = 1, ..., N$. We wish to choose the basis vectors $\boldsymbol{u}_i$ and coefficients b_i such that the approximation in Eq. 4.13 gives the best approximation to the original vector $\boldsymbol{x}$ on average for the entire data set. The error in the vector $\boldsymbol{x}^n$ introduced by the dimensionality reduction is given by

$$\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}^n = \sum_{i=M+1}^{d} (z_i^n - b_i) \boldsymbol{u}_i \tag{4.14}$$

The best approximation is defined as the one that minimizes the sum of the squares of the error over the data set. This is accomplished by minimizing

$$E_M = \frac{1}{2} \sum_{n=1}^{N} \left\| \boldsymbol{x}^n - \tilde{\boldsymbol{x}}^n \right\|^2 = \frac{1}{2} \sum_{n=1}^{N} \sum_{i=M+1}^{d} (z_i^n - b_i)^2 \tag{4.15}$$

The derivative of E_M in Eq. 4.15 with respect to b_i set to zero shows us that

$$b_i = \frac{1}{N}\sum_{n=1}^{N} z_i^n = \boldsymbol{u}_i^T \bar{\boldsymbol{x}} \tag{4.16}$$

where

$$\bar{\boldsymbol{x}} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}^n \tag{4.17}$$

Combining Eqs. 4.12 and 4.16, Eq. 4.15 can be written as

$$E_M = \frac{1}{2}\sum_{i=M+1}^{d}\sum_{n=1}^{N}\{\boldsymbol{u}_i^T(\boldsymbol{x}^n - \bar{\boldsymbol{x}})\}^2 = \frac{1}{2}\sum_{i=M+1}^{d} \boldsymbol{u}_i^T \Sigma \boldsymbol{u}_i \tag{4.18}$$

where Σ is the covariance matrix of the set of vectors $\{\boldsymbol{x}^n\}$ given by

$$\Sigma = \sum_n (\boldsymbol{x}^n - \bar{\boldsymbol{x}})(\boldsymbol{x}^n - \bar{\boldsymbol{x}})^T \tag{4.19}$$

It has been shown [Bish00, App. E] that minimizing E_M with respect to the basis vectors $\boldsymbol{u}_i$ occurs when the basis vectors satisfy

$$\Sigma \boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i \tag{4.20}$$

and are the eigenvectors of the covariance matrix. Substituting Eq. 4.20 into Eq. 4.18 and using Eq. 4.11, the error criterion at the minimum is

$$E_M = \frac{1}{2} \sum_{i=M+1}^{d} \lambda_i \tag{4.21}$$

Therefore, the minimum error is achieved by retaining the M largest eigenvalues and their corresponding eigenvectors, and discarding the $d-M$ smallest eigenvalues.

4.5 K-Means Clustering

If we had access to Sarda's data sets (as explained in section 2.6), then we would have a database of known signal classes that we could characterize. Signal A would have one set of characteristics, signal B would have another set of characteristics, and so on. If the signal classes are known, then sections 4.5 and 4.6 would not be necessary. However, what if the signal classes are not known? How could extracted characteristics be grouped, or *clustered*, and classes assigned to these clusters?

Figure 4.9 shows a simple problem where unknown hypothetical traffic is characterized and plotted in 2-dimensions. Poor clustering of these data points can lead to an incorrect number of identified classes, as shown by Fig. 4.10. Similarly, good clustering can lead to a more accurate (and possibly correct) number of identified classes, as shown by Fig. 4.11. The human visual system is very good at identifying clusters of data, and can immediately confirm that there are most likely four distinct classes within Fig. 4.10. Further imagination would find it difficult, if not impossible, to properly and intuitively find five or more classes in Fig. 4.9! The important topics of correct clustering and classification will be discussed further in sections 5.8.2 and 6.2.

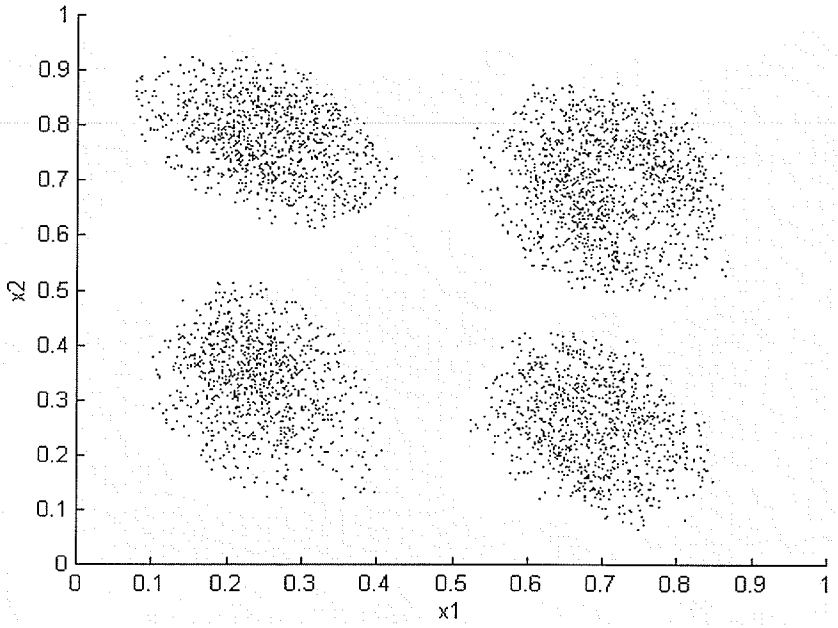

Fig. 4.9. Traffic characteristics plotted in 2-dimensions.

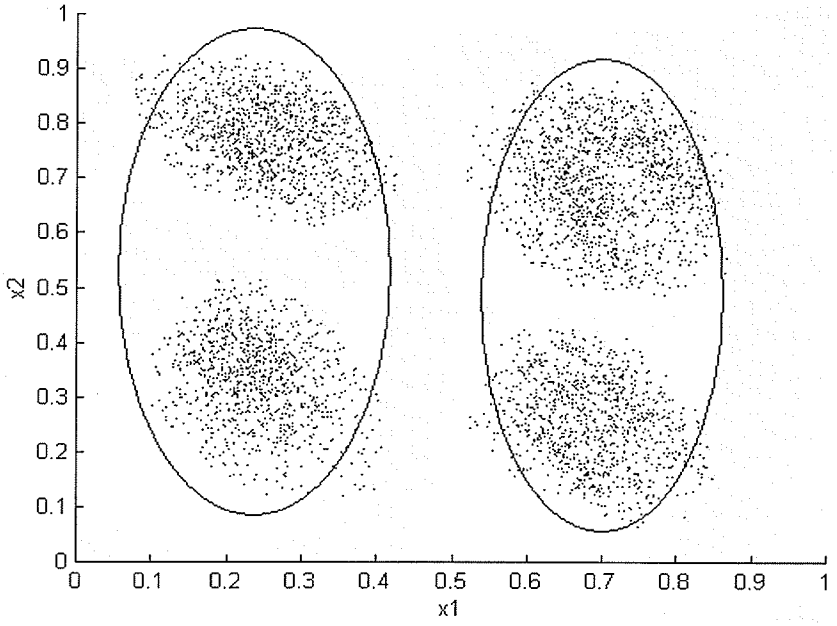

Fig. 4.10. A poor choice of two clusters (or classes) in Fig. 4.9.

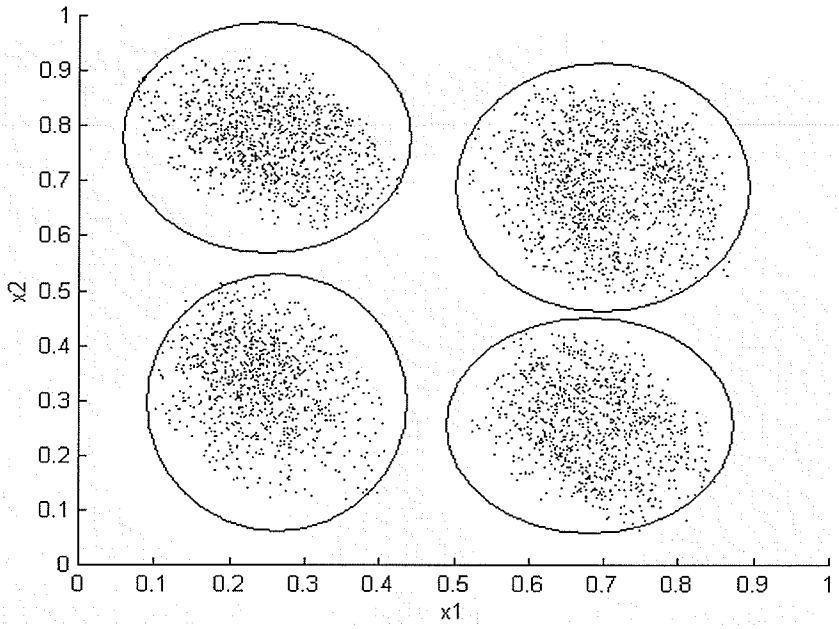

Fig. 4.11. A good choice of four clusters (or classes) in Fig. 4.9.

The K-means clustering algorithm [MacQ67] is used to cluster data points in N-dimensional space so that classes may be assigned to each cluster. The K-means algorithm was chosen because it is simple, intuitive, and preliminary experimentation showed that it gave good results. Hartigan provides the following description of the K-means clustering algorithm [Hart75, pp. 84-86].

Preliminaries:

- The I th case of the J th variable has value $A(I, J)$ where $1 \leq I \leq M$ and $1 \leq J \leq N$.
- The partition $P(M, K)$ is composed of the clusters 1, 2, 3, ..., K. Each of the M cases lies in just one of the K clusters.
- The mean of the J th variable over the cases in the L th cluster is denoted by $B(L, J)$.
- The number of cases in L is $N(L)$.

- The distance $D(I, L)$ between the I th case and L th cluster is

$$D(I,L) = \sqrt{\sum_{J=1}^{N} [A(I,J) - B(L,J)]^2} \tag{4.22}$$

The error $e[P(M, K)]$ of the partition is

$$e[P(M,K)] = \sum_{I=1}^{M} \{D[I, L(I)]\}^2 \tag{4.23}$$

where $L(I)$ is the cluster containing the I th case.

Step 1:

- Assume initial clusters 1, 2, ..., K and compute the cluster means $B(L, J)$ where $1 \le L \le K$ and $1 \le J \le N$, and the initial error as shown in Eq. 4.23

Step 2:

- For the first case, compute

$$\frac{N(L)[D(1,L)]^2}{N(L)+1} - \frac{N[L(1)]\{D[1,L(1)]\}^2}{N[L(1)]-1}$$

for every cluster L, which is the increase in the error by transferring the first case from cluster $L(1)$ to cluster L. If the minimum of this quantity over all $L \neq L(1)$ is negative, then transfer the first case from cluster $L(1)$ to this minimal L, update the cluster means of $L(1)$ and the minimal L, and then add the increase in error (which is negative) to $e[P(M, K)]$.

Step 3:

- **Repeat** Step 2 for the I th case, where $2 \leq I \leq M$.

Step 4:

- **Stop** if no movement of a case from one cluster to another occurs for any case. Otherwise, return to Step 2.

4.6 Self-Organizing Feature Map

Kohonen's self-organizing feature map (SOFM) [Koho88, Ch. 5], [Koho90] is a neural network that groups data based upon their topological similarity. Data points that are clustered in the same region, or *neighbourhood*, are assigned the same class. As its name suggests, the important features of the data organize themselves without the supervised control of a user. In this thesis, the SOFM could be used to cluster features; however, the task of clustering and dimensionality reduction is adequately performed by the K-means clustering algorithm and PCA. Therefore, the SOFM is used only for visual verification of the correctness of the K-means clustering. The clustering from the K-means algorithm and the SOFM are expected to be visually similar; if they are similar, then this is verification that the K-means algorithm is working as expected.

Kohonen provides the following discussion of the SOFM [Koho88, pp. 131-134]. The SOFM is a two-layered neural network with n-dimensional input nodes (denoted by $\boldsymbol{x}$) and p-dimensional output nodes (denoted by $\boldsymbol{y}$) with weights $\boldsymbol{w}$ connecting every input neuron to every output neuron, as shown in Fig. 4.12.

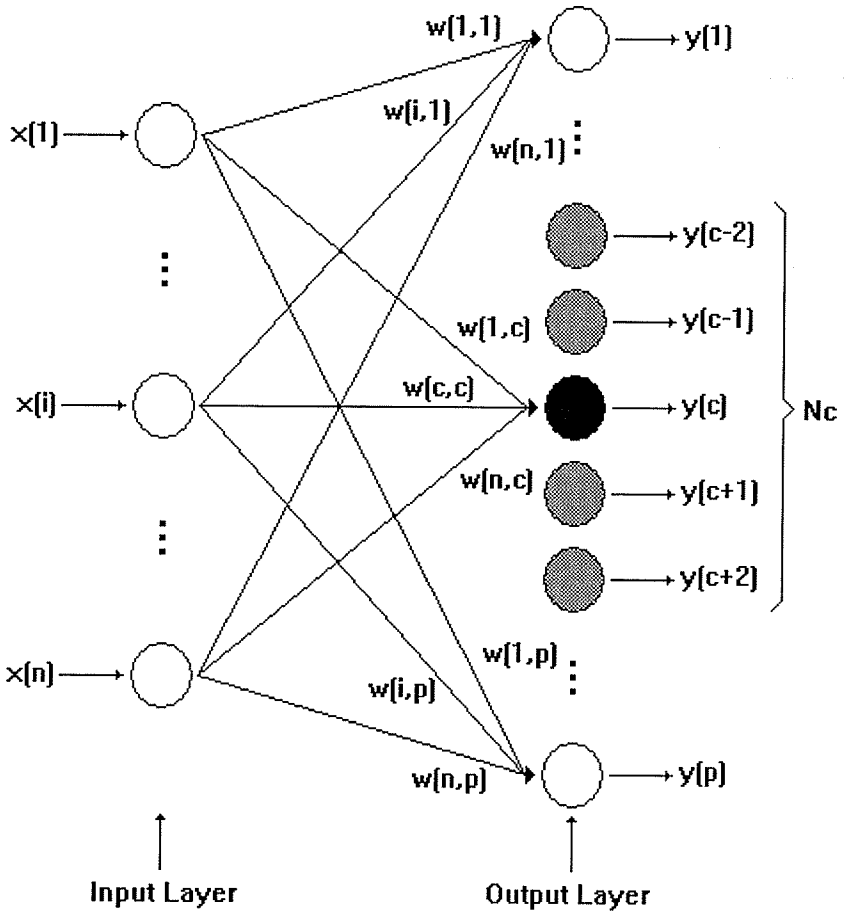

Fig. 4.12. SOFM architecture (after [Dans95]).

The SOFM trains itself through a *winner-takes-all* competitive learning scheme. In this scheme, only the winning output neuron $y(c)$ is activated for any given input vector $\boldsymbol{x}$ according to the condition

$$\|\boldsymbol{x} - \boldsymbol{w}(c)\| = \min_{i}\{\|\boldsymbol{x} - \boldsymbol{w}(i)\|\} \tag{4.24}$$

This evaluation is performed at discrete intervals of time, so Eq. 4.24 may be expressed as a function of time t

$$\|\boldsymbol{x}(t) - \boldsymbol{w}_c(t)\| = \min_i \{\|\boldsymbol{x}(t) - \boldsymbol{w}_i(t)\|\} \tag{4.25}$$

The weights $\boldsymbol{w}$ are updated by

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + h(t)[\boldsymbol{x}(t) - \boldsymbol{w}_i(t)] \ , \ \ i \in N_c \tag{4.26}$$

where N_c is the topological neighbourhood and contains all nodes within a certain radius from the winning node $y(c)$, and $h(t)$ is the neighbourhood function. A common definition for the neighbourhood function is the bubble function, defined as

$$h_{c,i}(t) = \alpha(t) \tag{4.27}$$

where $\alpha(t)$ is the learning rate (which usually decreases towards zero over time) which satisfies the restriction that $0 < \alpha(t) < 1$ [Zura92, p. 425]. Another commonly used neighbourhood function is

$$h_{c,i}(t) = \alpha(t)\, e^{-\frac{\|r_c - r_i\|^2}{[\sigma(t)]^2}} \tag{4.28}$$

where r_i and r_c are positional vectors for the winning neuron and neighbouring neuron, and $\sigma(t)$ is the decreasing radius function [Zura92, p. 425].

4.7 Probabilistic Neural Network

The probabilistic neural network (PNN) was introduced by Specht in 1988 [Spec88], although the necessary mathematical foundations were outlined 16 years earlier by Meisel in 1972 [Meis72, Ch. 2]. The PNN is a type of neural network that is specialized for classification problems, and works well when the training data are relatively sparse [Spec90b]. Its performance is generally very good, and asymptotically approaches Bayes optimality [Spec88]. Specht [Spec88], [Spec90a] provides the following discussion of the mathematical foundation for the PNN. Other detailed and comprehensive discussions are also provided by Wasserman [Wass93, Ch. 3] and Shaw [Shaw97].

4.7.1 Bayes Decision Rule

The Bayes decision rule for classification is constructed in a way that minimizes the "expected risk" of a decision [DuHa73, Ch. 2]. Consider the two-category situation in which the state of θ is either θ_A or θ_B. The decision of whether $\theta = \theta_A$ or $\theta = \theta_B$ is based on the set of measurements represented by the p-dimensional vector $\boldsymbol{X} = [X_1 \ldots X_p]$. Given that l_A is the loss associated with the decision $d(\boldsymbol{X}) = \theta_B$ when $\theta = \theta_A$ and l_B is the loss associated with the decision $d(\boldsymbol{X}) = \theta_A$ when $\theta = \theta_B$, h_A is the *a priori* probability of occurrence of patterns from category A and $h_B = 1 - h_A$ is the corresponding *a priori* probability of occurrence of patterns from category B, the Bayes decision rule is

$$d(\boldsymbol{X}) = \theta_A \quad \text{if } h_A l_A f_A(\boldsymbol{X}) > h_B l_B f_B(\boldsymbol{X}) \tag{4.29}$$

$$d(\boldsymbol{X}) = \theta_B \quad \text{if } h_A l_A f_A(\boldsymbol{X}) < h_B l_B f_B(\boldsymbol{X}) \tag{4.30}$$

where $f_A(\boldsymbol{X})$ and $f_B(\boldsymbol{X})$ are the probability density functions (PDFs) for categories A and B, respectively.

The boundary decision between categories A and B is therefore

$$f_A(\boldsymbol{X}) = K f_B(\boldsymbol{X}) \tag{4.31}$$

where

$$K = \frac{h_B l_B}{h_A l_A} \tag{4.32}$$

The problem with implementing the Bayes decision rule is that the PDFs $f_A(\boldsymbol{X})$ and $f_B(\boldsymbol{X})$ are generally unknown and may be too complicated to approximate with a simple distribution. However, these underlying probability densities may be estimated, and the estimates will converge asymptotically towards the true PDFs as the number of samples increases.

4.7.2 Parzen PDF Estimation

Parzen showed that if there are n_c training cases for a given class c, a PDF may be estimated [Parz62] by

$$g_c(\boldsymbol{X}) = \frac{1}{n_c \sigma} \sum_{i=1}^{n_c} W\left(\frac{\boldsymbol{X} - \boldsymbol{X}_i}{\sigma}\right) \tag{4.33}$$

where $W(\boldsymbol{X})$ is the weighting function and σ is the spread parameter for the width of this

weighting function that surrounds each data point. The weighting function $W(\boldsymbol{X})$ is usually chosen to be the Gaussian function

$$W(\boldsymbol{X}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\boldsymbol{X}^2}{2\sigma^2}} \tag{4.34}$$

By inserting Eq. 4.34 into Eq. 4.33, the PDF estimation becomes

$$g_c(\boldsymbol{X}) = \frac{1}{(2\pi)^{p/2}\sigma^p n_c} \sum_{i=1}^{n_c} e^{-\frac{\|X - X_i\|^2}{2\sigma^2}} \tag{4.35}$$

where $\boldsymbol{X} = [X_1 \ldots X_p]$. The spread parameter σ can take on a wide range of values. If σ is too small, then the PNN performs as a nearest neighbour classifier [Spec88], [CoHa67]; if σ is too large, then the PNN acts as a matched filter [Spec88]. As shown by Fig. 4.13, there exists a range of σ for most classification problems that provides good results [Spec88], [Wass93, p. 44].

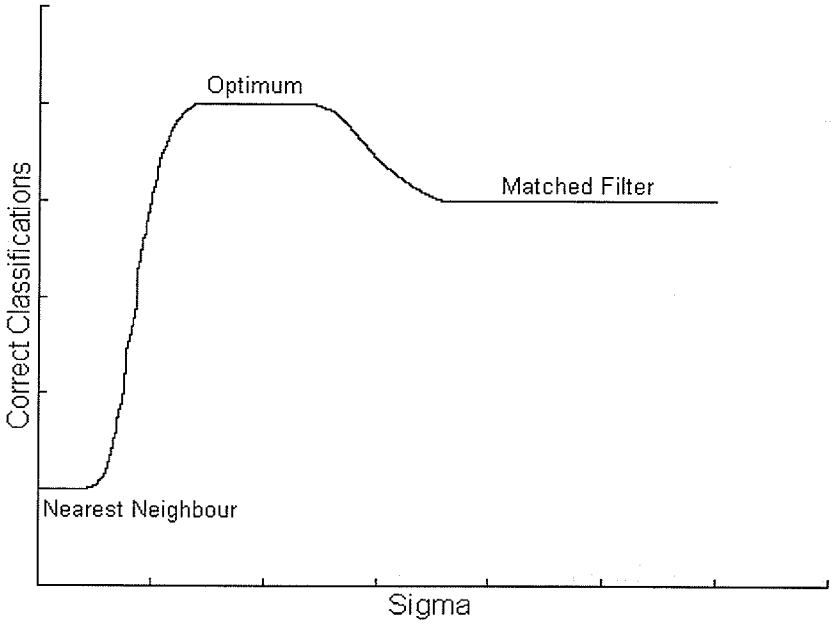

Fig. 4.13. Varying the spread parameter σ (after [Spec88], [CaCh03]).

For the sake of completeness, it should also be mentioned that in 1966, Cacoullos [Caco66] extended Eq. 4.35 to allow for the multivariate case where $\boldsymbol{\sigma} = [\boldsymbol{\sigma}_1 \ldots \sigma_p]$. This generalized expression is

$$g_c(\boldsymbol{X}) = \frac{1}{(2\pi)^{p/2} \sigma_c^{\,p} n_c} \sum_{i=1}^{n_c} e^{-\frac{\|X - X_i\|^2}{2\sigma_c^{\,2}}} \tag{4.36}$$

4.7.3 PNN Architecture

The PNN is designed to implement Bayes decision rule using Parzen's method of PDF estimation. Figure 4.14 shows the four layer architecture of the PNN.

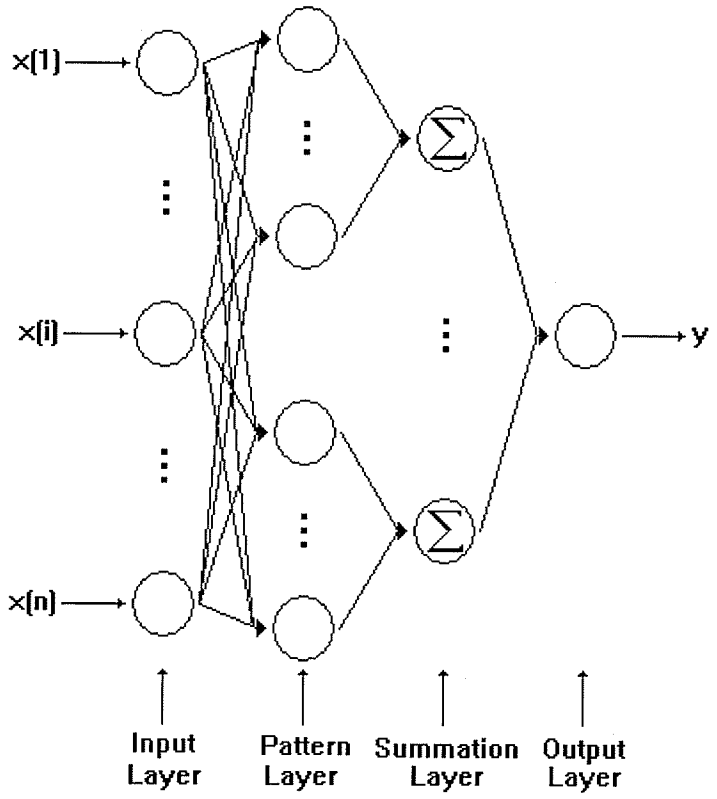

Fig. 4.14. PNN architecture (after [Spec88]).

The number of neurons in the input layer is equal to the dimensionality of the input data, and each neuron in the pattern layer corresponds to a training sample. The summation layer consists of neurons for each possible output class and performs the PDF estimation. The output layer usually consists of only one neuron which selects the largest value in the summation layer as the class output for the PNN.

In addition to providing very good results, a PNN is also very fast to train. Training typically takes seconds where similar training of a back-propagation neural network may take hours or days [Spec90a]. The main disadvantages of a PNN is that it

requires a substantial amount of memory to store every training sample; however, in today's computerized world, memory is generally abundant and inexpensive, so this "disadvantage" does not hinder the use of the PNN for classification problems.

4.8 Summary

This chapter provides an overview of the techniques used in feature extraction and classification. Firstly, a study of the higher-order statistics and the modelling of the histograms are used to extract the important characteristic features of the data. Secondly, PCA is performed to further compress this representation, and the K-means algorithm is used to cluster these features with verification from a SOFM. Finally, a PNN is trained to reliably classify newly observed traffic patterns.

CHAPTER V

SYSTEM DESIGN AND VERIFICATION

5.1 Verification of Sampling Frequency

In Pear's experiments, the position of the *Betta splendens* is recorded ten times per second [PeMa02]. Therefore, the sampling frequency is f_s = 10 Hz. Intuitively, this sampling frequency seems adequate to accurately observe and record the motion of the *Betta splendens*; however, one's intuition must be confirmed with a proof before these data may be used in this thesis.

In 1928, Harry Nyquist introduced the fundamental idea that for an analog signal to be properly represented by samples at discrete intervals of time, the sampling frequency must be at least twice the maximum frequency in the signal [Nyqu24], [Nyqu28], [Beau02]. If sampling was performed at less than twice the maximum frequency, then *aliasing* – a process whereby higher frequencies are incorrectly perceived to be lower frequencies – would occur, and the signal would be irrecoverably distorted. This idea was rigorously proven by Shannon in 1949 [Shan49] and is widely known as the *Nyquist sampling theorem*.

The sampling frequency used in Pear's experiments will be shown to be adequate using two different proofs.

Proof #1: The *Betta splendens* used in the experiments are approximately 2.5 cm in length, and their position is recorded with a maximum error of about 0.5 cm [PeMa02]. If

the sampling frequency of 10 Hz is inadequate, then the *Betta splendens* would have to evade the camera system at some time t by swimming 3.0 cm (2.5 cm for its length + 0.5 cm for the maximum error in its position) away, turning around, and swimming back to the same position before time (t + 0.1) sec. If this scenario did take place, then this high frequency movement would be incorrectly perceived as low frequency movement (i.e., the fish did not move); once again, this phenomenon is known as aliasing. For the *Betta splendens* to move as described above, it would have to swim 6.0 cm in 0.1 sec, or 60 cm / sec. However, Pear and Martin have observed that the *Betta splendens* swims at a maximum speed of approximately 10 cm / sec [PeMa02]. Therefore, the sampling frequency $f_s = 10\,\text{Hz}$ is clearly acceptable.

Proof #2: Assume that the *Betta splendens* moves along a sinusoidal trajectory [CaCh03]

$$x(t) = A\sin(2\pi f_N t) \tag{5.1}$$

where f_N is the Nyquist frequency given by

$$f_N = \frac{1}{2} f_s \tag{5.2}$$

and A is the amplitude, or length of the *Betta splendens*.

The velocity of the *Betta splendens* may be calculated as follows by taking the first derivative of Eq. 5.1 with respect to time

$$v(t) = \frac{\mathrm{d}}{\mathrm{d}t} x(t) = A\pi f_s \cos(\pi f_s t) \tag{5.3}$$

The velocity is at a maximum when

$$\cos(\pi f_s t) = 1 \tag{5.4}$$

and therefore

$$v_{max} = A\pi f_s \tag{5.5}$$

If we set $A = 2.0$ cm (2.5 cm for its length - 0.5 cm for the maximum error in its position) and $f_s = 10\,\text{Hz}$, then $v_{max} = 62.8\,\text{cm / sec}$, or approximately 60.0 cm / sec. Therefore, the sampling frequency $f_s = 10\,\text{Hz}$ is once again shown to be acceptable.

5.2 Verification of Self-Affinity

As discussed in section 3.1, self-affinity implies fractality (and vice versa). Therefore, to show that Pear's data sets are self-affine, it must be shown that they are fractal. As explained in section 3.3.2, to show that a signal is fractal, a power-law relationship must exist between the variance of the amplitude of a signal and its time increments. In other words, if a log-log plot of the above relationship yields a straight line, then the signal is fractal and therefore self-affine.

Figures 5.1 to 5.8 show four segments of Record 11020219, each with a length of 1024 data points, and the corresponding log-log plots for these segments. In these plots,

$$X = \Delta t = |t_2 - t_1| \tag{5.6}$$

and

$$Y = (\Delta B)_{\Delta t} = B(t_2) - B(t_1) \tag{5.7}$$

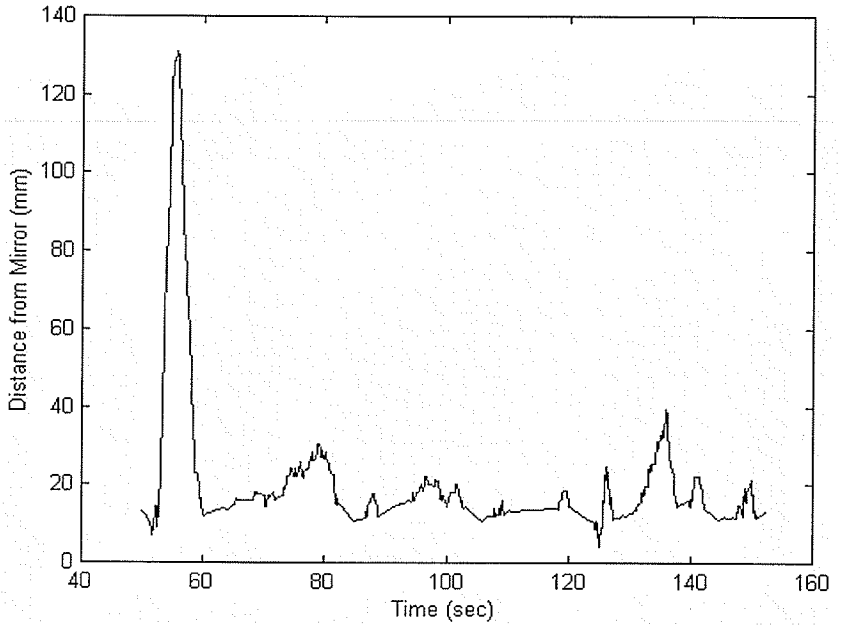

Fig. 5.1. Record 11020219 from $t = 50.1$ to 152.4 sec.

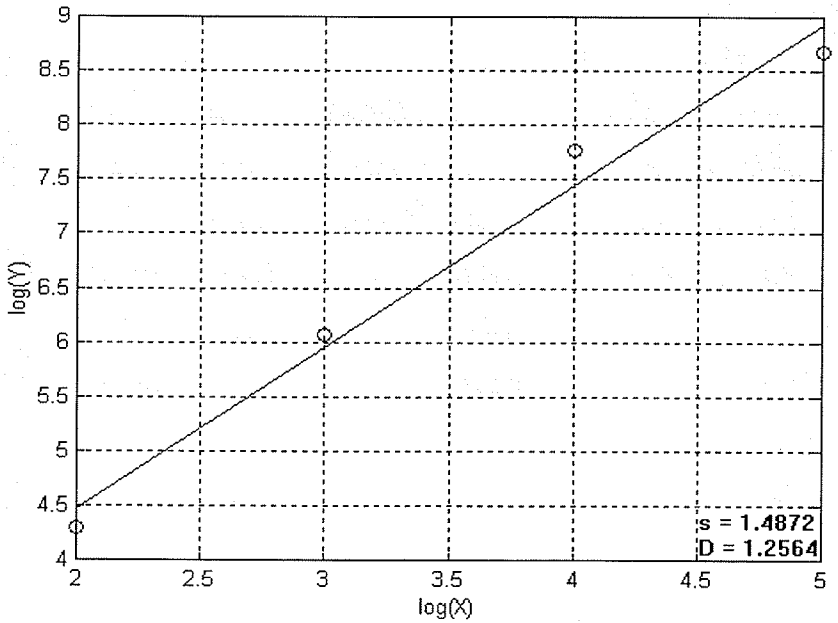

Fig. 5.2. Calculation of variance fractal dimension of Fig. 5.1.

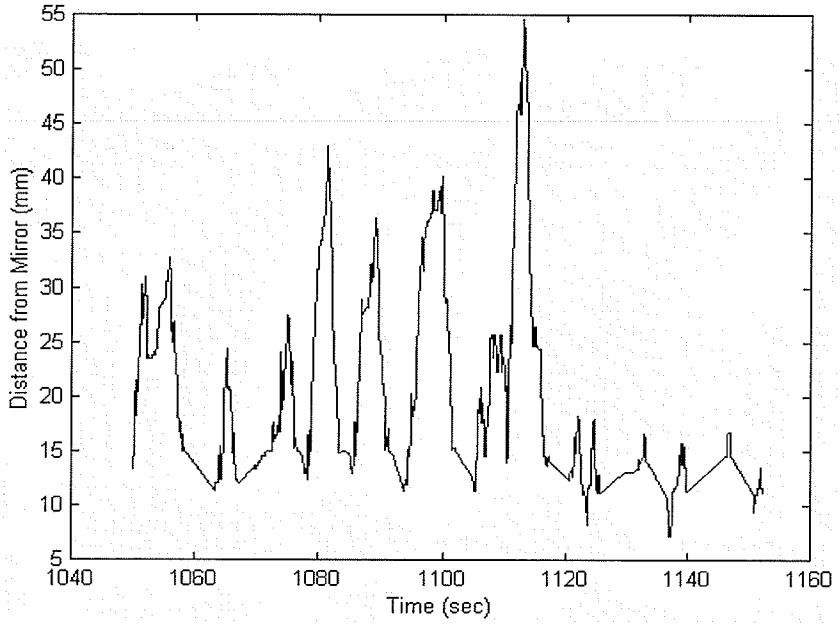

Fig. 5.3. Record 11020219 from $t = 1050.1$ to 1152.4 sec.

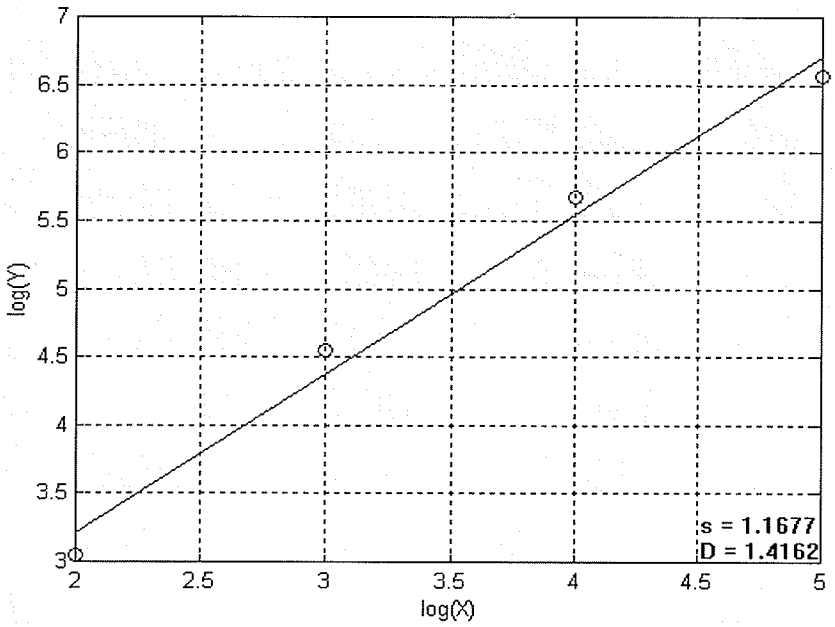

Fig. 5.4. Calculation of variance fractal dimension of Fig. 5.3.

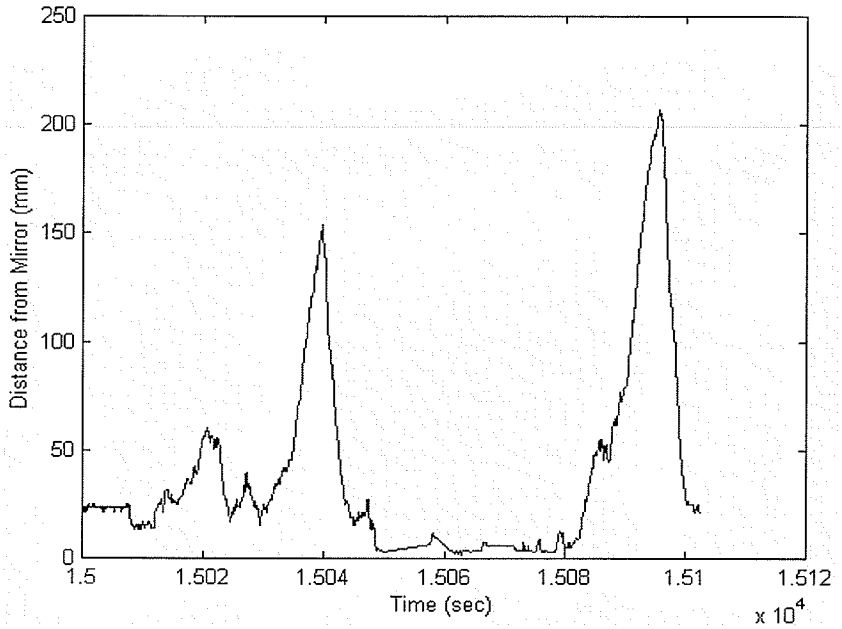

Fig. 5.5. Record 11020219 from $t = 15000.1$ to 15102.4 sec.

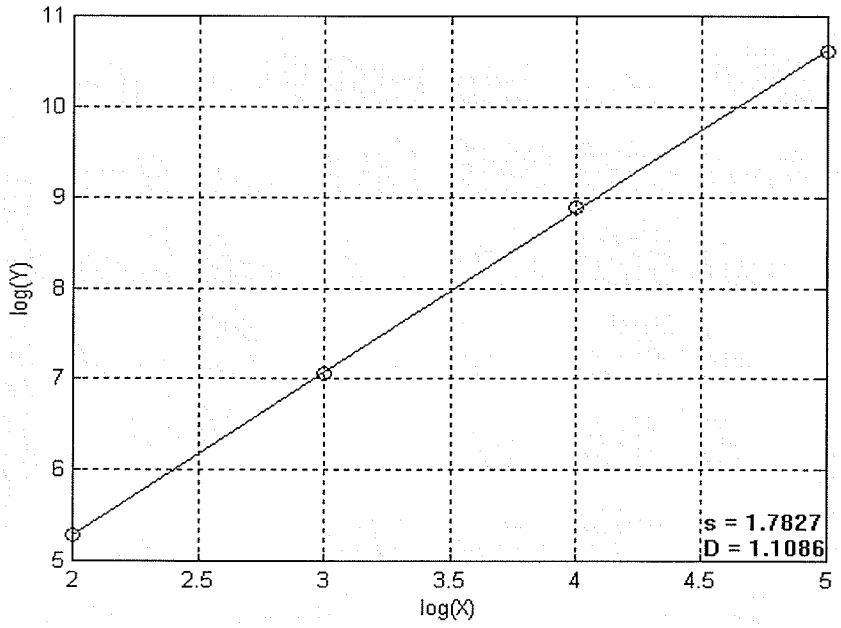

Fig. 5.6. Calculation of variance fractal dimension of Fig. 5.5.

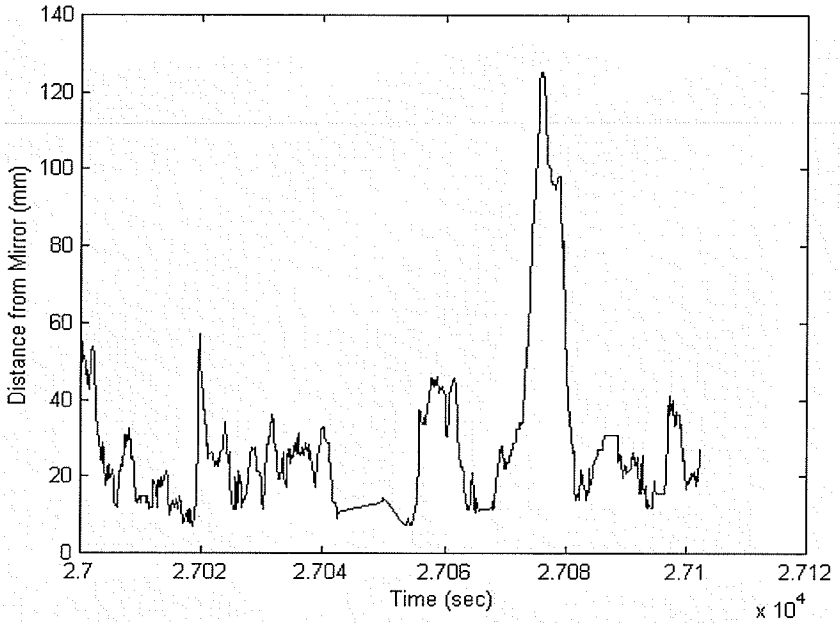

Fig. 5.7. Record 11020219 from $t = 27000.1$ to 27102.4 sec.

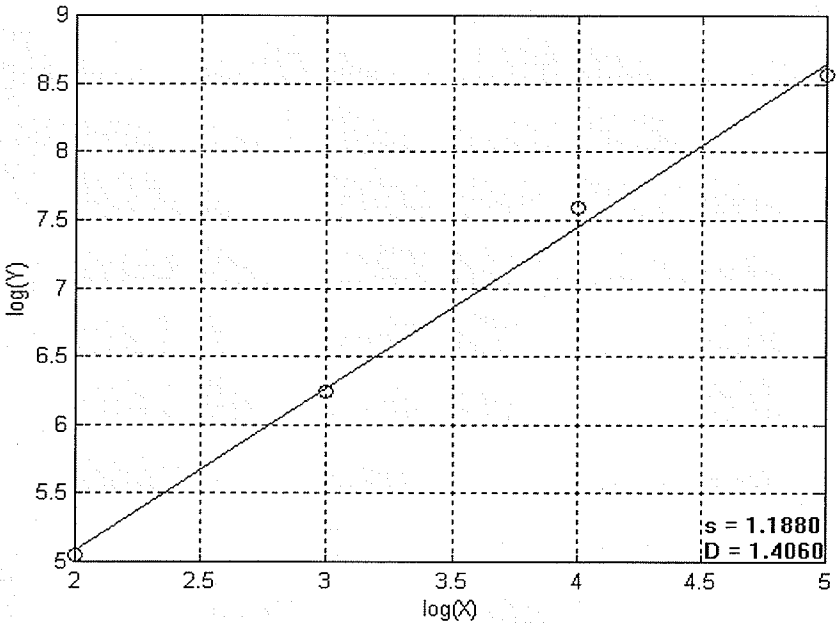

Fig. 5.8. Calculation of variance fractal dimension of Fig. 5.7.

Figures 5.1, 5.3, 5.5, and 5.7 represent the X-coordinates of the *Betta splendens* in Record 11020219 for 102.4 seconds starting at approximately 1, 18, 250, and 450 minutes, respectively. Visually speaking, these plots look very different. However, although these time segments were selected arbitrarily without *a priori* knowledge of what the log-log plots would look like, the plots shown in Figs. 5.2, 5.4, 5.6, and 5.8 reveal an exciting and consistent underlying self-affine structure.

However, a proof that Record 11020219 is self-affine cannot be constructed from four encouraging examples alone. To demonstrate the self-affinity of Record 11020219, we must consider similar log-log plots for the entire time series, and the accuracy of the line of best fit for each plot. The mean square error (MSE) is calculated for each log-log plot as follows

$$MSE = \frac{1}{N}\sum_{i=1}^{N}[Y(i) - \tilde{Y}(i)]^2 \tag{5.8}$$

where N is the number of points in the log-log plot, $Y(i)$ are the actual data points, and $\tilde{Y}(i)$ are the corresponding points on the line of best fit. As in Figs. 5.2, 5.4, 5.6, and 5.8, a window size of 1024 points is used to calculate the VFDT (which means that $N = 4$) with no overlapping between successive windows. The resulting histogram of the MSEs is shown in Fig. 5.9.

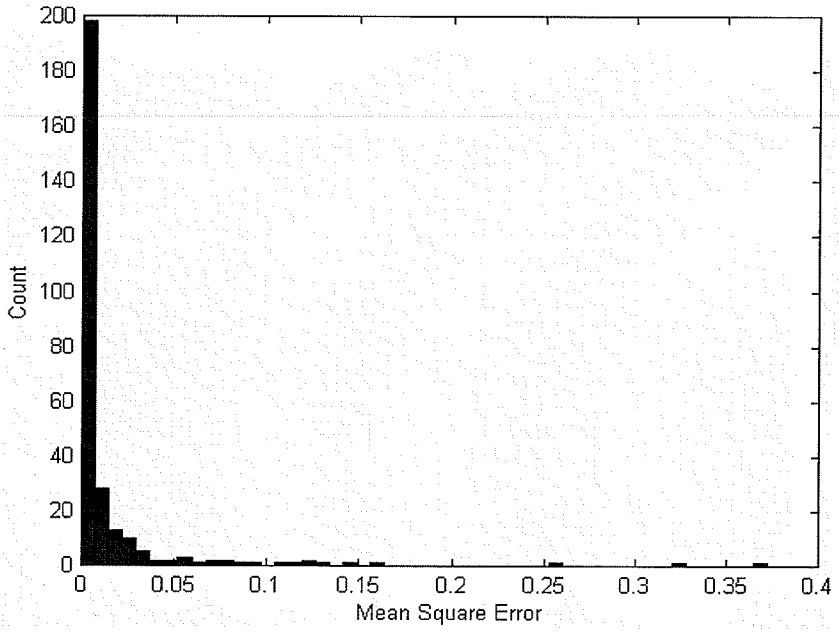

Fig. 5.9. Histogram of the MSE of the VFDT in Record 11020219.

Figure 5.9, consisting of 278 points, shows that the vast majority of the VFDT has a very low MSE; this means that the line of best fit is very close to the actual data points. In fact, the average MSE is 0.0147, and only 10 points have a MSE greater than 0.1. Therefore, since 96.4% of Record 11020219 has a MSE less than 0.1, we may conclude that Record 11202019 is self-affine.

5.3 Verification of Spatial Multifractality

As explained in section 3.4.2, the test to determine if an object is multifractal in space is to calculate its Rényi multifractal dimension spectrum. Figures 5.10 to 5.13 show the Rényi dimension spectra for Figs. 5.1, 5.3, 5.5, and 5.7, which represent four selected segments of 1024 points in Record 11020219.

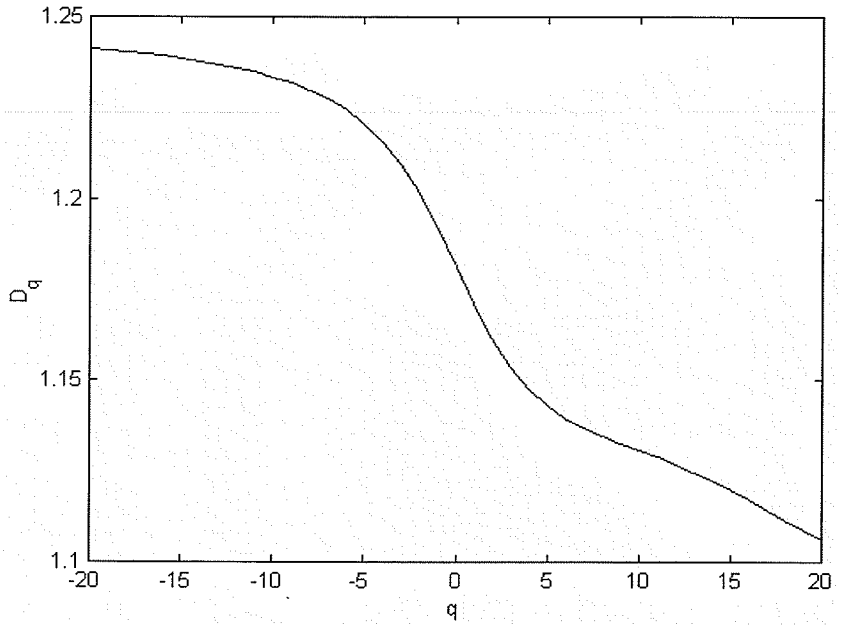

Fig. 5.10. Rényi dimension spectrum of Fig. 5.1.

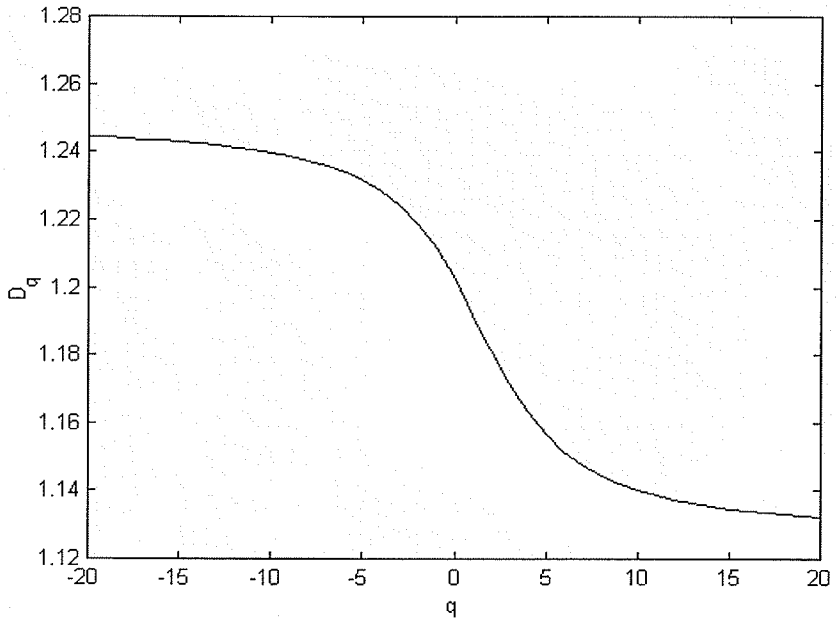

Fig. 5.11. Rényi dimension spectrum of Fig. 5.3.

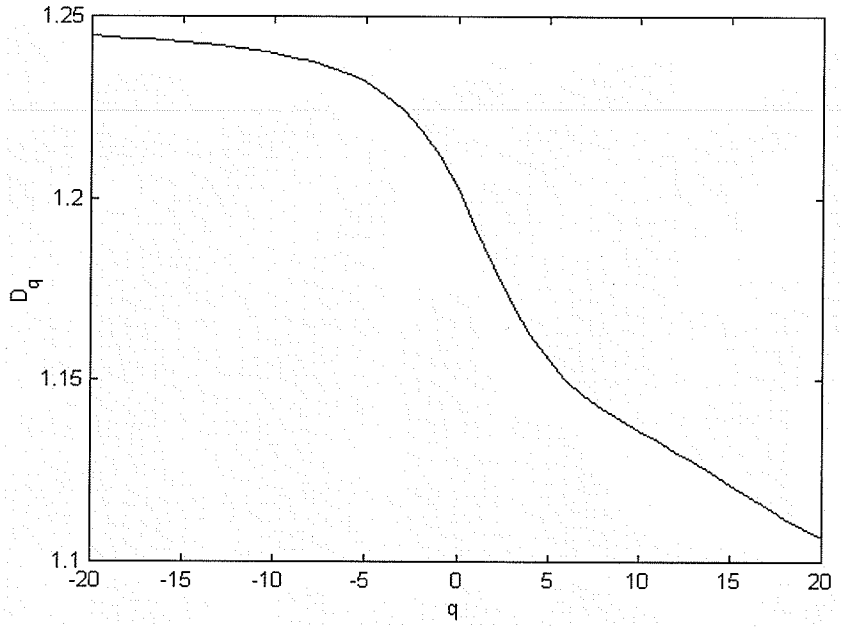

Fig. 5.12. Rényi dimension spectrum of Fig. 5.5.

