COMPRESSION OF ECG SIGNALS WITH FEATURE EXTRACTION, CLASSIFICATION, AND BROWSABILITY

by
Bin Huang

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

DOCTOR OF PHILOSOPHY

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

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

(xx+247+A-7+B-84) = 358 pp.

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

Compression of ECG Signals with Feature Extraction, Classification, and Browsability

BY

Bin Huang

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

Of

DOCTOR OF PHILOSOPHY

ABSTRACT

This thesis presents nonlinear analysis techniques to characterize and compress a nonstationary electrocardiogram (ECG) signal in order to alleviate some of the limitations of linear methods in biological signals. The motivation behind the study is the need to reduce the size of storage and the time of transmission and analysis of the ECG signal since a long-term ECG recording produces large amounts of data. This research has resulted in three methods for ECG data compression: (i) nonlinear iterated function systems (IFS) compression and compositional complexity partitioning, (ii) statistical feature extraction and classification to compress and browse the ECG signal, and (iii) dynamic time warping (DTW) classification and block encoding.

A nonlinear IFS (NIFS) has been developed to compress ECG data based on self-similarity of the signal. Compared to the traditional IFS, the NIFS provides more flexible modelling. It gives a compression ratio of 7.3:1, which is higher than that of 6.0:1 in orthogonal fractal technique of Øien and Nårstad, under a reconstruction error of 5.8%. Furthermore, to reduce computational complexity, a variance fractal dimension trajectory (VFDT) is applied to partition the ECG signal by measuring the local compositional complexity of the signal. The segmented NIFS technique reduces computational complexity to $O(N)$ for a time series with length N, compared to $O(N^2)$ of the IFS, and achieves a compression ratio of 5.7:1 under the same reconstruction error.

A feature extraction and classification method is also proposed to reduce the redundancy among the beats of the ECG signal. Instead of comparing the beats of the quasiperiodic ECG signal directly, statistical features with the same dimension are extracted from the beats with various lengths. First, we propose to extract moment-invariants by treating ECG beats as character images, thus preserving the shape information of ECG waveforms. A probabilistic neural network is used to classify ECG beats through such features. The classification information is applied to remove the redundancy among ECG beats and to browse the long-term recording in ECG analysis. Secondly, the Rényi multifractal dimension spectrum based on the Rényi entropy measure of the object is extracted by modelling the ECG signal through its underlying strange attractor. The investigation of the ECG strange attractor helps us recognize limitations of the low-sampling frequency (360 Hz) and noise in phase space reconstruction of the ECG signal. Therefore, denoising techniques are applied to the ECG signal to remove the noise and thus improve the convergence of the Rényi dimension spectrum. A mean absolute difference (MAD) between two Rényi dimension spectra is proposed to measure the convergence of the Rényi dimension spectrum. Experimental results show that chaos denoising improves the convergence about five times under the MAD, while wavelet denoising degrades it about two times.

Finally, a modified