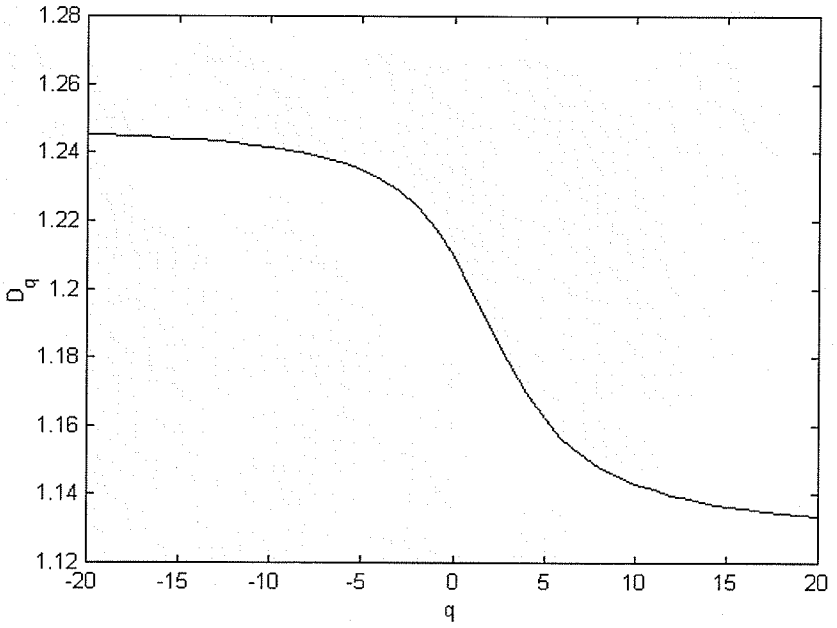

Fig. 5.13. Rényi dimension spectrum of Fig. 5.7.

To calculate the Rényi dimension spectrum of a temporal signal, it has to be embedded into a 2D plane as an image. A mesh is then constructed over the image and the frequency of points is calculated for every bin. If the points are evenly spaced throughout the image, then the probability distribution is uniform, the Rényi dimension spectrum is a horizontal line, and the object is a single fractal. However, if the image has a varying probability distribution, then the spectrum will be curved, and the object is multifractal.

These Rényi dimension spectra all indicate degrees of spatial multifractality in the four segments of Record 11020219, with $1.1 < D_q < 1.25$. Figures 5.10 and 5.12 have similar endpoints when $q = 20$ and $q = -20$, but the curves are different in the middle around $q = 0$. This statement is also true for Figs. 5.11 and 5.13.

The construction of the Rényi dimension spectrum is another method of multifractal characterization for complex temporal signals. In these examples, 1024 points are considered in each successive window of the original signal, and the spectrum constructed for each window contains 41 points between $q = 20$ and $q = -20$. However, the majority of these 41 points may not change very much from one spectrum to another, so a study of the spectra may reveal the fewest number of values for carefully selected q which adequately characterizes the original signal.

The calculation of the Rényi dimension spectrum for 278 sequential windows (which is, in fact, the Rényi dimension spectrum *trajectory*) reveal that the four Rényi dimension spectra displayed in this section are characteristic of the spatial multifractality found throughout Record 11020219.

5.4 Selection of Window Size and Offset to Calculate VFDT

The variance fractal dimension may be calculated as discussed in section 3.3.2. However, careful consideration must be taken when selecting the window size (N_T) and window offset (w_{off}) for the calculation of the variance fractal dimension. The window size determines the number of data points considered in the calculation of the variance fractal dimension. A larger window size results in a greater compression of the data. In an analogous way, this window acts as a low-pass filter (LPF): the larger the window, the more the data is "smoothed" and compressed. However, if the window size is too large, then important high-frequency data may be lost. The window offset may be seen as controlling the resolution of the window, or LPF, through time. A window offset equal to the length of the window means that there is no overlapping between successive windows, and each data point is considered in the calculation of the VFDT only once. As the window offset decreases, data points are considered in the calculation of more than one variance fractal dimension along the trajectory; this process may be referred to as *fractal amplification* [Kins94a]. When $w_{off} = 1$, successive windows overlap as much as possible and maximum resolution of the VFDT occurs through time.

To demonstrate the effect of decreasing the size of non-overlapping windows, Figs. 5.14 to 5.18 show the calculation of the VFDT with window sizes of 8192, 4096, 2048, 1024, and 512, respectively. In our previous log-log plots, a window size of 1024 was used and resulted in $N = 4$ points for the calculation of the slope. 512 points would result in $N = 3$, which is a fundamental lower limit for the number of points needed to reliably calculate the slope of this line, and therefore the variance fractal dimension.

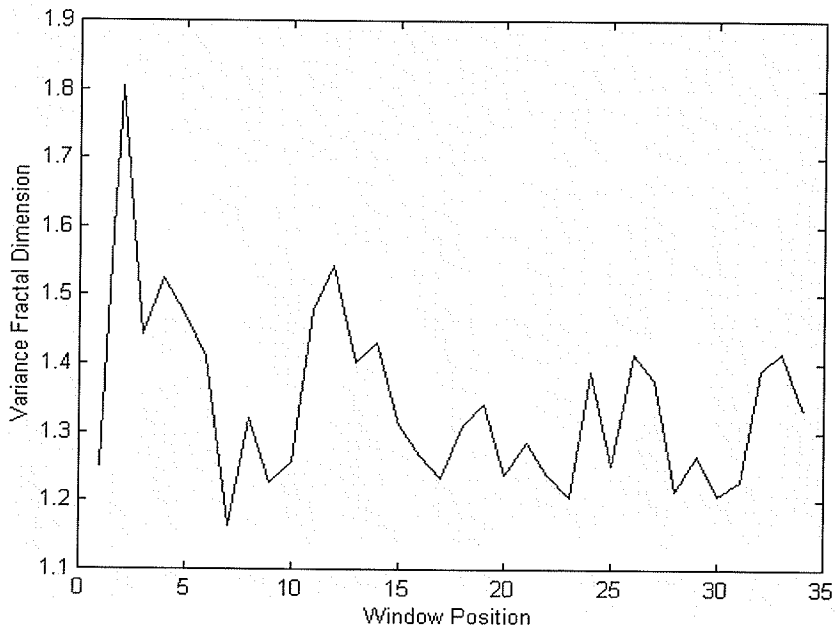

Fig. 5.14. VFDT calculated using a non-overlapping window size of 8192.

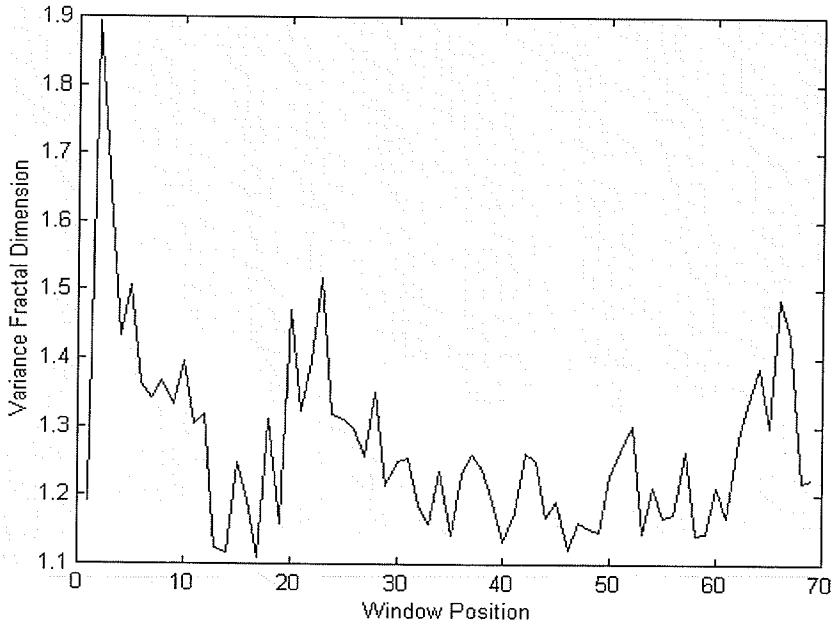

Fig. 5.15. VFDT calculated using a non-overlapping window size of 4096.

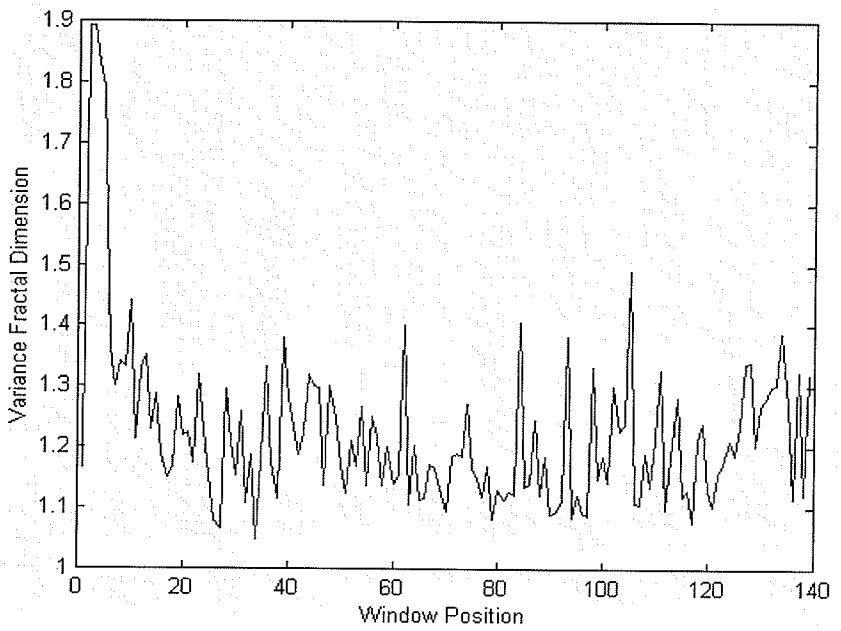

Fig. 5.16. VFDT calculated using a non-overlapping window size of 2048.

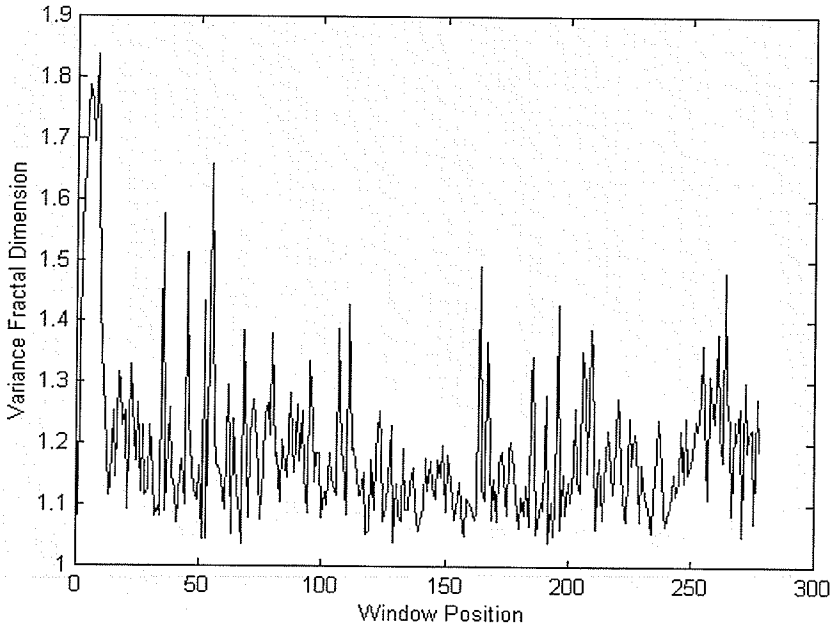

Fig. 5.17. VFDT calculated using a non-overlapping window size of 1024.

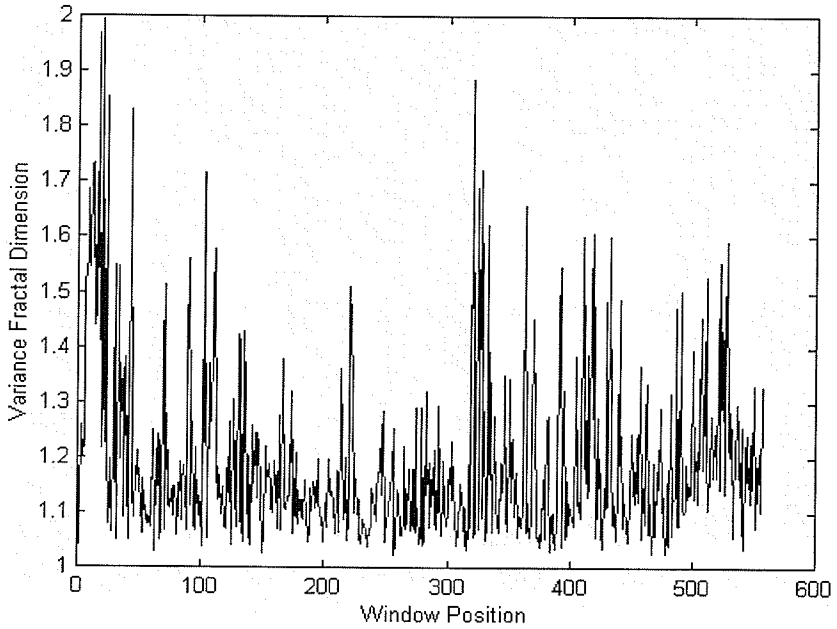

Fig. 5.18. VFDT calculated using a non-overlapping window size of 512.

As the window size decreases from 8192 to 512, the VFDT becomes less smooth and contains more of the subtle variations of the variance fractal dimension. In this way, a larger window size may be seen as acting like a LPF.

Keeping a window size of 512 (as shown in Fig. 5.18), the effects of an overlapping window on the construction of the VFDT will now be illustrated. Figs. 5.19, 5.20, and 5.21 show the VFDT calculated with a window size of 512 and window offset of 256, 128, and 8, respectively. The plots using a window offset of 64, 32, and 16 are not shown here because they look quite similar given the finite resolution on a printed page.

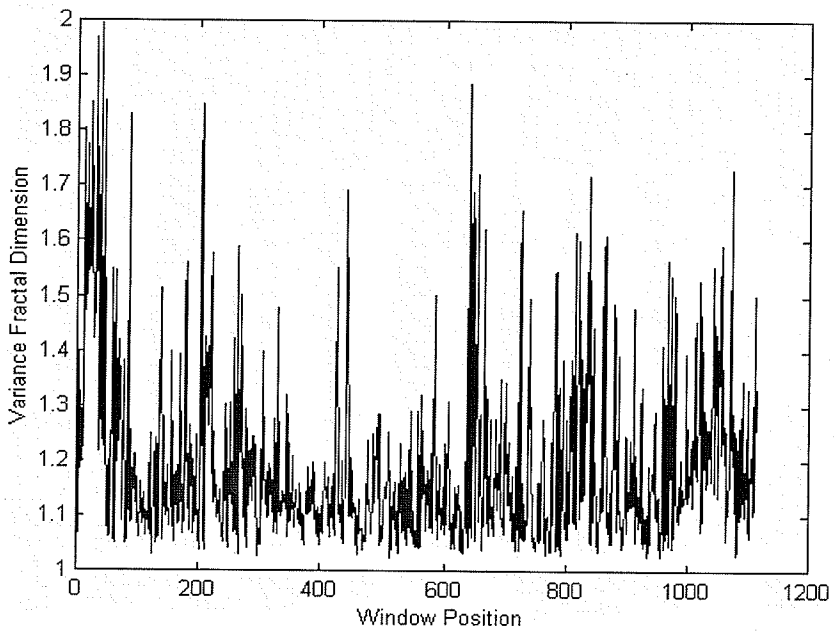

Fig. 5.19. VFDT calculated using a window size of 512 and window offset of 256.

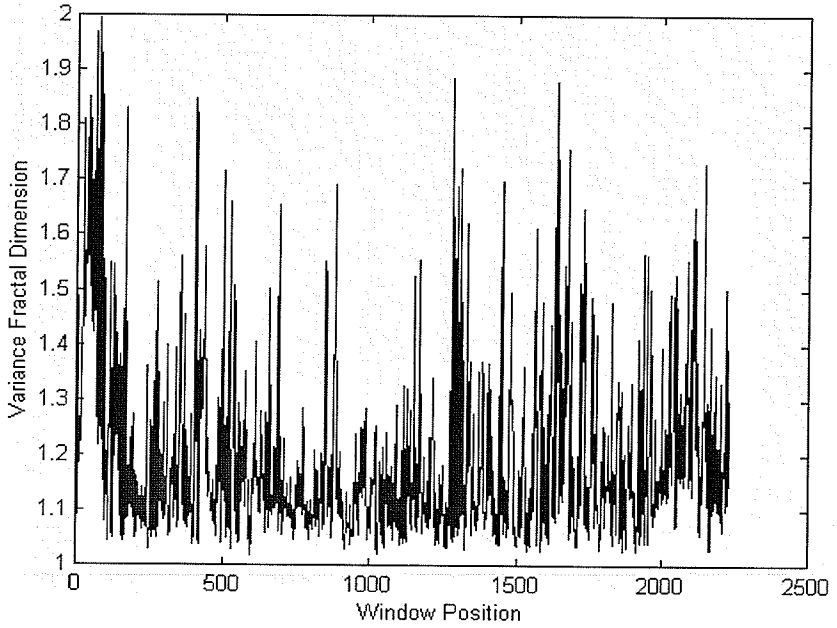

Fig. 5.20. VFDT calculated using a window size of 512 and window offset of 128.

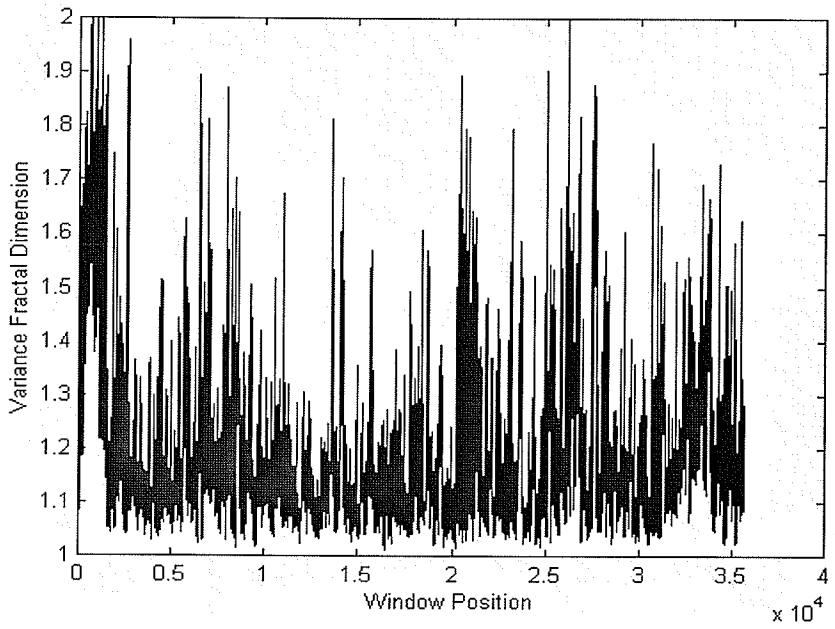

Fig. 5.21. VFDT calculated using a window size of 512 and window offset of 8.

As the window offset decreases, more of each successive window overlaps the previous, and the resolution of the VFDT in time increases.

For this analysis of Pear's data sets and Record 11020219, it was decided that a VFDT with a window length of 512 and window offset of 8 will be used. With a sampling frequency f_s = 10 Hz, this corresponds to a window size of 51.2 sec and a window offset of 0.8 sec. Roughly speaking, the behaviour of the *Betta splendens* is studied for 1 minute intervals every second.

5.5 Construction of Statistical Trajectories and Histograms

To further study the statistics of the VFDT shown in Fig. 5.21, the statistical trajectories – namely the mean, variance, skewness, and kurtosis trajectories – of the VFDT are calculated using Eqs. 4.1 to 4.4, respectively. These four new trajectories are calculated using a window size of 512 with a window offset of 1. The smallest possible window offset was chosen so the resulting trajectories would have the same number of points as the original VFDT. Figs. 5.22 to 5.25 show the mean, variance, skewness, and kurtosis trajectories of the VFDT.

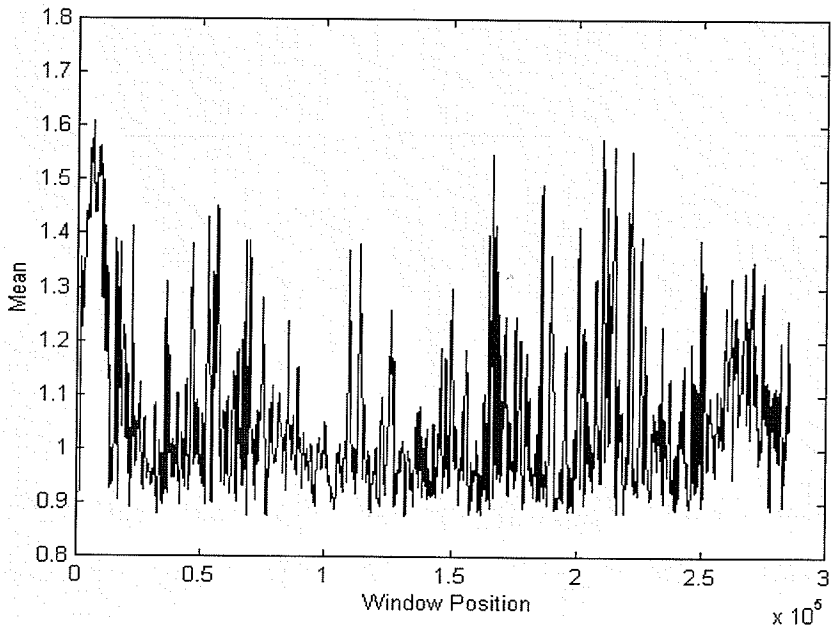

Fig. 5.22. Mean trajectory of the VFDT.

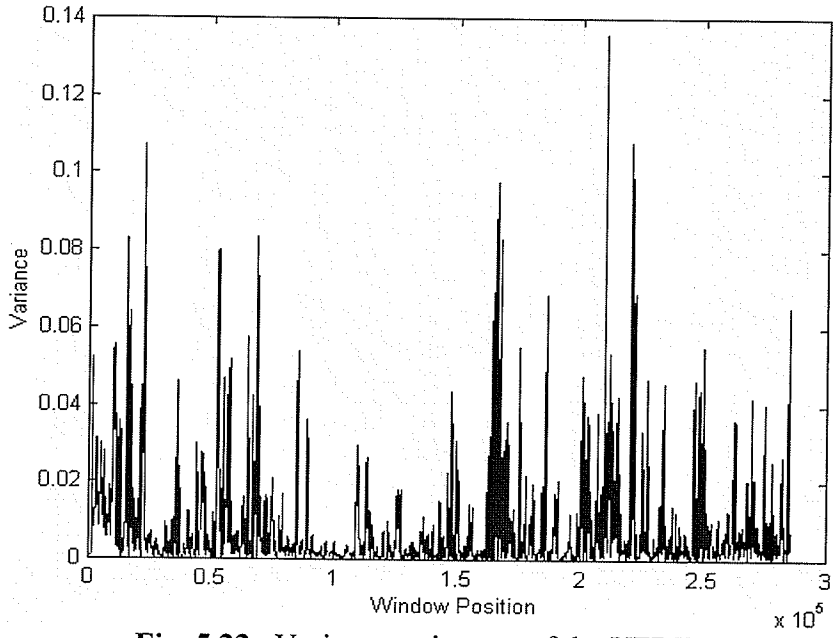

Fig. 5.23. Variance trajectory of the VFDT.

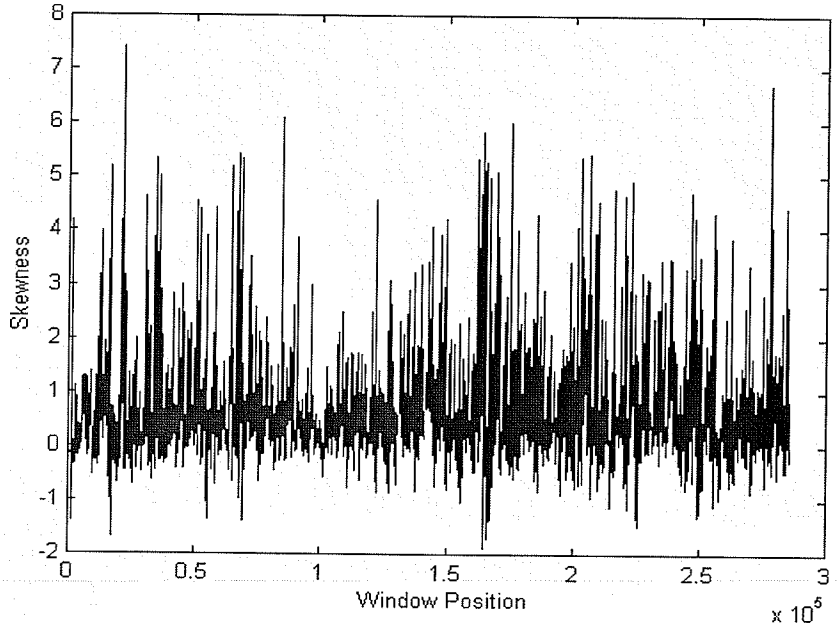

Fig. 5.24. Skewness trajectory of the VFDT.

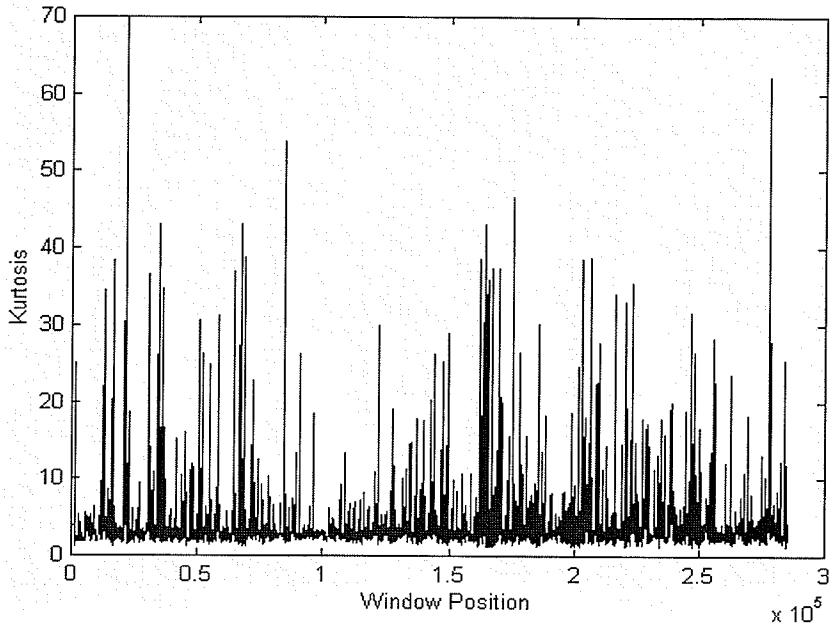

Fig. 5.25. Kurtosis trajectory of the VFDT.

The *global* histograms of Figs. 5.22 to 5.25 are shown in Figs. 5.26, 5.28, 5.30, and 5.32, respectively. These histograms represent the distribution of the mean, variance, skewness, and kurtosis for the entire corresponding trajectory.

Visually speaking, the structure of the global histogram in Fig. 5.26 resembles a log-normal distribution, Fig. 5.30 resembles a skewed Gaussian distribution, and Figs. 5.28 and 5.32 resemble exponential distributions. These histograms strongly reinforce the existence of an underlying behaviour in the generation of the data. The log-normal, Gaussian, and exponential distributions may be used to accurately model the four histograms, but none of these distributions would succeed in modelling all four histograms. For this reason, the gamma distribution was chosen to model the histograms, as shown in Figs. 5.27, 5.29, 5.31 and 5.33, respectively.

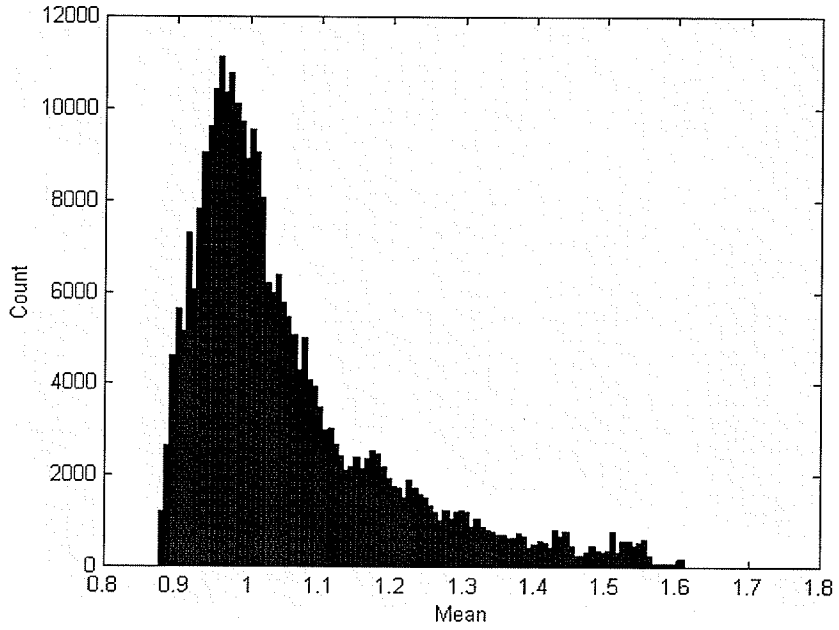

Fig. 5.26. Global histogram of the mean trajectory of the VFDT.

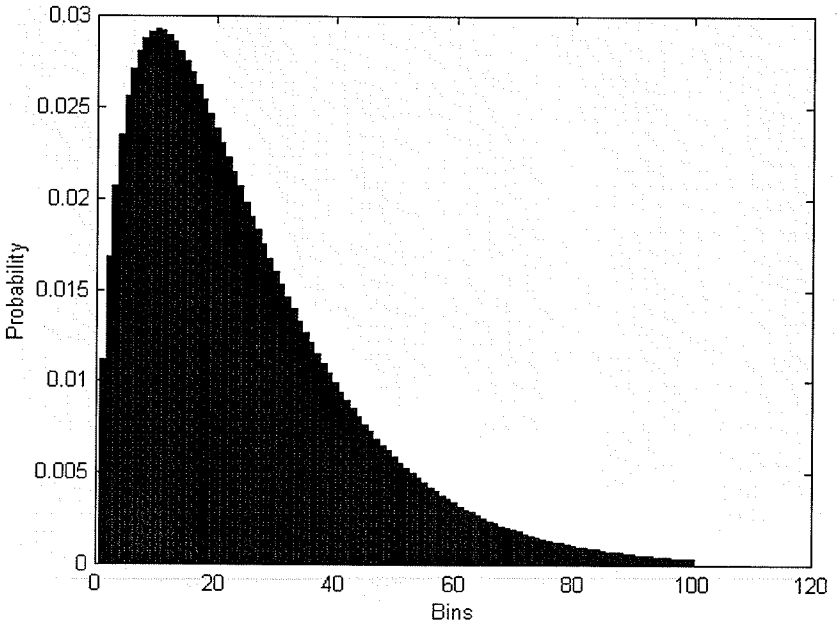

Fig. 5.27. Gamma distribution model of Fig. 5.26 with $\alpha = 1.6816$ and $\beta = 14.804$.

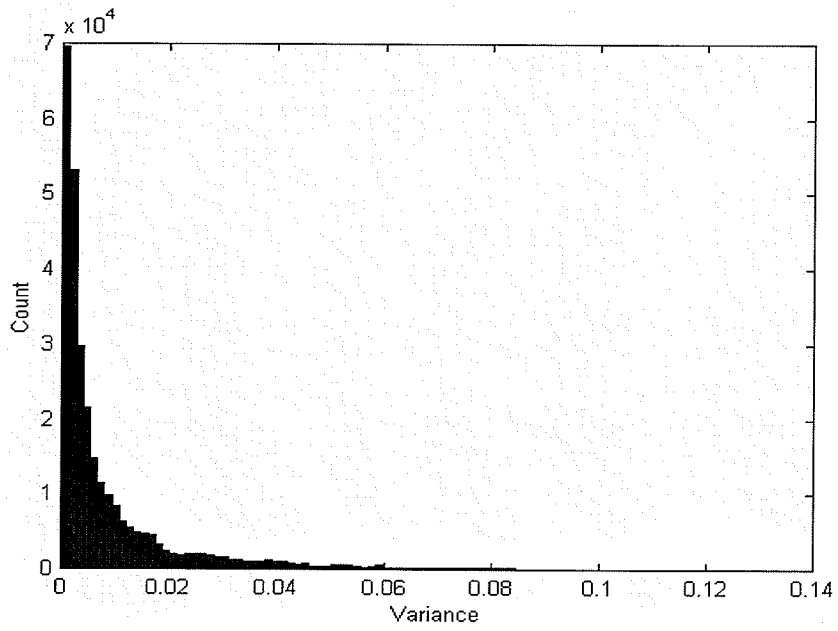

Fig. 5.28. Global histogram of the variance trajectory of the VFDT.

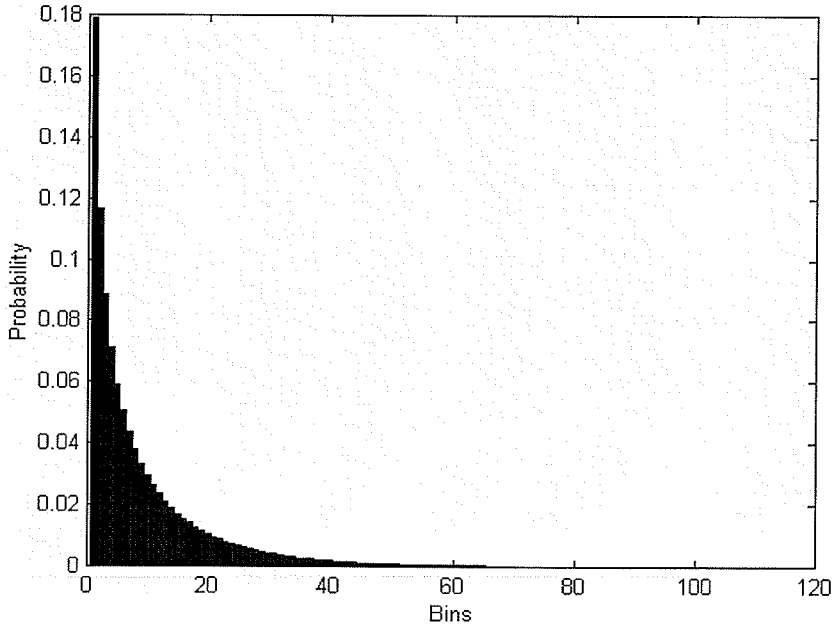

Fig. 5.29. Gamma distribution model of Fig. 5.28 with $\alpha = 0.48026$ and $\beta = 14.885$.

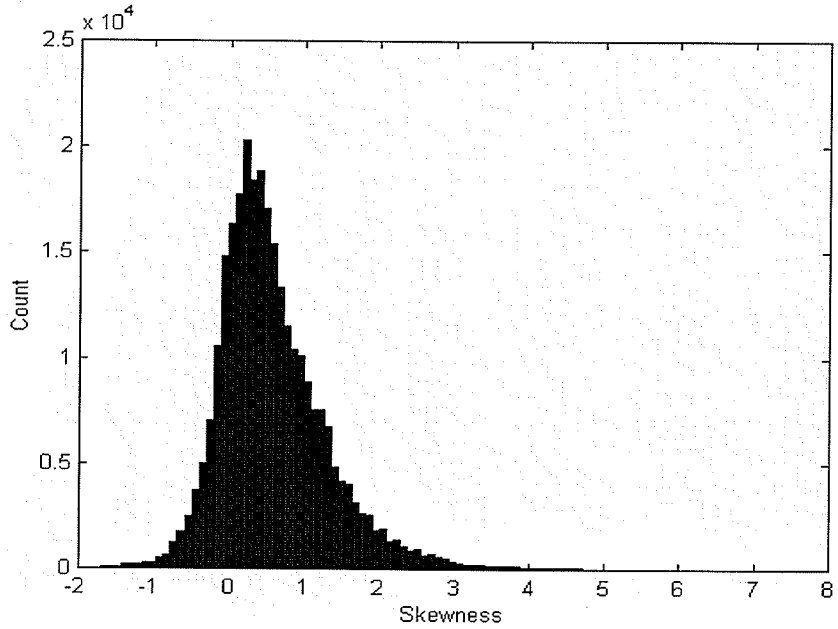

Fig. 5.30. Global histogram of the skewness trajectory of the VFDT.

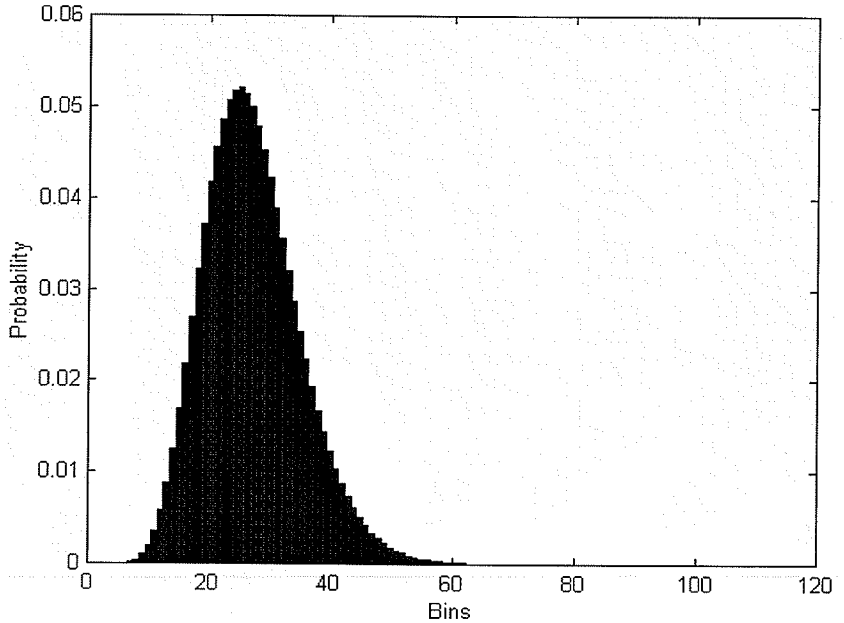

Fig. 5.31. Gamma distribution model of Fig. 5.30 with $\alpha = 11.575$ and $\beta = 2.3385$.

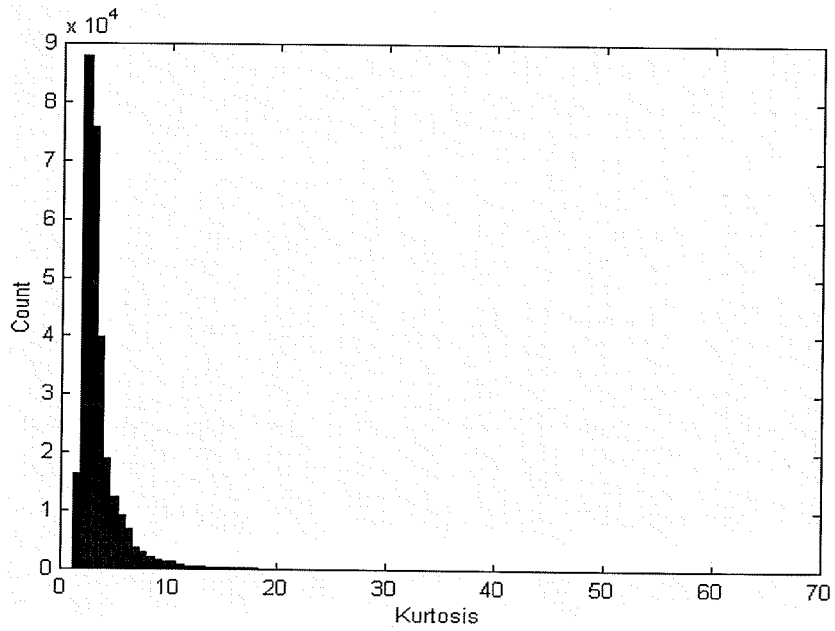

Fig. 5.32. Global histogram of the kurtosis trajectory of the VFDT.

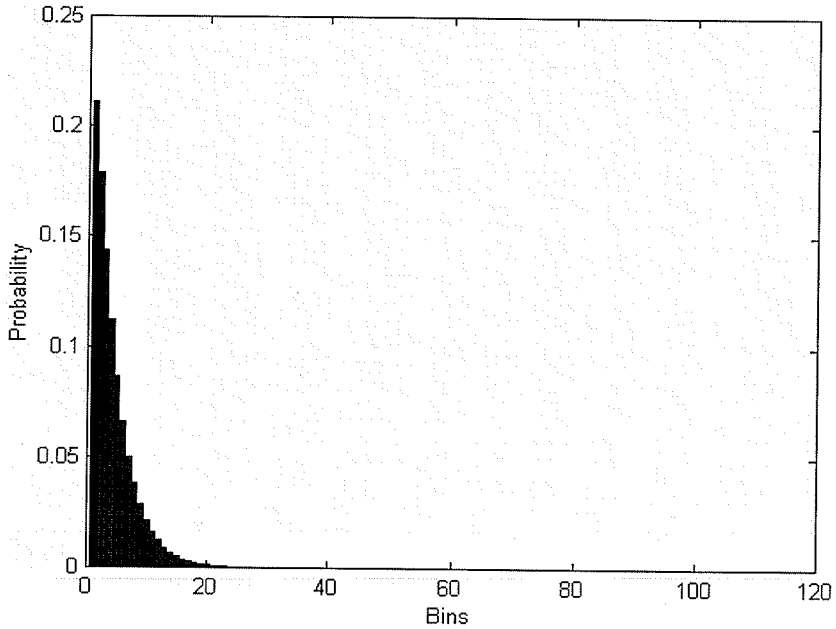

Fig. 5.33. Gamma distribution model of Fig. 5.32 with $\alpha = 1.2060$ and $\beta = 3.2648$.

Given the gamma distribution in Eq. 4.7, the mean and variance are [Weis99a]

$$\mu = \alpha\beta \tag{5.9}$$

and

$$\sigma^2 = \alpha\beta^2 \tag{5.10}$$

Since μ and σ^2 are calculated from the data, we can re-arrange Eqs. 5.9 and 5.10 to determine the appropriate α and β to model the histogram with the gamma distribution. These expressions are

$$\beta = \frac{\sigma^2}{\mu} \tag{5.11}$$

and

$$\alpha = \frac{\mu^2}{\sigma^2} = \frac{\mu}{\beta} \tag{5.12}$$

To study the changes in the behaviour of the *Betta splendens*, we must consider the stationarity of the *local* histograms formed from the statistical trajectories within certain ranges of time. Clearly, there is some lower bound for the number of points needed to construct a histogram so that it may be accurately modelled. After visually inspecting the local statistical histograms for many different regions of time, it was decided that 9000 points, or 15 minutes, of data were sufficient for constructing a reasonable histogram. Figs. 5.34, 5.36, 5.38, and 5.40 show the local statistical histograms for a period of 15 minutes starting at time t = 20,000 sec (or 5.56 hrs). Figs. 5.35, 5.37, 5.39, and 5.41, respectively, show the gamma distribution models for these local statistical histograms.

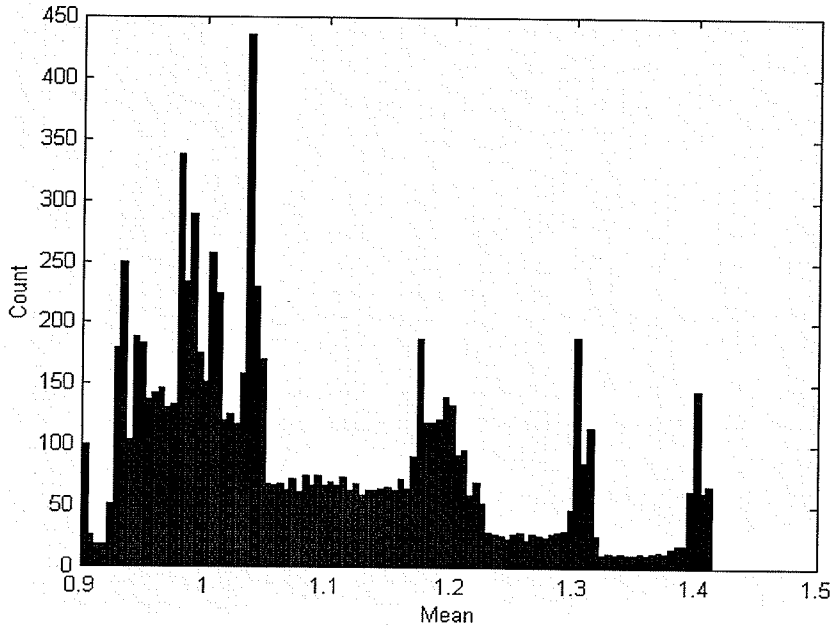

Fig. 5.34. Local mean trajectory histogram between time $t = 20{,}000$ and 29,000 sec.

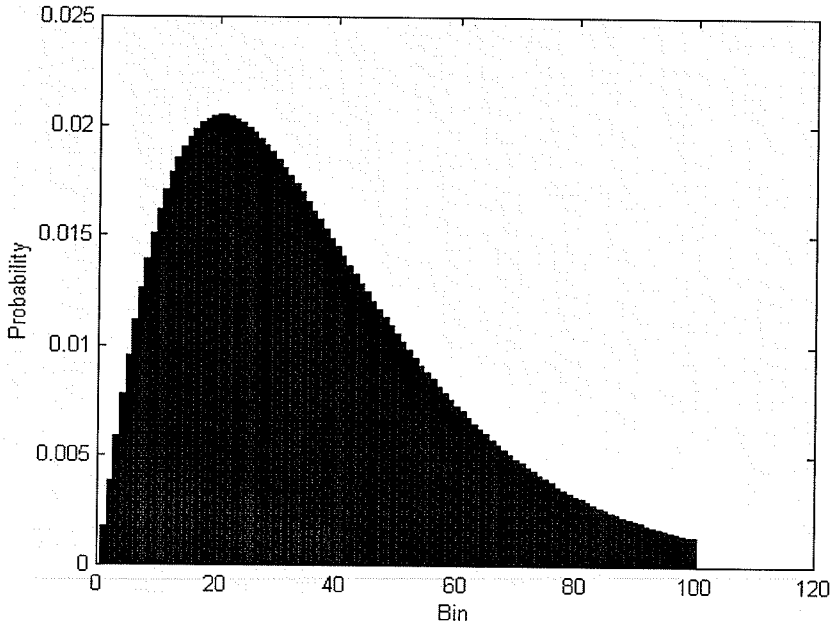

Fig. 5.35. Gamma distribution model of Fig. 5.34 with $\alpha = 2.1853$ and $\beta = 17.113$.

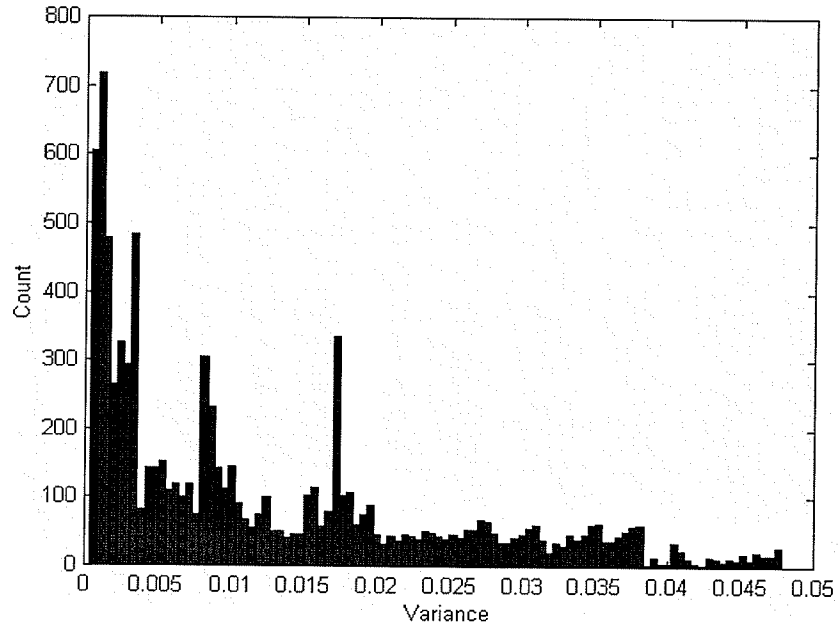

Fig. 5.36. Local variance trajectory histogram between time $t = 20{,}000$ and 29,000 sec.

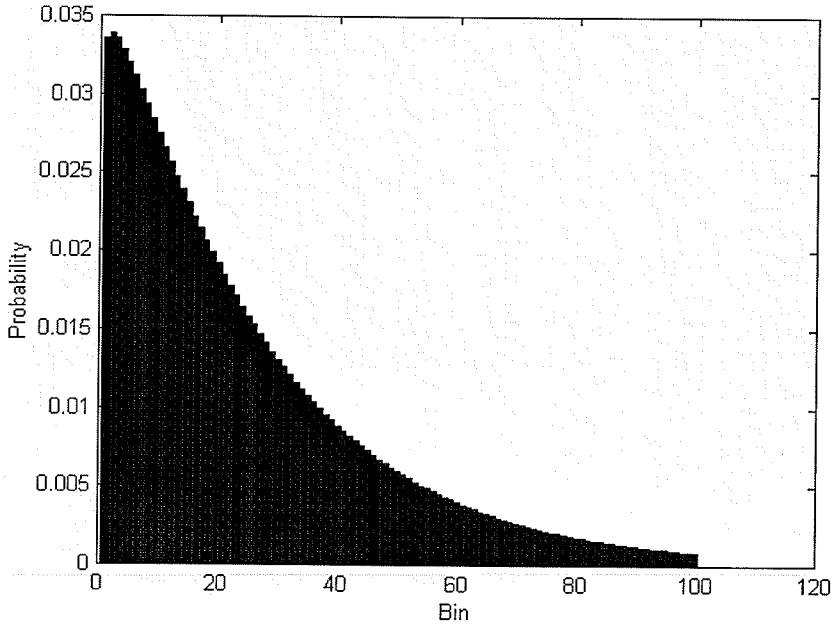

Fig. 5.37. Gamma distribution model of Fig. 5.36 with $\alpha = 1.0744$ and $\beta = 24.249$.

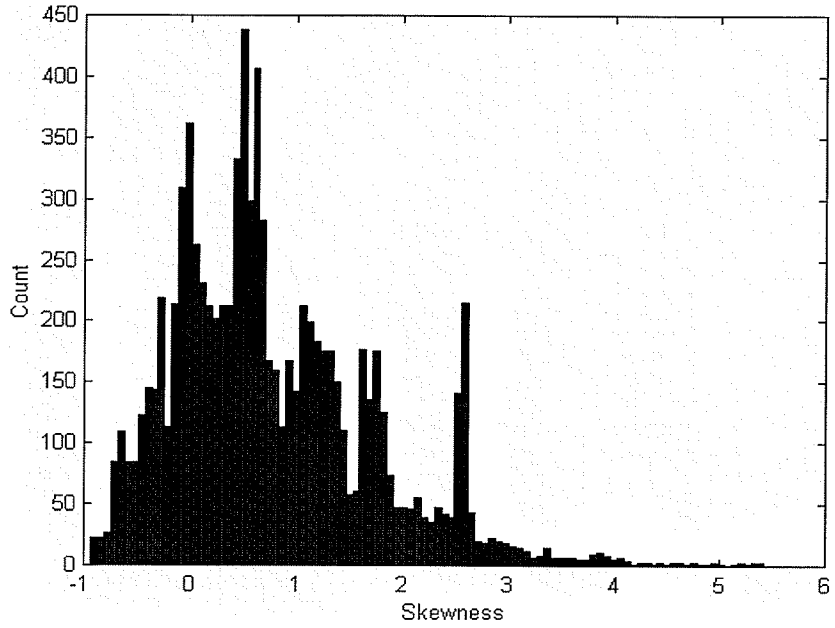

Fig. 5.38. Local skewness trajectory histogram between time $t = 20{,}000$ and 29,000 sec.

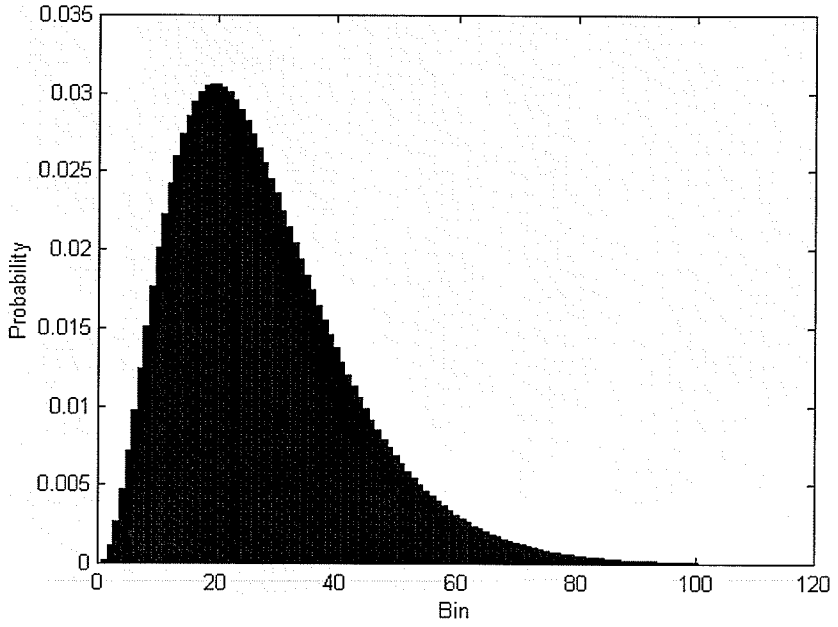

Fig. 5.39. Gamma distribution model of Fig. 5.38 with $\alpha = 3.3753$ and $\beta = 8.1635$.

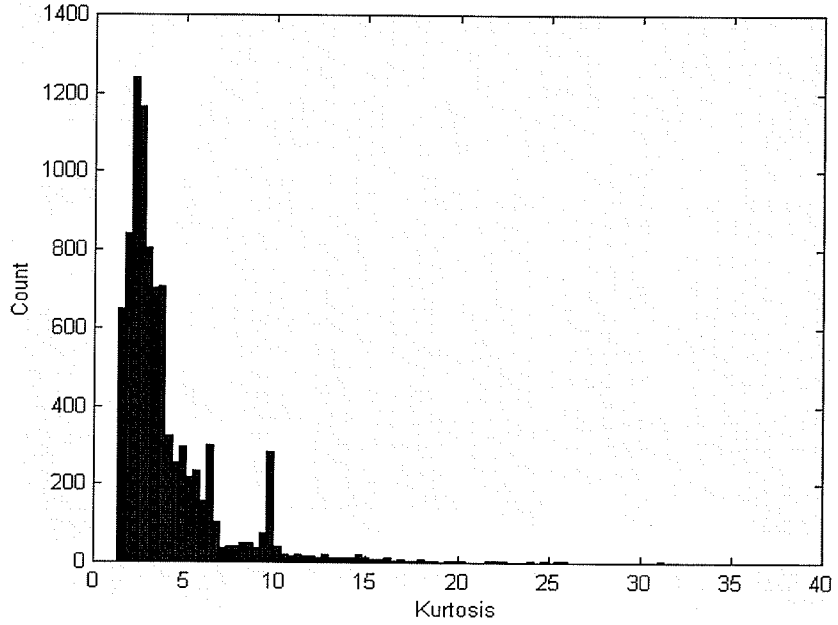

Fig. 5.40. Local kurtosis trajectory histogram between time $t = 20{,}000$ and 29,000 sec.

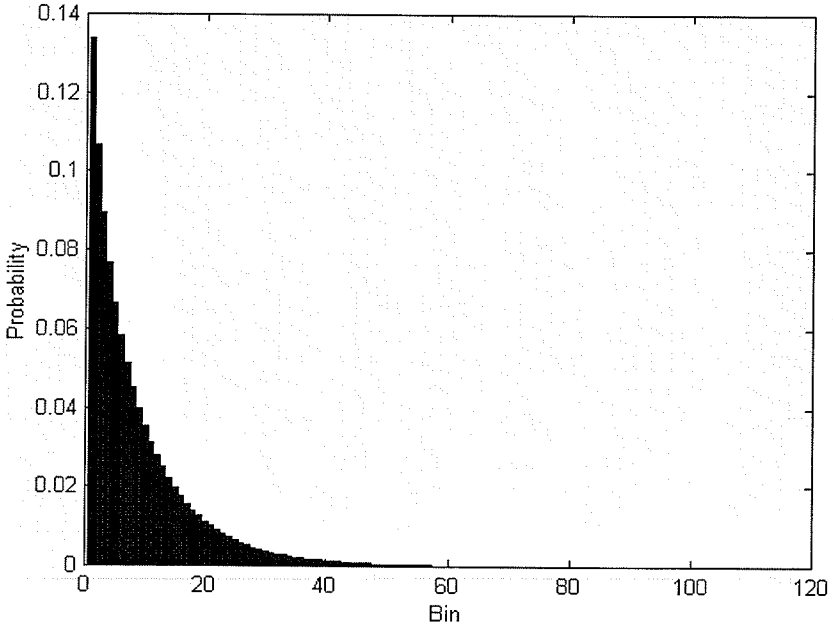

Fig. 5.41. Gamma distribution model of Fig. 5.40 with $\alpha = 0.81900$ and $\beta = 9.8329$.

5.6 Dimensionality Reduction using PCA

A sudden change in the behaviour of the VFDT would be reflected in its statistical trajectories, so a significant degree of correlation should be expected between the mean, variance, skewness, and kurtosis trajectories of the VFDT. Figures 5.42 to 5.47 show the relationships between the mean, variance, skewness, and kurtosis trajectories. Indeed, this correlation does exist between all four trajectories to varying degrees.

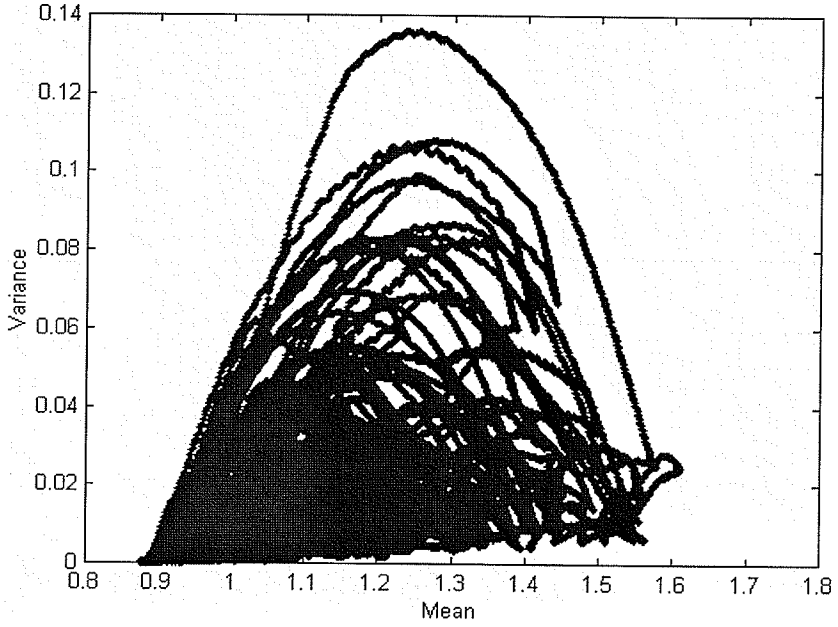

Fig. 5.42. Relationship between mean and variance trajectories.

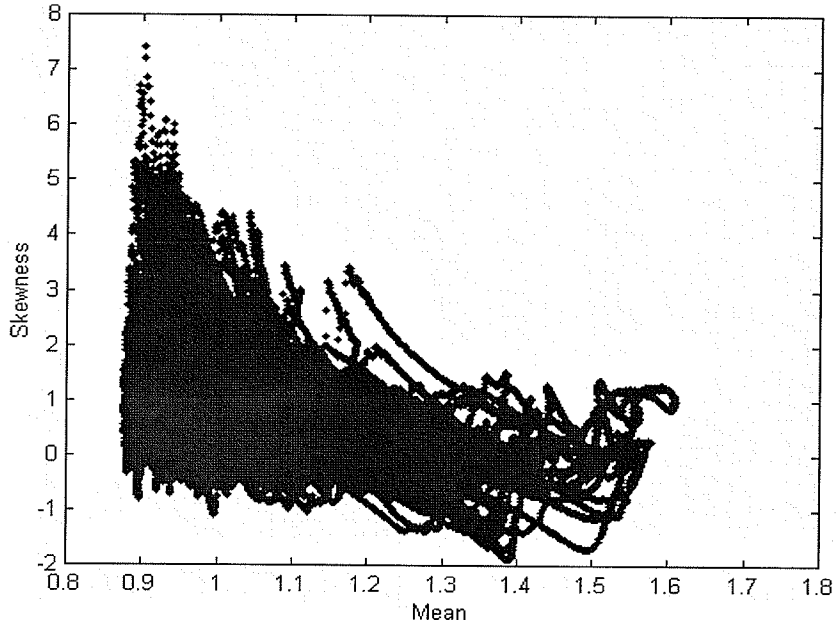

Fig. 5.43. Relationship between mean and skewness trajectories.

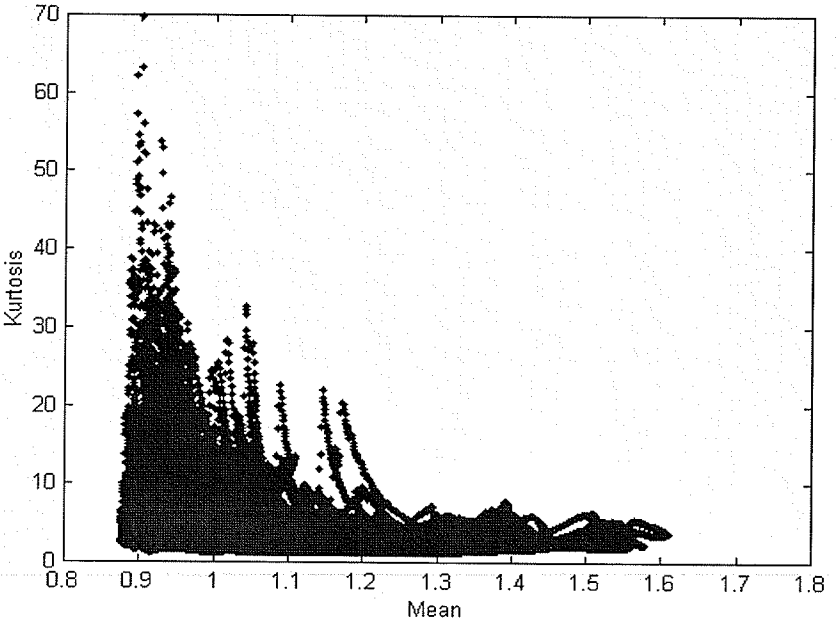

Fig. 5.44. Relationship between mean and kurtosis trajectories.

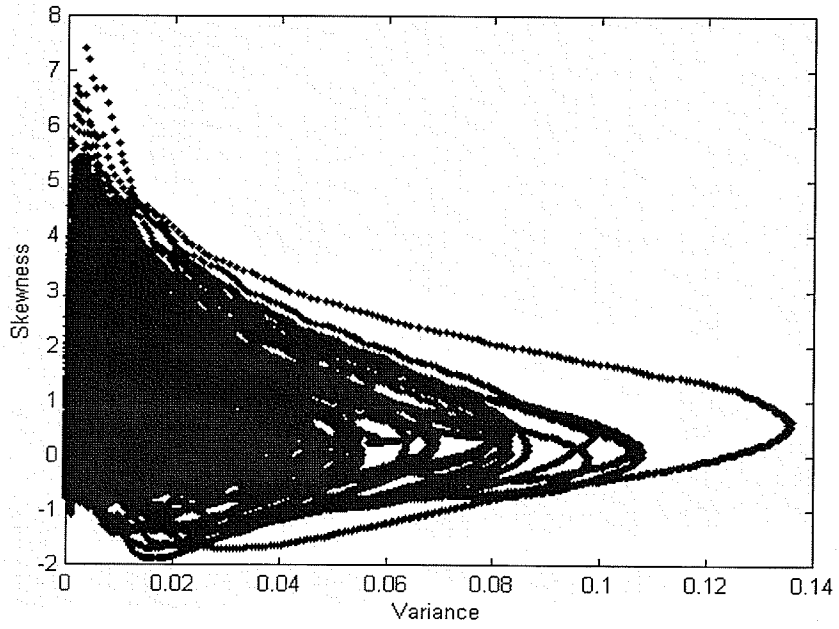

Fig. 5.45. Relationship between variance and skewness trajectories.

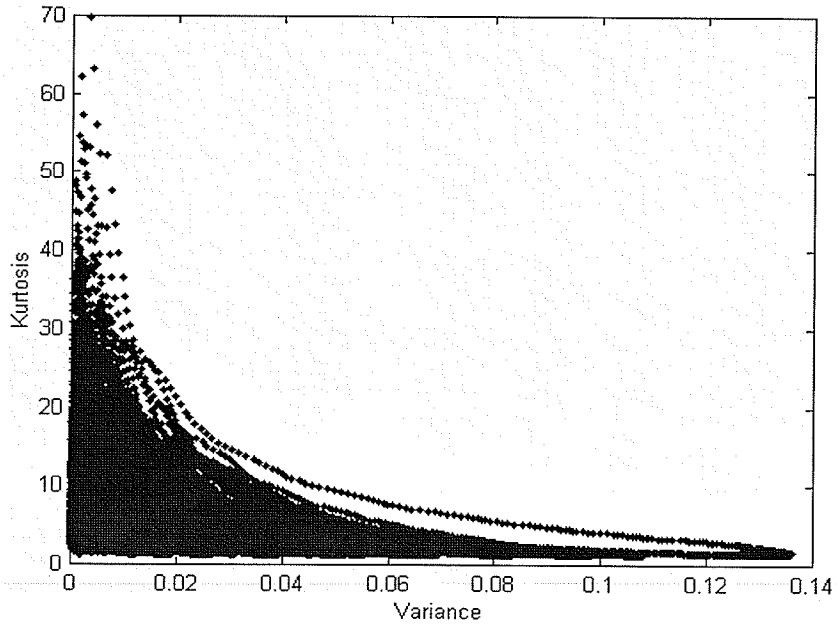

Fig. 5.46. Relationship between variance and kurtosis trajectories.

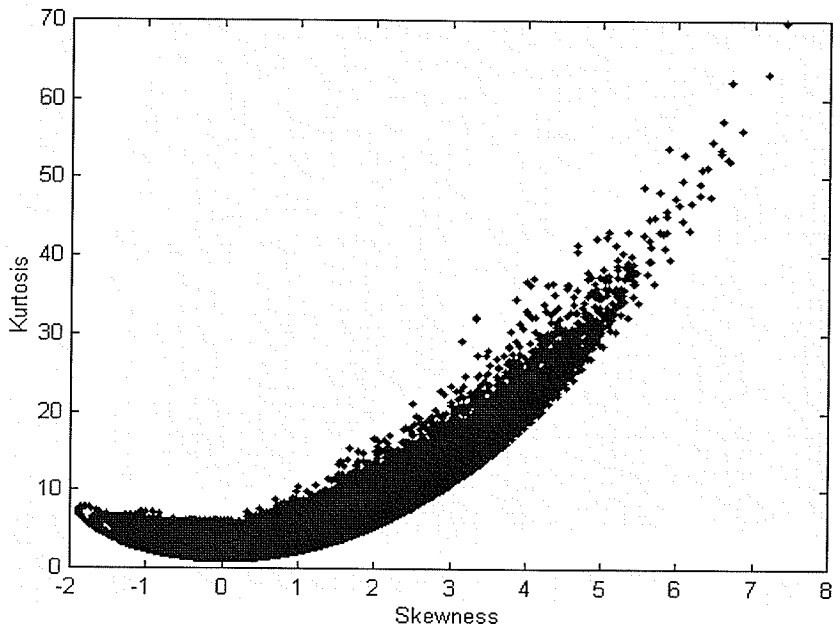

Fig. 5.47. Relationship between skewness and kurtosis trajectories.

Some general statements may be made about the relationships between the four statistical trajectories shown in these six figures:

- Figures 5.42 and 5.45 show the plots between the mean and variance, and the variance and skewness, to be the least correlated. Firstly, the variance can take on a wide range of values when the mean is between 1.1 and 1.5, the variance is below about 0.08 when the mean is less than 1.1 or greater than 1.5. Secondly, the skewness is greater than 3 when the variance is greater than 0.04.

- Figures 5.43 and 5.44 show that as the mean increases, the skewness and kurtosis both tend to decrease. When the mean is greater than 1.3, the skewness is less than 2 and the kurtosis is less than 10.

- Figure 5.46 shows a steep decrease in the kurtosis as the variance increases. When the variance is greater than 0.06, the kurtosis is less than 10.

- Figure 5.47 reveals a highly correlated and almost linear relationship between the skewness and kurtosis. The kurtosis is less than 10 when the skewness is between -2 and 1, and increases approximately linearly from 10 to 70 as the skewness increases from 1 to 7.

Once the histograms of these four statistical trajectories are constructed and modelled, there are eight parameters (an α and a β for each of the four histograms) which characterize the 9000 consecutive points in the VFDT. These eight parameters may be concatenated into a vector format to form an 8-dimensional (8D) signature. Since the four statistical trajectories have been shown to be correlated, it intuitively follows that the αs and βs should also be correlated because they model the histograms of these correlated statistical trajectories.

Figure 5.48 shows an interesting plot of the values for all of the 8D signatures which characterize Record 11020219. $T_8(i)$ is the concatenated vector where:

- $T_8(1,2) = (\alpha_1, \beta_1)$ models the histogram of the mean trajectory,
- $T_8(3,4) = (\alpha_2, \beta_2)$ models the histogram of the variance trajectory,
- $T_8(5,6) = (\alpha_3, \beta_3)$ models the histogram of the skewness trajectory, and
- $T_8(7,8) = (\alpha_4, \beta_4)$ models the histogram of the kurtosis trajectory.

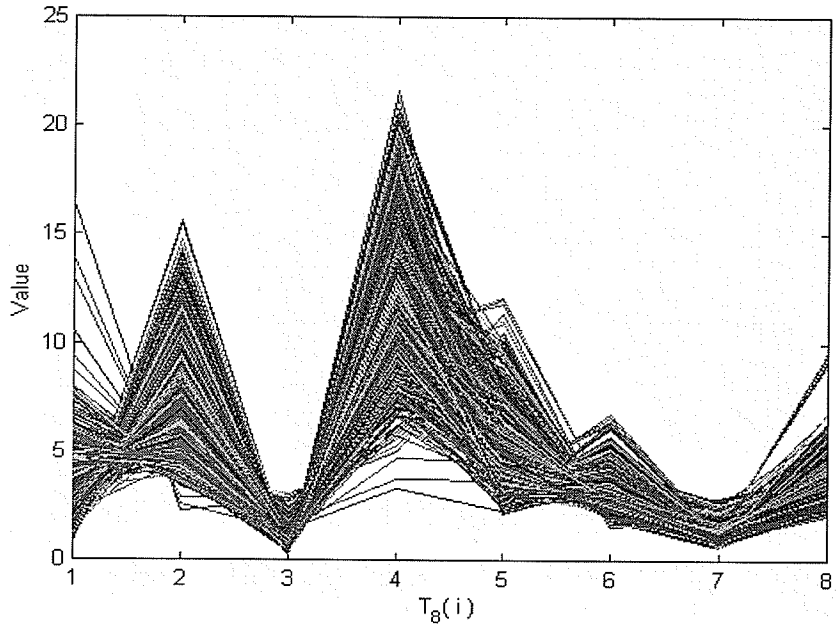

Fig. 5.48. Values of all 8D signatures which characterize Record 11020219.

Viewing all of the signatures simultaneously highlights the global behaviour of the 8D signatures. It is interesting to notice that the variability between parameters is not the same. For example, $T_8(3) = \alpha_2$ and $T_8(7) = \alpha_4$ have lowest variability across all signatures with values ranging between 0.21 and 3.1, and $T_8(4) = \beta_2$ has the highest variability with values ranging between 3.3 and 22.

As discussed in section 4.4, PCA may be used to remove the correlation between the parameters in the 8D signature. Once PCA has been performed, the M largest eigenvalues, and their corresponding eigenvectors, contain the majority of the variance between the parameters, and may be kept to form a further compressed M-dimensional signature of the traffic. A plot of the cumulative variance for an increasing number of principal components for the 8D signatures is shown in Fig. 5.49.

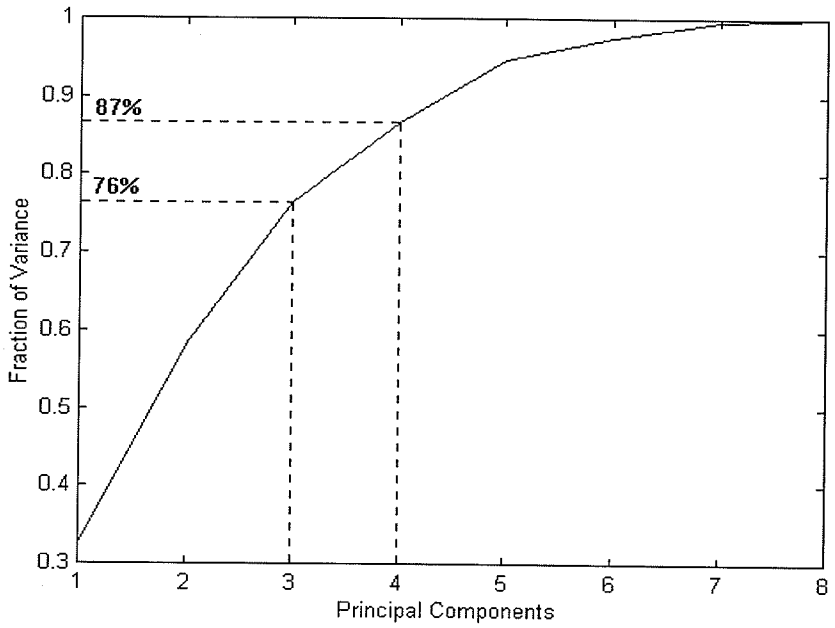

Fig. 5.49. Cumulative variance for the principal components.

The increase of the cumulative variance gets progressively smaller with the addition of each successive principal component. Approximately 76% and 87% of the variance is contained within the first three and four principal components, respectively. If at least 80% of the variance must be preserved after the dimensionality reduction, then the first four principal components must be kept to form a compressed 4-dimensional (4D) signature of the traffic. A comprehensive overview of the construction of this 4D signature is shown in the section 5.7.

Figure 5.50 illustrate the removal of the correlation after PCA is performed on the 8D signatures shown in Fig. 5.48. As expected, the majority of the variability is in the first few principal components. Figure 5.51 shows the first four principal components which will be used as the final compressed 4D signature for the traffic.

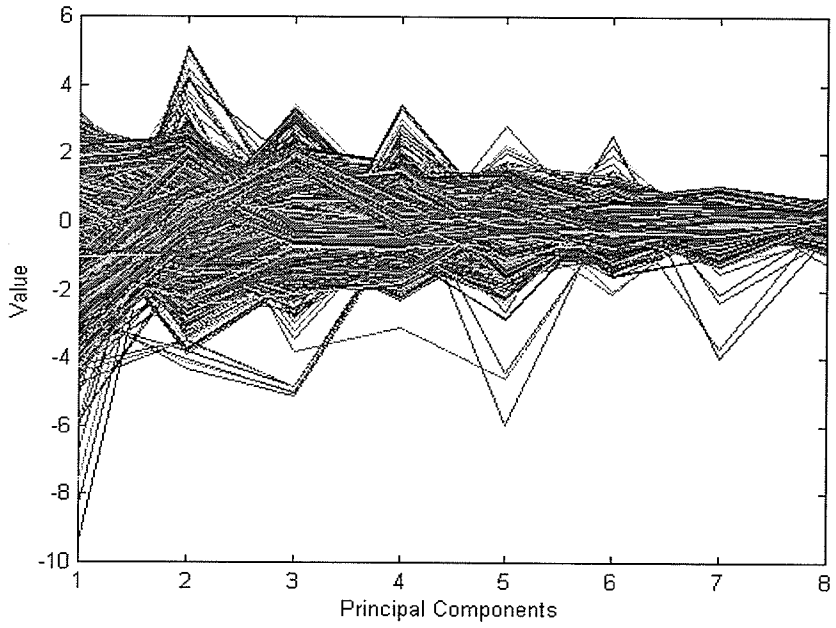

Fig. 5.50. Values of all 8D signatures in Fig. 5.48 after PCA.

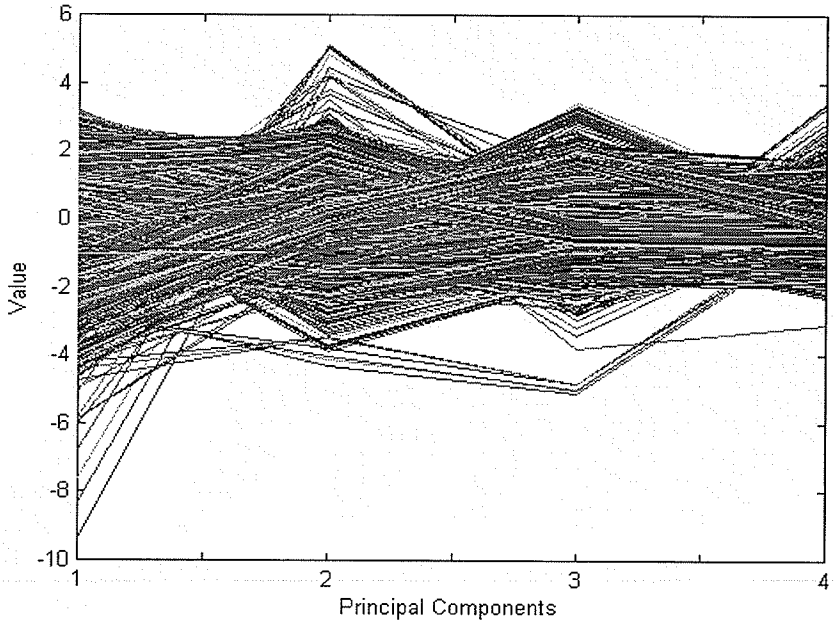

Fig. 5.51. Values of all compressed 4D signatures.

5.7 Construction of 4D Traffic Signature Trajectory

The 4D traffic signature, or "fingerprint," of a window of traffic is constructed through five steps, as illustrated by Fig. 5.52.

Step 1:

- Calculate the VFDT of the self-affine traffic using a window size of 512 and a window offset of 8.

Step 2:

- Calculate the mean, variance, skewness, and kurtosis trajectories of the VFDT using a window size of 512 and a window offset of 1.

Step 3:

- Construct histograms of the mean, variance, skewness, and kurtosis trajectories using a window size of 9000 (and a window offset of 150 when iterating Step 3 to Step 5 to create a 4D signature trajectory).

Step 4:

- Model each histogram using the gamma distribution and construct an 8D signature by concatenating the α and β parameters for each model.

Step 5:

- Perform PCA on the 8D signature and compress it into a 4D signature.

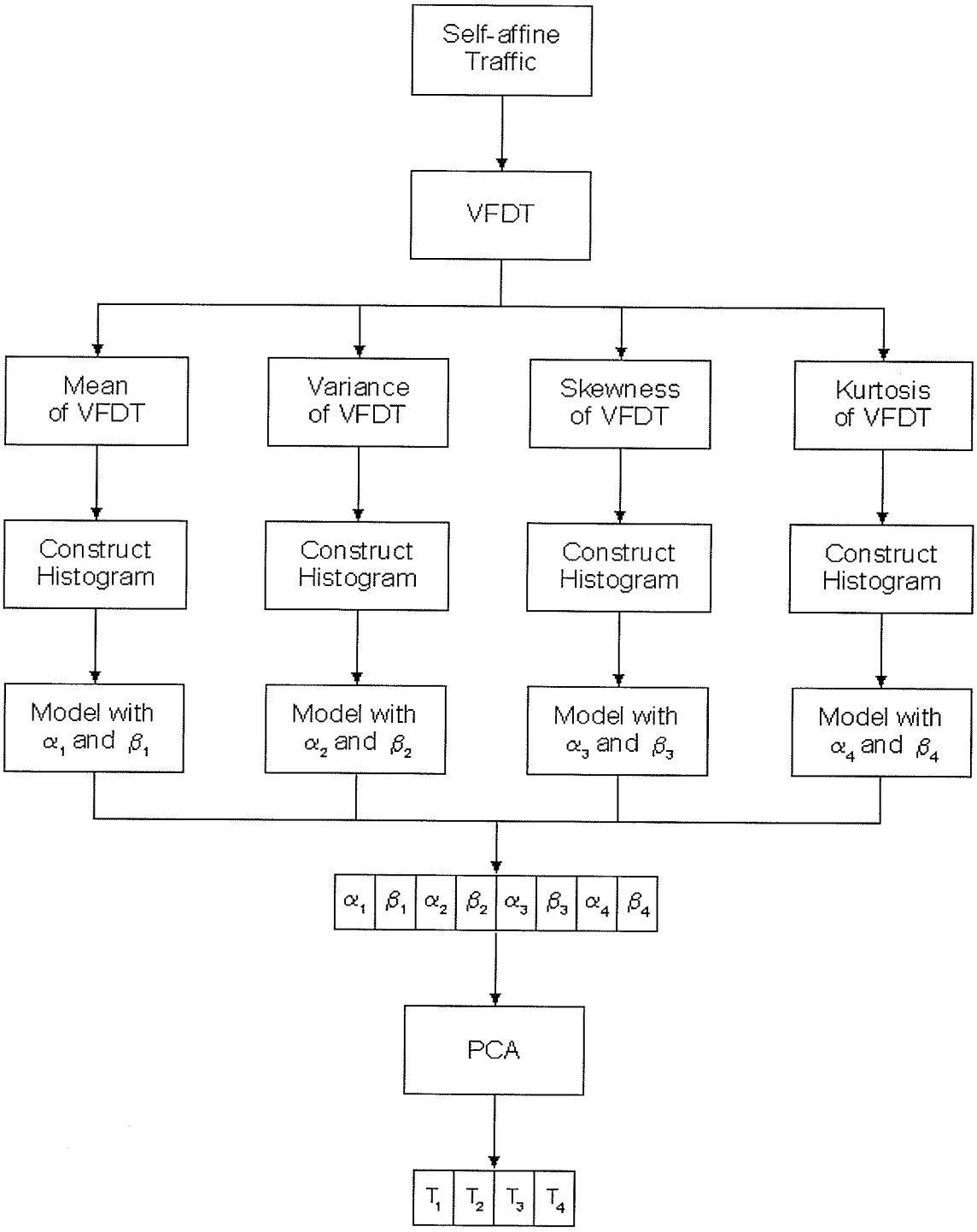

Fig. 5.52. Construction of the 4D signature T of a window of traffic.

Steps 1 and 2 create the VFDT and its statistical trajectories for the entire self-affine traffic recording. Step 3 uses a window size of 9000 to select an appropriate number of points in the statistical trajectories to construct their histograms, Step 4 models these histograms and forms an 8D signature, and Step 5 compresses this 8D signature into a 4D signature.

A trajectory of these compressed 4D signatures may be formed by iterating the last three steps for successive windows through time. For each successive iteration, a window offset of 150 is used. With the sampling frequency $f_s = 10\,\text{Hz}$, the construction of the final 4D signature trajectory uses a window size of 15 min and a window offset of 15 sec (since a minimum number of 9000 points are needed to construct a reasonable histogram). Therefore, the behaviour of the *Betta splendens* is studied for 15 minute intervals every 15 seconds.

5.8 Neural Network Processing

The final topic to discuss in the system design is the use of neural networks for (1) verification of the K-means clustering, (2) selection of the most likely number of classes, and (3) classification.

As described in section 4.5, the K-means algorithm is used to cluster the data into groups based on their topographical similarity. However, two concerns must be addressed while using the K-means algorithm. Firstly, are the results produced by the K-means algorithm *reliable*? Secondly, are the results produced by the K-means algorithm *correct*? This second question is directly related to the selection of the most likely number of classes in the data because of one important weakness in the K-means algorithm (and

many other clustering algorithms): the number of classes to find must be specified prior to the execution of the algorithm. Therefore, if there are actually six classes in a data set (although this knowledge may not be available to the researcher), and the K-means algorithm is set to find ten classes, it will "create" four new classes that do not actually exist. Similarly, if the algorithm is set to find two classes, it will merge four existing classes and the uniqueness of those existing classes will be lost.

5.8.1 Verification of Clustering

To show that the clusters generated by the K-means algorithm are reliable, these results must be somehow verified by an independent method. As discussed in section 4.6, a SOFM may be used to independently demonstrate the reliable clustering of the K-means algorithm. However, a SOFM is usually 2-dimensional [Bish00, p. 188], so for the purposes of visualization and verification, the first two principal components from the 4D signature are used with the assumption that a successful verification in two dimensions would not simply be coincidence, and that these conclusions could be extended without further proof to higher dimensions. Figure 5.53 shows the relationship between the first two principal components, $T(1)$ and $T(2)$, of the 4D signature from Record 11020219. Figure 5.54 shows the decision boundaries between six clusters using the K-means algorithm, and the nodes of a two-by-three SOFM in Fig. 5.55 show the clusters with the highest densities.

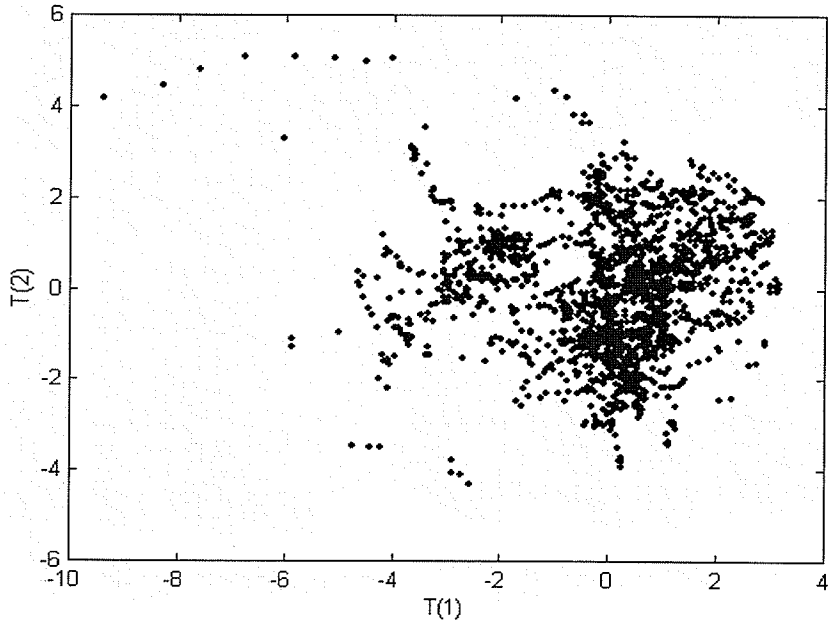

Fig. 5.53. Relationship between first two principal components from Record 11020219.

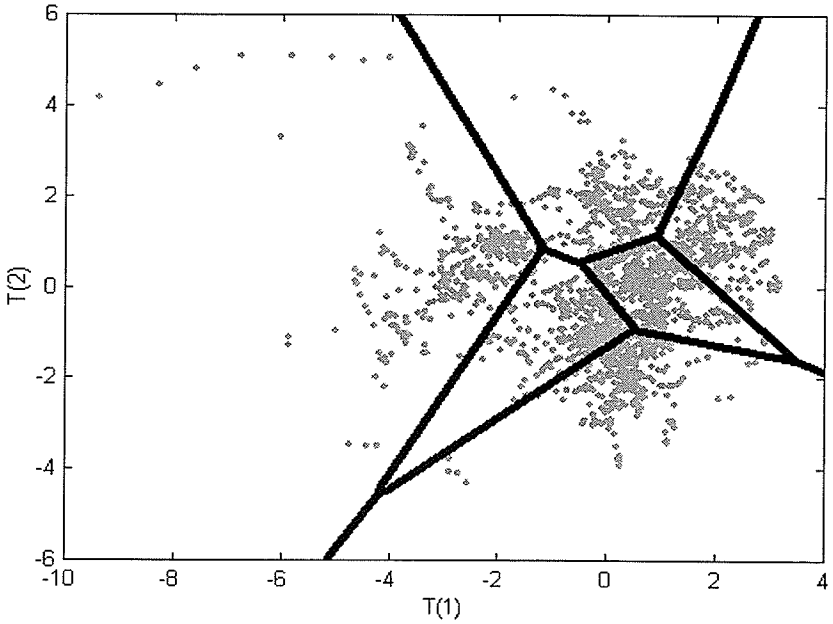

Fig. 5.54. K-means clustering boundaries of Fig. 5.53 with 6 clusters.

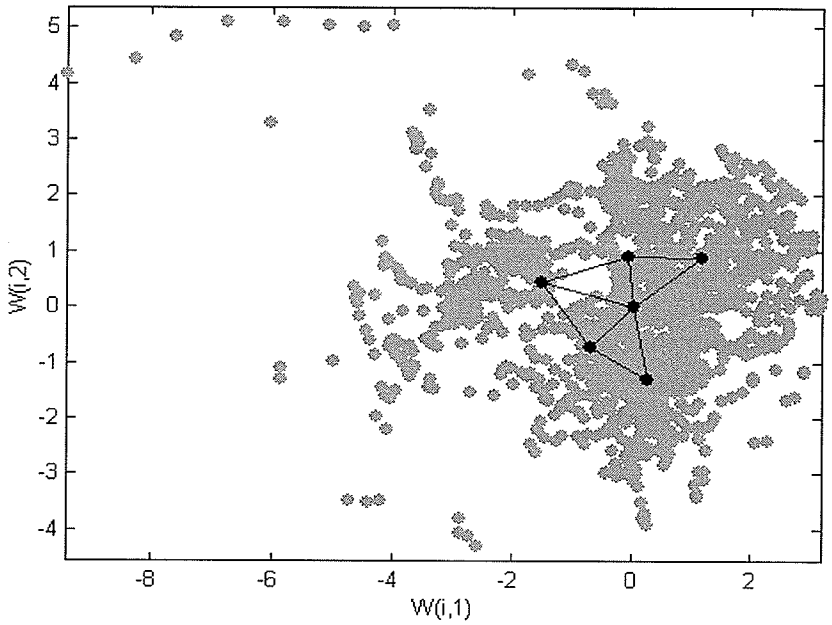

Fig. 5.55. SOFM identification of 6 clusters in Fig. 5.53.

A visual inspection of Figs. 5.54 and 5.55 show a distinct similarity between the results of the two clustering algorithms: the nodes in Fig. 5.55 lie within the boundaries of Fig. 5.54. If these figures were very different, then there would be cause for concern regarding the validity of the K-means algorithm, but since they are very similar, we may conclude with confidence that the K-means clustering algorithm produces reliable results.

5.8.2 Class Assignment

If the classes in the data are not known *a priori*, then the K-means algorithm alone cannot correctly assign classes to the clusters. As explained in section 5.8, the K-means algorithm will generate the number of clusters specified, regardless of whether or not the classes within the data are accurately represented by these clusters. Therefore, a method of measuring the correctness of choosing C clusters to model the actual classes within the

data must be developed with the intent of determining the most likely number of classes. The following intuitive argument is made to assess the correctness of the clustering algorithm in determining the most likely number of classes in the data.

- Assume that a choice C is made for the number of clusters in a sample of data, and that classes are assigned to each cluster.
- A good choice $C = C_{opt}$ for the number of classes exists.
- $C = 1$ is the trivial case and will not be considered.
- A PNN could be trained with a sufficient percentage of the data (i.e., the training set) and used to classify the remaining data (i.e., the test set).
- Choices of C where $C < C_{opt}$ may result in a clustering model that is too simplistic in its representation of the classes within the data.
- Choices of C where $C > C_{opt}$ may result in a clustering model that is too complex in its representation of the classes within the data.
- PNN misclassifications may occur less frequently when $C < C_{opt}$ because the model used is too simplistic to accurately reflect the classes within the data.
- PNN misclassifications may occur more frequently when $C > C_{opt}$ because the model used is too complex to accurately reflect the classes within the data.
- As C increases, the percentage correct classification on the test set will tend to decrease. Therefore, C_{opt} may be revealed by a dramatic change (such as a significant increase or decrease) in the rate of misclassifications.

Section 6.2 will investigate the performance of the PNN when the number of classes C increases and identify the most likely number of classes in the data.

5.8.3 Classification

Once the self-affine traffic has been characterized by a 4D signature trajectory and the most likely number of classes C_{opt} is known, classes may be assigned to the C_{opt} clusters and a PNN trained to classify previously unobserved 4D signatures.

The PNN, as explained in section 4.7, is a supervised neural network that requires a training set to train the neural network and a test set to test its performance. To obtain meaningful results from the PNN, the training and test sets must be mutually exclusive. Furthermore, to properly demonstrate the ability of the PNN to learn and generalize its knowledge on previously unobserved data, the training set should only be as large as it needs to be to be statistically representative of the data [Wass93, p. 224]. Overtraining the PNN would require more resources and, more importantly, would leave fewer data with which to construct the test set, and a test set that is too small may not be able to accurately measure the overall performance of the PNN. These requirements for the PNN are illustrated in Fig. 5.56.

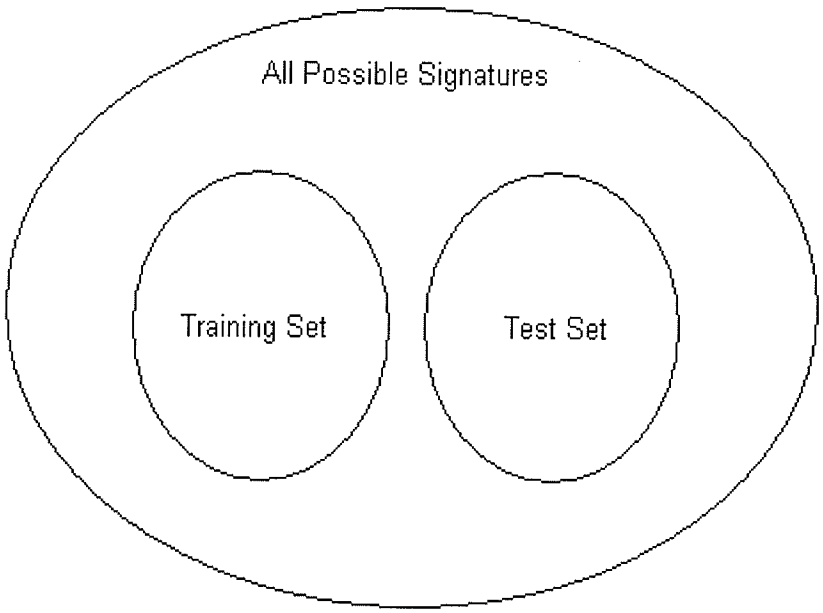

Fig. 5.56. PNN training and test sets.

The 4D signatures for the training set may be selected at *regular intervals* throughout Record 11020219, or *randomly*. Since these signatures change with time, construction of the training set by sampling the signatures at regular intervals would ensure that the traffic is equally represented throughout time. However, ignoring any *a priori* knowledge of the self-affine traffic and sampling the signatures at random would ensure that the classification accuracies of the PNNs would be as unbiased as possible. Both sampling methods for the construction of the training sets will be investigated in sections 6.2 and 6.3.

5.9 Summary

This chapter provides a comprehensive overview of the design of the system for the characterization and classification of self-affine traffic. In particular, this design includes verification of the sampling frequency used to record the data in Record 11020219, verification of the self-affinity and spatial multifractality of Record 11020219, selection of the window size and offset for the calculation of the variance fractal dimension, selection of the window size and offset for the construction of the statistical histograms of the VFDT, and modelling these statistical histograms using the gamma distribution. The verification of the K-means algorithm is also presented, as well as a discussion of the role of the PNN in determining the most likely number of classes in the data and classifying previously unobserved traffic.

CHAPTER VI

EXPERIMENTAL RESULTS AND DISCUSSION

6.1 Multifractal Characterization

Record 11020219, as introduced in section 2.7.5, is the self-affine data set selected for the final experiments to demonstrate the abilities of new multifractal classifier to characterize and classify previously unobserved traffic. Figure 6.1 once again displays Record 11020219 from Pear's data sets.

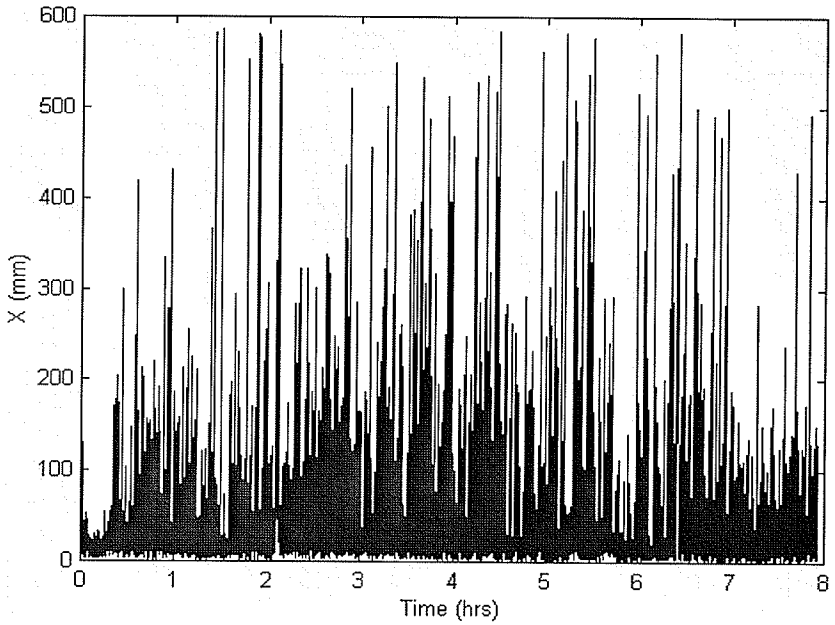

Fig. 6.1. Record 11020219.

As discussed in section 5.4, the variance fractal dimension trajectory of Fig. 6.1 is calculated using a window size of 512 and window offset of 8. The resulting VFDT is

shown again in Fig. 6.2, and is the first stage of the characterization.

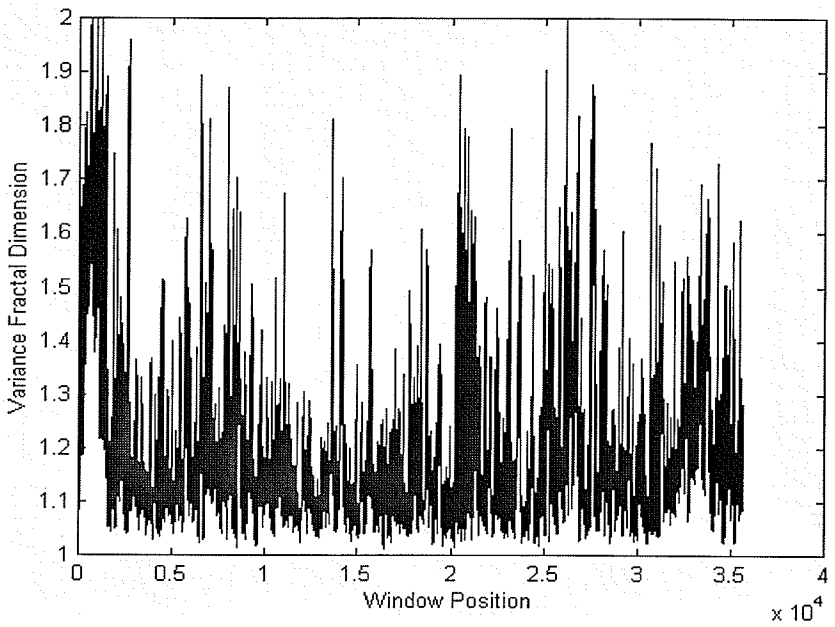

Fig. 6.2. VFDT of Fig. 6.1 using a window size of 512 and window offset of 8.

The second stage of characterization is the construction of the mean, variance, skewness, and kurtosis trajectories of the VFDT using a window size of 512 and window offset of 1. The plots of these statistical trajectories are re-displayed in Figs. 6.3 to 6.6.

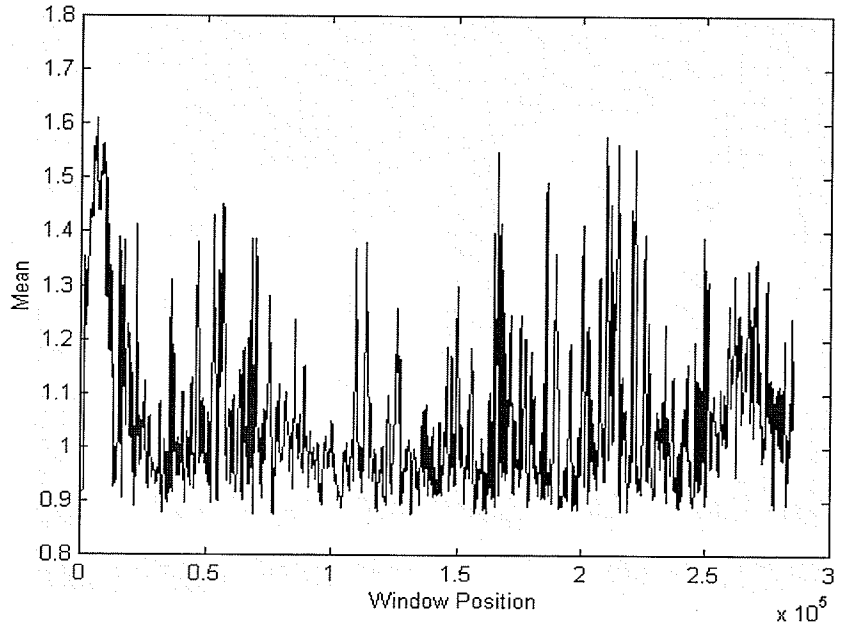

Fig. 6.3. Mean trajectory of the VFDT.

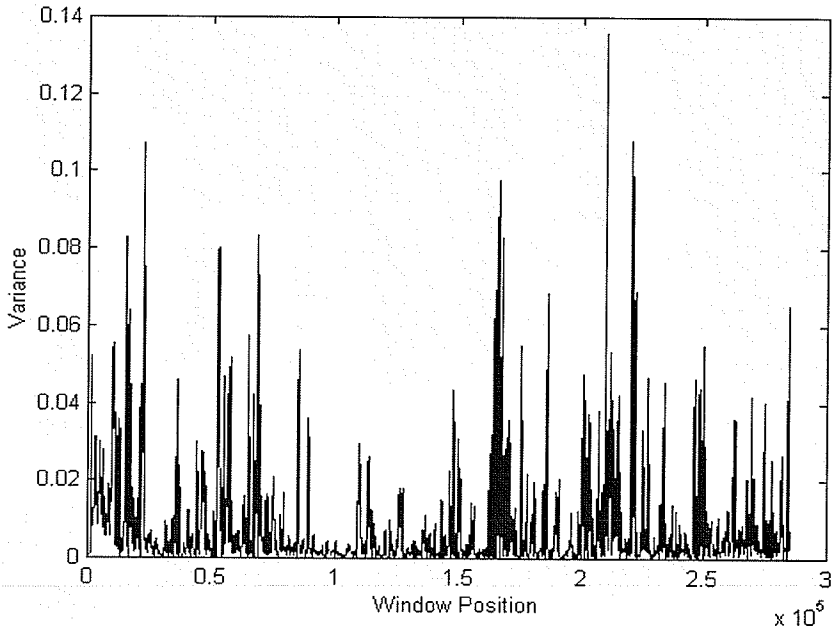

Fig. 6.4. Variance trajectory of the VFDT.

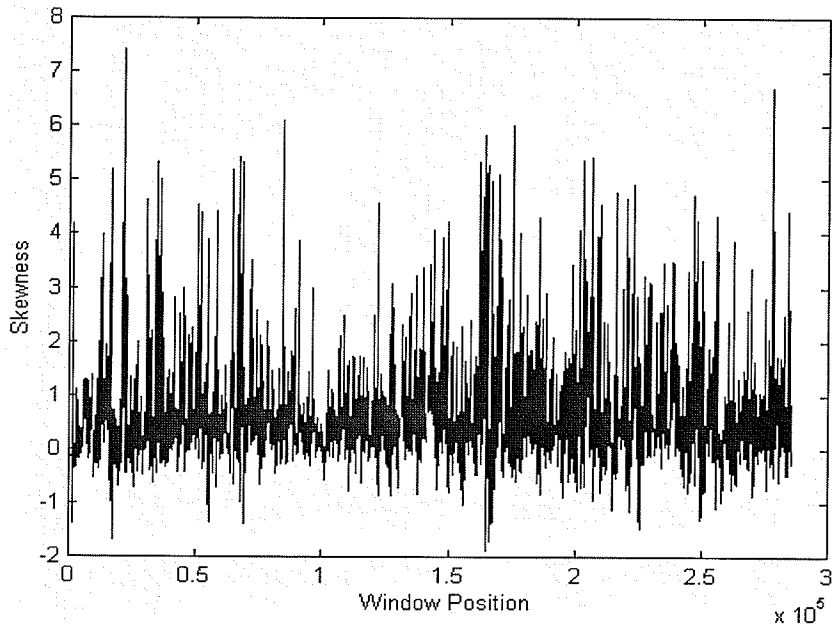

Fig. 6.5. Skewness trajectory of the VFDT.

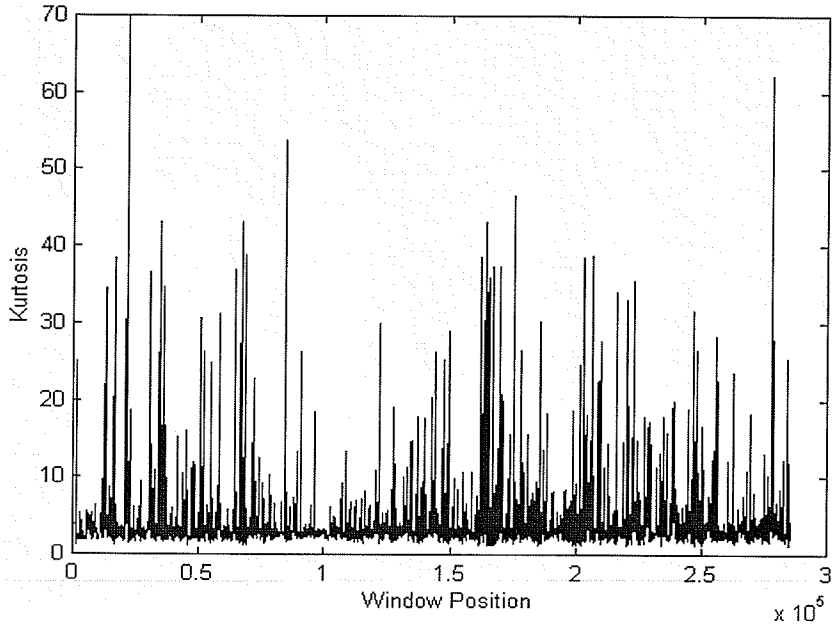

Fig. 6.6. Kurtosis trajectory of the VFDT.

The third stage of characterization is the representation and compression of the underlying non-stationary processes which generate these trajectories by modelling these four histograms using gamma distributions. A window of length 9000 and window offset of 150 is found to be sufficient in the construction of the histograms. The resulting 8D trajectory consists of eight parameters: two parameters for each of the four histograms. This 8D trajectory is then further compressed to a 4D trajectory using PCA while still preserving more than 80% of the variance in the original trajectory. The final 1844 compressed 4D signatures are shown in Fig. 6.7, and will be used as the input vectors for the training and testing of the PNN in sections 6.2 and 6.3.

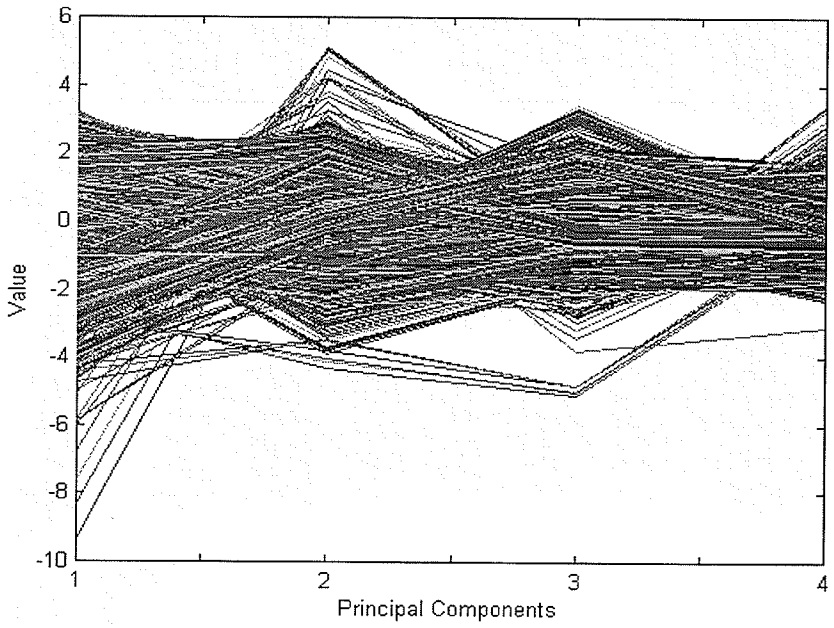

Fig. 6.7. Final 1844 compressed 4D signatures.

6.2 Optimal Class Assignment and Verification

As introduced in section 5.8.2, a weakness of the K-means clustering algorithm (as well as other algorithms) is that the user must specify the number of clusters, and therefore classes, for the algorithm to find. Therefore, making the statement that the most likely number of classes may be revealed by a high percentage classification accuracy of an optimized PNN leads us to the following simulations to determine the optimal class assignment within Record 11020219.

Firstly, there are nine possible class configurations (c = 2, 3, ..., 10) generated by the K-means algorithm, so the performance of the PNN will be investigated for each configuration. Secondly, a good value (which will be a *locally* optimal value) of the spread parameter σ will need to be determined for each configuration to ensure a fair comparison is maintained as c increases, since a good value of σ for one value of c may not be the best for another value. To determine a good value of σ for each class configuration, three training sets and three test sets are generated for the PNN for a given value p, which is the percentage of the 1844 vectors which compose the training set. One training and test set pair are generated by sampling signatures from regular intervals (denoted on the table as "Regular"), and the remaining two training and test set pairs are generated by randomly sampling the signatures (denoted as "Random #1" and "Random #2"). The generation of the training and test set pairs were chosen in this way so the value of σ chosen was not dependent on how the vectors were acquired. Thirdly, these simulations are repeated for p = 30%, p = 40%, and p = 50% to see if the most likely number of classes changes based upon the percentage of vectors used to train the PNN.

Preliminary experiments showed that a small value of σ sometimes provided good results, and other times a larger value of σ provided better results. A nearly exhaustive search through all values of σ between 0.001 and 10.0 at small intervals would have been extremely time-consuming, and the percentage correct classification may not change by much as σ is slowly varied [Wass93, p. 43]. Therefore, the practical decision was made to try values of at least σ = 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, and 5.0. In a few instances, larger values of σ were also used to determine the location of a local maximum or plateau in the percentage correct classification.

Tables 6.1 to 6.9 display the percentage correct classification using 30% of the vectors for training the PNN with varying σ and *c*. Tables 6.10 to 6.18 display the percentage correct classification using 40% of the vectors for training the PNN with varying σ and *c*. Finally, tables 6.19 to 6.27 display the percentage correct classification using 50% of the vectors for training the PNN with varying σ and *c*.

The highest percentage classification accuracy is made **bold** in each column of every table. A detailed discussion of the results will follow the presentation of these simulation results.

Table 6.1: Percentage correct classification using 30% of the vectors for training with varying σ when $c = 2$.

$c = 2$	Regular	Random #1	Random #2
σ = 0.001	54.38	54.53	**54.14**
σ = 0.005	54.84	55.38	51.51
σ = 0.01	55.54	53.76	50.27
σ = 0.05	55.62	55.31	47.56
σ = 0.1	**55.77**	**55.54**	47.87
σ = 0.5	51.90	53.45	48.18
σ = 1.0	49.65	51.43	45.39
σ = 1.5	50.12	49.73	47.25
σ = 2.0	46.63	46.63	45.93
σ = 2.5	45.93	45.86	45.78
σ = 3.0	45.86	45.86	45.86
σ = 3.5	45.86	"	45.86
σ = 4.0	"	"	"
σ = 4.5	"	"	"
σ = 5.0	"	"	"

Table 6.2: Percentage correct classification using 30% of the vectors for training with varying σ when $c = 3$.

$c = 3$	Regular	Random #1	Random #2
σ = 0.001	**48.64**	**49.11**	**48.88**
σ = 0.005	41.60	45.24	43.14
σ = 0.01	34.70	38.26	38.50
σ = 0.05	32.07	35.79	31.91
σ = 0.1	32.15	35.94	31.99
σ = 0.5	31.76	34.62	35.01
σ = 1.0	33.46	36.56	34.86
σ = 1.5	35.63	36.56	32.92
σ = 2.0	35.32	36.25	33.46
σ = 2.5	35.24	34.86	34.00
σ = 3.0	34.16	33.93	34.39
σ = 3.5	33.93	33.85	33.85
σ = 4.0	33.93	33.93	33.93
σ = 4.5	"	33.93	33.93
σ = 5.0	"	"	"

Table 6.3: Percentage correct classification using 30% of the vectors for training with varying σ when $c = 4$.

$c = 4$	Regular	Random #1	Random #2
σ = 0.001	16.73	16.81	16.89
σ = 0.005	20.91	19.44	20.45
σ = 0.01	23.47	22.77	21.84
σ = 0.05	23.08	24.17	23.70
σ = 0.1	23.01	24.01	23.70
σ = 0.5	22.77	23.16	23.70
σ = 1.0	26.65	27.34	24.40
σ = 1.5	29.12	28.43	22.39
σ = 2.0	28.74	29.05	25.17
σ = 2.5	**29.98**	**31.14**	**26.49**
σ = 3.0	29.90	30.21	26.34
σ = 3.5	28.66	27.50	25.64
σ = 4.0	26.26	24.71	24.63
σ = 4.5	23.78	23.01	24.24
σ = 5.0	22.39	21.53	23.16

Table 6.4: Percentage correct classification using 30% of the vectors for training with varying σ when $c = 5$.

$c = 5$	Regular	Random #1	Random #2
σ = 0.001	12.39	12.55	12.39
σ = 0.005	9.60	9.60	13.01
σ = 0.01	7.44	7.90	11.23
σ = 0.05	8.06	7.90	14.25
σ = 0.1	8.13	8.06	14.41
σ = 0.5	7.13	8.13	14.87
σ = 1.0	9.45	9.99	13.63
σ = 1.5	11.46	11.46	11.70
σ = 2.0	15.49	13.56	14.10
σ = 2.5	15.80	14.72	16.50
σ = 3.0	15.80	14.87	16.03
σ = 3.5	15.57	14.56	16.34
σ = 4.0	16.42	16.03	16.89
σ = 4.5	**16.81**	**16.65**	16.73
σ = 5.0	16.58	16.50	**17.04**
σ = 5.5			16.96

Table 6.5: Percentage correct classification using 30% of the vectors for training with varying σ when $c = 6$.

$c = 6$	Regular	Random #1	Random #2
σ = 0.001	6.43	6.58	6.51
σ = 0.005	7.20	6.43	7.51
σ = 0.01	6.58	6.27	8.06
σ = 0.05	7.05	6.97	11.77
σ = 0.1	7.13	7.05	11.70
σ = 0.5	5.96	6.51	11.70
σ = 1.0	8.29	7.82	11.93
σ = 1.5	10.15	10.84	11.39
σ = 2.0	11.93	11.39	11.54
σ = 2.5	13.40	12.24	12.55
σ = 3.0	14.48	13.01	14.25
σ = 3.5	15.57	13.25	14.95
σ = 4.0	16.27	14.79	15.65
σ = 4.5	16.89	15.65	15.96
σ = 5.0	**16.96**	**15.88**	**16.27**
σ = 5.5	16.34	15.34	16.11

Table 6.6: Percentage correct classification using 30% of the vectors for training with varying σ when $c = 7$.

$c = 7$	Regular	Random #1	Random #2
σ = 0.001	16.50	16.50	16.58
σ = 0.005	10.69	12.24	14.02
σ = 0.01	8.68	9.76	10.77
σ = 0.05	8.13	8.83	13.09
σ = 0.1	8.06	8.99	13.48
σ = 0.5	6.82	8.37	13.94
σ = 1.0	8.83	9.91	12.24
σ = 1.5	11.39	11.00	11.46
σ = 2.0	15.34	13.56	15.41
σ = 2.5	15.41	14.64	15.72
σ = 3.0	15.88	15.26	16.19
σ = 3.5	15.96	15.57	16.34
σ = 4.0	15.80	15.80	16.42
σ = 4.5	16.81	16.58	15.72
σ = 5.0	17.43	16.89	15.57
σ = 6.0	18.75	18.51	17.66
σ = 7.0	19.75	**20.53**	18.20
σ = 8.0	**20.53**	**20.53**	19.60
σ = 9.0	**20.53**	"	**20.53**

Table 6.7: Percentage correct classification using 30% of the vectors for training with varying σ when $c = 8$.

$c = 8$	Regular	Random #1	Random #2
σ = 0.001	16.11	16.19	16.42
σ = 0.005	14.33	14.95	15.41
σ = 0.01	11.77	13.63	12.39
σ = 0.05	9.60	10.38	12.47
σ = 0.1	9.53	10.30	12.47
σ = 0.5	9.22	11.00	12.32
σ = 1.0	8.44	9.76	12.94
σ = 1.5	11.23	12.08	11.62
σ = 2.0	14.25	14.79	14.56
σ = 2.5	15.03	16.58	16.42
σ = 3.0	17.89	**18.44**	17.74
σ = 3.5	17.97	18.20	**18.20**
σ = 4.0	**18.13**	18.13	17.97
σ = 4.5	17.97	17.97	17.97
σ = 5.0	17.97	17.97	"

Table 6.8: Percentage correct classification using 30% of the vectors for training with varying σ when $c = 9$.

$c = 9$	Regular	Random #1	Random #2
σ = 0.001	7.05	7.05	7.13
σ = 0.005	8.83	7.82	7.98
σ = 0.01	8.91	9.22	9.30
σ = 0.05	9.06	9.76	11.62
σ = 0.1	8.99	9.91	11.70
σ = 0.5	8.75	10.30	12.70
σ = 1.0	8.21	9.60	12.63
σ = 1.5	11.31	12.32	12.24
σ = 2.0	14.48	15.26	15.41
σ = 2.5	16.42	17.20	16.73
σ = 3.0	17.74	**18.44**	17.82
σ = 3.5	**17.89**	18.13	**18.05**
σ = 4.0	**17.89**	17.89	17.89
σ = 4.5	"	17.89	17.89
σ = 5.0	"	"	"

Table 6.9: Percentage correct classification using 30% of the vectors for training with varying σ when $c = 10$.

$c = 10$	Regular	Random #1	Random #2
$\sigma = 0.001$	7.05	7.05	7.05
$\sigma = 0.005$	8.06	7.05	7.20
$\sigma = 0.01$	7.59	7.13	7.75
$\sigma = 0.05$	7.67	6.74	8.68
$\sigma = 0.1$	7.59	6.97	8.83
$\sigma = 0.5$	6.43	6.74	9.30
$\sigma = 1.0$	5.81	6.89	8.21
$\sigma = 1.5$	7.51	7.82	8.68
$\sigma = 2.0$	7.98	8.68	9.53
$\sigma = 2.5$	10.53	9.45	10.92
$\sigma = 3.0$	10.22	9.84	11.31
$\sigma = 3.5$	10.92	10.38	11.77
$\sigma = 4.0$	11.77	9.99	11.70
$\sigma = 4.5$	12.01	10.46	11.70
$\sigma = 5.0$	11.85	10.61	11.15
$\sigma = 6.0$	11.77	10.69	11.00
$\sigma = 7.5$	12.47	**12.78**	15.49
$\sigma = 8.5$	**14.79**	12.39	16.34
$\sigma = 9.0$	14.56	12.24	**16.50**

Table 6.10: Percentage correct classification using 40% of the vectors for training with varying σ when $c = 2$.

$c = 2$	Regular	Random #1	Random #2
σ = 0.001	49.46	49.91	49.82
σ = 0.005	42.22	45.48	45.12
σ = 0.01	37.61	45.12	39.87
σ = 0.05	37.25	43.49	39.51
σ = 0.1	37.43	43.67	39.78
σ = 0.5	40.42	47.02	42.31
σ = 1.0	41.14	46.29	43.67
σ = 1.5	45.57	47.83	46.93
σ = 2.0	49.46	49.37	47.65
σ = 2.5	**50.45**	**50.63**	**50.54**
σ = 3.0	49.73	49.73	50.27
σ = 3.5	49.73	49.73	49.73
σ = 4.0	"	"	49.73
σ = 4.5	"	"	"
σ = 5.0	"	"	"

Table 6.11: Percentage correct classification using 40% of the vectors for training with varying σ when $c = 3$.

$c = 3$	Regular	Random #1	Random #2
σ = 0.001	**43.58**	**44.03**	**43.94**
σ = 0.005	35.26	36.08	36.44
σ = 0.01	30.38	31.19	29.66
σ = 0.05	29.29	27.49	27.94
σ = 0.1	29.57	27.76	28.12
σ = 0.5	30.38	30.11	27.49
σ = 1.0	30.02	31.10	27.22
σ = 1.5	30.56	32.73	33.00
σ = 2.0	32.91	35.35	33.63
σ = 2.5	32.73	33.63	32.73
σ = 3.0	35.90	34.45	32.55
σ = 3.5	36.89	34.27	34.90
σ = 4.0	35.44	34.99	35.44
σ = 4.5	35.44	35.35	35.44
σ = 5.0	"	35.44	"

Table 6.12: Percentage correct classification using 40% of the vectors for training with varying σ when $c = 4$.

$c = 4$	Regular	Random #1	Random #2
σ = 0.001	19.89	19.53	19.53
σ = 0.005	19.53	20.52	20.61
σ = 0.01	19.53	19.62	20.07
σ = 0.05	20.16	18.08	18.81
σ = 0.1	20.34	18.35	18.81
σ = 0.5	19.26	20.98	18.08
σ = 1.0	17.45	19.17	20.52
σ = 1.5	**21.34**	25.68	24.14
σ = 2.0	20.16	27.12	24.05
σ = 2.5	20.61	**28.66**	24.77
σ = 3.0	20.98	28.39	**24.86**
σ = 3.5	20.61	28.39	24.32
σ = 4.0	20.16	28.21	23.78
σ = 4.5	20.25	28.12	23.78
σ = 5.0	20.07	27.67	23.51

Table 6.13: Percentage correct classification using 40% of the vectors for training with varying σ when $c = 5$.

$c = 5$	Regular	Random #1	Random #2
σ = 0.001	13.20	13.29	13.38
σ = 0.005	13.02	13.38	12.30
σ = 0.01	15.55	14.47	12.93
σ = 0.05	16.64	14.29	14.56
σ = 0.1	16.91	14.29	14.74
σ = 0.5	16.27	15.82	13.56
σ = 1.0	15.10	14.65	15.64
σ = 1.5	19.71	17.63	19.44
σ = 2.0	17.90	18.63	22.06
σ = 2.5	18.72	22.78	**23.60**
σ = 3.0	21.07	**25.05**	22.88
σ = 3.5	**21.88**	22.24	22.78
σ = 4.0	20.25	21.34	22.69
σ = 4.5	19.26	20.43	20.61
σ = 5.0	19.53	20.07	20.34

Table 6.14: Percentage correct classification using 40% of the vectors for training with varying σ when $c = 6$.

$c = 6$	Regular	Random #1	Random #2
σ = 0.001	7.05	7.59	7.59
σ = 0.005	9.13	8.50	8.68
σ = 0.01	12.12	10.76	10.31
σ = 0.05	13.92	11.57	12.30
σ = 0.1	14.20	11.84	12.48
σ = 0.5	14.74	13.11	13.29
σ = 1.0	13.47	12.66	15.10
σ = 1.5	16.00	14.38	17.18
σ = 2.0	**17.72**	16.18	**18.08**
σ = 2.5	16.91	16.91	16.82
σ = 3.0	16.27	**19.44**	16.27
σ = 3.5	16.00	18.81	15.46
σ = 4.0	16.09	17.00	16.37
σ = 4.5	15.46	16.73	15.82
σ = 5.0	15.37	16.46	16.09

Table 6.15: Percentage correct classification using 40% of the vectors for training with varying σ when $c = 7$.

$c = 7$	Regular	Random #1	Random #2
σ = 0.001	15.28	15.46	15.28
σ = 0.005	13.56	12.03	14.20
σ = 0.01	11.39	11.30	12.66
σ = 0.05	11.75	9.86	12.48
σ = 0.1	11.93	10.04	12.66
σ = 0.5	12.48	11.39	12.12
σ = 1.0	11.93	10.67	11.12
σ = 1.5	14.29	14.01	13.38
σ = 2.0	16.00	14.74	17.90
σ = 2.5	14.20	18.35	**21.07**
σ = 3.0	14.74	19.26	20.89
σ = 3.5	15.37	19.71	20.07
σ = 4.0	16.27	20.89	19.80
σ = 4.5	**16.73**	**22.69**	18.63
σ = 5.0	16.37	20.34	17.99

Table 6.16: Percentage correct classification using 40% of the vectors for training with varying σ when $c = 8$.

$c = 8$	Regular	Random #1	Random #2	Random #3
σ = 0.001	**18.81**	18.90	**18.90**	18.81
σ = 0.005	15.28	15.91	17.09	19.26
σ = 0.01	12.66	12.21	15.10	17.99
σ = 0.05	12.21	10.22	12.57	13.47
σ = 0.1	12.30	10.22	12.57	13.47
σ = 0.5	12.30	12.48	12.66	14.01
σ = 1.0	12.57	13.83	15.73	16.27
σ = 1.5	**18.35***	19.53	**18.72***	**23.15**
σ = 2.0	17.81	**23.60**	18.54	21.25
σ = 2.5	17.09	18.44	17.45	17.99
σ = 3.0	17.09	18.08	17.45	17.63
σ = 3.5	17.09	17.99	17.45	17.81
σ = 4.0	17.45	18.08	18.08	18.26
σ = 4.5	17.36	18.08	17.90	18.17
σ = 5.0	17.72	17.90	17.72	17.72

The two bold numbers with a * beside them are the second highest percentage classification accuracy. Their use in the calculation of the average value of σ for this table will be explained in the following discussion.

Table 6.17: Percentage correct classification using 40% of the vectors for training with varying σ when $c = 9$.

$c = 9$	Regular	Random #1	Random #2
σ = 0.001	5.70	5.70	5.70
σ = 0.005	8.05	6.69	7.41
σ = 0.01	10.58	8.41	10.40
σ = 0.05	11.57	9.95	12.12
σ = 0.1	11.75	10.13	12.30
σ = 0.5	12.03	11.66	11.75
σ = 1.0	12.84	13.02	14.47
σ = 1.5	**17.72**	17.63	15.46
σ = 2.0	17.54	**21.16**	16.37
σ = 2.5	17.09	18.08	17.27
σ = 3.0	16.73	17.81	17.09
σ = 3.5	16.64	17.45	17.00
σ = 4.0	17.09	17.63	**17.72**
σ = 4.5	17.18	17.63	17.63
σ = 5.0	17.63	"	17.63

Table 6.18: Percentage correct classification using 40% of the vectors for training with varying σ when $c = 10$.

$c = 10$	Regular	Random #1	Random #2
$\sigma = 0.001$	5.70	5.70	5.70
$\sigma = 0.005$	6.69	6.15	6.51
$\sigma = 0.01$	8.59	7.05	8.32
$\sigma = 0.05$	10.04	8.50	9.58
$\sigma = 0.1$	10.22	8.68	9.76
$\sigma = 0.5$	10.58	9.95	9.95
$\sigma = 1.0$	11.12	10.31	12.03
$\sigma = 1.5$	15.01	13.65	13.20
$\sigma = 2.0$	**15.46**	14.47	13.47
$\sigma = 2.5$	15.19	**16.00**	14.20
$\sigma = 3.0$	14.38	15.64	15.10
$\sigma = 3.5$	14.65	15.46	15.10
$\sigma = 4.0$	14.83	15.55	15.01
$\sigma = 4.5$	15.01	15.64	**15.64**
$\sigma = 5.0$	15.01	15.64	**15.64**

Table 6.19: Percentage correct classification using 50% of the vectors for training with varying σ when $c = 2$.

$c = 2$	Regular	Random #1	Random #2
σ = 0.001	41.14	43.38	44.36
σ = 0.005	79.61	62.69	61.93
σ = 0.01	91.32	76.25	73.54
σ = 0.05	**95.99**	**83.95**	80.15
σ = 0.1	95.34	**83.95**	**80.26**
σ = 0.5	89.15	80.80	76.57
σ = 1.0	76.57	71.91	67.79
σ = 1.5	67.14	67.90	63.56
σ = 2.0	61.39	60.74	61.06
σ = 2.5	58.89	57.70	59.11
σ = 3.0	57.59	56.72	57.38
σ = 3.5	56.72	"	56.72
σ = 4.0	"	"	"
σ = 4.5	"	"	"
σ = 5.0	"	"	"

Table 6.20: Percentage correct classification using 50% of the vectors for training with varying σ when $c = 3$.

$c = 3$	Regular	Random #1	Random #2
σ = 0.001	41.00	40.24	41.00
σ = 0.005	77.77	58.89	58.46
σ = 0.01	90.24	71.26	68.55
σ = 0.05	**94.79**	**77.77**	73.43
σ = 0.1	94.58	77.55	**73.64**
σ = 0.5	88.39	72.56	70.50
σ = 1.0	69.63	61.93	59.65
σ = 1.5	59.11	53.36	51.84
σ = 2.0	55.31	51.08	49.67
σ = 2.5	53.47	52.28	48.05
σ = 3.0	51.52	51.95	45.34
σ = 3.5	48.26	50.98	44.03
σ = 4.0	41.65	45.55	41.87
σ = 4.5	43.17	43.17	40.67
σ = 5.0	41.11	42.08	41.87

Table 6.21: Percentage correct classification using 50% of the vectors for training with varying σ when $c = 4$.

$c = 4$	Regular	Random #1	Random #2
σ = 0.001	25.05	24.40	24.84
σ = 0.005	70.07	50.65	47.07
σ = 0.01	88.83	67.25	61.82
σ = 0.05	**93.82**	**73.32**	**66.27**
σ = 0.1	93.60	73.10	66.16
σ = 0.5	85.79	68.66	63.99
σ = 1.0	68.33	59.11	58.24
σ = 1.5	54.56	46.96	48.81
σ = 2.0	40.89	41.32	40.67
σ = 2.5	38.39	38.39	38.39
σ = 3.0	37.85	37.96	37.74
σ = 3.5	37.74	37.85	37.74
σ = 4.0	37.64	37.74	37.64
σ = 4.5	"	37.64	"
σ = 5.0	"	"	"

Table 6.22: Percentage correct classification using 50% of the vectors for training with varying σ when $c = 5$.

$c = 5$	Regular	Random #1	Random #2
$\sigma = 0.001$	14.43	13.99	13.77
$\sigma = 0.005$	67.90	42.52	41.97
$\sigma = 0.01$	88.50	65.18	61.06
$\sigma = 0.05$	**94.14**	**73.21**	68.55
$\sigma = 0.1$	**94.14**	72.89	**68.98**
$\sigma = 0.5$	85.57	68.98	66.59
$\sigma = 1.0$	66.59	59.33	55.86
$\sigma = 1.5$	49.89	46.10	43.38
$\sigma = 2.0$	42.19	41.43	37.31
$\sigma = 2.5$	39.37	34.49	35.36
$\sigma = 3.0$	36.12	32.43	31.34
$\sigma = 3.5$	34.49	30.26	29.28
$\sigma = 4.0$	31.89	28.74	27.98
$\sigma = 4.5$	30.37	27.77	27.44
$\sigma = 5.0$	30.04	27.33	27.22

Table 6.23: Percentage correct classification using 50% of the vectors for training with varying σ when $c = 6$.

$c = 6$	Regular	Random #1	Random #2
$\sigma = 0.001$	5.64	4.99	5.31
$\sigma = 0.005$	64.43	38.72	37.09
$\sigma = 0.01$	87.53	61.50	56.29
$\sigma = 0.05$	**94.03**	**70.61**	64.21
$\sigma = 0.1$	**94.03**	**70.61**	**64.43**
$\sigma = 0.5$	85.68	67.68	62.15
$\sigma = 1.0$	67.14	60.20	55.97
$\sigma = 1.5$	48.16	48.59	41.43
$\sigma = 2.0$	40.35	45.34	36.12
$\sigma = 2.5$	39.37	39.59	34.71
$\sigma = 3.0$	37.64	36.55	34.27
$\sigma = 3.5$	32.43	34.38	33.30
$\sigma = 4.0$	32.86	31.45	31.67
$\sigma = 4.5$	30.59	31.24	31.67
$\sigma = 5.0$	30.15	31.13	30.04

Table 6.24: Percentage correct classification using 50% of the vectors for training with varying σ when $c = 7$.

$c = 7$	Regular	Random #1	Random #2
σ = 0.001	9.22	8.57	8.68
σ = 0.005	63.99	38.39	35.25
σ = 0.01	85.36	57.38	53.69
σ = 0.05	**92.62**	**64.97**	61.28
σ = 0.1	92.52	**64.97**	**61.71**
σ = 0.5	85.14	61.93	59.22
σ = 1.0	63.34	51.30	49.13
σ = 1.5	46.53	40.78	39.26
σ = 2.0	39.48	35.25	33.19
σ = 2.5	36.01	32.00	30.69
σ = 3.0	33.30	30.26	27.01
σ = 3.5	33.30	28.52	27.33
σ = 4.0	32.32	28.74	28.31
σ = 4.5	29.18	25.81	27.87
σ = 5.0	28.52	27.22	29.18

Table 6.25: Percentage correct classification using 50% of the vectors for training with varying σ when $c = 8$.

$c = 8$	Regular	Random #1	Random #2
σ = 0.001	24.30	23.64	23.97
σ = 0.005	69.52	46.20	43.17
σ = 0.01	88.29	61.61	54.66
σ = 0.05	**92.95**	66.70	**57.59**
σ = 0.1	**92.95**	**66.92**	57.48
σ = 0.5	84.71	63.34	54.23
σ = 1.0	62.15	52.17	46.75
σ = 1.5	44.47	44.58	37.85
σ = 2.0	38.72	36.66	35.90
σ = 2.5	33.73	31.89	32.00
σ = 3.0	28.74	28.63	27.22
σ = 3.5	28.74	25.92	26.90
σ = 4.0	28.74	25.81	25.92
σ = 4.5	28.20	26.25	25.60
σ = 5.0	27.11	27.55	25.70

Table 6.26: Percentage correct classification using 50% of the vectors for training with varying σ when $c = 9$.

$c = 9$	Regular	Random #1	Random #2
σ = 0.001	5.86	5.21	5.53
σ = 0.005	63.45	35.57	32.65
σ = 0.01	85.47	55.53	48.81
σ = 0.05	**92.52**	63.34	55.10
σ = 0.1	92.41	**63.67**	**55.31**
σ = 0.5	83.73	59.87	51.84
σ = 1.0	62.04	48.59	43.82
σ = 1.5	45.12	38.61	33.62
σ = 2.0	33.41	30.69	31.34
σ = 2.5	28.52	27.98	27.33
σ = 3.0	23.97	24.08	23.21
σ = 3.5	22.13	22.45	22.13
σ = 4.0	21.37	21.58	21.58
σ = 4.5	21.37	21.48	21.26
σ = 5.0	21.15	21.37	"

Table 6.27: Percentage correct classification using 50% of the vectors for training with varying σ when $c = 10$.

$c = 10$	Regular	Random #1	Random #2
σ = 0.001	5.86	5.21	5.53
σ = 0.005	63.56	36.55	34.06
σ = 0.01	85.68	56.51	51.08
σ = 0.05	**92.84**	64.75	57.81
σ = 0.1	**92.84**	**65.08**	**58.13**
σ = 0.5	84.38	60.41	54.66
σ = 1.0	64.53	50.65	47.07
σ = 1.5	47.94	41.32	35.14
σ = 2.0	35.36	32.10	31.02
σ = 2.5	29.07	27.33	26.90
σ = 3.0	25.05	26.03	24.40
σ = 3.5	23.32	22.78	23.32
σ = 4.0	20.93	20.82	21.04
σ = 4.5	19.63	19.74	19.52
σ = 5.0	18.87	19.09	18.87

In general, the values of σ that corresponded to a high percentage classification accuracy were close to each other, which means that the selection of σ is not very dependent on the method of generation of the training and test set pairs. However, the

percentage classification accuracies were more sensitive to changes in σ than originally anticipated. To determine a good value of σ for each simulation, the average value of σ for each table was calculated. The only exception to this was in Table 6.16 where the first three simulations gave maximum classification values of 18.81%, 23.60%, and 18.90% when $\sigma = 0.001$, 2.0, and 0.001, respectively. These values of σ were very different, and the classification accuracy at their average value of $\sigma = 0.67$ resulted in a relatively poor rate of about 12%. More results were needed to obtain a better result, so a fourth simulation (denoted by the added column titled "Random #3") was performed which resulted in a maximum classification of 23.15% when $\sigma = 1.5$. Closer inspection of these data revealed that although $\sigma = 0.001$ resulted in the highest classification accuracy for Regular and Random #2 data sets, the *second* highest classification accuracy was when $\sigma = 1.5$ for both sets, which is much closer to the other values of σ. Therefore, the values of $\sigma = 0.001$ in this table alone are labelled as outliers, and the values of $\sigma = 1.5$ will be used in their place for the calculation of a good value of σ.

Tables 6.28, 6.29, and 6.30 show the average value of σ which results in the highest percentage classification accuracy for each value of c when 30%, 40%, and 50% of the vectors, respectively, are used for training the PNN.

Table 6.28: Average values of σ which achieves the highest percentage correct classification using 30% of the vectors for training.

30%	Average σ
$c = 2$	0.067
$c = 3$	0.001
$c = 4$	2.5
$c = 5$	4.7
$c = 6$	5.0
$c = 7$	8.0
$c = 8$	3.5
$c = 9$	3.5
$c = 10$	8.3

Table 6.29: Average values of σ which achieves the highest percentage correct classification using 40% of the vectors for training.

40%	Average σ
$c = 2$	2.5
$c = 3$	0.001
$c = 4$	2.3
$c = 5$	3.0
$c = 6$	2.3
$c = 7$	3.8
$c = 8$	1.6
$c = 9$	2.8
$c = 10$	3.5

Table 6.30: Average values of σ which achieves the highest percentage correct classification using 50% of the vectors for training.

50%	Average σ
$c = 2$	0.075
$c = 3$	0.067
$c = 4$	0.050
$c = 5$	0.075
$c = 6$	0.080
$c = 7$	0.075
$c = 8$	0.075
$c = 9$	0.083
$c = 10$	0.088

Tables 6.31, 6.32, and 6.33 show the PNN percentage correct classification using the average values of σ for each value of c when 30%, 40%, and 50% of the vectors sampled at regular intervals, respectively, are used for training the PNN.

Table 6.31: PNN percentage correct classification using the average values of σ for each value of c when 30% of the vectors sampled at regular intervals are used for training.

30%	% Correct Classification
$c = 2$	55.54
$c = 3$	48.64
$c = 4$	29.98
$c = 5$	16.73
$c = 6$	16.96
$c = 7$	20.53
$c = 8$	17.97
$c = 9$	17.89
$c = 10$	14.49

Table 6.32: PNN percentage correct classification using the average values of σ for each value of c when 40% of the vectors sampled at regular intervals are used for training.

40%	% Correct Classification
$c = 2$	50.45
$c = 3$	43.58
$c = 4$	20.89
$c = 5$	21.07
$c = 6$	17.27
$c = 7$	15.73
$c = 8$	17.81
$c = 9$	16.55
$c = 10$	14.65

Table 6.33: PNN percentage correct classification using the average values of σ for each value of c when 50% of the vectors sampled at regular intervals are used for training.

50%	% Correct Classification
$c = 2$	95.55
$c = 3$	94.79
$c = 4$	93.82
$c = 5$	94.14
$c = 6$	94.14
$c = 7$	92.41
$c = 8$	92.84
$c = 9$	92.41
$c = 10$	92.84

Figure 6.8 shows the percentage correct classification results listed in Table 6.33 with an optimized PNN trained with 50% of the vectors sampled at regular intervals. Upon first glance, there seems to be an interesting plateau in the classification rate when $c = 5$ and $c = 6$ followed by a steep drop in correct classification. However, a brief glance at the y-axis reveals that all of the classification rates are between 92.5% and 95.5%, and therefore within only 3% of each other! Therefore, this network seems to be well trained since all of the classification accuracies are above 90%, so no conclusive statements may be made regarding the most likely number of classes by studying this graph.

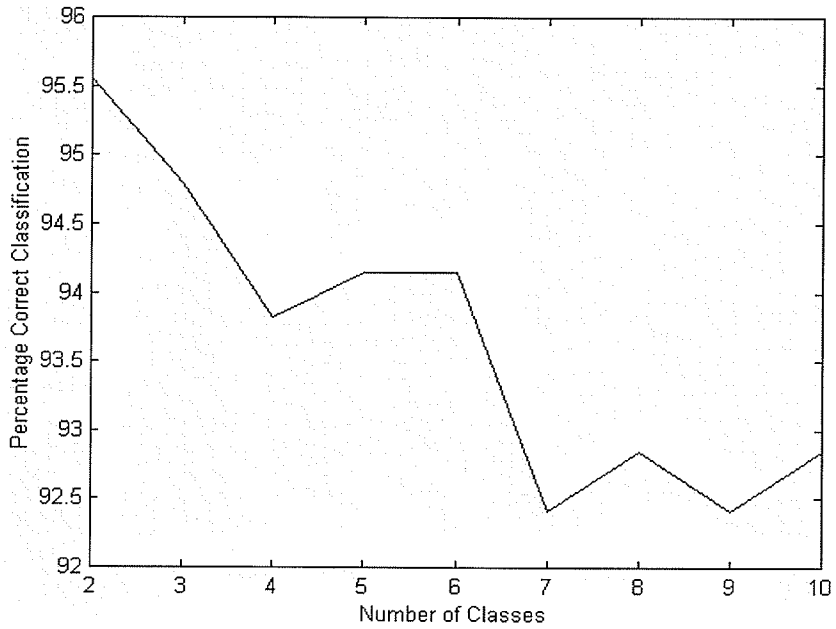

Fig. 6.8. Percentage correct classification with an optimized PNN trained with 50% of the vectors sampled at regular intervals.

Since Fig. 6.8 does not offer any insights regarding the most likely number of classes, perhaps plots of the percentage correct classification listed in Tables 6.31 and 6.32 with optimized PNNs trained with only 30% and 40% of the vectors sampled at regular intervals, respectively, will illuminate the most likely number of classes. These plots are shown together in Fig. 6.9.

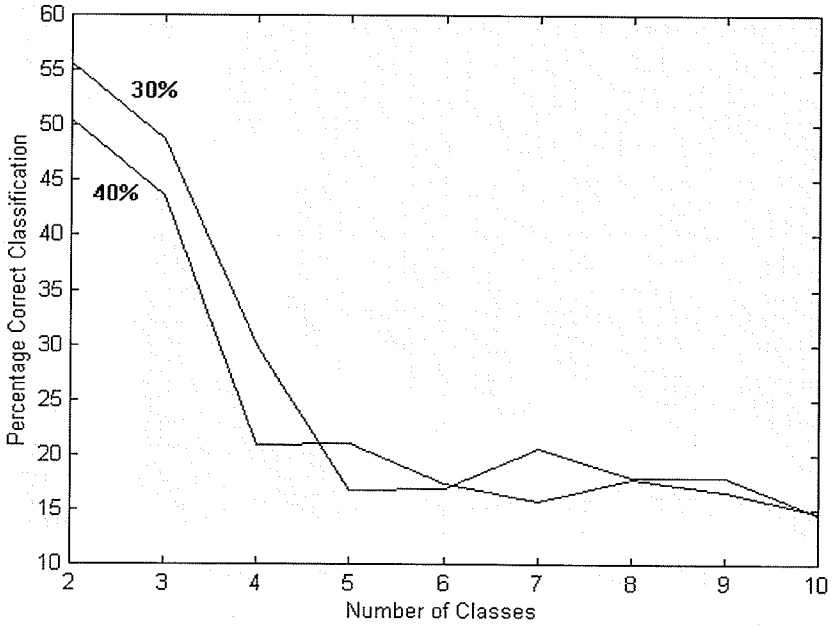

Fig. 6.9. Percentage correct classification with an optimized PNN trained with 30% and 40% of the vectors sampled at regular intervals.

Figure 6.9 is a very interesting graph, and several statements may be made about it. Firstly, both of the PNNs are undertrained with only 30% and 40% of the vectors, respectively, because the maximum percentage correct classification is only around 50%. Secondly, both of the curves look very similar. Thirdly, the PNN trained with 30% of the vectors has a higher classification rate than the PNN trained with 40% of the vectors with smaller values of c = 2, 3, and 4. This value seems to contradict the previous statement made that a PNN trained with more vectors performs well; however, as previously stated, the PNN is undertrained, and the small 5% difference between the curves is most likely inherent to the data used to test the PNNs. Fourthly, the percentage correct classification only declines slightly by about 5% for both curves when the number of classes increases from c = 2 to c = 3. Fifthly, a steep decrease of about 25% to 30% is observed on both

plots as the number of classes increases further from $c = 3$ to $c = 5$. Sixthly, the percentage correct classification maintains a low value between 15% and 20% for the remaining values of $c = 6$ to $c = 10$. Finally, this low percentage correct classification would most likely persist as the number of classes increased past $c = 10$, so we need not consider values higher than $c = 10$.

The discussion from section 5.8.2 stated that the most likely number of classes may be indicated by a marked change in the rate of misclassifications. Keeping this point in mind, our attention is drawn to the significant decrease in the percentage correct classification as the number of classes c increases beyond $c = 3$. Since this classification rate does not recover, it seems reasonable that our attention should be focused on $c = 2$ and $c = 3$ as possible candidates for the most likely number of classes.

If we recall the motivation discussion of clustering in section 4.5, Fig. 4.9 displays a hypothetical plot of traffic characterized in 2-dimensions. Figures 4.10 and 4.11 show these data with the assignment of two and four classes, respectively. If two classes are assigned to the data and a PNN trained with a sufficient percentage of the characteristic vectors, then it seems reasonable to expect that the PNN would perform quite well when tested with previously unobserved data. Similarly, if four classes are assigned to the data, the PNN would also perform well when trained in the same fashion. However, the *mis*classification rate would most likely increase when the number of assigned classes is three and five or more because models with numbers of assigned classes other than two and four would not accurately reflect the true number of classes in the data. Although a model with two classes would perform well in classifying previously unobserved data, and probably slightly better than a model with four classes, the model with four classes should

be selected as the most likely number of classes in this hypothetical example: a model with two classes is too simplistic and a model with five or more classes is too complex.

Relating this discussion of a hypothetical example back to the real characteristic signatures from Record 11020219, we should select $c = 3$ as the most likely number of classes over $c = 2$, as it is the largest number of classes with a relatively high percentage classification accuracy. Therefore, the most likely number of classes in Record 11020219 is *three*.

The previous statement is very important, and must be further verified through visualization of the clustering. Unfortunately, visualization of three classes in 4D data on 2D black and white paper is difficult task to accomplish! Therefore, the classes will be projected from 4-dimensions onto 2-dimensions in three different ways, keeping in mind that two dimensions are omitted for the sake of clarity. Figure 6.10 shows the projection of the three classes onto the two main principal components, namely PC(1) and PC(2). As before, the solid lines represent the decision boundaries between classes. If these were the only two dimensions, then it appears to the human visual classification system that there are only two classes in the data. However, these are only two out of four dimensions which, as shown in Fig. 5.49, only account for roughly 60% of the variance in the data. The remaining 40% of the variance is contained in PC(3) and PC(4), so we must also consider those dimensions.

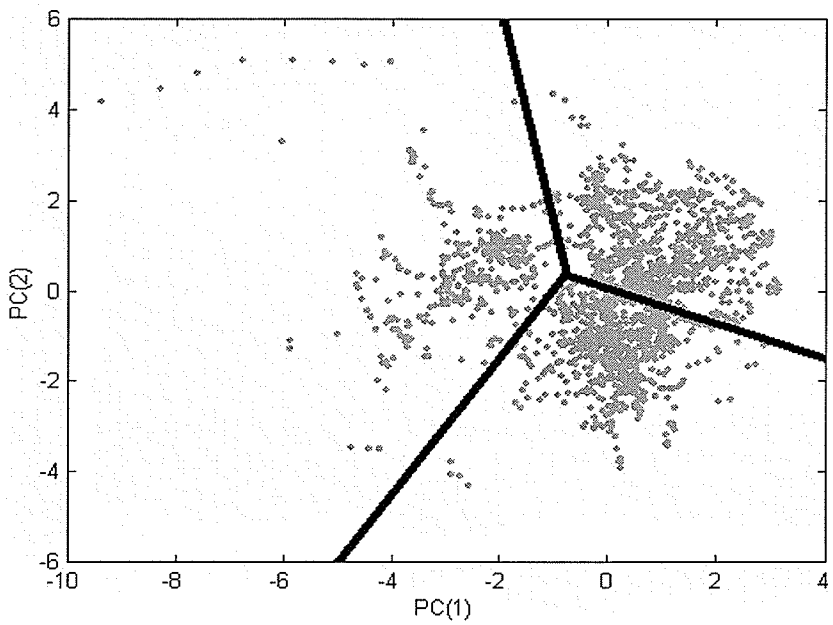

Fig. 6.10. Three classes in 4-dimensions projected onto PC(1) and PC(2).

Figures 6.11 and 6.12 show the classes projected onto PC(1) and PC(3), and PC(2) and PC(3), respectively, and complete the display of the relationships between the first three principal components. Figures 6.13, 6.14, and 6.15 show the information contained within the fourth dimension, as the classes are projected onto PC(1) and PC(4), PC(2) and PC(4), and PC(3) and PC(4). The decision boundaries overlap on all of these figures, so their inclusion would not add any more clarity to the figure. Therefore, the shaded symbols ".", "+", and "*" are used to indicate the clustering of the three classes.

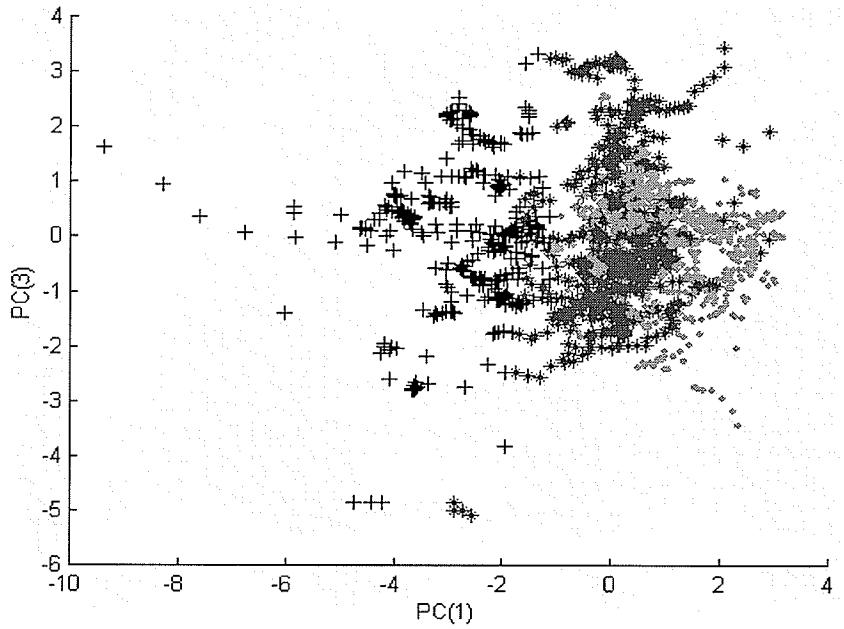

Fig. 6.11. Three classes in 4-dimensions projected onto PC(1) and PC(3).

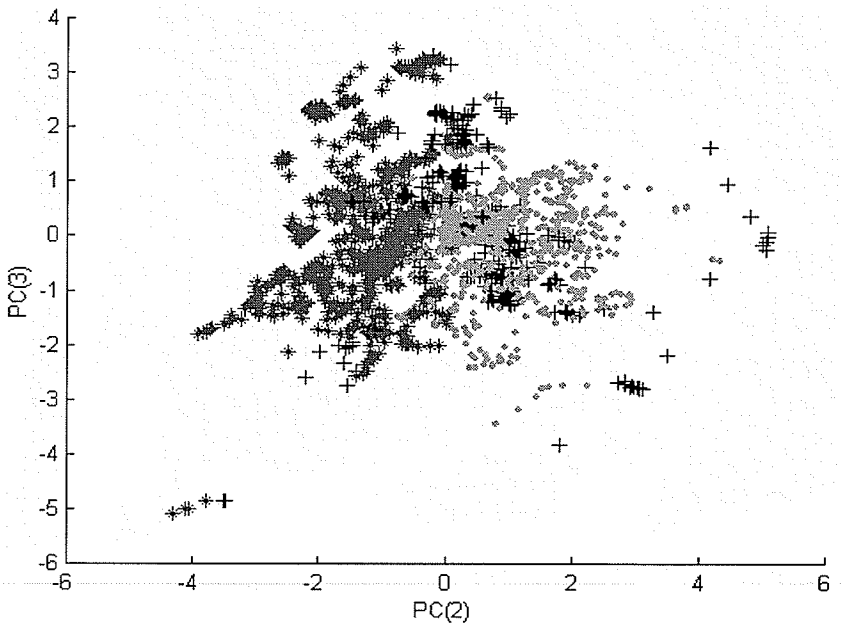

Fig. 6.12. Three classes in 4-dimensions projected onto PC(2) and PC(3).

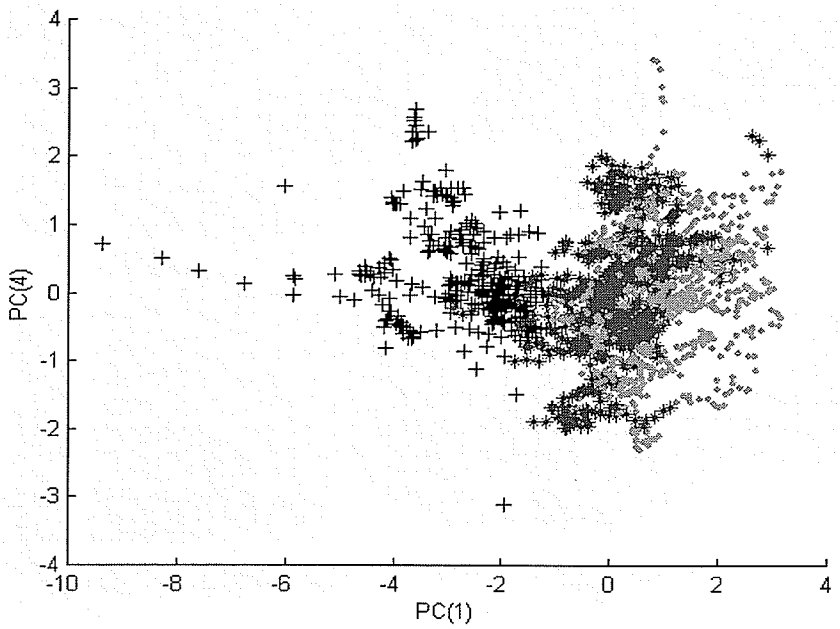

Fig. 6.13. Three classes in 4-dimensions projected onto PC(1) and PC(4).

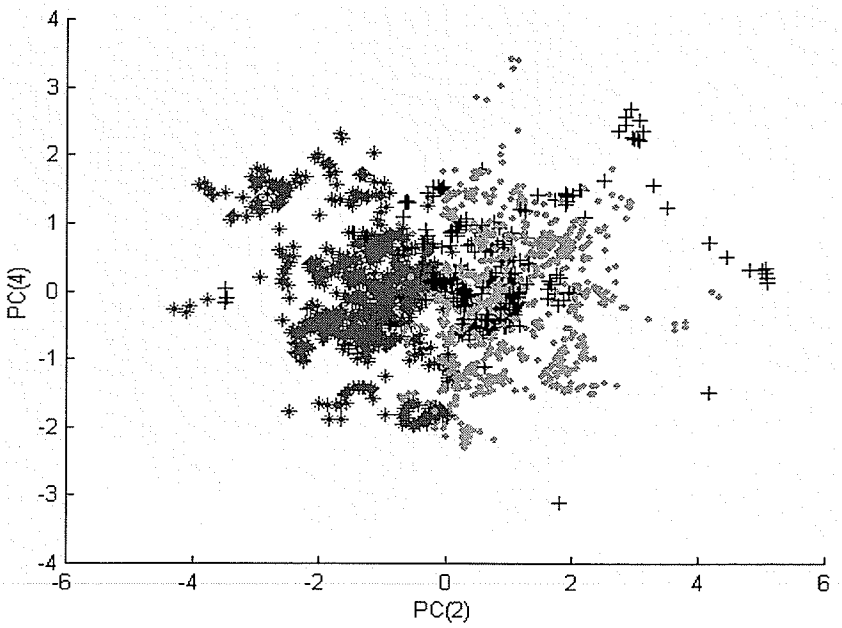

Fig. 6.14. Three classes in 4-dimensions projected onto PC(2) and PC(4).

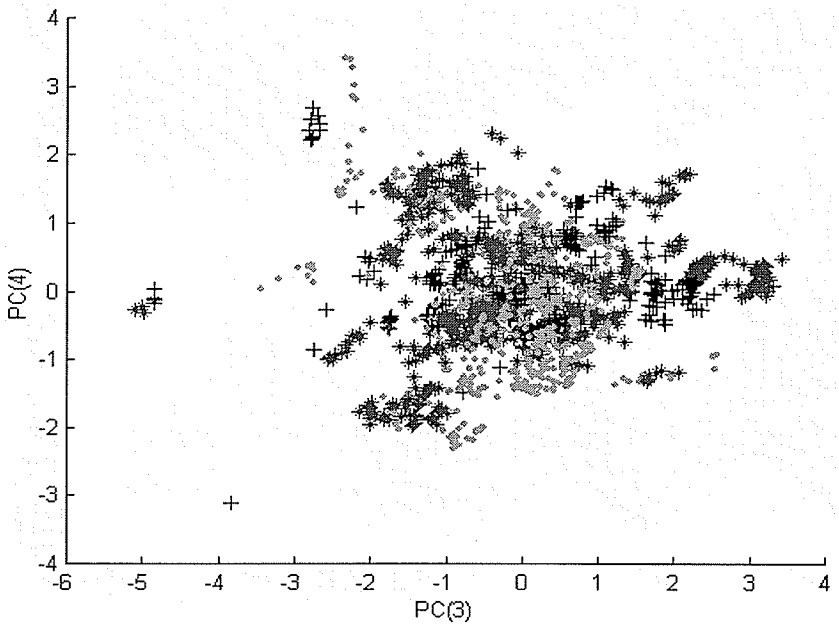

Fig. 6.15. Three classes in 4-dimensions projected onto PC(3) and PC(4).

Figure 6.11 shows a clear separation between the "+" class and the "." and "*" classes; this is the same separation that was observed in Fig. 6.10. Furthermore, Fig. 6.12 shows a good separation between the "." and "*" classes in front of the "+" class. Therefore, the first three principal components seem to suggest that there are three classes in the data. Figure 6.13 introduces the data contained in PC(4) and reaffirms the distinct separation between the "+" class and the "." and "*" classes. In the same fashion, Fig. 6.14 also reaffirms the separation between the "*" class and the "." and "+" classes. All three classes seem to overlap completely in Fig. 6.15 and no new class information is revealed. This is, however, expected because the relationship between PC(3) and PC(4) contains the least amount of information.

It has been shown, both mathematically and visually, that there are at least two and

most likely three classes in Record 11020219. These three classes unevenly account for 44%, 38%, and 18% of the data, respectively. Furthermore, there appears to be no viable evidence to entertain the possibility of a hidden fourth class within the first four principal components. For the sake of completeness and motivation for future research (which will be discussed in section 7.3), Fig. 6.16 shows the 1844-element behavioural classification trajectory for Record 11020219.

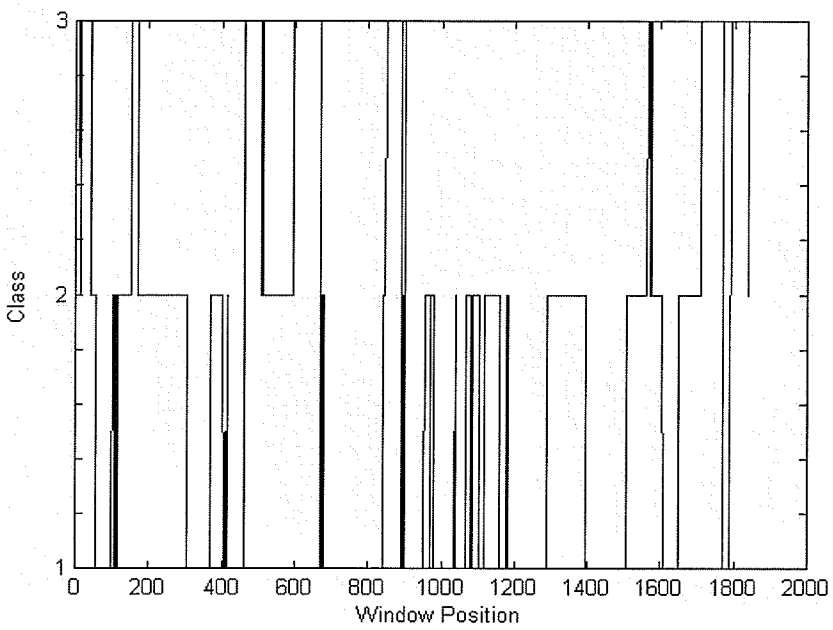

Fig. 6.16. Behavioural classification trajectory for Record 11020219.

Therefore, Record 11020219 contains three distinct classes.

6.3 PNN Classification

Since it has been determined that there are three classes in Record 11020219, the final task is to demonstrate the performance of the PNN in classifying previously

unobserved traffic. Section 6.2 showed that a PNN was sufficiently trained using 50% of the vectors for training, so the criterion will be set that 50% of the vectors are used to train the PNN.

An optimized PNN is created with $\sigma = 0.067$ and trained with 50% of the vectors sampled at regular intervals and random intervals. The performance of this PNN is shown in Table 6.34.

Table 6.34: PNN percentage correct classification with three classes when $\sigma = 0.067$ and 50% of the vectors are used for training.

50%	% Correct Classification
Regular	94.79
Random #1	77.77
Random #2	73.64
Random #3	65.73
Random #4	68.00
Random #5	73.43

As expected, the PNN achieves nearly 95% correct classification when the training and test sets are sampled at regular intervals, but the same PNN achieves an average of only 72% correct classification when the sets are sampled at random intervals. To investigate this 23% difference, Table 6.35 shows the confusion matrix for misclassifications when regularly sampled data is used to train the PNN. In a confusion matrix, the rows represent the actual classes and the columns represent the predicted

classes. Therefore, the diagonal of the matrix represents the correct classifications. In Table 6.35, for example, 350 vectors in Class 1 were *correctly* classified as Class 1, 5 vectors were *in*correctly classified as Class 2, and 13 vectors were *in*correctly classified as Class 3. Table 6.42 shows these numbers expressed as percentages.

Tables 6.36 to 6.40 show the confusion matrices for misclassifications when randomly sampled data are used to train the PNN. Table 6.41 shows the summation of these five tables, and Table 6.43 shows these summations expressed as percentages.

Table 6.35: PNN confusion matrix for simulation "Regular" in Table 6.34.

Regular	Class 1	Class 2	Class 3
Class 1	350	5	13
Class 2	6	209	7
Class 3	9	8	315

Table 6.36: PNN confusion matrix for simulation "Random #1" in Table 6.34.

Random #1	Class 1	Class 2	Class 3
Class 1	313	23	32
Class 2	30	151	41
Class 3	37	42	253

Table 6.37: PNN confusion matrix for simulation "Random #2" in Table 6.34.

Random #2	Class 1	Class 2	Class 3
Class 1	281	8	79
Class 2	55	151	16
Class 3	45	40	247

Table 6.38: PNN confusion matrix for simulation "Random #3" in Table 6.34.

Random #3	Class 1	Class 2	Class 3
Class 1	263	6	99
Class 2	42	135	45
Class 3	66	58	208

Table 6.39: PNN confusion matrix for simulation "Random #4" in Table 6.34.

Random #4	Class 1	Class 2	Class 3
Class 1	248	33	87
Class 2	32	160	30
Class 3	66	47	219

Table 6.40: PNN confusion matrix for simulation "Random #5" in Table 6.34.

Random #5	Class 1	Class 2	Class 3
Class 1	290	36	42
Class 2	22	161	39
Class 3	72	34	226

Table 6.41: Summation of PNN confusion matrices for all five "Random" simulations.

Random	Class 1	Class 2	Class 3
Class 1	1395	106	339
Class 2	181	758	171
Class 3	286	221	1153

Table 6.42: PNN *percentage* confusion matrix for the "Regular" simulation.

Regular	Class 1	Class 2	Class 3
Class 1	95.11	1.36	3.53
Class 2	2.70	94.15	3.15
Class 3	2.71	2.41	94.88

Table 6.43: PNN *percentage* confusion matrix for all five "Random" simulations.

Random	Class 1	Class 2	Class 3
Class 1	75.82	5.76	18.42
Class 2	16.31	68.29	15.40
Class 3	17.23	13.31	69.46

As expected, Table 6.42 shows us that very few misclassifications occurred when the training vectors are sampled at regular intervals: Classes 1, 2, and 3 were correctly classified 95.11%, 94.15%, and 94.88% of the time, respectively.

Table 6.43 is interesting because it reveals how the misclassifications occurred when the training vectors were sampled at random intervals. Class 1 was incorrectly classified as Class 3 about three times as often as Class 2. Class 2 was incorrectly classified equally between Classes 1 and 3, and Class 3 was incorrectly classified as Class 1 30% more often than as Class 2.

The performance of the PNN is very sensitive to the sampling methods used in constructing the training set. This difference is most likely due to the fact that the three classes are unequally represented in the data and unevenly distributed in its sequence. If a random sampling does not select enough vectors in Class 1, for example, then it will be undertrained in representing Class 1. Furthermore, the test set will be composed of those vectors that were not used to train that class. The resulting scenario is a PNN that is both undertrained and overtested in Class 1, resulting in more frequent misclassifications for that Class 1. As discussed in section 7.3, a better method of representing the classes when randomly sampling training vectors would be to randomly sample them within their respective classes, rather than randomly sampling them from the original classification trajectory. It is anticipated that a representative random sampling for each class would result in a percentage correct classification comparable to that achieved when sampling training vectors at regular intervals.

Therefore, when the three classes are properly represented in the training set through sampling at regular intervals, the optimized PNN achieves a *representative* percentage correct classification of approximately 95%.

6.4 Summary

This chapter summarizes the multifractal characterization of Record 11020219, and explores the optimal class assignment by selecting a good value of σ using training and test set pairs of different sizes formed by selected vectors at both regular and random intervals. The optimal class assignment is then verified through the 2D visualization of the first four principal components. Finally, an optimized PNN is trained with 50% of the vectors, and its performance in classifying previously unobserved traffic is evaluated.

CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS

7.1 Conclusions

This thesis sets out to improve upon existing traffic classifiers through the development of a new multifractal traffic classifier. Unfortunately, the data sets that were to have been used in this thesis became unavailable while the research was well underway, and therefore other self-affine data sets had to be acquired so that this thesis could be completed in a timely manner. Record 11020219 from Pear's data sets was chosen as the primary self-affine traffic recording with which to demonstrate the use of multifractal analysis and neural networks to reliably and accurately characterize and classify network traffic.

This thesis demonstrates the presence of both spatial and temporal multifractality in Record 11020219. The Rényi dimension spectrum was constructed using sequential non-overlapping windows of 1024 points, and the resulting monotonically decreasing curve for each window revealed consistent spatial multifractality in the traffic sequence. The variance fractal dimension was also constructed using a non-overlapping window size of 1024 points that resulted in a trajectory of dimensions of which 96.4% had a mean square error of less than 0.1, thereby proving that the traffic sequence is temporally multifractal and self-affine.

To characterize the self-affine traffic, the variance fractal dimension trajectory was once again calculated using a window size of 512 points and window offset of 8 points.

A sampling frequency of 10 Hz was used to record the traffic, which resulted in a temporal window size of 51.2 seconds and a window offset of 0.8 seconds. A minimum of 512 points are required for the calculation of the variance fractal dimension, which means that with a sampling frequency of 10 Hz, the temporal window size of 51.2 seconds was the highest level of temporal resolution possible. A higher sampling frequency would be required to study the behaviour of the self-affine traffic on a shorter time scale. For example, a sampling frequency of 256 Hz would enable a maximum temporal resolution of 2.0 seconds.

The mean, variance, skewness, and kurtosis trajectories of the variance fractal dimension trajectory were constructed using a window size of 512 points and window offset of 1 point. Histograms of each statistical trajectory were calculated using a window size of 9000 points and a window offset of 150 points. A sampling frequency of 10 Hz resulted in a temporal window size of 15 minutes and a window offset of 15 seconds. The gamma distribution was selected to model the strict stationarity of the histograms of these four statistical trajectories. Eight parameters (two for each of the four gamma distributions) were used to represent each set of four histograms, which resulted in the construction of a trajectory of 8-dimensional signatures.

A significant degree of correlation existed between these eight dimensions, so principal component analysis was used to decorrelate the dimensions. A plot of the cumulative variance for the principal components revealed that 87% of the variance was contained within the first four principal components, so the first four principal components were kept to form a compressed 4-dimensional multifractal signature for each window of traffic.

A trajectory of these 4-dimensional signatures is now the new compressed representation of Record 11020219. The K-means algorithm was used to cluster these signatures, with adequate verification from a self-organizing feature map. The performance of probabilistic neural networks trained with 30%, 40%, and 50% of the signatures, respectively, sampled at regular and random intervals were examined to reveal locally optimal value of σ for each network configuration for the selected number of classes. Probabilistic neural networks configured with these locally optimal σ parameters were then undertrained with 30% and 40% of the signatures, respectively, sampled at regular intervals to indicate the most likely number of classes. A plot of the percentage classification accuracy as a function of the increasing number of classes c revealed a slight decrease in classification accuracy of approximately 5% as the number of classes increased from $c = 2$ to $c = 3$, followed by a significant decrease of approximately 25% as the number of classes increased from $c = 3$ to $c = 4$ that did not recover as the number of classes increased to $c = 10$. Therefore, this significant decline in percentage classification accuracy indicated that there are at least two and most likely three distinct classes in Record 11020219.

A probabilistic neural network configured with the locally optimal σ is then sufficiently trained with 50% of the 4-dimensional signatures sampled at regular intervals from the trajectory, and achieved a representative correct classification accuracy of approximately 95% when classifying previously unobserved traffic signatures.

The methodologies that are presented for characterizing and classifying traffic may be directly applied to other classes of self-affine traffic that are of interest to TR*Labs* and its sponsors, thereby satisfying the mandate of TR*Labs* and the objectives of this thesis.

7.2 Contributions

This thesis and the research done towards its completion has provided the following contributions.

1. The capture of 24 hours of VoIP data at TR*Labs* (2,304,156 UDP packets), and the visualized demonstration of the self-affine nature of VoIP traffic.

2. The demonstration of the self-affine, or *fractal*, nature of the agonistic behaviour of the Siamese Fighting Fish (*Betta splendens*) through a comprehensive study of Record 11020219 from Pear's data sets.

3. The demonstration of the temporal multifractal nature of the *Betta splendens* through the calculation of the variance fractal dimension trajectory.

4. The demonstration of the spatial multifractal nature of the *Betta splendens* through the calculation of the Rényi multifractal dimension spectrum.

5. The selection of an appropriate window size and window offset for the calculation of the variance fractal dimension.

6. The selection of an appropriate window size and window offset for the construction of the statistical histograms of the variance fractal dimension trajectory.

7. The modelling of the statistical histograms of the variance fractal dimension trajectory using the gamma distribution.

8. The determination of the most likely number of classes in Record 11020219 by studying the performance of the probabilistic neural network.

9. The training and testing of a probabilistic neural network to accurately classify previously unobserved self-affine traffic.

7.3 Recommendations for Future Work

Based on the work done in this thesis, several recommendations are presented for future work in this area.

1. A study of the sections of Record 11020219 where the mean square error (MSE) is greater than 0.1. Of the 278 windows where the MSE error was calculated (to generate Fig. 5.9), only 10 had a MSE greater than 0.1, located at window positions 2, 3, 4, 5, 7, 8, 35, 55, 80, and 95. Since 6 of these 10 windows are located at the very beginning of Record 11020219, further study of these sections would contribute to a better understanding of the self-affine nature of the motion of *Betta splendens*.

2. A study of the Kullback-Leibler distance [KuLe51] to measure the accuracy of the gamma distribution in modelling the statistical histograms of the variance fractal dimension trajectory. Relationships may exist between the sections of Record 11020219 where the Kullback-Leibler distance is significantly high, the MSE is greater than 0.1, and the variance fractal dimension is close to 2. Furthermore, a better measurement of the number of points needed to construct the statistical histograms may be uncovered through the calculation of the Kullback-Leibler distance trajectory.

3. A refinement in the usage of the probabilistic neural network by construction of the training and test set pairs through representative random sampling.

4. An extension of the probabilistic neural network by incorporating a spread parameter σ for each class (as shown in Eq. 4.36), and selecting optimal values for each parameter.

5. A comparison between these results (achieved when the first four principal components are used), and the results obtained when the first three or five principal components are used, respectively.

6. A study of the Rényi multifractal dimension spectrum trajectory and how it could be used to improve the multifractal characterization.

7. The characterization of the Z-coordinates of Experiment 11020219 and the construction of a 4D signature trajectory for the Z-coordinates in the same way as presented for the X-coordinates. The study of the correlation between the 4D-X and 4D-Z trajectories may lead to better characterization and classification of the traffic [CaCh03].

8. An independent study by Dr. Pear's research group of the behaviour of the *Betta splendens* as recorded by the regular video camera for Experiment 11020219 could possibly result in a temporal behavioural classification trajectory based solely upon the characteristic features visually extracted from this video. A direct comparison between this behavioural classification trajectory and the statistical classification trajectory constructed in this thesis would be an excellent method of independent verification of the development of the statistical classification trajectory. This comparison might also suggest possible refinements of the techniques and methodologies presented by this thesis.

9. The application of these techniques and methodologies in characterizing and classifying other experiments from Pear's data sets.

10. The application of these techniques and methodologies in characterizing and classifying Sarda's data sets.

REFERENCES

[Abou03] About, Inc., "aquarium fish profile photo of the Siamese Fighting Fish (Betta Splendens)," 2003. http://freshaquarium.about.com/library/profiles/blfw4071.htm

[BaKi02] R. L. Barry and W. Kinsner, "Multifractal characterization for classification of telecommunications traffic," *Proc. IEEE CCECE '02*, pp. 1538-1544, May 2002.

[Beau02] N. C. Beaulieu, "Introduction to 'Certain topics in telegraph transmission theory'," *Proc. IEEE*, vol. 90, no. 2, pp. 276-279, Feb. 2002.

[Benv93] N. Benvenuto, "A speech / voiceband data discriminator," *IEEE Trans. Communications*, vol. 31, no. 4, pp. 539-543, Apr. 1993.

[BKPM03] R. L. Barry, W. Kinsner, J. Pear, and T. Martin, "Multifractal characterization for classification of self-affine signals," *Proc. IEEE CCECE '03*, May 2003.

[Bish00] C. M. Bishop, *Neural Networks for Pattern Recognition.* New York, NY: Oxford University Press, 482 pp., 2000.

[BlMe70] L. M. Blumenthal and K. Menger, *Studies in Geometry.* San Francisco, CA: W. H. Freeman, 512 pp., 1970.

[Bron94] P. M. Bronstein, "On the predictability, sensitization, and habituation of aggression in male Bettas (*Betta splendens*)," *J. Comparative Psychology*, vol. 108, no. 1, pp. 45-57, 1994.

[Brow28] R. Brown, "A brief account of microscopical observations made in the months of June, July, and August 1827, on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies," *The London and Edinburgh Philosophical Magazine and Annals of Philosophy*, vol. 4, no. 21, pp. 161-173, 1828.

[BSTW95] J. Beran, R. Sherman, M. S. Taqqu, and W. Willinger, "Long-range dependence in Variable-Bit-Rate video traffic," *IEEE/ACM Trans. Communications*, vol. 43, no. 2, pp. 1566-1579, 1995.

[CaCh03] K. Cannons and V. Cheung, "Signal classification through multifractal analysis and neural networks," B.Sc. thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 2003.

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

[ChBa97] K. Christensen and V. Ballingam, "Reduction of self-similarity by application-level traffic shaping," *Proc. IEEE Conf. Local Computer Networks*, pp. 511-518, 1997.

[Chen97] H. Chen, "Accuracy of fractal and multifractal measures for signal analysis," M.Sc thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 1997.

[Chen02] J. Chen, "Classification of transients in power systems," M.Sc. thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 2002.

[CiPo96] K. Ciesielski and Z. Pogoda, "The beginning of Polish topology," *The Mathematical Intelligencer*, vol. 18, no. 3, pp. 32-39, 1996.

[CoHa67] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Information Theory*, vol. 13, no. 1, pp. 21-27, Jan. 1967.

[CoSa98] B. F. Cockburn and D. P. Sarda, "Implementation and evaluation of an accurate real-time voiceband signal classifier," *Proc. IEEE CCECE '98*, pp. 133-136, May 1998.

[CrBe97] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 835-846, Dec. 1997.

[CuBC95] C. A. Cunha, A. Bestavros, and M. E. Crovella, Boston University Department of Computer Science, Boston, MA, *Tech. Rep. TR-95-010*, Apr. 1995.

[Dans95] R. M. Dansereau, "Codebook image compression with neural networks," B.Sc. thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 1995.

[Dans01] R. M. Dansereau, "Progressive image transmission using fractal and wavelet techniques with image complexity measures," Ph.D. thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 2001.

[Deva92] R. L. Devaney, *A First Course in Chaotical Dynamical Systems: Theory and Experiments*. Reading, MA: Addison-Wesley, 302 pp., 1992.

[DMPS03a] P. Danzig, J. Mogul, V. Paxson, and M. Schwartz, The Internet Traffic Archive, "BC - Ethernet Traces of LAN and WAN Traffic," 2003. http://ita.ee.lbl.gov/html/contrib/BC.html

[DMPS03b] P. Danzig, J. Mogul, V. Paxson, and M. Schwartz, The Internet Traffic Archive, "BU-Web-Client - Six Months of Web Client," 2003. http://ita.ee.lbl.gov/html/contrib/BU-Web-Client.html

[DuHa73] R. O. Duda and P. E. Hart, *Pattern classification and scene analysis*. New York, NY: John Wiley & Sons, 482 pp., 1973.

[DuWi94] D. E. Duffy and W. Willinger, "Statistical analysis of CCSN / SS7 traffic data from working CCS subnetworks," *IEEE J. Selected Areas in Communications*, 1994.

[Edga90] G. A. Edgar, *Measure, Topology, and Fractal Geometry*. New York, NY: Springer-Verlag, 230 pp., 1990.

[Ehti99] T. Ehtiati, "Multifractal characterization of electromyogram signals," M.Sc. thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 1999.

[FoLe91] H. J. Fowler and W. E. Leland, "Local area network traffic characteristics, with implications for broadband network congestion management," *IEEE J. Selected Areas in Communications*, vol. 9, no. 7, pp. 1139-1149, Sep. 1991.

[FoWi98] F. M. Foo and C. L. Williamson, "Network traffic measurements of IP / Frame Relay / ATM," *Proc. Workshop on Workload Characterization in High Performance Computing Environments*, pp. 1-14, July 1998.

[GaWi94] M. W. Garrett and W. Willinger, "Analysis, modeling and generation of self-similar VBR video traffic," *Proc. ACM SIGCOMM '94*, pp. 269-280, Sep. 1994.

[Glei87] J. Gleick, *Chaos: Making a New Science*. New York, NY: Viking Penguin, 352 pp., 1987.

[Grie96] W. S. Grieder, "Variance fractal dimension for signal feature enhancement and segmentation from noise," M.Sc. thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 1996.

[Hart75] J. A. Hartigan, *Clustering Algorithms*. New York, NY: John Wiley & Sons, 351 pp., 1975.

[HePr83] H. G. E. Hentschel and I. Procaccia, "The infinite number of generalized dimensions of fractals and strange attractors," *Physica D: Nonlinear Phenomena*, vol. 8D, no. 1 & 2, pp. 435-444, July 1983.

[HyKO01] A. Hyvarinen, J. Karhunen, and E. Oja, *Independent Component Analysis.* New York, NY: John Wiley & Sons, 481 pp., 2001.

[KaWo99] K. Kant and Y. Won, "Server capacity planning for web traffic workload," *IEEE Trans. Knowledge and Data Engineering*, vol. 11, no. 5, pp. 731-747, Sep. 1999.

[Kins94a] W. Kinsner, "Fractal dimensions: Morphological, entropy, spectrum, and variance classes," University of Manitoba and TR*Labs*, Winnipeg, MB, Canada, *Tech. Rep. DEL94-4*, 1994.

[Kins94b] W. Kinsner, "The Hausdorff-Besicovitch dimension formulation for fractals and multifractals," University of Manitoba and TR*Labs*, Winnipeg, MB, Canada, *Tech. Rep. DEL94-7*, 1994.

[Koch04] H. von Koch, "Sur une courbe continue sans tangente obtenue par une construction géometrique élémentaire," *Arkiv för Matematik, Astronomi och Fysik*, vol. 1, pp. 681-702, 1904.

[Koho88] T. Kohonen, *Self-organization and associative memory.* Berlin, Germany: Springer-Verlag, 2nd ed., 312 pp., 1988.

[Koho90] T. Kohonen, "The self-organizing map," *Proc. IEEE*, vol. 78, no. 9, pp. 1464-1480, Sep. 1990.

[KuLe51] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79-86, Mar. 1951.

[LeWi91] W. E. Leland and D. V. Wilson, "High time-resolution measurements and analysis of LAN traffic: Implications for LAN interconnection," *Proc. IEEE INFOCOM '91*, pp. 1360-1366, Apr. 1991.

[LLTW94] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of Ethernet traffic," *IEEE/ACM Trans. Networking*, vol. 2, no. 1, pp. 1-15, Feb. 1994.

[MacQ67] J. MacQueen, "Some methods for classification and analysis of multivariate observations," *Proc. Fourth Berkeley Symposium on Mathematical Statistics and Probability*, University of California Press, Berkeley, vol. 1, pp. 281-297, 1967.

[Mand67] B. B. Mandelbrot, "How long is the coast of Britain? Statistical self-similarity and fractal dimension," *Science*, vol. 156, pp. 636-638, May 1967.

[Mand69] B. B. Mandelbrot, "Long-run linearity, locally Gaussian processes, H-spectra and infinite variances," *Intern. Economic Rev.*, vol. 10, pp. 82-113, 1969.

[Mand71] B. B. Mandelbrot, "A fast fractional Gaussian noise generator," *Water Resources Research*, vol. 7, no. 3, pp. 543-553, 1971.

[Mand75] B. B. Mandelbrot, *Les Objects Fractals: Forme, hasard et dimension.* Paris, France: Flammarion, 192 pp., 1975.

[Mand83] B. B. Mandelbrot, *The Fractal Geometry of Nature*. New York, NY: W. H. Freeman, 468 pp., 1983.

[Mand85] B. B. Mandelbrot, "Self-affine fractals and fractal dimensions," *Physica Scripta*, vol. 32, pp. 257-260, 1985.

[Mand02] B. B. Mandelbrot, *Gaussian Self-Affinity and Fractals.* New York, NY: Springer-Verlag, 654 pp., 2002.

[MaPe02] G. Martin and J. Pear, *Behavior Modification: What it is and How to do it*. Upper Saddle River, NJ: Prentice Hall, 7th ed., 470 pp., 2002.

[Mart02] T. Martin, "Attempts to produce dishabituation of agonistic behavior in *Betta splendens*," Internal Rep., University of Manitoba, Winnipeg, Manitoba, Canada, June 2002.

[MaSu93] P. Maragos and F.-K. Sun, "Measuring the fractal dimension of signals: morphological covers and iterative optimization," *IEEE Trans. Signal Processing*, vol. 41, no. 1, pp. 108-121, Jan. 1993.

[Math03a] The Mathworks, Inc., "skewness (Statistics Toolbox)," 2003. http://www.mathworks.com/access/helpdesk/help/toolbox/stats/skewness.shtml

[Math03b] The Mathworks, Inc., "kurtosis (Statistics Toolbox)," 2003. http://www.mathworks.com/access/helpdesk/help/toolbox/stats/kurtosis.shtml

[MaVa68] B. B. Mandelbrot and J. W. Van Ness, "Fractional Brownian motions, fractional noises and applications," *SIAM Rev.*, vol. 10, no. 4, pp. 422-437, 1968.

[MaWa68] B. B. Mandelbrot and J. R. Wallis, "Noah, Joseph, and Operational Hydrology," *Water Resources Research*, vol. 4, no. 5, pp. 909-918, 1968.

[MaWa69a] B. B. Mandelbrot and J. R. Wallis, "Computer experiments with fractional Gaussian noises. Part 1, Averages and variances," *Water Resources Research*, vol. 5, no. 1, pp. 228-241, Feb. 1969.

[MaWa69b] B. B. Mandelbrot and J. R. Wallis, "Computer experiments with fractional Gaussian noises. Part 2, Rescaled ranges and spectra," *Water Resources Research*, vol. 5, no. 1, pp. 242-259, Feb. 1969.

[MaWa69c] B. B. Mandelbrot and J. R. Wallis, "Computer experiments with fractional Gaussian noises. Part 3, Mathematical appendix," Water Resources Research, vol. 5, no. 1, pp. 260-267, Feb. 1969.

[McSH90] R. D. McLeod, J. J. Schellenberg, and P. D. Hortensius, "Percolation and anomalous transport as tools in analyzing parallel processing interconnection networks," *J. Parallel and Distributed Computing*, vol. 8, pp. 376-387, 1990.

[Meis72] W. S. Meisel, *Computer-oriented approaches to pattern recognition.* New York, NY: Academic Press, 250 pp., 1972.

[Meng26] K. Menger, "Allgemeine Räume und charakteristische Räume, Zweite Mitteilung: Über umfassenste *n*-dimensionale Mengen," *Proc. Royal Academy of Science (Amsterdam)*, vol. 29, pp. 1125-1128, 1926.

[Nyqu24] H. Nyquist, "Certain factors affecting telegraph speed," *Bell System Tech. J.*, vol. 3, pp. 324-346, Apr. 1924.

[Nyqu28] H. Nyquist, "Certain topics in telegraph transmission theory," *Trans. AIEE*, vol. 47, pp. 617-644, Apr. 1928.

[PaFl95] V. Paxson and S. Floyd, "Wide-area traffic: the failure of the Poisson modeling," *IEEE/ACM Trans. Networking*, vol. 3, no. 3, pp. 226-244, June 1995.

[PaKC96] K. Park, G. T. Kim, and M. E. Crovella, "On the relationship between file sizes, transport protocols, and self-similar network traffic," *Proc. IEEE Intern. Conf. Network Protocols*, pp. 171-180, 1996.

[PaKC97] K. Park, G. T. Kim, and M. E. Crovella, "On the effect of traffic self-similarity on network performance," *Proc. SPIE Intern. Conf. Performance and Control of Network Systems*, pp. 296-310, 1997.

[Parz62] E. Parzen, "On estimation of a probability density function and mode," *Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065-1076, Sep. 1962.

[PaWi00] K. Park and W. Willinger, Ed., *Self-Similar Network Traffic and Performance Evaluation.* New York, NY: John Wiley & Sons, 558 pp., 2000.

[Pear01] J. J. Pear, *The Science of Learning.* Philadelphia, PA: Psychology Press, 524 pp., 2001.

[PeJS92] H.-O. Peitgen, H. Jürgens, and D. Saupe, *Chaos and Fractals: New Frontiers of Science*. New York, NY: Springer-Verlag, 984 pp., 1992.

[PeMa02] J. J. Pear and T. Martin (private communication), 07 June 2002.

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

[Rény59] A. Rényi, "On the dimension and entropy of probability distributions," *Acta Mathematica Academiae Scientiarum Hungaricae*, vol. 10, pp. 193-226, 1959.

[Rény61] A. Rényi, "On measures of entropy and information," *Proc. Fourth Berkeley Symposium on Mathematical Statistics and Probability*, University of California Press, Berkeley, vol. 1, pp. 547-561, 1961.

[Rifa98] R. Rifaat, "Multifractal analysis of DNA," M.Sc. thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 1998.

[Sard99] D. P. Sarda, "Implementation and evaluation of an accurate real-time voiceband signal classifier," M.Sc. thesis, Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada, 1999.

[SaTe99] Z. Sahinoglu and S. Tekinay, "On multimedia networks: self-similar traffic and network performance," *IEEE Communications*, pp. 48-52, Jan. 1999.

[Sewa96] J. S. Sewall, "Signal classification in digital telephone networks," M.Sc. thesis, Department of Electrical Engineering, University of Alberta, Edmonton, AB, Canada, 1996.

[Shan48a] C. E. Shannon, "A mathematical theory of communications," *Bell System Tech. J.*, vol. 27, pp. 379-423, July 1948.

[Shan48b] C. E. Shannon, "A mathematical theory of communications," (continued) *Bell System Tech. J.*, vol. 27, pp. 623-656, Oct. 1948.

[Shan49] C. E. Shannon, "Communication in the presence of noise," *Proc. Institute of Radio Engineers*, vol. 37, no. 1, pp. 10-21, Jan. 1949.

[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, Winnipeg, MB, Canada, 1997.

[ShSh71] S. Shapiro and H. Shuckman, "Habituation and covariation of the components of the threat display in *Betta splendens*," *Psychology Reports*, vol. 28, pp. 827-837, 1971.

[Sier15] W. Sierpinski, "Sur une courbe dont tout point est un point de ramification," *Comptes Rendus Académie des Sciences (Paris)*, vol. 160, pp. 302-305, 1915.

[Sier16] W. Sierpinski, "Sur une courbe cantorienne qui contient une image biunivoque et continue de toute courbe donnée," *Comptes Rendus Académie des Sciences (Paris)*, vol. 162, pp. 629-632, 1916.

[SiFe03] A. A. Sirotinsky and O. V. Fedorenko, "Fractal Explorer," 2003. http://www.eclectasy.com/Fractal-Explorer/index.html

[Simp68] M. J. A. Simpson, "The display of the Siamese Fighting Fish, *Betta splendens*," *Animal Behaviour Monographs*, vol. 1, pp. 1-73, 1968.

[SnTr71] E. Snapper and R. J. Troyer, *Metric Affine Geometry.* New York, NY: Academic Press, 435 pp., 1971.

[SoNT99] S. Song, J. K.-Y. Ng, and B. Tang, "On the self-similarity property of the output process from a network server with self-similar input traffic," *Proc. IEEE Intern. Conf. Real-Time Computing Systems and Applications*, pp. 128-132, 1999.

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

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

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

[Sun99] L. Sun, "A fast radio transmitter identification system," M.Sc. thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 1999.

[TaWS97] M. S. Taqqu, W. Willinger, and R. Sherman, "Proof of a fundamental result in self-similar traffic modeling," *ACM SIGCOMM Computer Communication Rev.*, pp. 5-23, 1997.

[TEMM01] D. A. Tikhonov, J. Enderlein, H. Malchow, and A. B. Medvinsky, "Chaos and fractals in fish school motion," *Chaos, Solitons and Fractals*, vol. 12, no. 2, pp. 277-288, 2001.

[ThSp66] R. F. Thompson and W. A. Spencer, "Habituation: A model phenomenon for the study of neuronal substrates of behavior," *Psychology Rev.*, vol. 73, no. 1, pp. 16-43, 1966.

[[Toon97] J. P. Toonstra, "A radio transmitter fingerprinting system," M.Sc. thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 1997.

[Vics92] T. Vicsek, *Fractal Growth Phenomena.* Singapore: World Scientific Publishing, 2nd ed., 488 pp., 1992.

[WaMM98] R. E. Walpole, R. H. Myers, and S. L. Myers, *Probability and Statistics for Engineers and Scientists.* Upper Saddle River, NJ: Prentice Hall, 739 pp., 1998.

[Wass93] P. D. Wasserman, *Advanced Methods in Neural Computing.* New York, NY: Van Nostrand Reinhold, 255 pp., 1993.

[Weis99a] E. W. Weisstein, Eric Weisstein's World of Mathematics, Wolfram Research, Inc., "Gamma Distribution -- from MathWorld," 2003. http://mathworld.wolfram.com/GammaDistribution.html

[Weis99b] E. W. Weisstein, Eric Weisstein's World of Mathematics, Wolfram Research, Inc., "Gamma Function -- from MathWorld," 2003. http://mathworld.wolfram.com/GammaFunction.html

[WTSW95] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level," *Proc. ACM SIGCOMM '95*, pp. 100-113, 1995.

[Zhan02] H. Zhang, "Compositional complexity measures of DNA sequence using multifractal techniques," M.Sc. thesis, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 2002.

[ZhBM90] P. Zhang, H. Barad, and A. Martinez, "Fractal dimension estimation of fractional Brownian motion," *Proc. IEEE Southeastcon '90*, vol. 3, pp. 934-939, 1990.

[Zura92] J. M. Zurada, *Introduction to Artificial Neural Systems.* St. Paul, MN: West Publishing, 683 pp., 1992.

APPENDIX A

PEAR'S DISHABITUATION DATA SETS

This appendix presents the plots for all 13 dishabituation experiments performed by Dr. Pear and his research group at the University of Manitoba.

Dishabituation stimulus : MO = *mirror off*

MC = *mirror off and live conspecific shown*

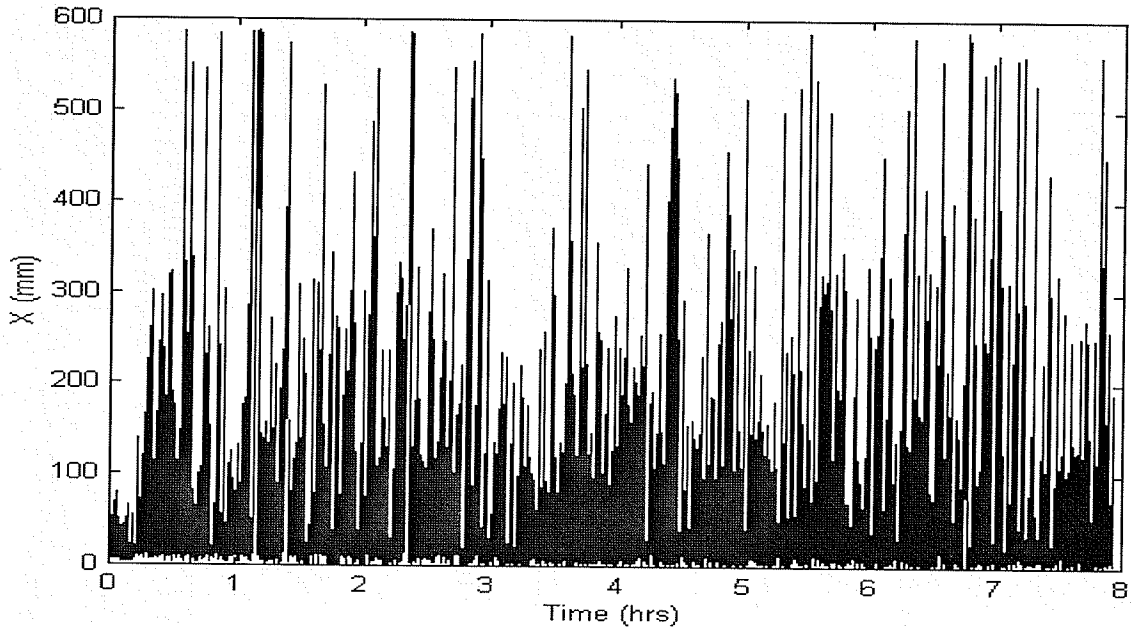

Fig. A.1. X-coordinates of Experiment 11020218 (MO).

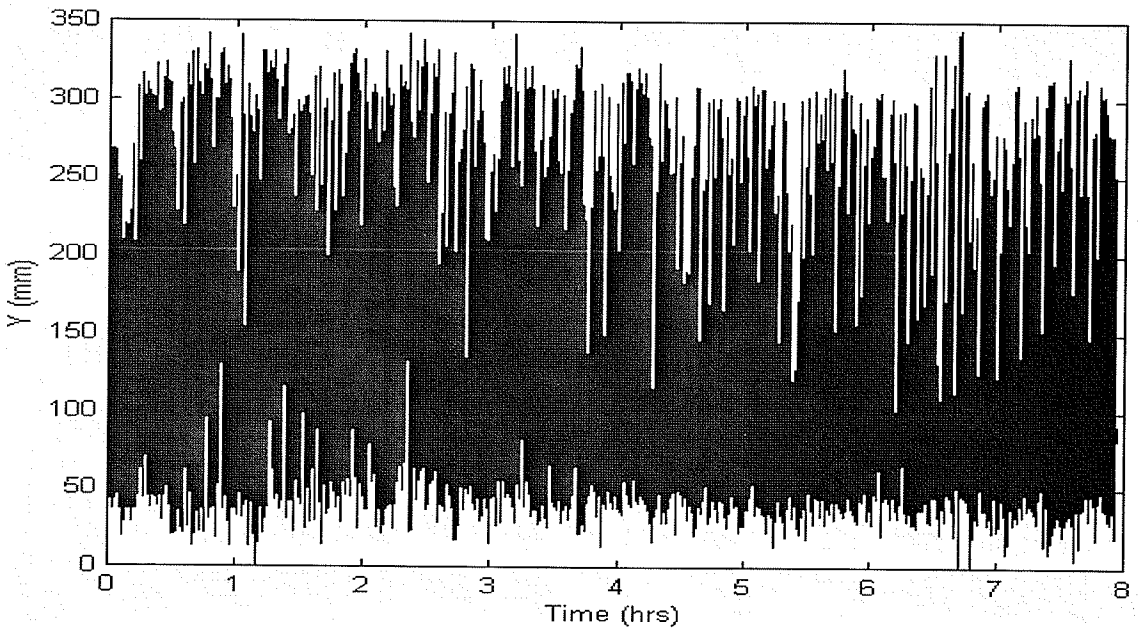

Fig. A.2. Y-coordinates of Experiment 11020218 (MO).

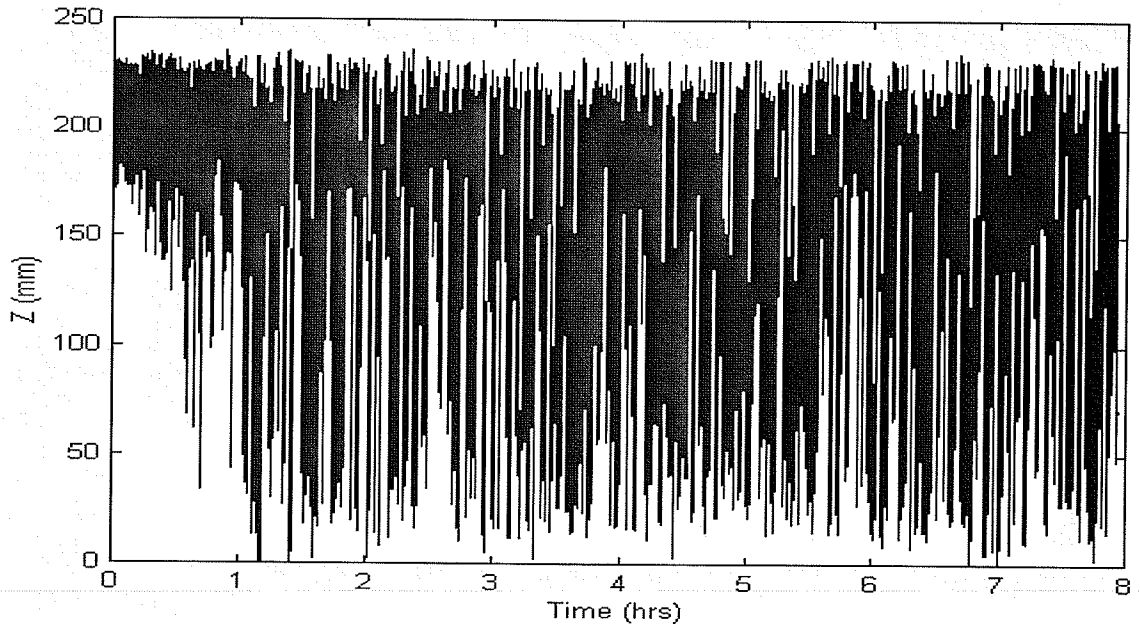

Fig. A.3. Z-coordinates of Experiment 11020218 (MO).

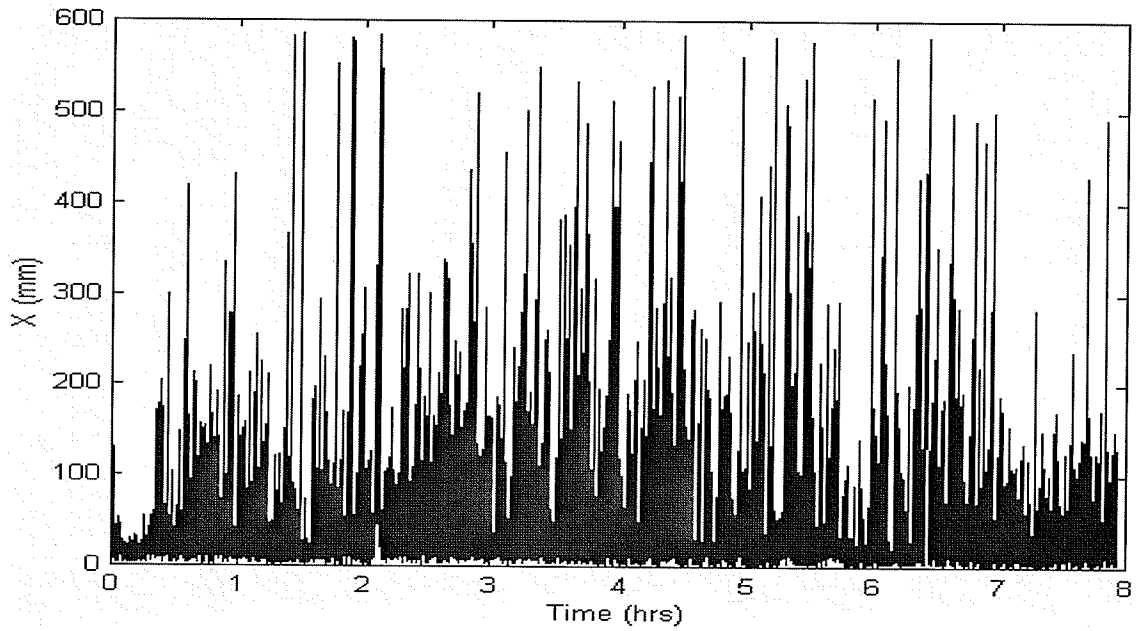

Fig. A.4. X-coordinates of Experiment 11020219 (MO).

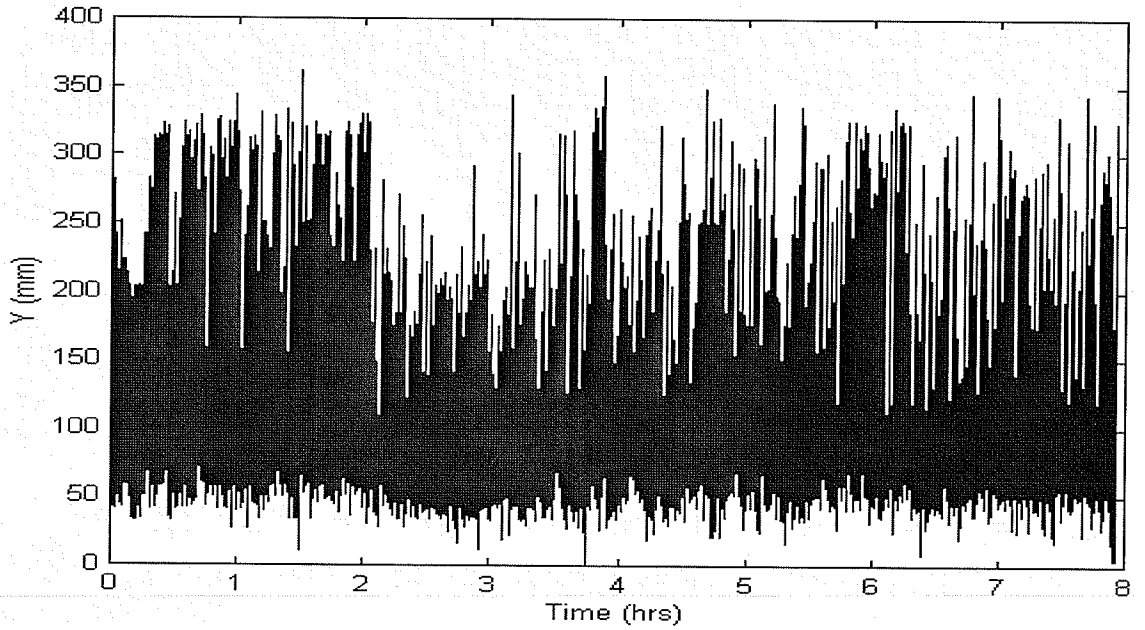

Fig. A.5. Y-coordinates of Experiment 11020219 (MO).

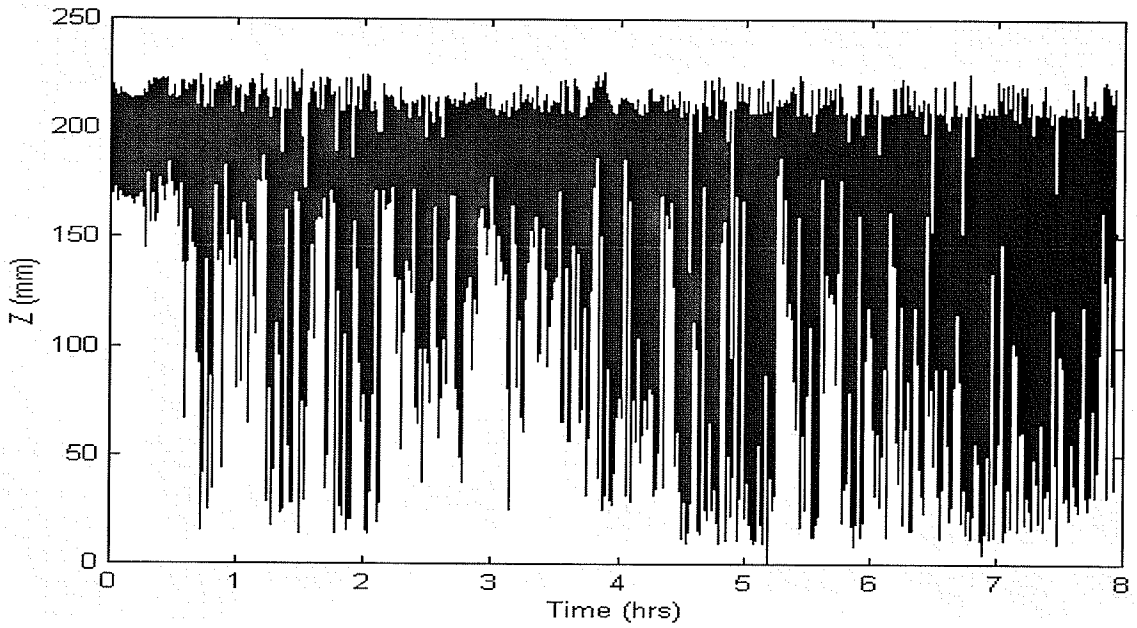

Fig. A.6. Z-coordinates of Experiment 11020219 (MO).

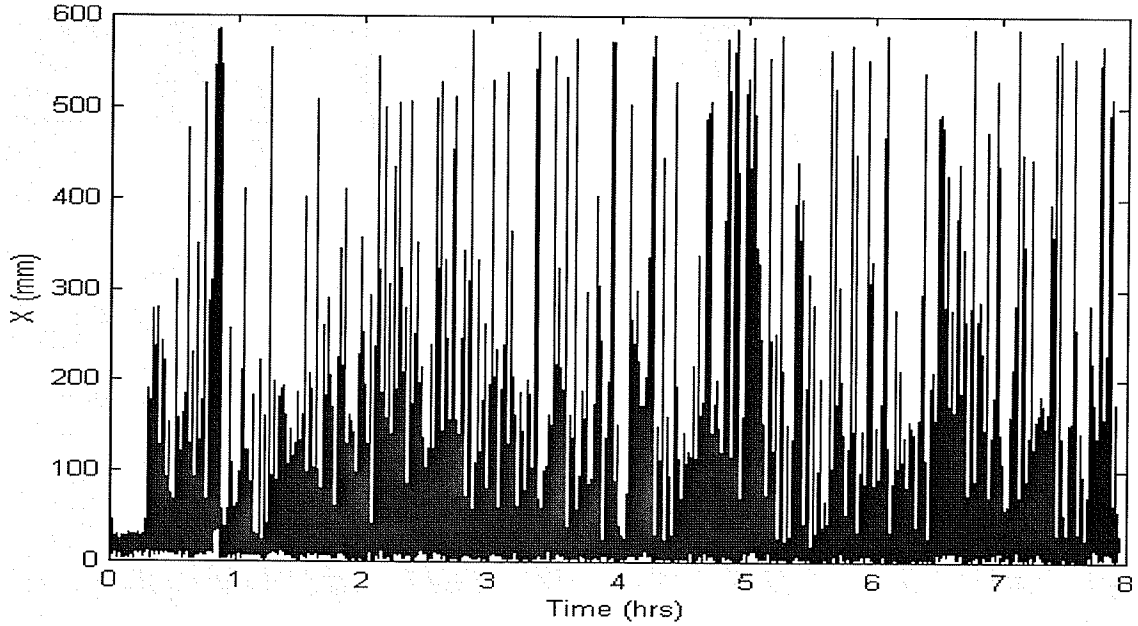

Fig. A.7. X-coordinates of Experiment 11020220 (MO).

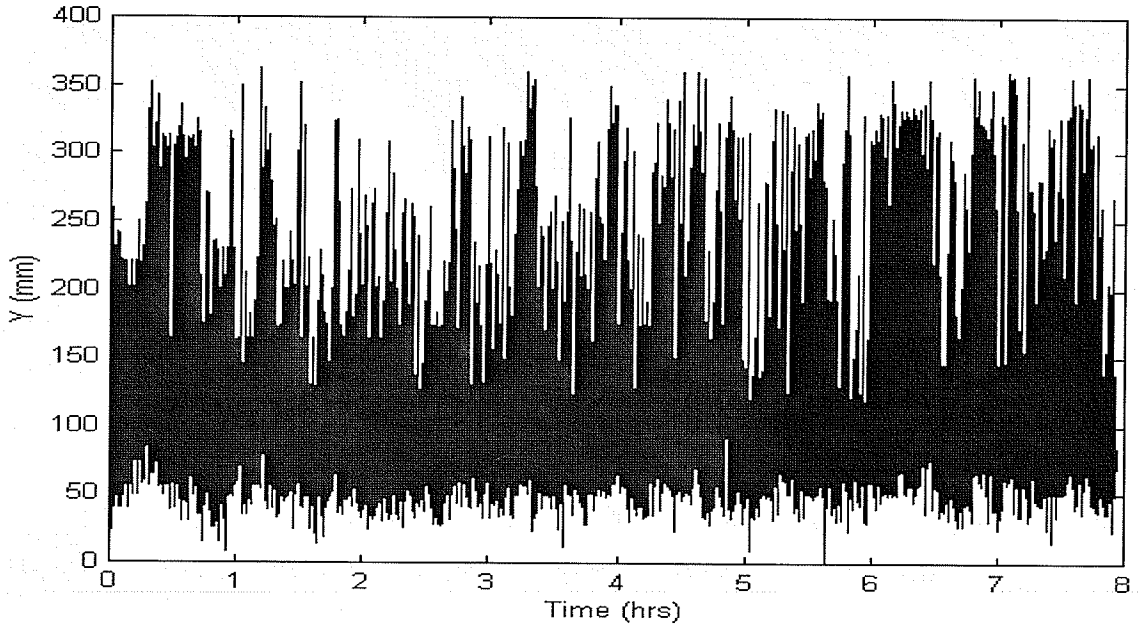

Fig. A.8. Y-coordinates of Experiment 11020220 (MO).

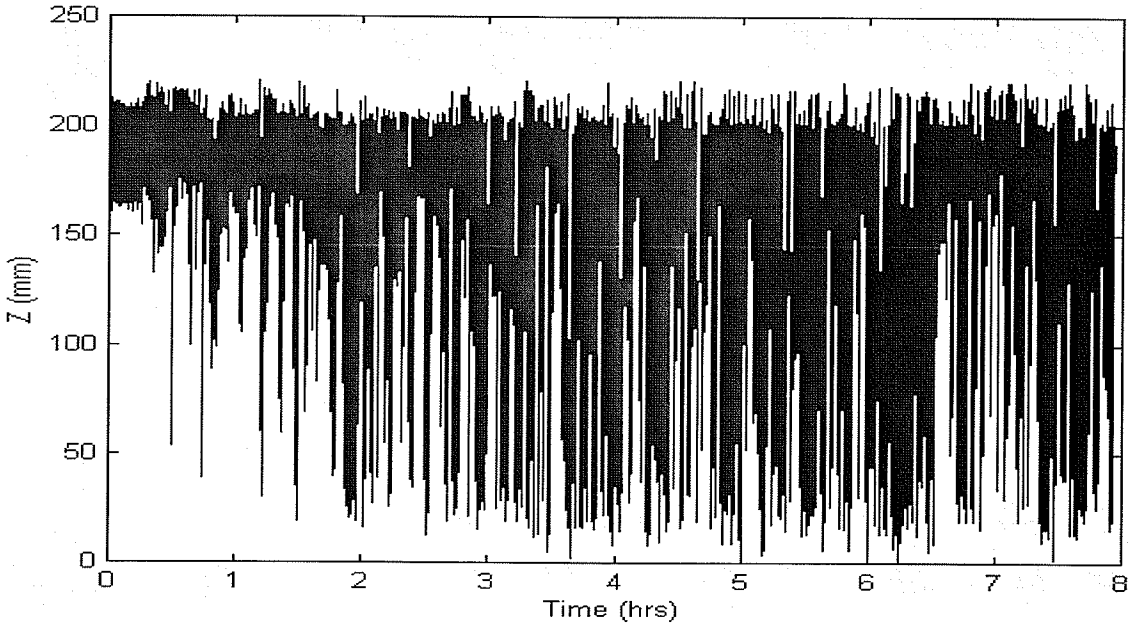

Fig. A.9. Z-coordinates of Experiment 11020220 (MO).

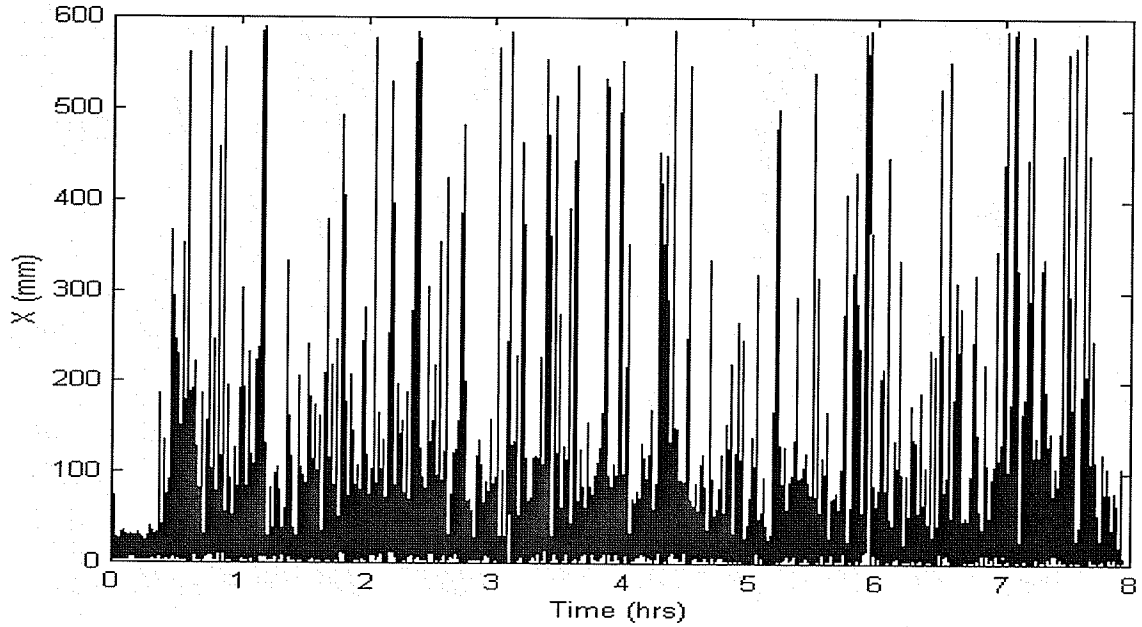

Fig. A.10. X-coordinates of Experiment 11020221 (MC).

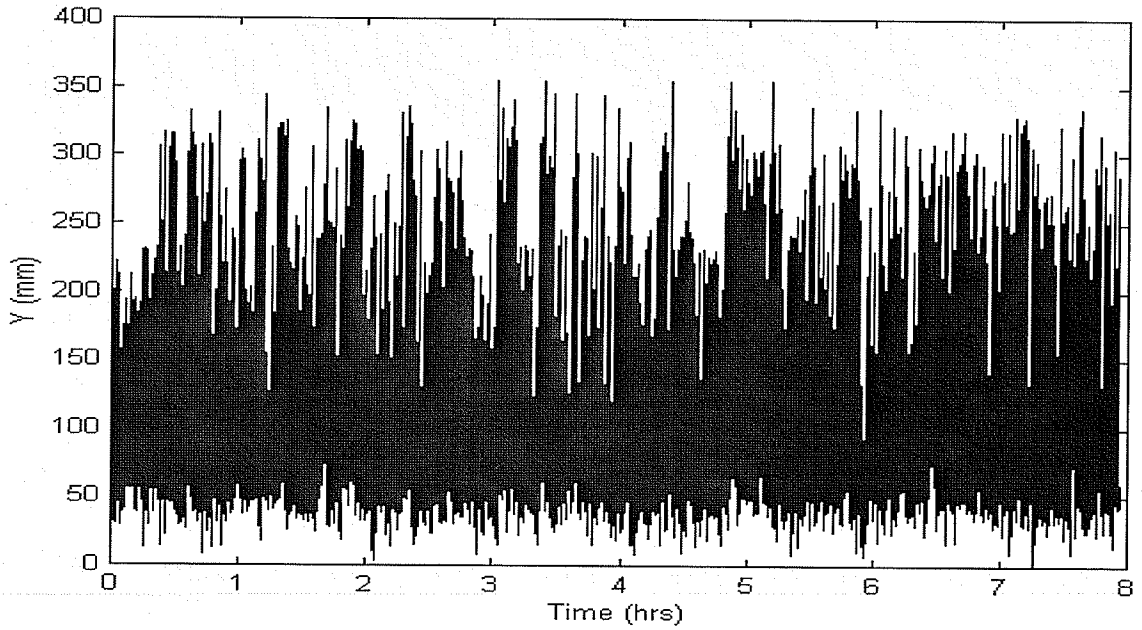

Fig. A.11. Y-coordinates of Experiment 11020221 (MC).

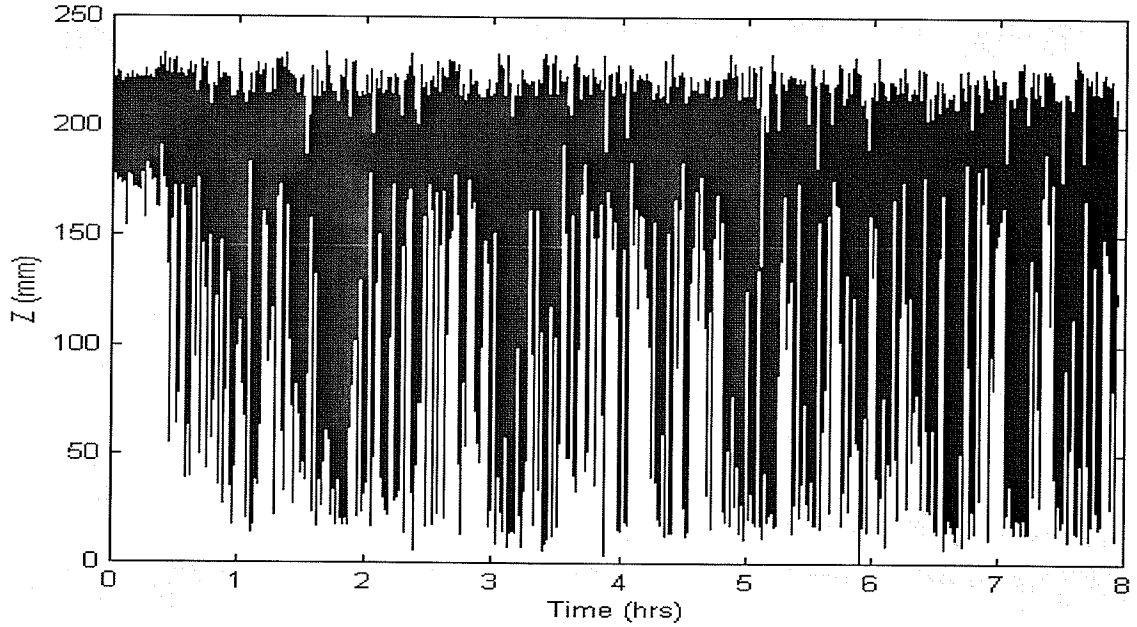

Fig. A.12. Z-coordinates of Experiment 11020221 (MC).

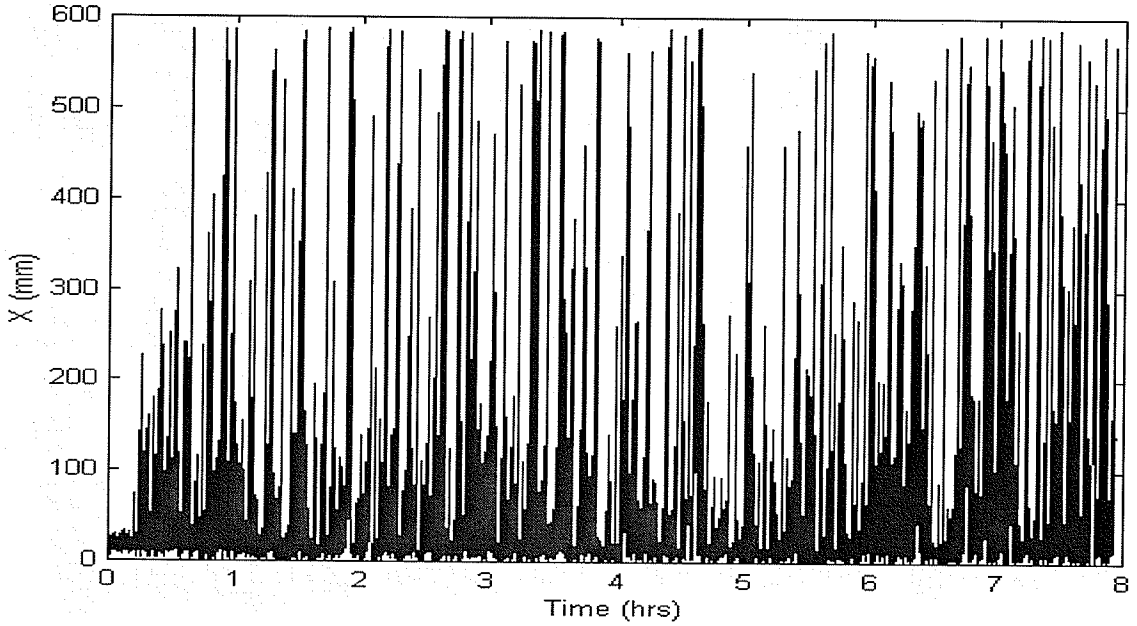

Fig. A.13. X-coordinates of Experiment 11020222 (MC).

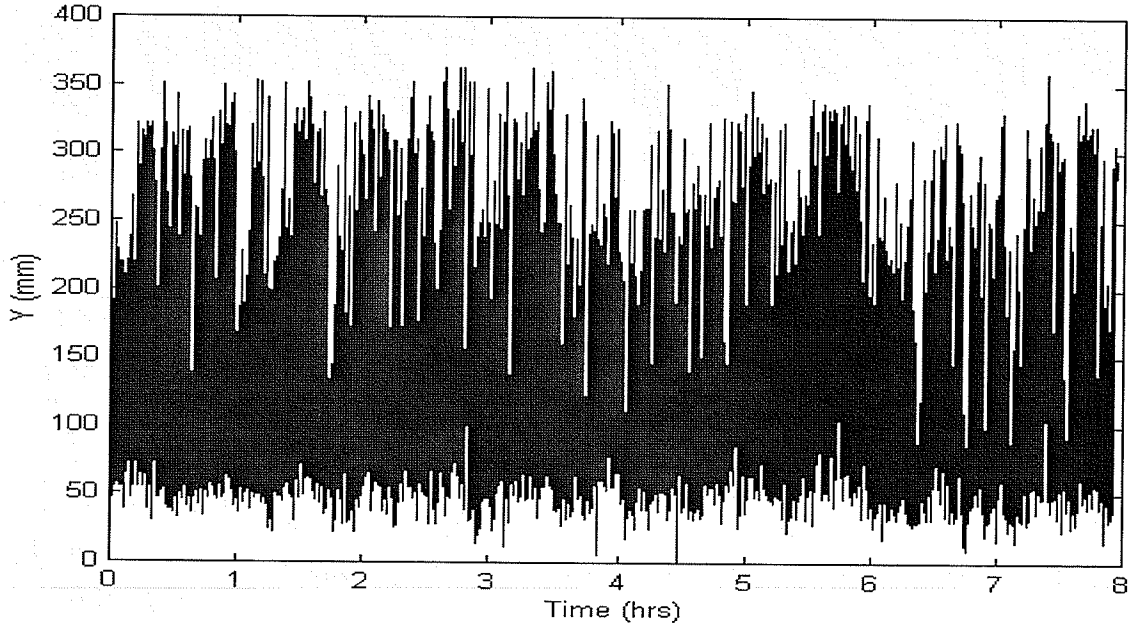

Fig. A.14. Y-coordinates of Experiment 11020222 (MC).

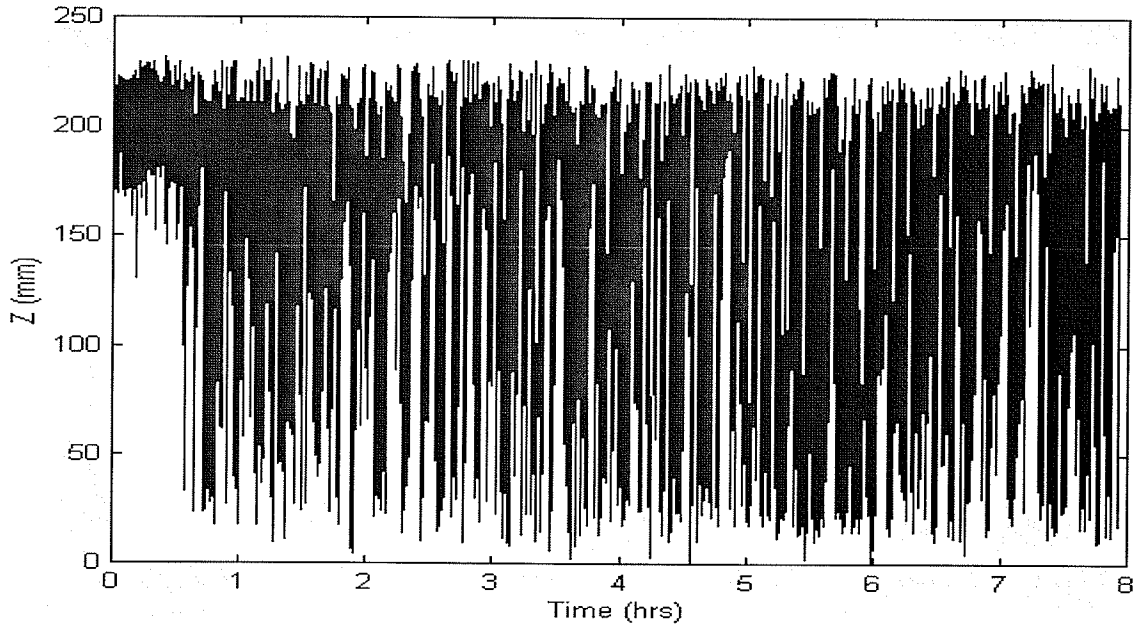

Fig. A.15. Z-coordinates of Experiment 11020222 (MC).

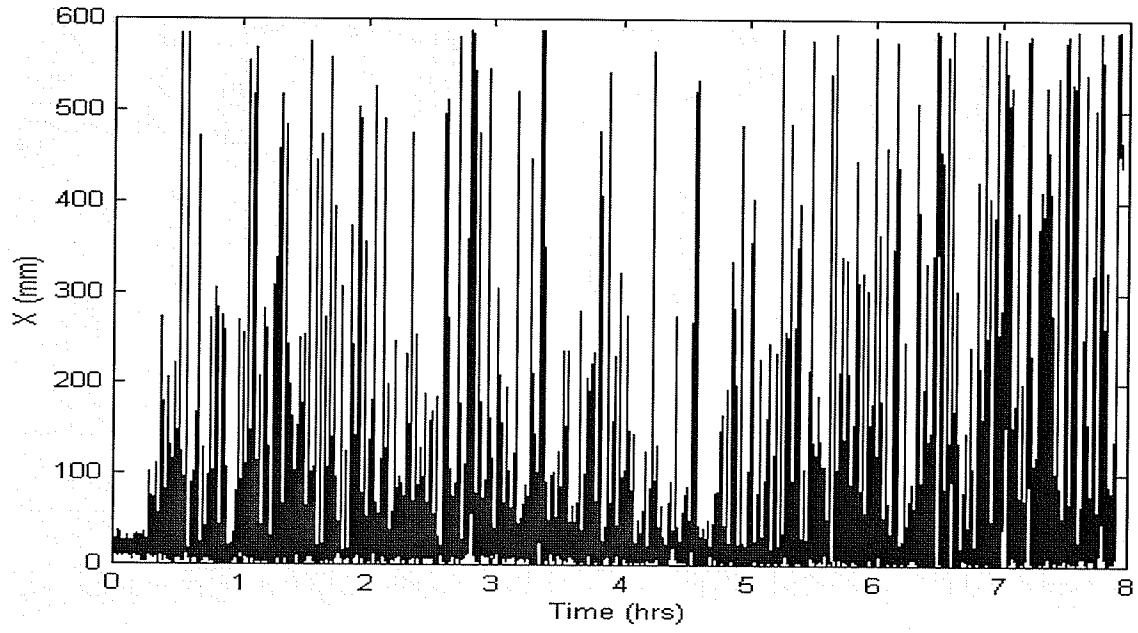

Fig. A.16. X-coordinates of Experiment 11020223 (MC).

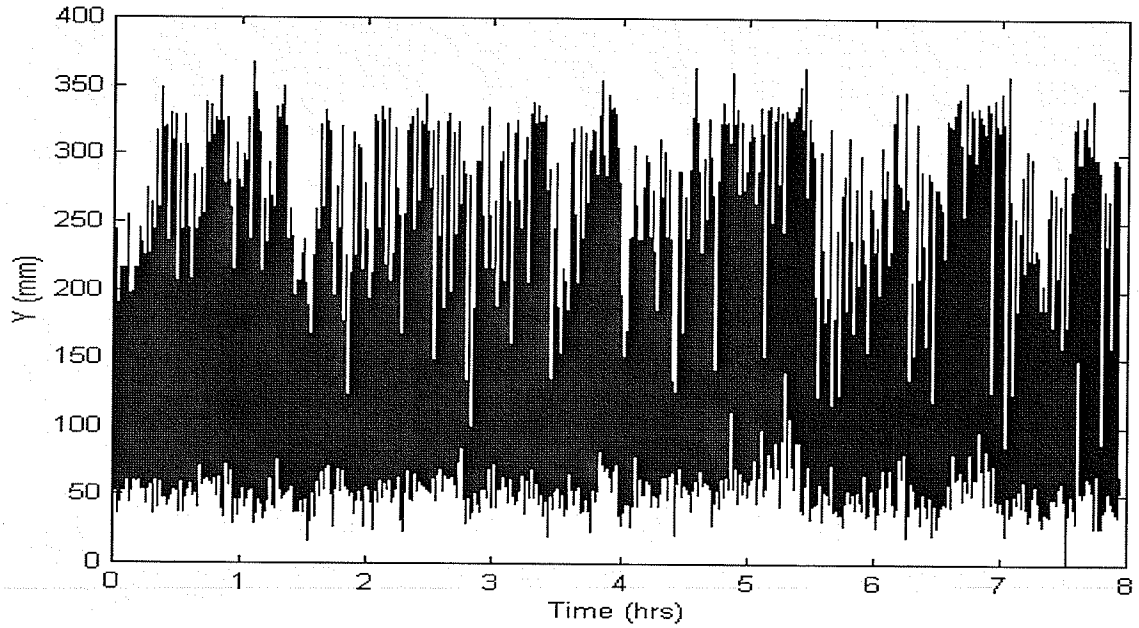

Fig. A.17. Y-coordinates of Experiment 11020223 (MC).

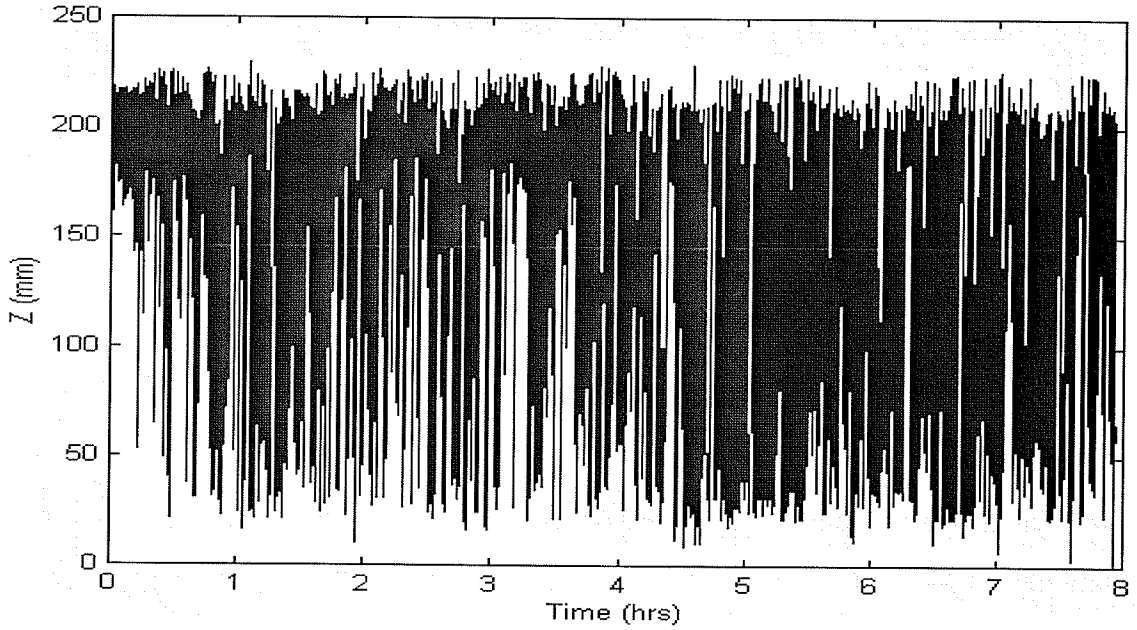

Fig. A.18. Z-coordinates of Experiment 11020223 (MC).

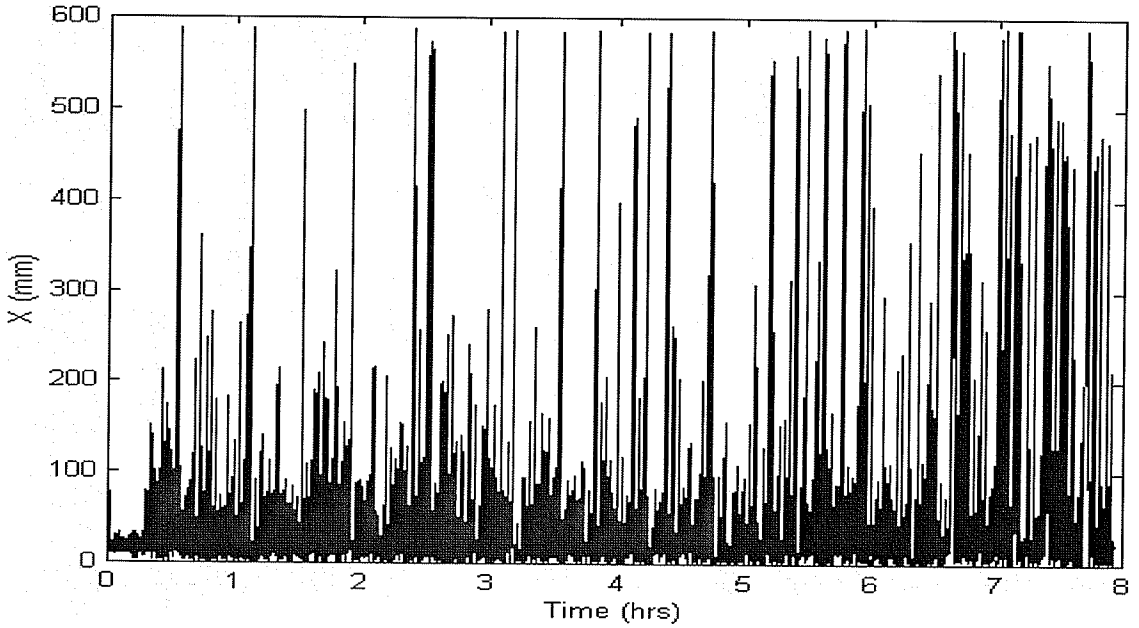

Fig. A.19. X-coordinates of Experiment 11020224 (MC).

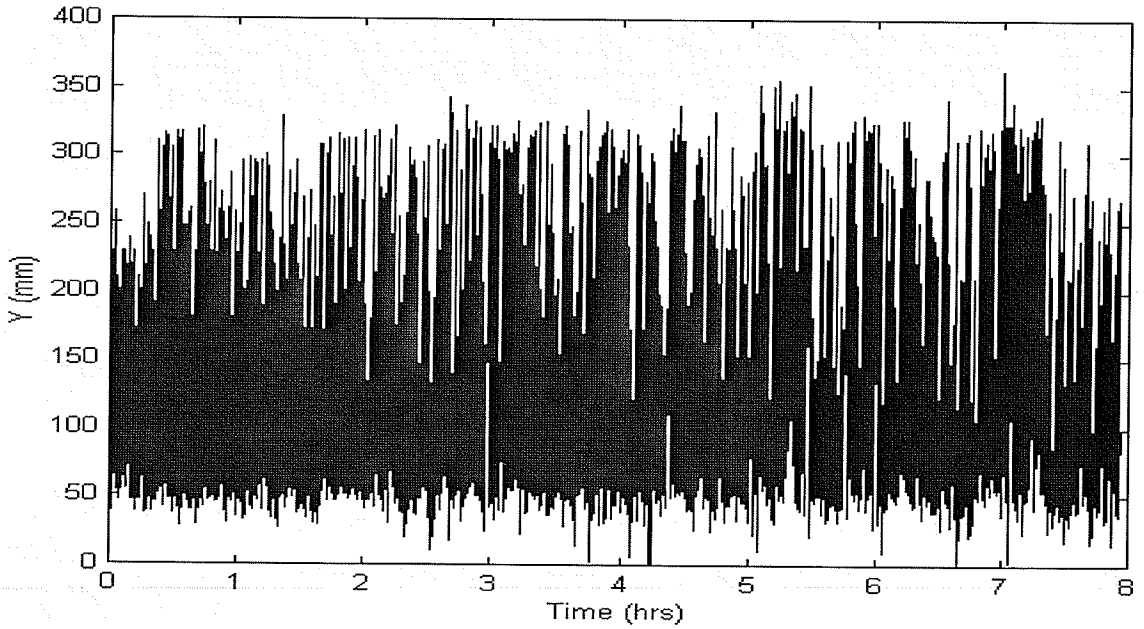

Fig. A.20. Y-coordinates of Experiment 11020224 (MC).

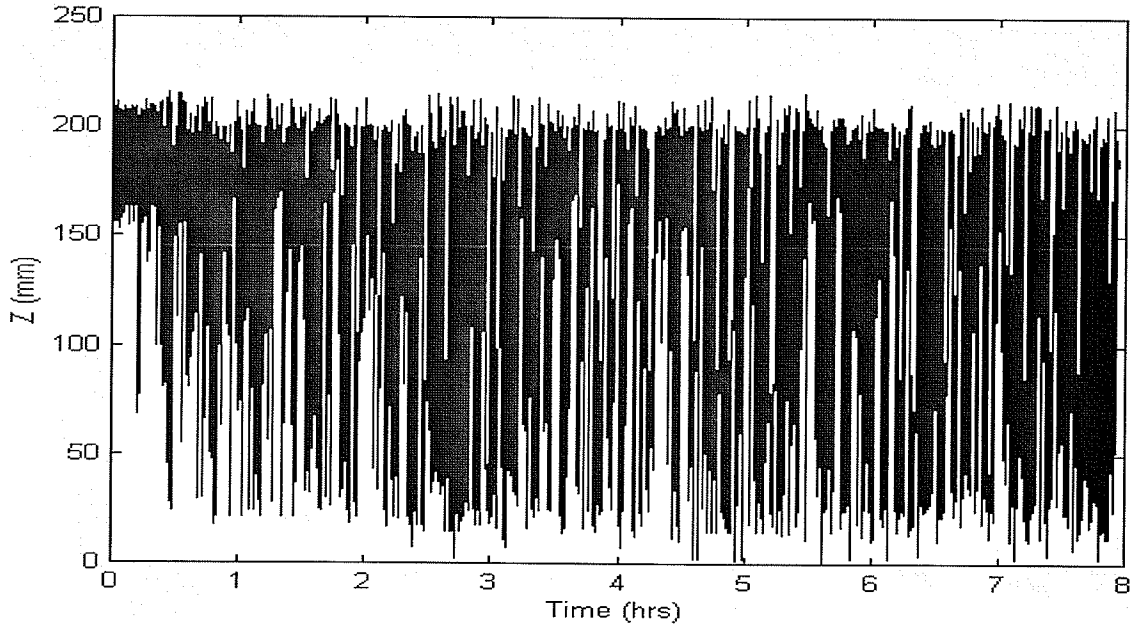

Fig. A.21. Z-coordinates of Experiment 11020224 (MC).

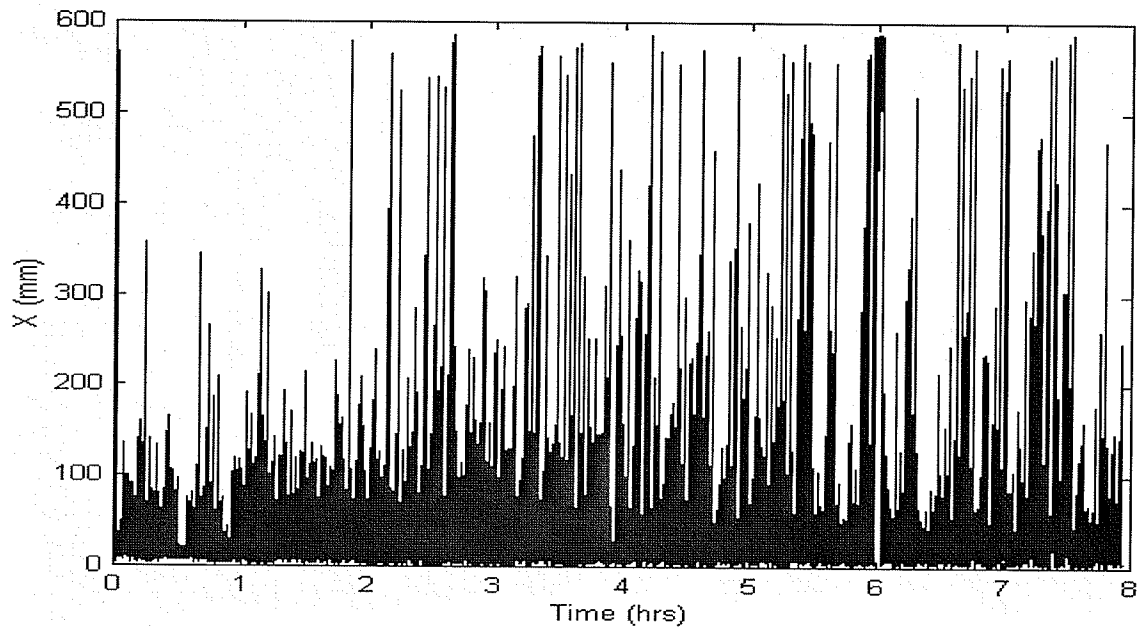

Fig. A.22. X-coordinates of Experiment 14020309 (MO).

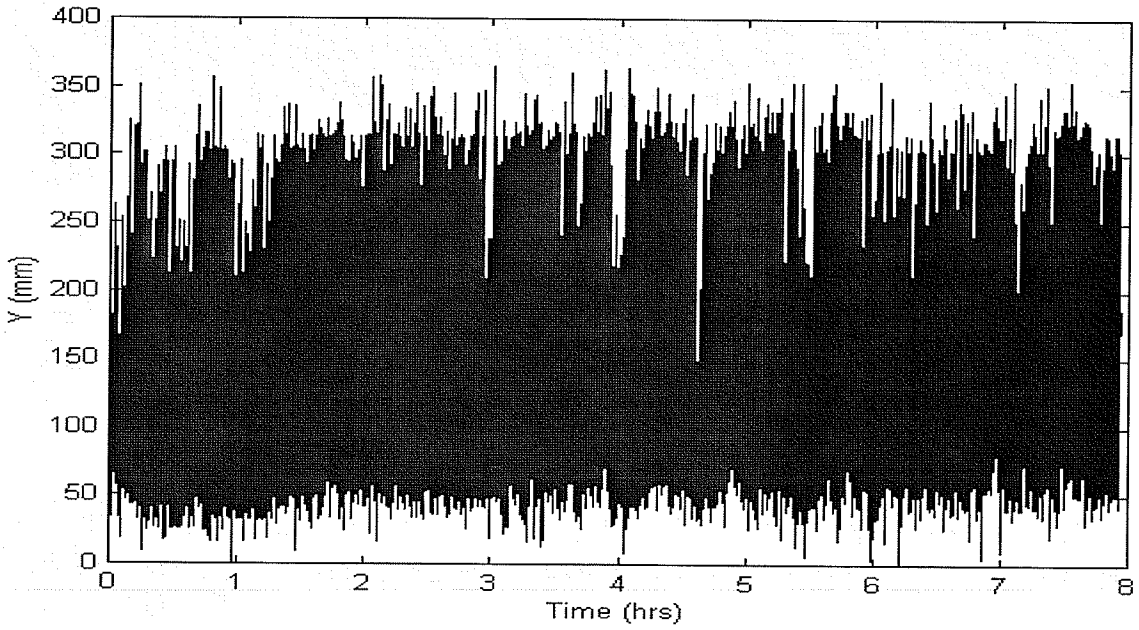

Fig. A.23. Y-coordinates of Experiment 14020309 (MO).

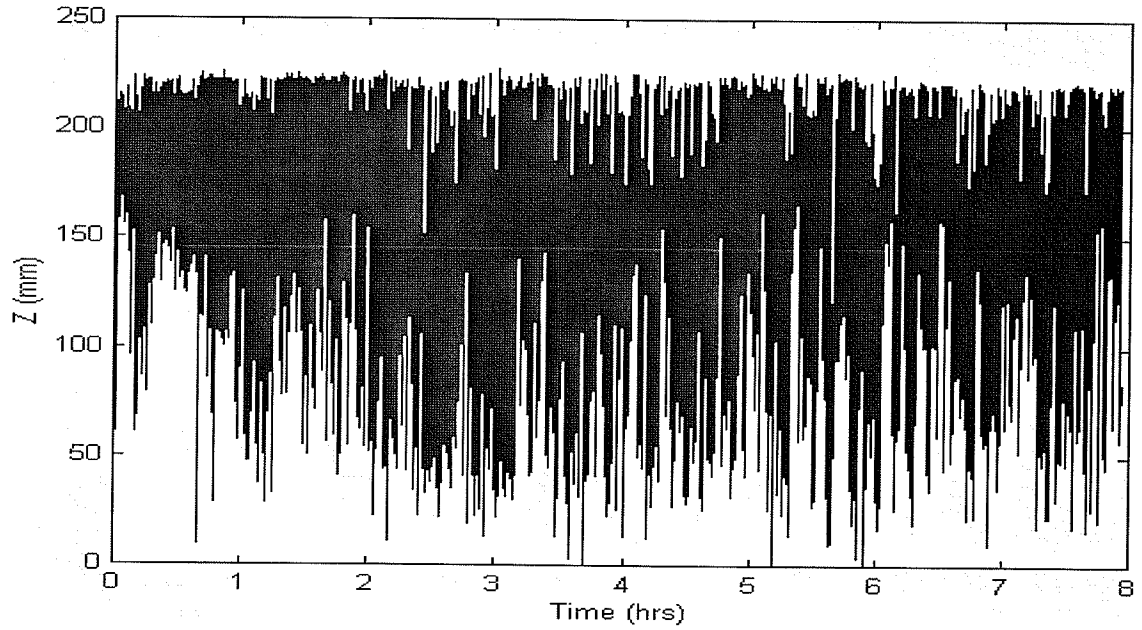

Fig. A.24. Z-coordinates of Experiment 14020309 (MO).

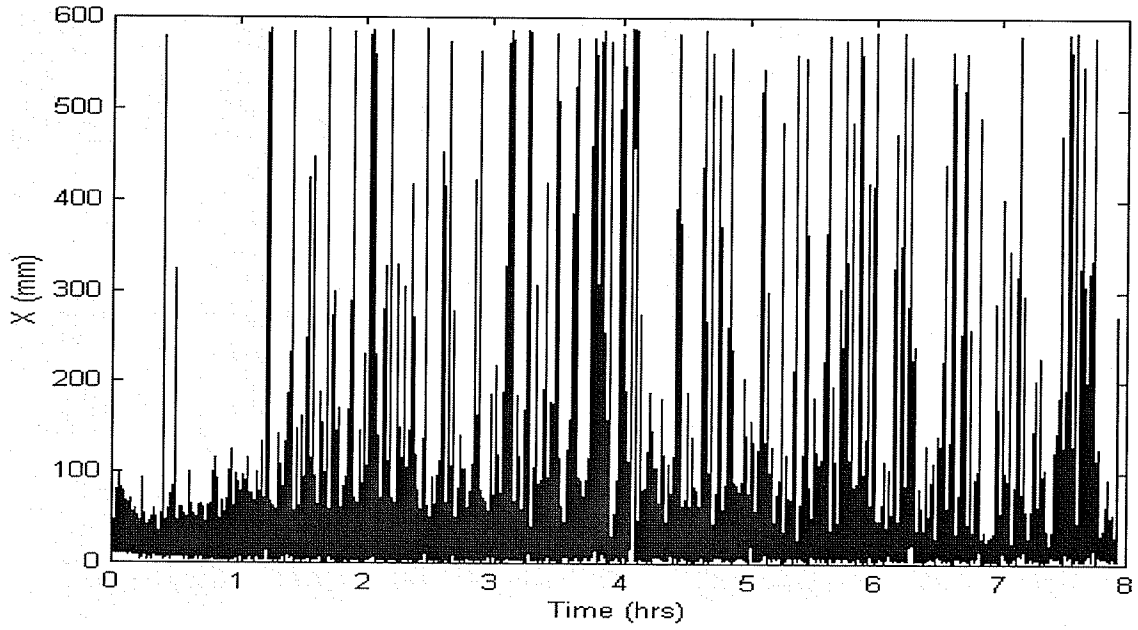

Fig. A.25. X-coordinates of Experiment 14020310 (MO).

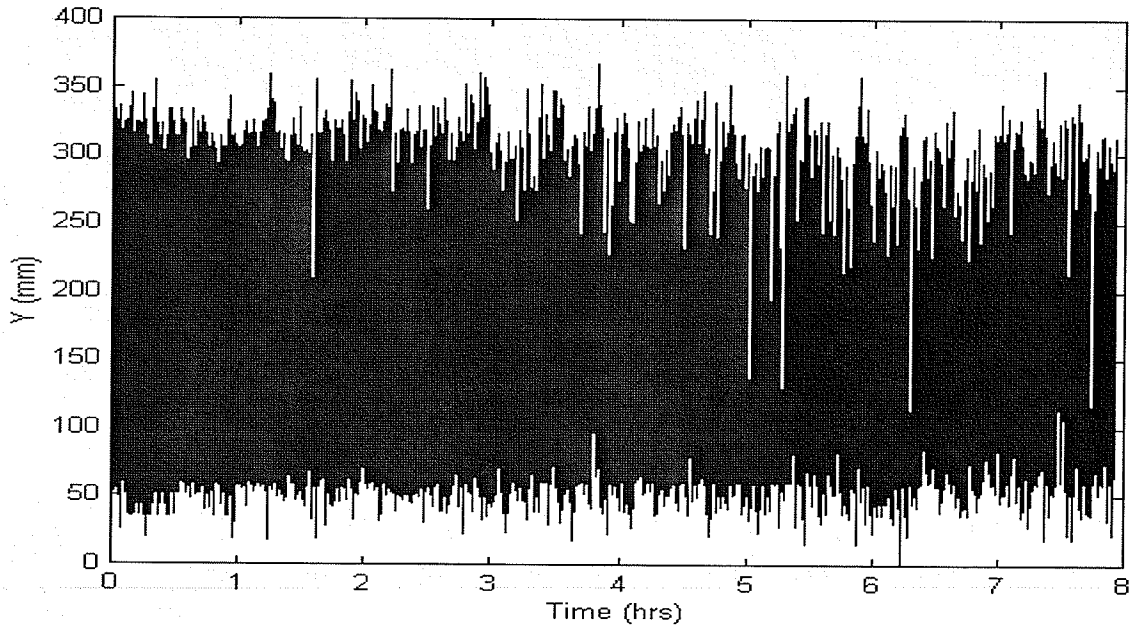

Fig. A.26. Y-coordinates of Experiment 14020310 (MO).

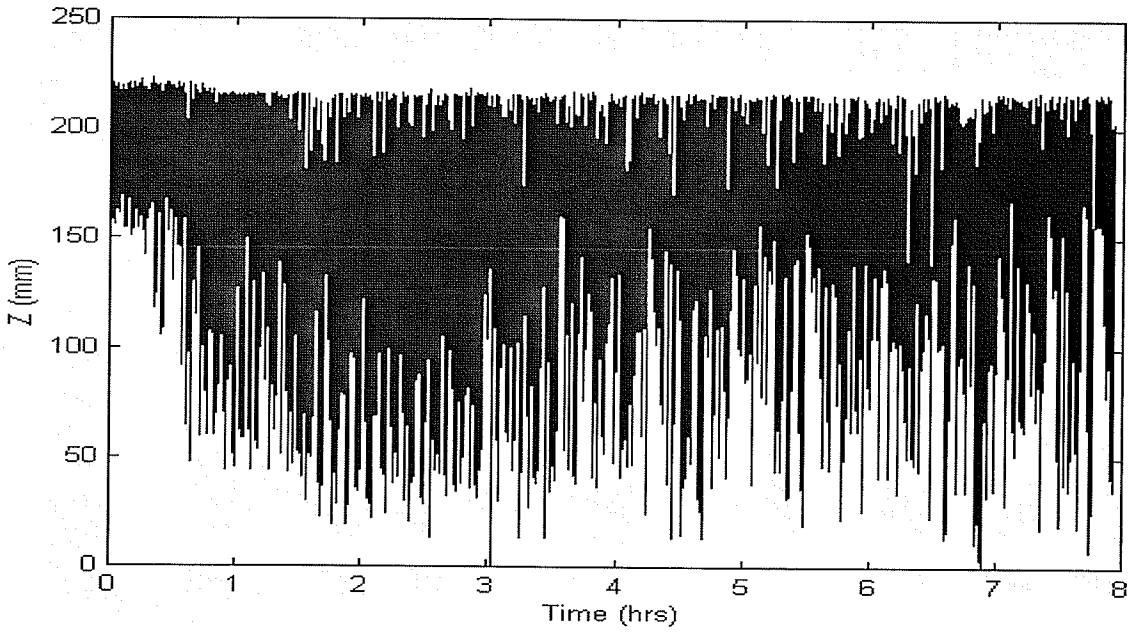

Fig. A.27. Z-coordinates of Experiment 14020310 (MO).

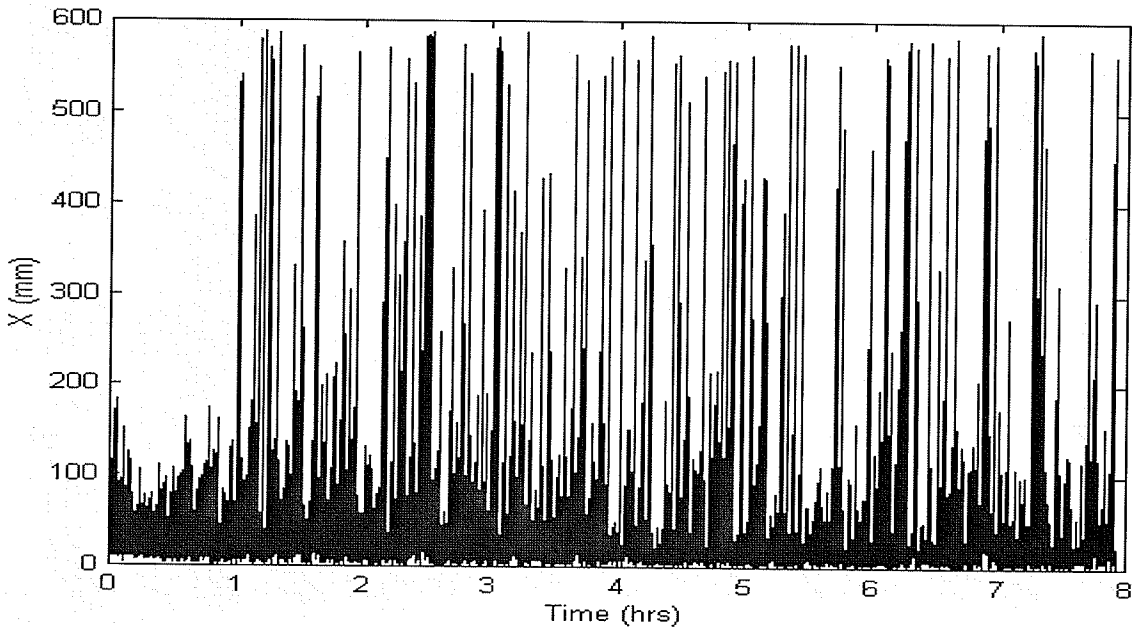

Fig. A.28. X-coordinates of Experiment 14020311 (MO).

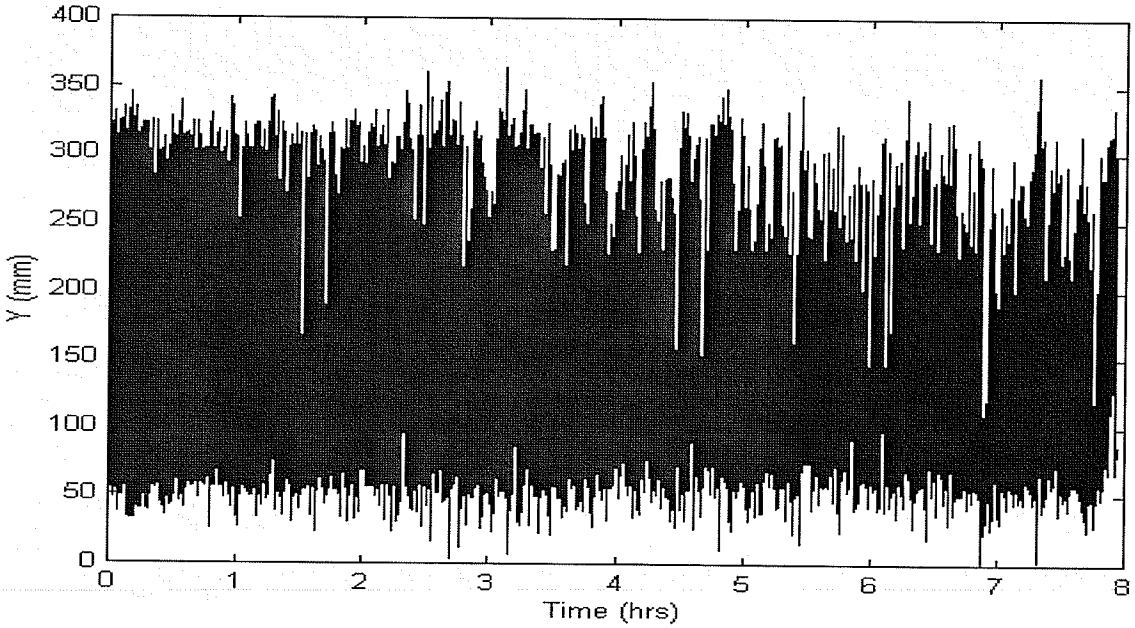

Fig. A.29. Y-coordinates of Experiment 14020311 (MO).

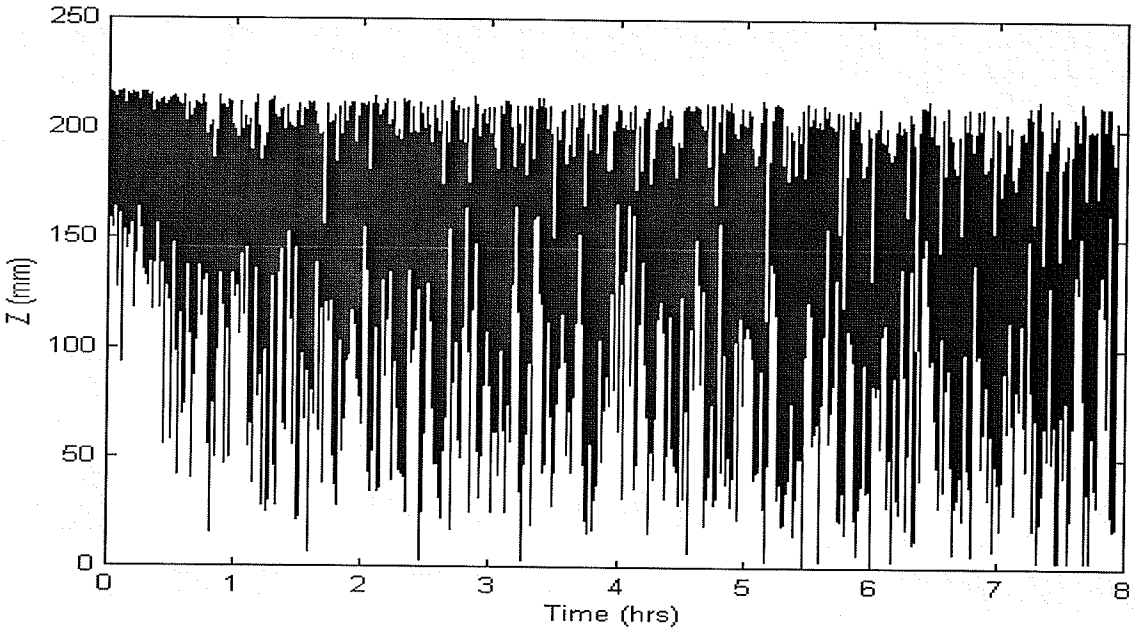

Fig. A.30. Z-coordinates of Experiment 14020311 (MO).

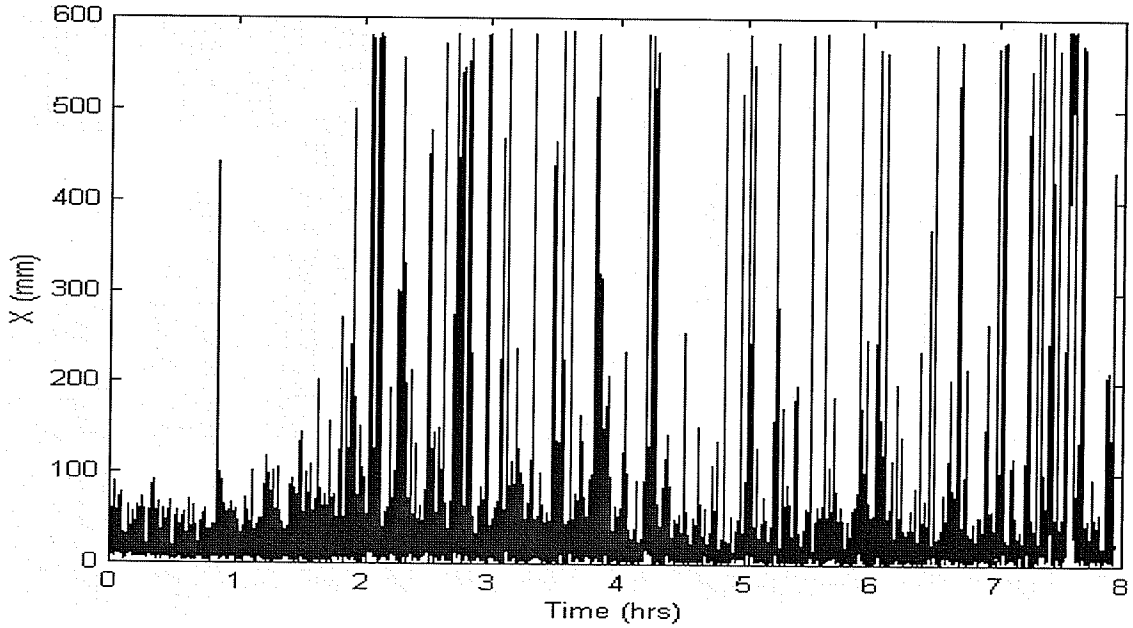

Fig. A.31. X-coordinates of Experiment 14020312 (MC).

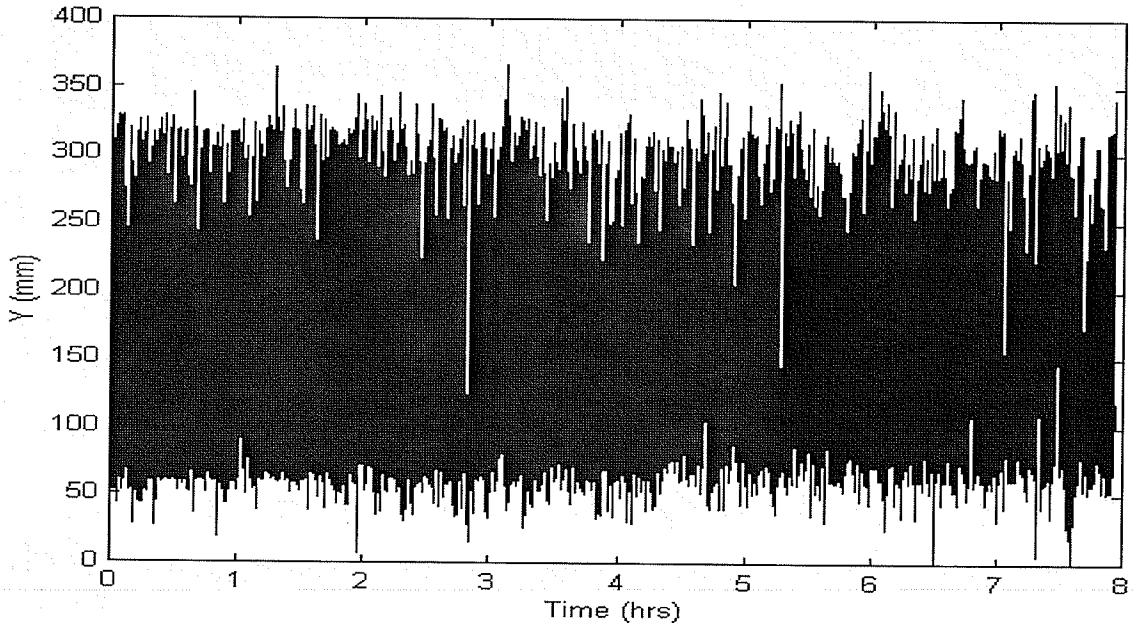

Fig. A.32. Y-coordinates of Experiment 14020312 (MC).

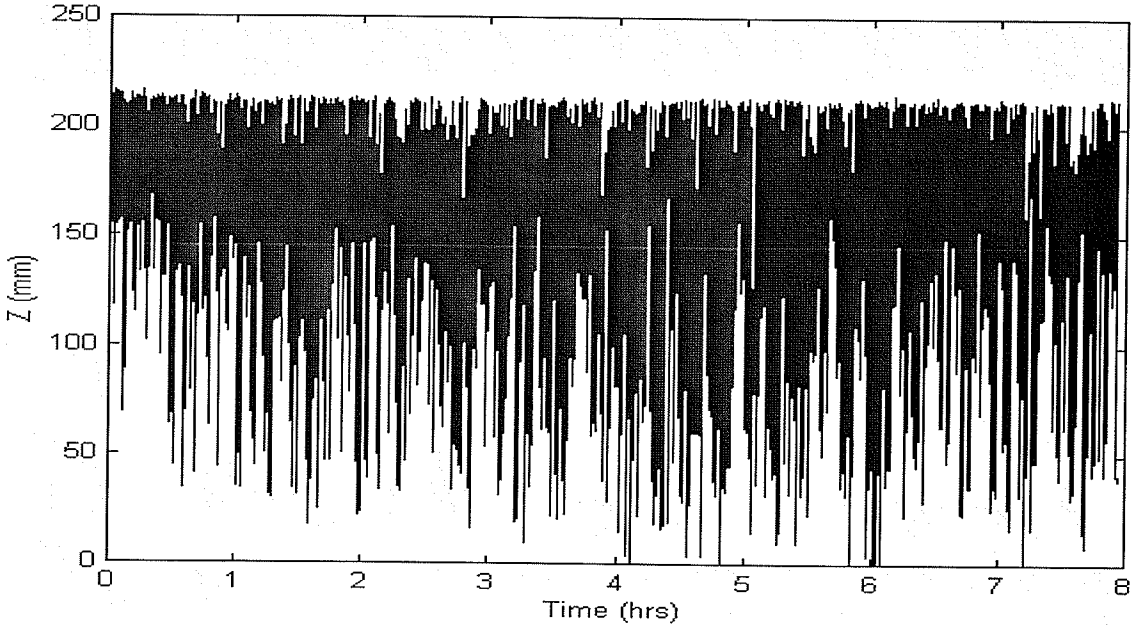

Fig. A.33. Z-coordinates of Experiment 14020312 (MC).

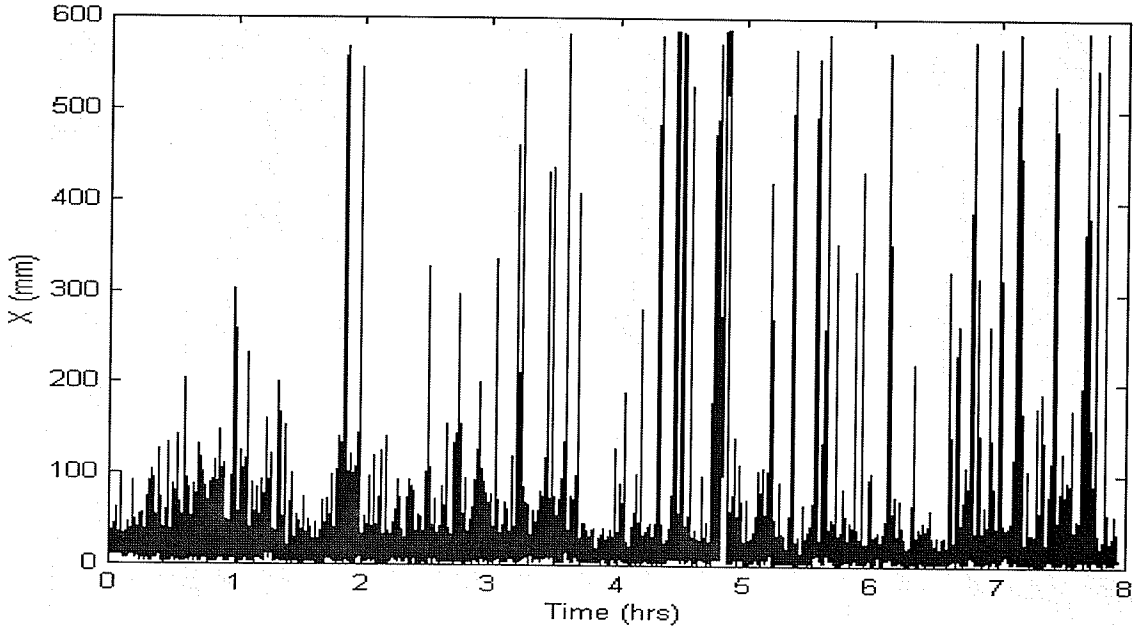

Fig. A.34. X-coordinates of Experiment 14020313 (MC).

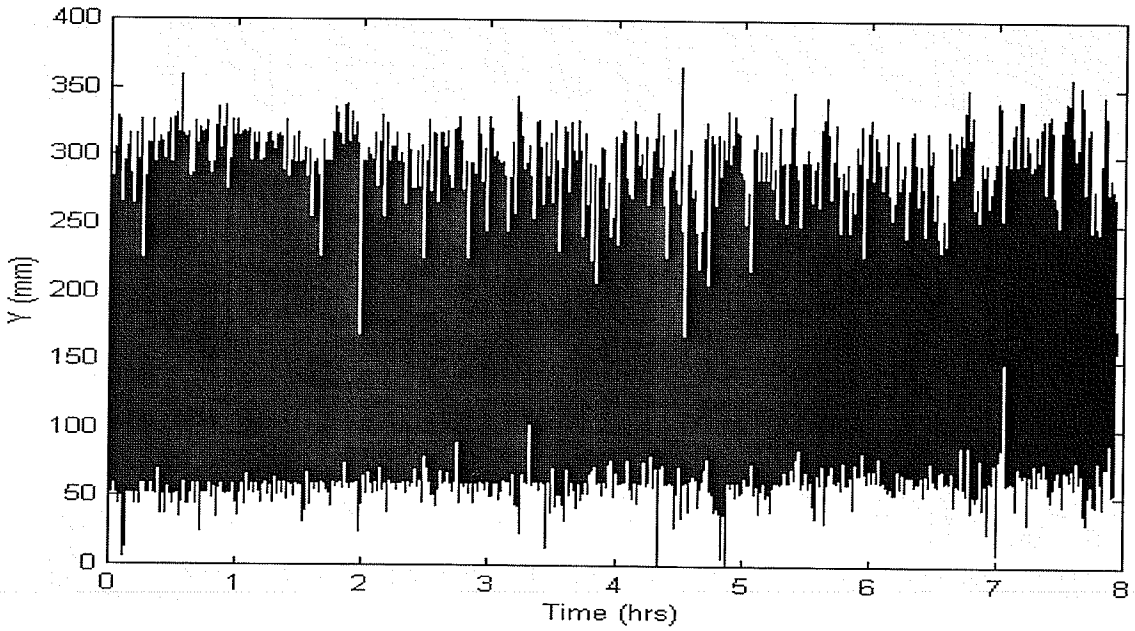

Fig. A.35. Y-coordinates of Experiment 14020313 (MC).

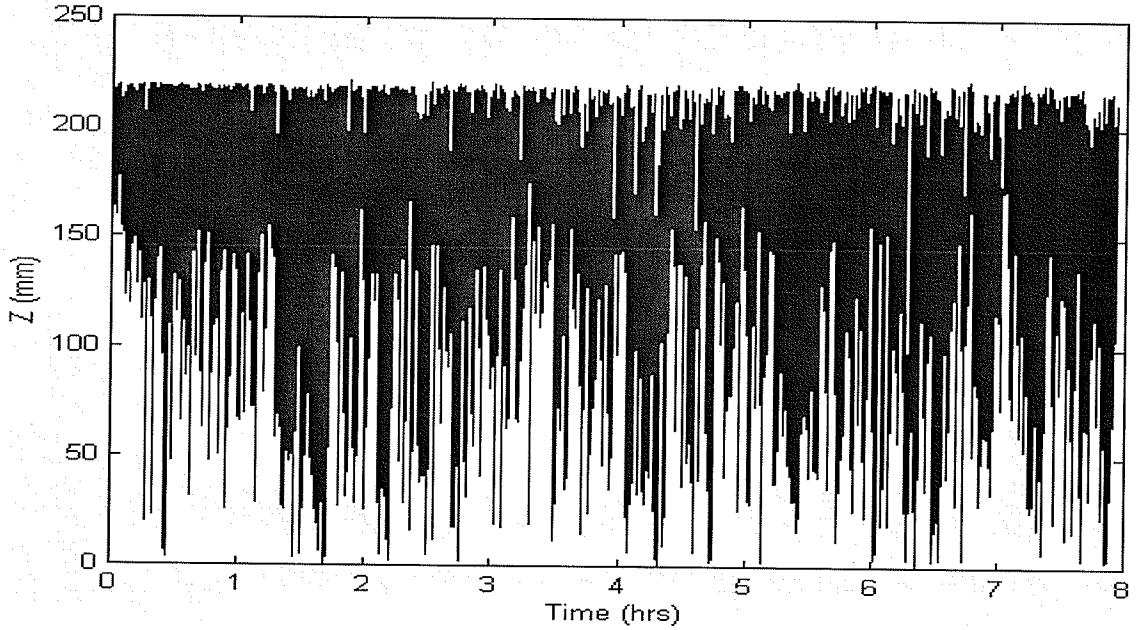

Fig. A.36. Z-coordinates of Experiment 14020313 (MC).

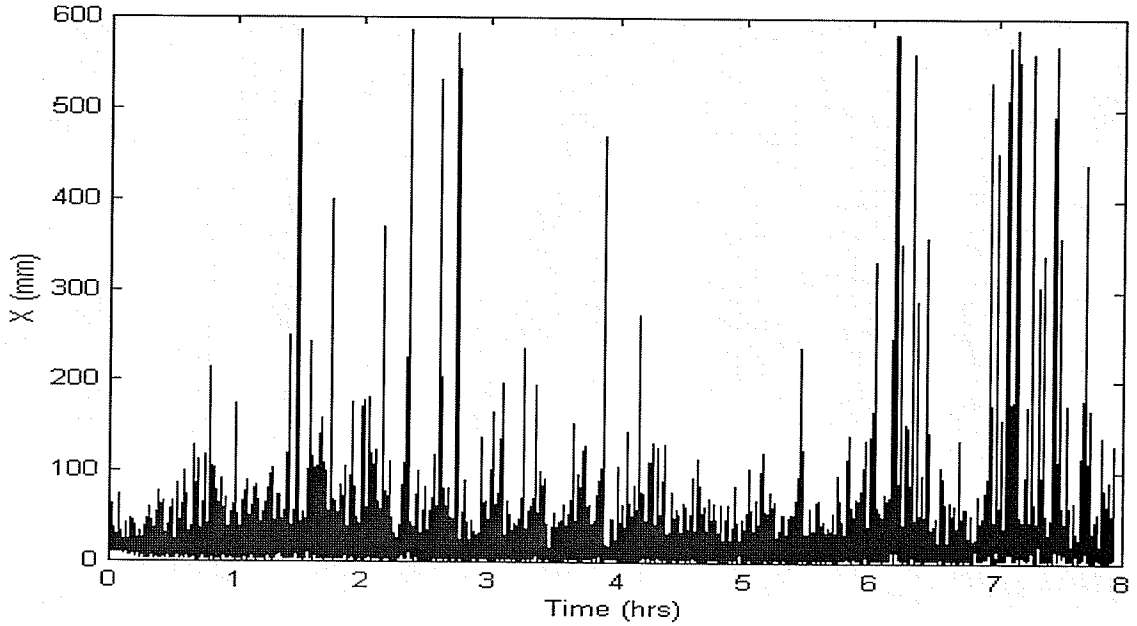

Fig. A.37. X-coordinates of Experiment 14020314 (MC).

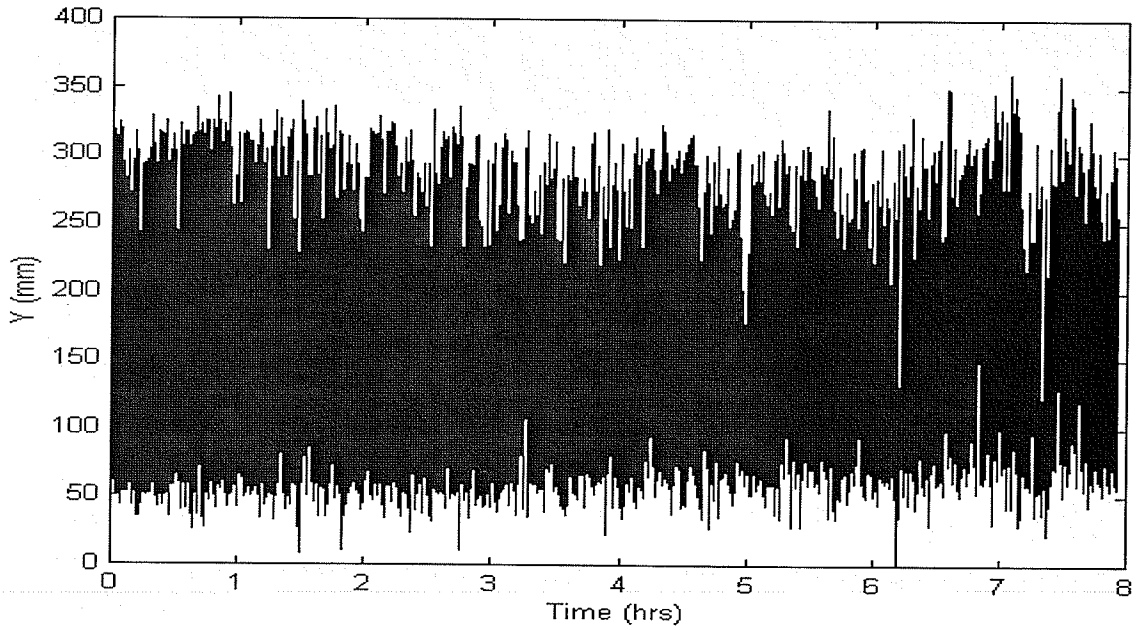

Fig. A.38. Y-coordinates of Experiment 14020314 (MC).

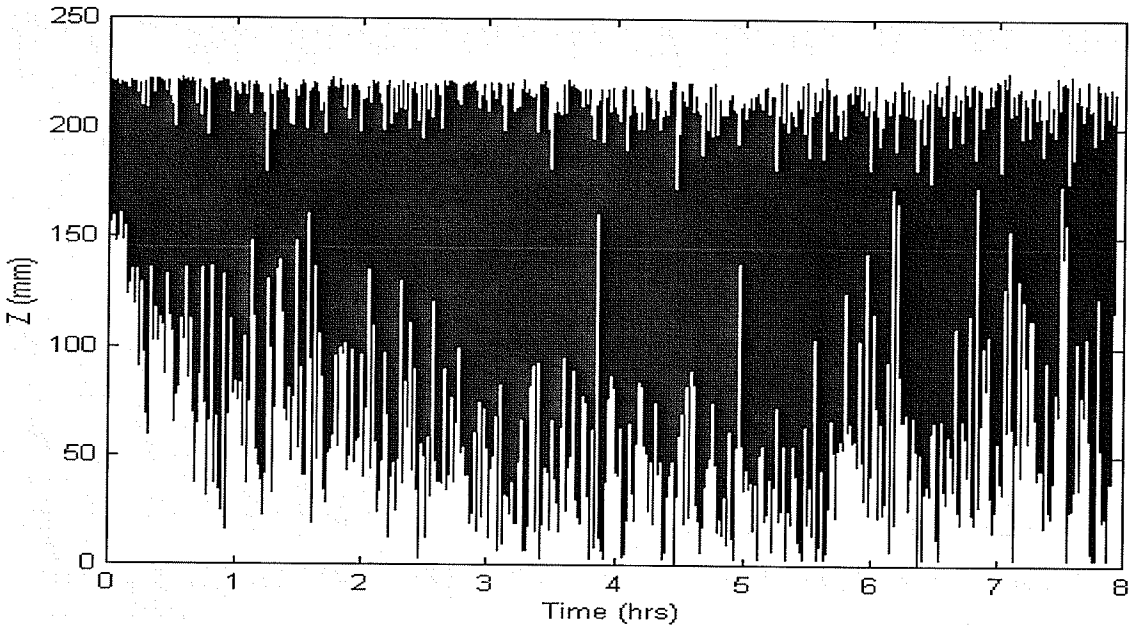

Fig. A.39. Z-coordinates of Experiment 14020314 (MC).

APPENDIX B

MATLAB CODE

This appendix provides the MATLAB code used in this thesis.

B.1 Best Classes (bestclass.m)

```
function bc = bestclass(c,k,reps);

% bestclass selects the best K-means clustering classes
% c - K-means clustering results
% k - number of classes to find (2, 3, 4, ..., 10)
% reps - number of repetitions (1, 2, ..., N)
%
% BC = BESTCLASS(C,K,REPS) returns the best clusters
% C with K classes using REPS repetitions.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.01 on July 28, 2003 at 7:20 am (Central)

i1 = zeros(2,reps);
for i = 1:reps i1(2,i) = i; end
i2 = i1;
if (k > 2) i3 = i1;
   if (k > 3) i4 = i1;
      if (k > 4) i5 = i1;
         if (k > 5) i6 = i1;
            if (k > 6) i7 = i1;
               if (k > 7) i8 = i1;
                  if (k > 8) i9 = i1;
                     if (k > 9) i10 = i1;
                     end
                  end
               end
            end
         end
      end
```

```
   end
end % if

for h = 1:reps          % for each repetition...
   for i = 1:length(c) % for each vector...
      if (c(i,h) == 1) i1(1,h) = i1(1,h) + 1; end
      if (c(i,h) == 2) i2(1,h) = i2(1,h) + 1; end
      if (k > 2)
         if (c(i,h) == 3) i3(1,h) = i3(1,h) + 1; end
         if (k > 3)
            if (c(i,h) == 4) i4(1,h) = i4(1,h) + 1; end
            if (k > 4)
               if (c(i,h) == 5) i5(1,h) = i5(1,h) + 1; end
               if (k > 5)
                  if (c(i,h) == 6) i6(1,h) = i6(1,h) + 1; end
                  if (k > 6)
                     if (c(i,h) == 7) i7(1,h) = i7(1,h) + 1; end
                     if (k > 7)
                        if (c(i,h) == 8) i8(1,h) = i8(1,h) + 1; end
                        if (k > 8)
                           if (c(i,h) == 9) i9(1,h) = i9(1,h) + 1; end
                           if (k > 9)
                              if (c(i,h) == 10) i10(1,h) = i10(1,h) + 1;
end
                           end
                        end
                     end
                  end
               end
            end
         end
      end % if
   end % for i
end % for h

cc = zeros(k,reps);
for i = 1:reps
   cc(1,i) = i1(1,i);
   cc(2,i) = i2(1,i);
   if (k > 2)
      cc(3,i) = i3(1,i);
      if (k > 3)
         cc(4,i) = i4(1,i);
         if (k > 4)
            cc(5,i) = i5(1,i);
            if (k > 5)
               cc(6,i) = i6(1,i);
               if (k > 6)
```

```
                cc(7,i) = i7(1,i);
                if (k > 7)
                   cc(8,i) = i8(1,i);
                   if (k > 8)
                      cc(9,i) = i9(1,i);
                      if (k > 9)
                         cc(10,i) = i10(1,i);
                      end
                   end
                end
             end
          end
       end
    end % if
end % for

ccs = sort(cc);                    % sorted classes
ccm = zeros(k+2,reps);             % matched classes
m = 1;                             % number of individual classes
ccm(1:k,m) = ccs(:,1);             % insert first class
ccm(k+1,m) = 1;                    % one instance so far
ccm(k+2,m) = 1;                    % location of instance
for i = 2:reps                     % for each simulation
   cct = ccs(:,i);                 % temporary class to compare
   j = 1;                          % match index
   while ((j <= m) & not(sum(ccm(1:k,j) == cct) == k))
      j = j + 1;
   end % while
   if (j > m)                      % if no match is found
      m = m + 1;                   % one more class
      ccm(1:k,m) = cct;            % new class
      ccm(k+1,m) = 1;              % one instance
      ccm(k+2,m) = i;
   elseif (j <= m)                 % a match is found
      ccm(k+1,j) = ccm(k+1,j) + 1; % increment instance
   end % if
end % for
ccm = ccm(:,1:m);
ccm = sortrows(ccm',k+1); % sort based on most likely clustering
bc = ccm(m,k+2);          % most likely (best) class
```

B.2 Characteristic Trajectory (chartraj.m)

```
% chartraj creates an 8D trajectory
%
% This program models the four statistical histograms
% and creates an 8D characteristic trajectory.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.01 on July 28, 2003 at 9:55 am (Central)

clear all variables;
load ../mat/VFDTstats.mat;
fs = 10;                          % sampling frequency of 10 Hz
wn = 15 * 60 * fs;                % histogram for 15 minutes
woff = 0.25 * 60 * fs;            % window offset of 15 seconds
len = length(Tm);                 % length of original vectors
maxloops = floor((len-wn)/woff); % number of windows that can be
                                  % sampled from original vectors
traj8D = zeros(8,maxloops);       % create empty char traj vec

r1 = 1;
r2 = wn;
for m = 1 : maxloops
   T1 = Tm(r1:r2);
   T2 = Tv(r1:r2);
   T3 = Ts(r1:r2);
   T4 = Tk(r1:r2);
   charvec = gammamodel(T1,T2,T3,T4);
   traj8D(:,m) = charvec;
   r1 = r1 + woff;
   r2 = r2 + woff;
end % for
save ../mat/traj8D.mat traj8D;
```

B.3 Create Neural Network Sets (createNNsets.m)

```
function [settrain,settest] = createNNsets(T,NV,P)

% createNNsets stands for "create neural network sets"
% T - type of set (0 = random, 1 = regular intervals)
% NV - total number of vectors
% P - percentage of data for training set (0 < P < 1)
%
% [SETTRAIN,SETTEST] = CREATENNSETS(T,NV,P) returns the
% index pointers for the training set SETTRAIN and test
% set SETTEST method T with P% used for SETTRAIN.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.1 on July 18, 2003 at 1:25 am (Central)

NTrain = round(NV*P);              % # of training vectors
NTest = NV - NTrain;               % # of test vectors
settrain = zeros(1,NTrain);
settest = zeros(1,NTest);

if (T == 0)                        % RANDOM SELECTION

i = 0;
while (i < NTrain)                 % select random numbers
   temp = round(rand(1,1)*NV);
   while (not(isempty(find(settrain == temp))) | (temp == 0))
      temp = round(rand(1,1)*NV);
   end % while
   i = i + 1;
   settrain(1,i) = temp;           % put into training set
end % while
settrain = sort(settrain);

j = 0;
for i = 1 : NV                     % select the rest
   if isempty(find(settrain == i))
      j = j + 1;
      settest(j) = i;              % put into test set
   end % if
end % for

elseif (T == 1)                    % REGULAR INTERVALS
```

```
inc = NV / NTrain;                  % training vector offset
settrain(1,1) = 1;
i1 = 1; i2 = 0;                     % data set counters
test = 0;
for i = 2 : NV
   test = test + 1;
   if (test >= inc)                 % put into training set
      test = mod(test,inc);
      i1 = i1 + 1;
      settrain(1,i1) = i;
   elseif (test < inc)              % put into test set
      i2 = i2 + 1;
      settest(1,i2) = i;
   end % if
end % for

end % if (T)
```

B.4 Do K-Means Clustering (doCluster.m)

```
function c = doCluster(PCn,k,reps,doplot,dim)

% doCluster performs K-means clustering on a data set
% PCn - number of principal components (from PC.mat)
% k - number of classes to find (2, 3, 4, ..., 10)
% reps - number of repetitions (1, 2, ..., N)
% doplot - do a plot? (boolean)
% dim - plot in 2D or 3D? (2 or 3, if k <= 6)
%
% C = DOCLUSTER(PCN,K,REPS,DOPLOT,DIM) returns the clusters
% of the first PCN principal components for K classes using REPS
% repetitions for the K-means algorithm, and plots the clusters
% in DIM dimensions if DIM < 7 and DOPLOT = 1.
%
% Required functions: - bestclass.m
%                     - doclustering.m
%                     - plotclasses.m

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.1 on July 8, 2003 at 9:20 pm (Central)

load ../mat/PC.mat;
data = PC(:,1:PCn);              % isolate principal components
                                 % do K-means clustering
[rit, ris, c] = doclustering('kmeans', data, k, [], reps);
corig = c';
bc = bestclass(corig,k,reps);    % find the best clusters
c = corig(:,bc);
if (dim) & (k < 7)               % plot cluster?
   plotclasses(data,c,dim);      % up to 6 clusters in 2D or 3D
end % if
```

B.5 Do Self-Organizing Feature Map Verification (doSOFM.m)

```
% doSOFM models data with a self-organizing feature map
%
% This program displays the first two principal components
% of the data, and the trained N by M SOFM which indicates
% the classes within the 2D data.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.1 on July 8, 2003 at 6:25 pm (Central)

clear all variables;
load ../mat/PC.mat;
x = PC; x = x';
clear PC;

P = x(1:2,:);                    % first two PCs
net = newsom([0 2; 0 1],[2 3]);  % new SOFM
net.trainParam.epochs = 100000;  % how many iterations
net = train(net,P);              % train the SOFM
plot(P(1,:),P(2,:),'.g','markersize',20)     % plot data
hold on
plotsom(net.iw{1,1},net.layers{1}.distances) % plot SOFM
hold off
```

B.6 Gamma Distribution Model (gammamodel.m)

```
function charvec = gammamodel(Tm,Tv,Ts,Tk);

% Tm,Tv,Ts,Tk - vectors for the mean, variance, skewness, and
% kurtosis trajectories of the VFDT
%
% CHARVEC = GAMMAMODEL(TM,TV,TS,TK) returns an eight-dimensional
% vector represeting the alpha and beta parameters of the Gamma
% distribution modeling the mean, variance, skewness, and kurtosis
% trajectory histograms.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.0 on October 21, 2002 at 5:30 pm (Central)

charvec = zeros(1,8);

numbins = 50;
TmH = hist(Tm,numbins);          % create histograms
TvH = hist(Tv,numbins);
TsH = hist(Ts,numbins);
TkH = hist(Tk,numbins);

Hm = TmH ./ sum(TmH);            % create PDFs
Hv = TvH ./ sum(TvH);
Hs = TsH ./ sum(TsH);
Hk = TkH ./ sum(TkH);

f = Hm;                          % Mean PDF
x = 1:length(f);
exp_x = 0; % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for
exp_x2 = 0; % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for

b = exp_x2 / exp_x;              % model parameters
a = exp_x / b;
charvec(1,1:2) = [a b];
```

```
f = Hv;                              % Variance PDF
exp_x = 0; % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for
exp_x2 = 0; % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for

b = exp_x2 / exp_x;                  % model parameters
a = exp_x / b;
charvec(1,3:4) = [a b];

f = Hs;                              % Skewness PDF
exp_x = 0; % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for
exp_x2 = 0; % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for

b = exp_x2 / exp_x;                  % model parameters
a = exp_x / b;
charvec(1,5:6) = [a b];

f = Hk;                              % Kurtosis PDF
exp_x = 0; % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for
exp_x2 = 0; % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for

b = exp_x2 / exp_x;                  % model parameters
a = exp_x / b;
charvec(1,7:8) = [a b];
charvec = charvec';
```

B.7 Get LAN Graph Info (getfilegraphLAN.m)

```
% getfilegraphLAN creates the file trafficLAN.mat
%
% This program extracts the LAN traffic info from the file
% trafficLANraw.mat and saves the compressed time and data
% to the file trafficLAN.mat.  The data are also plotted.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.11 on July 12, 2003 at 7:00 pm (Central)

clear all variables;
load ../mat/trafficLANraw.mat;
T1 = trafLAN;
clear trafLAN;                      % save memory

L = ceil(T1(length(T1))*10);        % new length
T2 = zeros(L,2);
T2(:,1) = ((1 : 1 : L)') ./ 10;     % 0.1 sec time scale

i1 = 1; i2 = 1;
T2(i2,2) = T1(i1,2);
rinc = 0.1;                         % compression rate
r = rinc;
for i1 = 2 : length(T1)             % compressed time
    if (T1(i1,1) > r)
        i2 = i2 + 1;
        r = r + rinc;
    end % if
    T2(i2,2) = T2(i2,2) + T1(i1,2);
end % for
T = T2;
clear T2;                           % save memory
save ../mat/trafficLAN.mat T;

L1 = 1;
L2 = length(T);
plot(T(L1:L2,1),(T(L1:L2,2)/1024));
xlabel('Time (sec)');
ylabel('KB requested / 100 ms');
title('Sampled LAN traffic on 29 August 1989');
%orient landscape;
%print;
```

B.8 Get VoIP Graph Info (getfilegraphVoIP.m)

```
% getfilegraphVoIP creates the file trafficVoIP.mat
%
% This program extracts the VoIP traffic info from the
% file trafficVoIPraw.mat, converts it into a more useful
% rate format, and saves the compressed time and data to
% the file trafficVoIP.mat.  The data are also plotted.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.02 on July 14, 2003 at 9:45 pm (Central)

clear all variables;
load ../mat/trafficVoIPraw.mat;
T1 = trafVoIP;
clear trafVoIP;                    % save memory

Tx = zeros(size(T1));
i2 = 0;
for i1 = 1 : length(T1)            % isolate UDP packets
   if (strcmp(VoIPprot(i1,1:3),'UDP'))
      i2 = i2 + 1;
      Tx(i2,1:2) = T1(i1,1:2);
   end % if
end % for
T1 = Tx(1:i2,:);
clear Tx VoIPprot;                 % save memory

L = ceil(T1(length(T1))*10);       % new length
T2 = zeros(L,2);
sts = 10;
T2(:,1) = ((1 : 1 : L)') ./ sts; % 0.1 sec time scale

for i1 = 1 : length(T1)            % compressed time
   lc = ceil(T1(i1,1)*sts);
   T2(lc,2) = T2(lc,2) + T1(i1,2);
end % for
T = T2;
clear T1 T2;                       % save memory
save ../mat/trafficVoIP.mat T;

L1 = 1;
L2 = length(T);
```

```
T(:,2) = T(:,2) ./ 1024;          % KB scale
plot((T(L1:L2,1)./(60*60)),T(L1:L2,2));
xlabel('Time (hrs)');
ylabel('KB transmitted / 100 ms');
title('Sampled VoIP traffic at TRLabs on 09 July 2003');
%orient landscape;
%print;
```

B.9 Get WWW Graph Info (getfilegraphWWW.m)

```
% getfilegraphWWW creates the file trafficWWW.mat
%
% This program extracts the WWW traffic info from the
% file trafficLANraw.mat, converts it into a more useful
% rate format, and saves the compressed time and data to
% the file trafficLAN.mat.  The data are also plotted.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 2.01 on July 13, 2003 at 7:45 pm (Central)

clear all variables;
load ../mat/trafficWWWraw.mat;
time = trafWWW(:,1);
bytes = trafWWW(:,2);
extime = trafWWW(:,3);
clear trafWWW;                    % save memory

Lold = length(time);
Lnew = sum(ceil(extime));         % new length (uncomp'd)
Ntime = zeros(Lnew,1);            % rate format
Nrate = zeros(Lnew,1);

i2 = 0;
for i1 = 1 : Lold                 % convert all data to
   intervals = ceil(extime(i1)); % bytes / sec format
   rate = bytes(i1) / intervals;
   ttime = time(i1);
   for j = 1 : intervals
      i2 = i2 + 1;
      Nrate(i2) = rate;
      Ntime(i2) = ttime;
      ttime = ttime + 1;
   end % for
end % for
T0 = zeros(Lnew,2);
T0(:,1) = Ntime;
T0(:,2) = Nrate;
T1 = sortrows(T0);
clear Ntime Nrate T0;             % save memory

T = zeros(max(T1(:,1)),1);
```

```
for i1 = 1 : Lnew
   T(T1(i1,1)) = T(T1(i1,1)) + T1(i1,2);
end % for

save ../mat/trafficWWW.mat T;

L1 = 1;
L2 = length(T);
sect = linspace(1,L2,L2);          % time scale
plot((sect(L1:L2)./(60*60*24)),(T(L1:L2)./(1024*1024)));
xlabel('Time (days)');
ylabel('MB requested / sec');
title('Sampled WWW traffic in February 1995');
%orient landscape;
%print;
```

B.10 Get LAN File Info (getfileinfoLAN.m)

```
% getfileinfoLAN creates the file trafficLANraw.mat
%
% This program extracts the LAN traffic info from the file
% BC-pAug89.TL and saves the times and data to the file
% trafficLANraw.mat for further processing.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 2.01 on July 12, 2003 at 2:45 am (Central)

clear all variables;
ddir = '../../Traffic Recordings/LAN - Bellcore';
cd(ddir);

count = 0;                          % initial variables
len = 1000001;                      % number of packets
time = zeros(1,len);
bytes = zeros(1,len);

fid = fopen('BC-pAug89.TL','r');
L = num2str(fgets(fid));
i = 0;
while not(strcmp(L,'-1'))           % for each packet...
   i = i + 1;
   while (L(1,1) == ' ')            % remove leading spaces
      L = L(1,2:length(L));
   end % while
   spaces = findstr(L, ' ');        % find spaces
   j = spaces(1);
   while (L(1,j:j+1) == '  ')       % remove middle spaces
      L = strcat(L(1,1:j),L(1,j+1:length(L)));
   end % while
   spaces = findstr(L, ' ');        % find spaces
   j = spaces(2);
   L = L(1,1:j-1);                  % remove trailing spaces

   j = spaces(1);                   % find marker
   time(1,i) = str2num(L(1,1:j-1));
   bytes(1,i) = str2num(L(1,j+1:length(L)));
   L = num2str(fgets(fid));
end % while
fclose(fid);
```

```
if (i < len)                        % crop final variables
   time = time(1,1:i);
   bytes = bytes(1,1:i);
end % if
trafLAN = zeros(length(time),2); % save final variables
trafLAN(:,1) = time';
trafLAN(:,2) = bytes';
save ../../matlab/mat/trafficLANraw.mat trafLAN;
```

B.11 Get VoIP File Info (getfileinfoVoIP.m)

```
% getfileinfoVoIP creates the file trafficVoIPraw.mat
%
% This program extracts the VoIP traffic info from the file
% voip1d.txt and saves the times, data, and protocol info
% to the file trafficVoIPraw.mat for further processing.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.0 on July 12, 2003 at 7:15 pm (Central)

clear all variables;
ddir = '../../Traffic Recordings/VoIP - TRLabs';
cd(ddir);
maxlen = 5000000;
time = zeros(1,maxlen);
bytes = zeros(1,maxlen);
prot = char(zeros(maxlen,3));     % protocol (UDP, etc.)

fid = fopen('voip1d.txt','r');
L = num2str(fgets(fid));          % ignore column headers
L = num2str(fgets(fid));

i = 0;
while not(strcmp(L,'-1'))         % for each packet...
   i = i + 1;
   while (L(1,1) == ' ')          % remove leading spaces
      L = L(1,2:length(L));
   end % while
   spaces = findstr(L, ' ');      % find spaces
   j = spaces(1);
   while (L(1,j:j+1) == '  ')     % remove middle spaces
      L = strcat(L(1,1:j),L(1,j+1:length(L)));
   end % while
   spaces = findstr(L, ' ');      % find spaces
   j = spaces(2);
   while (L(1,j:j+1) == '  ')     % remove middle spaces
      L = strcat(L(1,1:j),L(1,j+1:length(L)));
   end % while
   spaces = findstr(L, ' ');      % find spaces
   j = spaces(3);
   while (L(1,j:j+1) == '  ')     % remove middle spaces
      L = strcat(L(1,1:j),L(1,j+1:length(L)));
```

```
   end % while
   L = deblank(L);                % remove trailing spaces

   spaces = findstr(L, ' ');      % find spaces
   time(1,i) = str2num(L((spaces(1)+1):(spaces(2)-1)));
   bytes(1,i) = str2num(L((spaces(3)+1):length(L)));
   z = L((spaces(2)+1):(spaces(3)-1));
   prot(i,1:length(z)) = z;

   L = num2str(fgets(fid));
end % while
fclose(fid);

if (i < maxlen)                   % crop final variables
   time = time(1,1:i);
   bytes = bytes(1,1:i);
   VoIPprot = prot(1:i,:);
end % if
trafVoIP = zeros(length(time),2);
trafVoIP(:,1) = time';            % save final variables
trafVoIP(:,2) = bytes';

save ../../matlab/mat/trafficVoIPraw.mat trafVoIP VoIPprot;
```

B.12 Get WWW File Info (getfileinfoWWW.m)

```
% getfileinfoWWW creates the file trafficWWWraw.mat
%
% This program extracts the WWW traffic info from the files
% listed in WWWnames.mat and saves the times and data to the
% file trafficWWWraw.mat for further processing.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 2.02 on July 12, 2003 at 1:10 am (Central)

clear all variables;
load ../mat/WWWnames.mat;
ddir = '../../Traffic Recordings/WWW - Boston University';
cd(ddir);
numfiles = length(WWWnames);      % get WWW files names

count = 0;                        % initial variables
startlen = 110000;
time = zeros(1,startlen);
bytes = zeros(1,startlen);
extime = zeros(1,startlen);

for i = 1 : numfiles              % for each file...
   i
   fid = fopen(deblank(WWWnames(i,:)),'r');
   L = fgets(fid);
   while (L ~= -1)
      wadd = findstr(L, '"');
      L = strcat(L(1:(wadd(1)-1)),L((wadd(2)+1):length(L)));
      dspaces = findstr(L, '  '); % find double spaces
      while (not(isempty(dspaces)))
         strrep(L, '  ', ' ');       % replace double spaces
         dspaces = findstr(L, '  '); % find double spaces
      end % while
      spaces = findstr(L, ' ');   % find markers
      tmptime = str2num(L((spaces(1)+1):(spaces(2)-1)));
      tmpbytes = str2num(L((spaces(3)+1):(spaces(4)-1)));
      tmpextime = str2num(L((spaces(4)+1):length(L)));
      if ((tmpbytes ~= 0) & (tmpextime ~= 0.0))
         count = count + 1;       % save new data
         time(1,count) = str2num(L((spaces(1)+1):(spaces(2)-1)));
         bytes(1,count) = str2num(L((spaces(3)+1):(spaces(4)-1)));
```

```
         extime(1,count) = str2num(L((spaces(4)+1):length(L)));
      end % if
      L = fgets(fid);
   end % while
   fclose(fid);
end % for

if (count < startlen)            % crop final variables
   time = time(1,1:count);
   bytes = bytes(1,1:count);
   extime = extime(1,1:count);
end % if
time = time - min(time) + 1;     % change time offset
trafWWW = zeros(length(time),3);
trafWWW(:,1) = time';            % save final variables
trafWWW(:,2) = bytes';
trafWWW(:,3) = extime';
save ../../matlab/mat/trafficWWWraw.mat trafWWW;
```

B.13 Graph Siamese Fighting Fish Traffic (graphFish.m)

```
% graphFish graphs the Siamese Fighting Fish traffic at
% different time scales
%
% To visually demonstrate the self-affine nature of the
% Siamese Fighting Fish traffic, this program graphs the
% fish traffic at three different time scales.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.11 on July 17, 2003 at 2:05 am (Central)

clear all variables;
load ../mat/11020219.mat;        % load traffic
x1 = x;
x1 = x1 ./ 10;                   % length in cm
clear x y z t;

k = 10;                          % scaling rate

t1 = zeros(1,length(x1));        % time (100 ms)
for i = 1:length(t1)
   t1(i) = i / (60 * 10);        % time (min)
end % for

x2 = zeros(1,floor(length(x1)/k)); % second scale
for i = 1:(length(x1)/k)
   x2(i) = sum(x1((i*k)-(k-1):(i*k)));
end % for
t2 = t1(1:length(x2)); t2 = t2 .* k;

x3 = zeros(1,floor(length(x2)/k)); % third scale
for i = 1:(length(x2)/k)
   x3(i) = sum(x2((i*k)-(k-1):(i*k)));
end % for
t3 = t2(1:length(x3)); t3 = t3 .* k;

figure;
subplot(3,1,1); plot(t1,x1);
title('Averaged over 0.1 sec intervals');
ylabel('X (cm / 0.1 sec)');
subplot(3,1,2); plot(t2,x2);
title('Averaged over 1 sec intervals');
```

```
ylabel('X (cm / sec)');
subplot(3,1,3); plot(t3,x3);
title('Averaged over 10 sec intervals');
ylabel('X (cm / 10 sec)');
xlabel('Time (min)');
%orient landscape
%print
```

B.14 Graph LAN Traffic (graphLAN.m)

```
% graphLAN graphs LAN traffic at different time scales
%
% To visually demonstrate the self-affine nature of LAN
% traffic, this program graphs the LAN traffic at three
% different time scales.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.1 on July 14, 2003 at 4:30 pm (Central)

clear all variables;
load ../mat/trafficLANraw.mat;
T1 = trafLAN;
clear trafLAN;                    % save memory

L = ceil(T1(length(T1))*100);     % new length
T2 = zeros(L,2);
T2(:,1) = ((1 : 1 : L)') ./ 100; % 10 ms time scale

i1 = 1; i2 = 1;
T2(i2,2) = T1(i1,2);
rinc = 0.01;                      % compression rate
r = rinc;
for i1 = 2 : length(T1)           % compressed time
    if (T1(i1,1) > r)
        i2 = i2 + 1;
        r = r + rinc;
    end % if
    T2(i2,2) = T2(i2,2) + T1(i1,2);
end % for
T = T2;

x1 = T(:,2);
x1 = x1 ./ 1024;                  % KB scale
t1 = T(:,1);
clear T T2;                       % save memory

k = 10;                           % scaling rate

x2 = zeros(1,floor(length(x1)/k)); % second scale
for i = 1:(length(x1)/k)
   x2(i) = sum(x1((i*k)-(k-1):(i*k)));
```

```
end % for
t2 = t1(1:length(x2)); t2 = t2 .* k;

x3 = zeros(1,floor(length(x2)/k)); % third scale
for i = 1:(length(x2)/k)
   x3(i) = sum(x2((i*k)-(k-1):(i*k)));
end % for
t3 = t2(1:length(x3)); t3 = t3 .* k;

figure;
subplot(3,1,1); plot(t1,x1);
title('Averaged over 10 msec intervals');
ylabel('KB / 10 msec');
subplot(3,1,2); plot(t2,x2);
title('Averaged over 100 msec intervals');
ylabel('KB / 100 msec');
subplot(3,1,3); plot(t3,x3);
title('Averaged over 1 sec intervals');
ylabel('KB / sec');
xlabel('Time (sec)');
%orient landscape
%print
```

B.15 Graph VoIP Traffic (graphVoIP.m)

```
% graphVoIP graphs VoIP traffic at different time scales
%
% To visually demonstrate the self-affine nature of VoIP
% traffic, this program graphs the VoIP traffic at three
% different time scales.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.2 on July 16, 2003 at 11:55 pm (Central)

clear all variables;
load ../mat/trafficVoIP.mat;     % load traffic
x1 = T(:,2);
x1 = x1 ./ 1024;                 % KB scale
t1 = T(:,1);
t1 = t1 ./ (60*60);              % time (hours)
clear T;                         % save memory

k = 10;                          % scaling rate

x2 = zeros(1,floor(length(x1)/k)); % second scale
for i = 1:(length(x1)/k)
   x2(i) = sum(x1((i*k)-(k-1):(i*k)));
end % for
t2 = t1(1:length(x2)); t2 = t2 .* k;

x3 = zeros(1,floor(length(x2)/k)); % third scale
for i = 1:(length(x2)/k)
   x3(i) = sum(x2((i*k)-(k-1):(i*k)));
end % for
t3 = t2(1:length(x3)); t3 = t3 .* k;

x4 = zeros(1,floor(length(x3)/k)); % fourth scale
for i = 1:(length(x3)/k)
   x4(i) = sum(x3((i*k)-(k-1):(i*k)));
end % for
t4 = t3(1:length(x4)); t4 = t4 .* k;

figure;
subplot(4,1,1); plot(t1,x1);
title('Averaged over 0.1 sec intervals');
ylabel('KB / 0.1 sec');
```

```
subplot(4,1,2); plot(t2,x2);
title('Averaged over 1 sec intervals');
ylabel('KB / 1 sec');
subplot(4,1,3); plot(t3,x3);
title('Averaged over 10 sec intervals');
ylabel('KB / 10 sec');
subplot(4,1,4); plot(t4,x4);
title('Averaged over 100 sec intervals');
ylabel('KB / 100 sec');
xlabel('Time (hrs)');
%orient landscape
%print
```

B.16 Graph WWW Traffic (graphWWW.m)

```
% graphWWW graphs WWW traffic at different time scales
%
% To visually demonstrate the self-affine nature of WWW
% traffic, this program graphs the WWW traffic at three
% different time scales.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.1 on July 14, 2003 at 4:30 pm (Central)

clear all variables;
load ../mat/trafficWWW.mat;      % load traffic
x1 = T;
x1 = x1 ./ (1024*1024);          % MB scale
clear T;                         % save memory

t1 = linspace(1,length(x1),length(x1))'; % time (sec)
t1 = t1 ./ (60*60*24);           % time (days)

k = 10;                          % scaling rate

x2 = zeros(1,floor(length(x1)/k)); % second scale
for i = 1:(length(x1)/k)
   x2(i) = sum(x1((i*k)-(k-1):(i*k)));
end % for
t2 = t1(1:length(x2)); t2 = t2 .* k;

x3 = zeros(1,floor(length(x2)/k)); % third scale
for i = 1:(length(x2)/k)
   x3(i) = sum(x2((i*k)-(k-1):(i*k)));
end % for
t3 = t2(1:length(x3)); t3 = t3 .* k;

figure;
subplot(3,1,1); plot(t1,x1);
title('Averaged over 1 sec intervals');
ylabel('MB / sec');
subplot(3,1,2); plot(t2,x2);
title('Averaged over 10 sec intervals');
ylabel('MB / 10 sec');
subplot(3,1,3); plot(t3,x3);
title('Averaged over 100 sec intervals');
```

```
ylabel('MB / 100 sec');
xlabel('Time (days)');
%orient landscape
%print
```

B.17 Histogram Modelling (HistModel.m)

```
% HistModel models the statistical histograms
%
% This program uses gamma distributions to model the
% statistical histograms of the 8D trajectory.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.12 on July 28, 2003 at 10:15 am (Central)

clear all variables;
load ../mat/TXHist.mat;          % load histograms
TmXHistN = TmXHist ./ sum(TmXHist);
TvXHistN = TvXHist ./ sum(TvXHist);
TsXHistN = TsXHist ./ sum(TsXHist);
TkXHistN = TkXHist ./ sum(TkXHist);

% MEAN

figure(1);
f = TmXHistN;
x = 1:length(f);
bar(f); title('Histogram of the MEAN');
%orient landscape; print;

exp_x = 0;                       % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for

exp_x2 = 0;                      % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for

b = exp_x2 / exp_x;              % Gamma distribution
a = exp_x / b;
f_gammaK = ((x .^ (a-1)) .* exp((-x) ./ b)) / ((b^a) * gamma(a));
f_gammaK = f_gammaK ./ sum(f_gammaK);
figure(2);
bar(f_gammaK);
title(['MEAN - Gamma Distribution Model, a = ',num2str(a), ', b =
',num2str(b)]);
```

```
%orient landscape; print;

% VARIANCE

figure(3);
f = TvXHistN;
x = 1:length(f);
bar(f); title('Histogram of the VARIANCE');
%orient landscape; print;

exp_x = 0;                          % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for

exp_x2 = 0;                         % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for

b = exp_x2 / exp_x;                 % Gamma distribution
a = exp_x / b;
f_gammaK = ((x .^ (a-1)) .* exp((-x) ./ b)) / ((b^a) * gamma(a));
f_gammaK = f_gammaK ./ sum(f_gammaK);
figure(4);
bar(f_gammaK);
title(['VARIANCE - Gamma Distribution Model, a = ',num2str(a), ', b =
',num2str(b)]);
%orient landscape; print;

% SKEWNESS

figure(5);
f = TsXHistN;
x = 1:length(f);
bar(f); title('Histogram of the SKEWNESS');
%orient landscape; print;

exp_x = 0;                          % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for

exp_x2 = 0;                         % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for
```

```
b = exp_x2 / exp_x;                % Gamma distribution
a = exp_x / b;
f_gammaK = ((x .^ (a-1)) .* exp((-x) ./ b)) / ((b^a) * gamma(a));
f_gammaK = f_gammaK ./ sum(f_gammaK);
figure(6);
bar(f_gammaK);
title(['SKEWNESS - Gamma Distribution Model, a = ',num2str(a), ', b =
',num2str(b)]);
%orient landscape; print;

% KURTOSIS

figure(7);
f = TkXHistN;
x = 1:length(f);
bar(f); title('Histogram of the KURTOSIS');
%orient landscape; print;

exp_x = 0;                         % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for

exp_x2 = 0;                        % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for

b = exp_x2 / exp_x;                % Gamma distribution
a = exp_x / b;
f_gammaK = ((x .^ (a-1)) .* exp((-x) ./ b)) / ((b^a) * gamma(a));
f_gammaK = f_gammaK ./ sum(f_gammaK);
figure(8);
bar(f_gammaK);
title(['KURTOSIS - Gamma Distribution Model, a = ',num2str(a), ', b =
',num2str(b)]);
%orient landscape; print;
```

B.18 Local Histogram Modelling (LHistModel.m)

```
% LHistModel models local statistical histograms
%
% This program uses gamma distributions to model the
% local statistical histograms of the 8D trajectory.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.01 on July 28, 2003 at 10:25 am (Central)

clear all variables;
load ../mat/TmX.mat;              % load trajectories
load ../mat/TmX.mat;
load ../mat/TvX.mat;
load ../mat/TsX.mat;
load ../mat/TkX.mat;

TmXs = hist(TmX(200000:209000),100); % local histograms
TmXs = TmXs ./ sum(TmXs);
TvXs = hist(TvX(200000:209000),100);
TvXs = TvXs ./ sum(TvXs);
TsXs = hist(TsX(200000:209000),100);
TsXs = TsXs ./ sum(TsXs);
TkXs = hist(TkX(200000:209000),100);
TkXs = TkXs ./ sum(TkXs);

f = TmXs;
x = 1:length(f);
figure; bar(f);

exp_x = 0;                        % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for

exp_x2 = 0;                       % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for

b = exp_x2 / exp_x;               % Gamma distribution
a = exp_x / b;
f_gammaK = ((x .^ (a-1)) .* exp((-x) ./ b)) / ((b^a) * gamma(a));
```

```
f_gammaK = f_gammaK ./ sum(f_gammaK);
figure; bar(f_gammaK);
title(['Gamma Distribution Model, a = ',num2str(a), ', b =
',num2str(b)]);
xlabel('Bin'); ylabel('Probability');

f = TvXs;
x = 1:length(f);
figure; bar(f);

exp_x = 0;                          % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for

exp_x2 = 0;                         % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for

b = exp_x2 / exp_x;                 % Gamma distribution
a = exp_x / b;
f_gammaK = ((x .^ (a-1)) .* exp((-x) ./ b)) / ((b^a) * gamma(a));
f_gammaK = f_gammaK ./ sum(f_gammaK);
figure; bar(f_gammaK);
title(['Gamma Distribution Model, a = ',num2str(a), ', b =
',num2str(b)]);
xlabel('Bin'); ylabel('Probability');

f = TsXs;
x = 1:length(f);
figure; bar(f);

exp_x = 0;                          % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for

exp_x2 = 0;                         % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for

b = exp_x2 / exp_x;                 % Gamma distribution
a = exp_x / b;
f_gammaK = ((x .^ (a-1)) .* exp((-x) ./ b)) / ((b^a) * gamma(a));
f_gammaK = f_gammaK ./ sum(f_gammaK);
figure; bar(f_gammaK);
```

```
title(['Gamma Distribution Model, a = ',num2str(a), ', b =
',num2str(b)]);
xlabel('Bin'); ylabel('Probability');

f = TkXs;
x = 1:length(f);
figure; bar(f);

exp_x = 0;                          % mean (mu)
for i = 1 : length(f)
    exp_x = exp_x + x(i) * f(i);
end % for

exp_x2 = 0;                         % variance (sigma squared)
for i = 1 : length(f)
    exp_x2 = exp_x2 + (x(i)-exp_x)^2 * f(i);
end % for

b = exp_x2 / exp_x;                 % Gamma distribution
a = exp_x / b;
f_gammaK = ((x .^ (a-1)) .* exp((-x) ./ b)) / ((b^a) * gamma(a));
f_gammaK = f_gammaK ./ sum(f_gammaK);
figure; bar(f_gammaK);
title(['Gamma Distribution Model, a = ',num2str(a), ', b =
',num2str(b)]);
xlabel('Bin'); ylabel('Probability');
```

B.19 Plot Classes (plotclasses.m)

```
function plotclasses(data,c,d);

% data - data points
% c - classes (2, 3, ..., 10)
% d - dimensions to plot (2 or 3)
%
% PLOTCLASSES(DATA,C,D) returns a 2D or 3D plot of the
% classes C in DATA.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 2.02 on July 28, 2003 at 7:30 am (Central)

c1 = zeros(size(data)); i1 = 0;  % classes to plot
c2 = zeros(size(data)); i2 = 0;
c3 = zeros(size(data)); i3 = 0;
c4 = zeros(size(data)); i4 = 0;
c5 = zeros(size(data)); i5 = 0;
c6 = zeros(size(data)); i6 = 0;

for i = 1:length(data)           % assign data to classes
   if (c(i) == 1)
      i1 = i1 + 1;
      c1(i1,:) = data(i,:);
   elseif (c(i) == 2)
      i2 = i2 + 1;
      c2(i2,:) = data(i,:);
   elseif (c(i) == 3)
      i3 = i3 + 1;
      c3(i3,:) = data(i,:);
   elseif (c(i) == 4)
      i4 = i4 + 1;
      c4(i4,:) = data(i,:);
   elseif (c(i) == 5)
      i5 = i5 + 1;
      c5(i5,:) = data(i,:);
   elseif (c(i) == 6)
      i6 = i6 + 1;
      c6(i6,:) = data(i,:);
   end % if
end % for
c1 = c1(1:i1,:);
```

```
c2 = c2(1:i2,:);
c3 = c3(1:i3,:);
c4 = c4(1:i4,:);
c5 = c5(1:i5,:);
c6 = c6(1:i6,:);

if (d == 2)                          % plot in 2D
   figure;
   hold on;
   for i = 1:length(data)
      if (c(i) == 1)
         plot(data(i,1),data(i,2),'.r');
      elseif (c(i) == 2)
         plot(data(i,1),data(i,2),'.g');
      elseif (c(i) == 3)
         plot(data(i,1),data(i,2),'.b');
      elseif (c(i) == 4)
         plot(data(i,1),data(i,2),'.c');
      elseif (c(i) == 5)
         plot(data(i,1),data(i,2),'.y');
      elseif (c(i) == 6)
         plot(data(i,1),data(i,2),'.m');
      end % if
   end % for
   hold off;
elseif (d == 3)                      % or plot in 3D
   figure;
   plot3(c1(:,1),c1(:,2),c1(:,3),'.r');
   hold on;
   plot3(c2(:,1),c2(:,2),c2(:,3),'.g');
   plot3(c3(:,1),c3(:,2),c3(:,3),'.b');
   plot3(c4(:,1),c4(:,2),c4(:,3),'.c');
   plot3(c5(:,1),c5(:,2),c5(:,3),'.y');
   plot3(c6(:,1),c6(:,2),c6(:,3),'.m');
   hold off;
end % if
```

B.20 Plot Fish Record (plot_fish.m)

```
function plot_fish(x,t,traj,tmin,tmax,Nt,woff,bound,frec,fnum)

% plot_fish plots the fish trajectory and its VFDT.
% x - location of the fish (1D vector)
% t - time at each position (1D vector)
% traj - VFDT of x (they must be the same length) (1D vector)
% tmin - first point to plot (tmin >= 1)
% tmax - last point to plot (tmax <= length(t)) (tmin < tmax)
% Nt - size of window (256, 512, 1024, 2048, 4096, or 8192)
% woff - window offset (0 <= wn <= Nt)
% bound - ensure that traj <= 2 ? (boolean)
% frec - fish record identification
% fnum - figure number
%
% PLOT_FISH(X,T,TRAJ,TMIN,TMAX,NT,WOFF,BOUND,FREC,FNUM) plots the
% position X of a fish and the variance fractal dimension trajectory
% TRAJ at all points in time T (TMIN <= T <= TMAX).  If BOUND == 1,
% then the amplitude of TRAJ is ensured to be between 1 and 2 (incl).
% FREC, NT, and WOFF are the fish record identification, window size,
% and window offset, which are used in the title of figure FNUM.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.16 on July 5, 2003 at 6:00 pm (Central)

if (bound == 1)                 % ensure that traj <= 2 ?
    tb = traj;
    for k = 1 : length(tb)
        if (tb(k) > 2)
            tb(k) = 2;
        elseif (tb(k) < 1)
            tb(k) = 1;
        end % if
    end % for
    traj = tb;                  % save truncated trajectory
end % if

f = 10;                         % sampling frequency
t = t / (60*60*f);              % time in hours

figure(fnum);
subplot(2,1,1);
```

```
plot(t(tmin:tmax),x(tmin:tmax)); % plot fish traffic
title(['Fish Traffic - ', frec]);
xlabel('Time (hrs)');
ylabel('Distance from Mirror (mm)');
subplot(2,1,2);
plot(t(tmin:tmax),traj(tmin:tmax)); % plot VFDT
title(['VFDT - Window Size = ', int2str(Nt), ' , Window Displacement = ',
int2str(woff)]);
xlabel('Time (hrs)');
ylabel('Variance Fractal Dimension');
```

B.21 PNN Classification (PNNclassification.m)

```
% PNNclassification classifies previously unobserved
% traffic
%
% This program trains an optimal PNN with to classify
% previously unobserved traffic.  The percent correct
% classification and confusion matrix are displayed.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.0 on July 25, 2003 at 11:15 pm (Central)

clear all variables;
nPC = 4;                          % principal components
load ../mat/PC.mat; PC = PC';
PC = PC(1:nPC,:);                 % first 4 PCs

load ../mat/c3.mat; c = c3';      % c = 3

sigma = 0.067;                    % optimized sigma
                                  % when 50% and c = 3

load ../mat/PNNsetsSig.mat;       % training & test sets
SetTrain = SetTrainReg50; SetTest = SetTestReg50;
%SetTrain = SetTrainRan50_1; SetTest = SetTestRan50_1;
%SetTrain = SetTrainRan50_2; SetTest = SetTestRan50_2;
%SetTrain = SetTrainRan50_3; SetTest = SetTestRan50_3;
%SetTrain = SetTrainRan50_4; SetTest = SetTestRan50_4;
%SetTrain = SetTrainRan50_5; SetTest = SetTestRan50_5;

ctrain = zeros(nPC,length(SetTrain));
ctest = zeros(nPC,length(SetTest));
cltrain = zeros(1,length(SetTrain));
cltest =  zeros(1,length(SetTest));

for j = 1 : length(SetTrain)      % create training vectors
   ctrain(:,j) = PC(:,SetTrain(j));
   cltrain(j) = c(j);
end % for
for j = 1 : length(SetTest)       % create test vectors
   ctest(:,j) = PC(:,SetTest(j));
   cltest(j) = c(j);
end % for
```

```
T = ind2vec(cltrain);              % targets
net = newpnn(ctrain,T,sigma);      % train PNN
Y = sim(net,ctest);                % test PNN
Yc = vec2ind(Y);

conmatrix = zeros(3);              % confusion matrix
results = cltest - Yc;
numwrong = 0;
for j = 1:length(results)
   if (results(j) ~= 0)
      results(j) = -1;             % misclassification
      numwrong = numwrong + 1;
   end % if
   conmatrix(cltest(j),Yc(j)) = conmatrix(cltest(j),Yc(j)) + 1;
end % for
percor = (1-(numwrong/length(results)))*100
conmatrix
```

B.22 Read Fish Record (read_fish.m)

```
function [x,y,z,t] = read_fish(frec);

% read_fish reads a data record received from Dr. Pear (U of M).
% frec - fish record identification
%
% [X,Y,Z,T] = READ_FISH(FREC) reads the fish record FREC and returns
% X, Y, and Z at all times T.  Tracking errors are removed using
% linear interpolation.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.13 on July 8, 2003 at 6:05 pm (Central)

T = dlmread(frec);
t = T(:,1);
x = T(:,2);
y = T(:,3);
z = T(:,4);
e = T(:,5);
clear T;                          % save memory!

% For X coordinate...

k = 1;
while (e(k) == 0)                 % initial tracking errors?
    k = k + 1;                    % find the first instance of no errors
end % while
xt = x(k);                        % temporary value
for kt = 1 : k-1                  % remove initial tracking errors
    x(kt) = xt;
end % for
k1 = k+1;

k = length(e);
while (e(k) == 0)                 % final tracking errors?
    k = k - 1;                    % find the last instance of no errors
end % while
xt = x(k);                        % temporary value
for kt = k+1 : length(e)          % remove final tracking errors
    x(kt) = xt;
end % for
k2 = k;
```

```
while (k1 < k2)                    % remove all tracking errors by
    if (e(k1) == 0)                % linear interpolation
        xt = x(k1-1);              % store previous known point
        d = 1;                     % distance between known points
        while (e(k1) == 0)         % how many tracking errors?
            k1 = k1 + 1;
            d = d + 1;
        end % while
        inc = (x(k1) - x(k1-d)) / d; % increment
        for kt = (k1-d+1) : (k1-1)
            xt = xt + inc;             % interpolate
            x(kt) = xt;
        end % for
    end % if
    k1 = k1 + 1;
end % while

if (min(x) < 0)
    x = x + abs(min(x));           % make distances non-negative
end % if
if (min(x) > 0)
    x = x - min(x);                % or make smallest distance = 0
end % if

% For Y coordinate...

k = 1;
while (e(k) == 0)                  % initial tracking errors?
    k = k + 1;                     % find the first instance of no errors
end % while
yt = y(k);                         % temporary value
for kt = 1 : k-1                   % remove initial tracking errors
    y(kt) = yt;
end % for
k1 = k+1;

k = length(e);
while (e(k) == 0)                  % final tracking errors?
    k = k - 1;                     % find the last instance of no errors
end % while
yt = y(k);                         % temporary value
for kt = k+1 : length(e)           % remove final tracking errors
    y(kt) = yt;
end % for
k2 = k;

while (k1 < k2)                    % remove all tracking errors by
    if (e(k1) == 0)                % linear interpolation
```

```
        yt = y(k1-1);              % store previous known point
        d = 1;                     % distance between known points
        while (e(k1) == 0)         % how many tracking errors?
            k1 = k1 + 1;
            d = d + 1;
        end % while
        inc = (y(k1) - y(k1-d)) / d; % increment
        for kt = (k1-d+1) : (k1-1)
            yt = yt + inc;             % interpolate
            y(kt) = yt;
        end % for
    end % if
    k1 = k1 + 1;
end % while

if (min(y) < 0)
    y = y + abs(min(y));           % make distances non-negative
end % if
if (min(y) > 0)
    y = y - min(y);                % or make smallest distance = 0
end % if

% For Z coordinate...

k = 1;
while (e(k) == 0)                  % initial tracking errors?
    k = k + 1;                     % find the first instance of no errors
end % while
zt = z(k);                         % temporary value
for kt = 1 : k-1                   % remove initial tracking errors
    z(kt) = zt;
end % for
k1 = k+1;

k = length(e);
while (e(k) == 0)                  % final tracking errors?
    k = k - 1;                     % find the last instance of no errors
end % while
zt = z(k);                         % temporary value
for kt = k+1 : length(e)           % remove final tracking errors
    z(kt) = zt;
end % for
k2 = k;

while (k1 < k2)                    % remove all tracking errors by
    if (e(k1) == 0)                % linear interpolation
        zt = z(k1-1);              % store previous known point
        d = 1;                     % distance between known points
```

```
        while (e(k1) == 0)        % how many tracking errors?
            k1 = k1 + 1;
            d = d + 1;
        end % while
        inc = (z(k1) - z(k1-d)) / d; % increment
        for kt = (k1-d+1) : (k1-1)
            zt = zt + inc;            % interpolate
            z(kt) = zt;
        end % for
    end % if
    k1 = k1 + 1;
end % while

if (min(z) < 0)
    z = z + abs(min(z));          % make distances non-negative
end % if
if (min(z) > 0)
   z = z - min(z);                % or make smallest distance = 0
end % if

%save ../mat/11020219.mat x y z t; % save record
```

B.23 Rényi Multifractal Dimension Spectrum (Renyi.m)

```
function Dq = Renyi(traf,qr)

% Dq - Renyi dimension spectrum
% traf - traffic sequence
% qr - range of q
%
% DQ = RENYI(TRAF,QR) returns the Renyi dimension spectrum
% of a sequence of self-affine traffic TRAF with spectrum
% range QR.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 2.01 on July 26, 2003 at 7:30 pm (Central)

traf = traf ./ max(traf);          % traf amplitude is now [-1,1]
traf = traf / 1.001;               % traf amplitude is now (-1,1)
Mx = 1; Mn = -1;                   % max and min mesh ranges

LTr = length(traf);
for w = 1 : LTr                    % convert traffic sequence into a
   a = w - LTr/2;                  % complex number representation
   a = (a/(LTr/2)) / 1.001;        % traf range is now (-1,1)
   traf(w) = a + traf(w)*i;
end % for

bin = LTr / 2;
r = real(traf(2)-traf(1)) * 2;     % maximum reliable resolution
mx = linspace(Mn,Mx,bin+1);        % create a mesh where
my = mx;                           % mx - rows; my - columns
p = zeros(bin,bin);                % # of corners in vel

for b = 1 : LTr                    % calculate the location of a vel
   c = 1;                          % for a given point in the sequence
   target = imag(traf(b));         % row
   while not((target >= mx(c)) & (target < mx(c+1)))
      c = c + 1; end % while
   x = c; c = 1;
   target = real(traf(b));         % column
   while not((target >= my(c)) & (target < my(c+1)))
      c = c + 1; end % while
   y = c;
   p(x,y) = p(x,y) + 1;            % update bin
```

```
end % for b

[m,n] = find(p > 0);
p = p / length(traf);              % create probabilities
for q = -qr : qr
   H = 0;
   for pos = 1:length(m)
      b = m(pos); c = n(pos);
      H = H + ((p(b,c))^q);        % Renyi entropy
   end % for
   if (q ~= 1)                     % avoid division by 0
      H = (log2(H))/(1-q);
   end % if
   D(q+qr+1) = H / log2(1/r);
   Dq(q+qr+1) = q;
end % for q
D(qr+2) = (D(qr+1)+D(qr+3)) / 2; % when q = 1
plot(Dq,D);
xlabel('q');
ylabel('D_q');
```

B.24 Select Number of Classes (selectnumclasses.m)

```
% selectnumclasses indicates the most likely number of
% classes in the data
%
% This program helps the user detect the most likely
% number of classes in the data by calculating the
% classification accuracy with good values of sigma
% (as determined by the program 'selectsigmas') for
% an increasing number of classes from c = 2 to 10.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.02 on July 24, 2003 at 2:10 pm (Central)

clear all variables;
nPC = 4;                          % principal components
load ../mat/PC.mat; PC = PC';
PC = PC(1:nPC,:);                 % first 4 PCs

load ../mat/sigmas.mat;           % good sigmas for each
                                  % number of classes
sigma = sigma30;
%sigma = sigma40;
%sigma = sigma50;

load ../mat/PNNsetsReg.mat;       % training & test sets
SetTrain = SetTrainReg30; SetTest = SetTestReg30;
%SetTrain = SetTrainReg40; SetTest = SetTestReg40;
%SetTrain = SetTrainReg50; SetTest = SetTestReg50;

c = zeros(10,length(PC));         % all classes
load ../mat/c2.mat; c(2,:) = c2';
load ../mat/c3.mat; c(3,:) = c3';
load ../mat/c4.mat; c(4,:) = c4';
load ../mat/c5.mat; c(5,:) = c5';
load ../mat/c6.mat; c(6,:) = c6';
load ../mat/c7.mat; c(7,:) = c7';
load ../mat/c8.mat; c(8,:) = c8';
load ../mat/c9.mat; c(9,:) = c9';
load ../mat/c10.mat; c(10,:) = c10';

percor = zeros(1,length(sigma)); % percent correct
percor(1,1) = 100;                % always 100% @ c = 1
```

```
for i = 1 : (length(sigma)-1)     % 9 simulations
   i+1
   clear ctrain ctest cltrain cltest sig T; % clear
   clear net Y Yc results numwrong;

   ctrain = zeros(nPC,length(SetTrain));
   ctest = zeros(nPC,length(SetTest));
   cltrain = zeros(1,length(SetTrain));
   cltest =  zeros(1,length(SetTest));

   for j = 1 : length(SetTrain)  % create training vectors
      ctrain(:,j) = PC(:,SetTrain(j));
      cltrain(j) = c((i+1),j);
   end % for
   for j = 1 : length(SetTest)   % create test vectors
      ctest(:,j) = PC(:,SetTest(j));
      cltest(j) = c((i+1),j);
   end % for

   sig = sigma(i+1);
   T = ind2vec(cltrain);          % targets
   net = newpnn(ctrain,T,sig);    % train PNN
   Y = sim(net,ctest);            % test PNN
   Yc = vec2ind(Y);

   results = cltest - Yc;
   numwrong = 0;
   for j = 1:length(results)
      if (results(j) ~= 0)
         results(j) = -1;         % misclassification
         numwrong = numwrong + 1;
      end % if
   end % for
   percor(1,(i+1)) = (1-(numwrong/length(results)))*100;
end % for (sigma)
percor
%percor30 = percor;
%load ../mat/percor.mat;
%save ../mat/percor.mat percor*;

i = linspace(2,length(sigma),(length(sigma)-1));
plot(i,percor(2:length(sigma)));
xlabel('Number of Classes');
ylabel('Percentage Correct Classification');
```

B.25 Select Sigmas (selectsigmas.m)

```
% selectsigmas indicates a good value of sigma for a
% given number of classes
%
% This program finds a value of sigma which gives the
% highest correct classification for a given number of
% classes and a P%/(P-1)% training/test set ratio
% constructed by selecting data from regular or random
% intervals.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.12 on July 23, 2003 at 11:50 pm (Central)

clear all variables;
nPC = 4;                              % principal components
load ../mat/PC.mat; PC = PC';
PC = PC(1:nPC,:);                     % first 4 PCs

%load ../mat/c2.mat; c = c2';
load ../mat/c3.mat; c = c3';
%load ../mat/c4.mat; c = c4';
%load ../mat/c5.mat; c = c5';
%load ../mat/c6.mat; c = c6';
%load ../mat/c7.mat; c = c7';
%load ../mat/c8.mat; c = c8';
%load ../mat/c9.mat; c = c9';
%load ../mat/c10.mat; c = c10';

load ../mat/PNNsetsSig.mat;       % training & test sets

%SetTrain = SetTrainReg30; SetTest = SetTestReg30;
%SetTrain = SetTrainRan30_1; SetTest = SetTestRan30_1;
%SetTrain = SetTrainRan30_2; SetTest = SetTestRan30_2;

%SetTrain = SetTrainReg40; SetTest = SetTestReg40;
%SetTrain = SetTrainRan40_1; SetTest = SetTestRan40_1;
%SetTrain = SetTrainRan40_2; SetTest = SetTestRan40_2;
%SetTrain = SetTrainRan40_3; SetTest = SetTestRan40_3;

SetTrain = SetTrainReg50; SetTest = SetTestReg50;
%SetTrain = SetTrainRan50_1; SetTest = SetTestRan50_1;
%SetTrain = SetTrainRan50_2; SetTest = SetTestRan50_2;
```

```
ctrain = zeros(nPC,length(SetTrain));
ctest = zeros(nPC,length(SetTest));
cltrain = zeros(1,length(SetTrain));
cltest =  zeros(1,length(SetTest));

for i = 1 : length(SetTrain)     % create training vectors
   ctrain(:,i) = PC(:,SetTrain(i));
   cltrain(i) = c(i);
end % for
for i = 1 : length(SetTest)      % create test vectors
   ctest(:,i) = PC(:,SetTest(i));
   cltest(i) = c(i);
end % for

%sigma = zeros(1,25);             % try sigmas between
%sigma(7:25) = linspace(1,10,19); % 0.001 and 10.0

sigma = zeros(1,15);             % try sigmas between
sigma(7:15) = linspace(1,5,9);   % 0.001 and 5.0

sigma(1) = 0.001;                % constant sigmas
sigma(2) = 0.005;
sigma(3) = 0.01;
sigma(4) = 0.05;
sigma(5) = 0.1;
sigma(6) = 0.5;

percor = zeros(1,length(sigma));

slen = length(sigma);
for i = 1 : slen
   sig = sigma(i);
   T = ind2vec(cltrain);         % targets
   net = newpnn(ctrain,T,sig);   % train PNN
   Y = sim(net,ctest);           % test PNN
   Yc = vec2ind(Y);

   results = cltest - Yc;
   numwrong = 0;
   for j = 1:length(results)
      if (results(j) ~= 0)
         results(j) = -1;        % misclassification
         numwrong = numwrong + 1;
      end % if
   end % for
   percor(i) = 1 - (numwrong/length(results));
end % for (sigma)
percor
```

```
plot(sigma(1:slen),percor(1:slen));
xlabel('Sigma');
ylabel('Percentage Correct Classification');
sigma(find(percor == max(percor)))
```

B.26 Verify Rényi Dimension Spectrum (VerifyRD.m)

```
function [RYES,RNO,RNUM] = VerifyRD(xt,Nt)

% VerifyRD stands for "verify Rényi Dimension"
% xt - 1D vector (xt >= Nt)
% Nt - size of window (256, 512, 1024, 2048, 4096, or 8192)
%
% [RYES,RNO,RNUM] = VERIFYRD(XT,NT) returns the number of
% Rényi dimensions RYES that demonstrate multifractality,
% the number RNO that do not, and the number of non-zero
% cells RNUM of XT using windows of size NT.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.0 on June 30, 2003 at 9:30 pm (Central)

RYES = 0; RNO = 0;
r_min = 1;                          % min window value
r_max = Nt;                         % max window value
tj = zeros(1,length(xt));           % empty array (to speed up program)
j = 0;
while (r_max <= length(xt))
j = j + 1                           % counter
traf = xt(r_min : r_max);           % isolate a window of traffic

traf = traf ./ max(traf);           % traf amplitude is now [-1,1]
traf = traf / 1.001;                % traf amplitude is now (-1,1)
Mx = 1; Mn = -1;                    % max and min mesh ranges

LTr = length(traf);
for w = 1 : LTr                     % convert traffic sequence into a
   a = w - LTr/2;                   % complex number representation
   a = (a/(LTr/2)) / 1.001;         % traf range is now (-1,1)
   traf(w) = a + traf(w)*i;
end % for

bin = LTr / 2;
r = real(traf(2)-traf(1)) * 2;      % maximum reliable resolution
mx = linspace(Mn,Mx,bin+1);         % create a mesh where
my = mx;                            % mx - rows; my - columns
p = zeros(bin,bin);                 % # of corners in vel

for b = 1 : LTr                     % calculate the location of a vel
```

```
   c = 1;                          % for a given point in the sequence
   target = imag(traf(b));         % row
   while not((target >= mx(c)) & (target < mx(c+1)))
      c = c + 1; end % while
   x = c; c = 1;
   target = real(traf(b));         % column
   while not((target >= my(c)) & (target < my(c+1)))
      c = c + 1; end % while
   y = c;
   p(x,y) = p(x,y) + 1;            % update bin
end % for b

[m,n] = find(p > 0);
if (length(m) < Nt)
   RYES = RYES + 1;
else % (length(m) >= Nt)
   RNO = RNO + 1;
end % if
tj(j) = length(m);                 % store vector

r_max/length(xt)*100               % display percentage completion
r_min = r_min + Nt;                % update window position
r_max = r_max + Nt;
end % while
RNUM = tj(1:j);
```

B.27 Verify Self-Affinity (VerifySA.m)

```
function [X,Y,s,D] = VerifySA(traf,t,Nt)

% VerifySA verifies the self-affinity of a window of data
% traf - 1D vector (traf >= Nt)
% t - corresponding time segment
% Nt - size of window (256, 512, 1024, 2048, 4096, or 8192)
%
% [X,Y,S,D] = VERIFYSA(TRAF,T,NT) returns the X and Y
% coordinates of a log-log plot, slope S, and dimension
% D of TRAF during time T using windows of size NT.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.0 on June 23, 2003 at 6:30 pm (Central)

r_min = 1;                          % min window value
r_max = Nt;                         % max window value
b = 2;                              % dyadic sequence
K_low = 2;                          % minimum separation
K_max = fix(log(Nt)/log(b));        % maximum separation
K_buf = ceil(log(30)/log(b));
K_hi  = K_max - K_buf;
x = traf(r_min : r_max);            % isolate a window of traffic

n(1) = 1;                           % avoid division by zero (with N)
n(2) = 2^2;    n(3) = 2^3;
if (Nt > 256)  n(4) = 2^4; end      % points in interval
if (Nt > 512)  n(5) = 2^5; end
if (Nt > 1024) n(6) = 2^6; end
if (Nt > 2048) n(7) = 2^7; end
if (Nt > 4096) n(8) = 2^8; end

N = Nt ./ n;                        % number of such intervals
n(1) = 0; N(1) = 0;                 % mark invalid cells with zeros
fv = x(1)*ones(1,length(n));        % assign first values (fv)

ta = zeros(1,K_hi);                 % terms a and b in the variance
tb = ta;                            % equation that is used below

for k = 2 : Nt                      % look at each point in the window
   if (mod(k,b^2) == 0)             % n = 4
      ta(2) = ta(2) + (fv(2)-x(k))^2;
```

```
      tb(2) = tb(2) + (fv(2)-x(k));
      fv(2) = x(k); end % if
   if (mod(k,b^3) == 0)          % n = 8
      ta(3) = ta(3) + (fv(3)-x(k))^2;
      tb(3) = tb(3) + (fv(3)-x(k));
      fv(3) = x(k); end % if
if (Nt > 256)
   if (mod(k,b^4) == 0)          % n = 16
      ta(4) = ta(4) + (fv(4)-x(k))^2;
      tb(4) = tb(4) + (fv(4)-x(k));
      fv(4) = x(k); end; end % if / if
if (Nt > 512)
   if (mod(k,b^5) == 0)          % n = 32
      ta(5) = ta(5) + (fv(5)-x(k))^2;
      tb(5) = tb(5) + (fv(5)-x(k));
      fv(5) = x(k); end; end % if / if
if (Nt > 1024)
   if (mod(k,b^6) == 0)          % n = 64
      ta(6) = ta(6) + (fv(6)-x(k))^2;
      tb(6) = tb(6) + (fv(6)-x(k));
      fv(6) = x(k); end; end % if / if
if (Nt > 2048)
   if (mod(k,b^7) == 0)          % n = 128
      ta(7) = ta(7) + (fv(7)-x(k))^2;
      tb(7) = tb(7) + (fv(7)-x(k));
      fv(7) = x(k); end; end % if / if
if (Nt > 4096)
   if (mod(k,b^8) == 0)          % n = 256
      ta(8) = ta(8) + (fv(8)-x(k))^2;
      tb(8) = tb(8) + (fv(8)-x(k));
      fv(8) = x(k); end; end % if / if
end % for

K = 0;
for k = K_low : K_hi             % calculate the variance
    K = K + 1;
    Vr(K) = (ta(k)-(((tb(k))^2)/N(k)))/(N(k)-1);
    Vn(K) = n(k);
end % for
X = log2(Vn);                    % compute X and Y
Y = log2(Vr);

K = K_hi - K_low + 1;
s1 = (K*sum(X.*Y)) - (sum(X)*sum(Y));
s2 = (K*sum(X.^2)) - ((sum(X))^2);
s = s1 / s2;                     % calculate the slope s
H = s/2;                         % calculate the Hurst exponent
E = 1;                           % Euclidean embedding dimension
```

```
D = E + 1 - H;                          % calculate variance dimension

figure(1);                              % plot original data
plot((t./10),traf);
xlabel('Time (sec)');
ylabel('Distance from Mirror (mm)');

figure(2);
plot(X,Y,'o');                          % plot X and Y
grid on;
xlabel('log(X)');
ylabel('log(Y)');
[P,S] = polyfit(X,Y,1);                 % plot line of best fit
Y1 = P(1) .* X + P(2);
hold on;
plot(X,Y1);
hold off;
```

B.28 Verify Self-Affinity Histogram (VerifySAHist.m)

```
function VFDTMSEHist = VerifySAHist(traf,Nt,wn)

% VerifySAHist displays a histogram of the self-affinity MSE
% traf - 1D vector (traf >= Nt)
% Nt - size of window (256, 512, 1024, 2048, 4096, or 8192)
% wn - size of window overlap (0 <= wn <= Nt)
%    - if wn = 0 -> no overlap
%    - if wn = p (0 < p < 1) -> p percent overlap
%    - if wn = 1 -> maximum fractal amplification
%    - if wn = n (1 < n < Nt) -> n point offset
%
% [VFDTMSEHist] = VERIFYSAHIST(TRAF,NT,WN) returns the
% histogram of the mean square error between the points
% on the log-log plot and the line of best fit of TRAF
% using windows of size NT and offset WN.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.0 on June 24, 2003 at 5:30 pm (Central)

if (wn == 0)                        % calculate window offset
    woff = Nt;                      % no overlap
elseif (wn == 1)
    woff = 1;                       % maximum overlap
elseif ((wn > 0) & (wn < 1))
    woff = round(wn * Nt);
    woff = Nt - woff;               % percentage overlap
else % (wn > 1)
    woff = wn;                      % fixed offset value
end % if
if (woff > Nt)                      % check for rounding errors
    woff = Nt;
elseif (woff < 1)
    woff = 1;
end % if

r_min = 1;                          % min window value
r_max = Nt;                         % max window value
tj = zeros(1,length(traf));         % empty array (to speed up program)
j = 0;
while (r_max <= length(traf))
j = j + 1;                          % counter
```

```
b = 2;                              % dyadic sequence
K_low = 2;                          % minimum separation
K_max = fix(log(Nt)/log(b));        % maximum separation
K_buf = ceil(log(30)/log(b));
K_hi  = K_max - K_buf;
x = traf(r_min : r_max);            % isolate a window of traffic

n(1) = 1;                           % avoid division by zero (with N)
n(2) = 2^2;    n(3) = 2^3;
if (Nt > 256)  n(4) = 2^4; end      % points in interval
if (Nt > 512)  n(5) = 2^5; end
if (Nt > 1024) n(6) = 2^6; end
if (Nt > 2048) n(7) = 2^7; end
if (Nt > 4096) n(8) = 2^8; end

N = Nt ./ n;                        % number of such intervals
n(1) = 0; N(1) = 0;                 % mark invalid cells with zeros
fv = x(1)*ones(1,length(n));        % assign first values (fv)

ta = zeros(1,K_hi);                 % terms a and b in the variance
tb = ta;                            % equation that is used below

for k = 2 : Nt                      % look at each point in the window
   if (mod(k,b^2) == 0)             % n = 4
      ta(2) = ta(2) + (fv(2)-x(k))^2;
      tb(2) = tb(2) + (fv(2)-x(k));
      fv(2) = x(k); end % if
   if (mod(k,b^3) == 0)             % n = 8
      ta(3) = ta(3) + (fv(3)-x(k))^2;
      tb(3) = tb(3) + (fv(3)-x(k));
      fv(3) = x(k); end % if
if (Nt > 256)
   if (mod(k,b^4) == 0)             % n = 16
      ta(4) = ta(4) + (fv(4)-x(k))^2;
      tb(4) = tb(4) + (fv(4)-x(k));
      fv(4) = x(k); end; end % if / if
if (Nt > 512)
   if (mod(k,b^5) == 0)             % n = 32
      ta(5) = ta(5) + (fv(5)-x(k))^2;
      tb(5) = tb(5) + (fv(5)-x(k));
      fv(5) = x(k); end; end % if / if
if (Nt > 1024)
   if (mod(k,b^6) == 0)             % n = 64
      ta(6) = ta(6) + (fv(6)-x(k))^2;
      tb(6) = tb(6) + (fv(6)-x(k));
      fv(6) = x(k); end; end % if / if
if (Nt > 2048)
   if (mod(k,b^7) == 0)             % n = 128
```

```
      ta(7) = ta(7) + (fv(7)-x(k))^2;
      tb(7) = tb(7) + (fv(7)-x(k));
      fv(7) = x(k); end; end % if / if
if (Nt > 4096)
   if (mod(k,b^8) == 0)          % n = 256
      ta(8) = ta(8) + (fv(8)-x(k))^2;
      tb(8) = tb(8) + (fv(8)-x(k));
      fv(8) = x(k); end; end % if / if
end % for

K = 0;
for k = K_low : K_hi             % calculate the variance
    K = K + 1;
    Vr(K) = (ta(k)-(((tb(k))^2)/N(k)))/(N(k)-1);
    Vn(K) = n(k);
end % for
X = log2(Vn);                    % compute X and Y
Y = log2(Vr);

[P,S] = polyfit(X,Y,1);          % line of best fit
Y1 = P(1) .* X + P(2);
MSE = sum((Y-Y1).^2) / length(Y); % mean square error
tj(j) = MSE;                     % store vector

r_min = r_min + woff;            % update window position
r_max = r_max + woff;
(r_max-woff)/length(traf)*100    % display percentage completion
end % while
VFDTMSEHist = tj(1:j);           % final trajectory
```

B.29 Variance Fractal Dimension Trajectory (VFDT.m)

```
function [traj,woff] = vfdt(traf,Nt,wn)

% vfdt stands for "variance fractal dimension trajectory"
% traf - 1D vector (traf >= Nt)
% Nt - size of window (256, 512, 1024, 2048, 4096, or 8192)
% wn - size of window overlap (0 <= wn <= Nt)
%    - if wn = 0 -> no overlap
%    - if wn = p (0 < p < 1) -> p percent overlap
%    - if wn = 1 -> maximum fractal amplification
%    - if wn = n (1 < n < Nt) -> n point offset
%
% [TRAJ,WOFF] = VFDT(TRAF,NT,WN) returns the variance fractal
% dimension trajectory TRAJ and the window displacement WOFF of
% a 1D sequence TRAF using windows of size NT, and an overlap
% parameter WN.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.47 on July 14, 2002 at 4:00 pm (Central)

if (wn == 0)                        % calculate window offset
    woff = Nt;                      % no overlap
elseif (wn == 1)
    woff = 1;                       % maximum overlap
elseif ((wn > 0) & (wn < 1))
    woff = round(wn * Nt);
    woff = Nt - woff;               % percentage overlap
else % (wn > 1)
    woff = wn;                      % fixed offset value
end % if
if (woff > Nt)                      % check for rounding errors
    woff = Nt;
elseif (woff < 1)
    woff = 1;
end % if

r_min = 1;                          % min window value
r_max = Nt;                         % max window value
tj = zeros(1,length(traf));         % empty array (to speed up program)
j = 0;
while (r_max <= length(traf))
j = j + 1;                          % counter
```

```
b = 2;                                  % dyadic sequence
K_low = 2;                              % minimum separation
K_max = fix(log(Nt)/log(b));            % maximum separation
K_buf = ceil(log(30)/log(b));
K_hi  = K_max - K_buf;
x = traf(r_min : r_max);                % isolate a window of traffic

n(1) = 1;                               % avoid division by zero (with N)
n(2) = 2^2;    n(3) = 2^3;
if (Nt > 256)  n(4) = 2^4; end          % points in interval
if (Nt > 512)  n(5) = 2^5; end
if (Nt > 1024) n(6) = 2^6; end
if (Nt > 2048) n(7) = 2^7; end
if (Nt > 4096) n(8) = 2^8; end

N = Nt ./ n;                            % number of such intervals
n(1) = 0; N(1) = 0;                     % mark invalid cells with zeros
fv = x(1)*ones(1,length(n));            % assign first values (fv)

ta = zeros(1,K_hi);                     % terms a and b in the variance
tb = ta;                                % equation that is used below

for k = 2 : Nt                          % look at each point in the window
   if (mod(k,b^2) == 0)                 % n = 4
      ta(2) = ta(2) + (fv(2)-x(k))^2;
      tb(2) = tb(2) + (fv(2)-x(k));
      fv(2) = x(k); end % if
   if (mod(k,b^3) == 0)                 % n = 8
      ta(3) = ta(3) + (fv(3)-x(k))^2;
      tb(3) = tb(3) + (fv(3)-x(k));
      fv(3) = x(k); end % if
if (Nt > 256)
   if (mod(k,b^4) == 0)                 % n = 16
      ta(4) = ta(4) + (fv(4)-x(k))^2;
      tb(4) = tb(4) + (fv(4)-x(k));
      fv(4) = x(k); end; end % if / if
if (Nt > 512)
   if (mod(k,b^5) == 0)                 % n = 32
      ta(5) = ta(5) + (fv(5)-x(k))^2;
      tb(5) = tb(5) + (fv(5)-x(k));
      fv(5) = x(k); end; end % if / if
if (Nt > 1024)
   if (mod(k,b^6) == 0)                 % n = 64
      ta(6) = ta(6) + (fv(6)-x(k))^2;
      tb(6) = tb(6) + (fv(6)-x(k));
      fv(6) = x(k); end; end % if / if
if (Nt > 2048)
   if (mod(k,b^7) == 0)                 % n = 128
```

```
      ta(7) = ta(7) + (fv(7)-x(k))^2;
      tb(7) = tb(7) + (fv(7)-x(k));
      fv(7) = x(k); end; end % if / if
if (Nt > 4096)
   if (mod(k,b^8) == 0)          % n = 256
      ta(8) = ta(8) + (fv(8)-x(k))^2;
      tb(8) = tb(8) + (fv(8)-x(k));
      fv(8) = x(k); end; end % if / if
end % for

K = 0;
for k = K_low : K_hi             % calculate the variance
    K = K + 1;
    Vr(K) = (ta(k)-(((tb(k))^2)/N(k)))/(N(k)-1);
    Vn(K) = n(k);
end % for
X = log2(Vn);                    % compute X and Y
Y = log2(Vr);

K = K_hi - K_low + 1;
s1 = (K*sum(X.*Y)) - (sum(X)*sum(Y));
s2 = (K*sum(X.^2)) - ((sum(X))^2);
s = s1 / s2;                     % calculate the slope s
H = s/2;                         % calculate the Hurst exponent
E = 1;                           % Euclidean embedding dimension
D = E + 1 - H;                   % calculate variance dimension
tj(j) = D;                       % create fractal trajectory

r_min = r_min + woff;            % update window position
r_max = r_max + woff;
(r_max-woff)/length(traf)*100    % display percentage completion
end % while
traj = tj(1:j);                  % final trajectory
```

B.30 Variance Fractal Dimension Trajectory Interpolation (VFDTinter.m)

```
function traj = VFDTinter(tj,Nt,woff,newlen)

% VFDTinter interpolates the VFDT
% tj - 1D vector (tj >= Nt)
% Nt - size of window
% woff - window displacement (woff <= Nt)
% newlen - new trajectory length (newlen > length(tj))
%
% TRAJ = VFDTINTER(TJ,NT,WN,NEWLEN) returns the interpolated variance
% fractal dimension trajectory TRAJ with length NEWLEN of a sequence
% TJ, calculated using windows of size NT and displacement WOFF.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.0 on July 14, 2002 at 6:30 pm (Central)

traj = zeros(1,newlen);          % create empty trajectory
traj(1:(Nt-1)) = 1;
traj(Nt) = tj(1);
k = floor((newlen-Nt)/woff);     % remainder
r_max = Nt + woff*k;             % position of point in last window
traj((r_max-woff):newlen) = tj(length(tj)); % hold last point
trajgap = r_max - woff - 1 - Nt; % # of trajectory points to fill
m = trajgap / (length(tj)-2);    % average separation distance
newpt = Nt + floor(m/2);         % next point to fill
for k = 2 : (length(tj)-1)       % evenly spread out trajectory
    traj(floor(newpt)) = tj(k);
    newpt = newpt + m;
end % for

k1 = Nt + 1;
k2 = r_max - woff;
while (k1 < k2)                  % use linear interpolation to
    if (traj(k1) == 0)           % calculate intermediate dimensions
        trajt = traj(k1-1);      % store previous known point
        d = 1;                   % distance between known points
        while (traj(k1) == 0)    % how long is the gap?
            k1 = k1 + 1;
            d = d + 1;
        end % while
        inc = (traj(k1) - traj(k1-d)) / d; % increment
        for kt = (k1-d+1) : (k1-1)
```

```
            trajt = trajt + inc;            % interpolate
            traj(kt) = trajt;
        end % for
    end % if
    k1 = k1 + 1;
end % while
```

B.31 Variance Fractal Dimension Trajectory Statistics (VFDTstats.m)

```
function [Tm,Tv,Ts,Tk] = VFDTstats(traj,Nt)

% VFDTstats computes the statistics of the VFDT
% traj - 1D vector (traj >= Nt)
% Nt - size of window
%
% [TM,TV,TS,TK] = VFDTSTATS(TRAJ,NT) returns the mean, variance,
% skewness, and kurtosis trajectories of the variance fractal
% dimension trajectory TRAJ calculated using windows of size NT.
%
% Required functions: none

% Copyright (c) 2003 by Robert Barry (rbarry@pobox.com)
% Revision 1.01 on July 5, 2003 at 8:25 pm (Central)

ltraj = length(traj);
Tm = zeros(1,ltraj);              % create empty trajectories
Tv = Tm; Ts = Tm; Tk = Tm;
for k = 2*Nt : ltraj
    temp = traj((k-Nt+1):k);
    Tm(k) = mean(temp);           % mean
    Tv(k) = var(temp);            % variance (unbiased)
    Ts(k) = skewness(temp,1);     % skewness (unbiased)
    Tk(k) = kurtosis(temp,1);     % kurtosis (unbiased)
%    k
end % for

Tm(1:(2*Nt-1)) = Tm(2*Nt);        % set undefined values
Tv(1:(2*Nt-1)) = Tv(2*Nt);
Ts(1:(2*Nt-1)) = Ts(2*Nt);
Tk(1:(2*Nt-1)) = Tk(2*Nt);
```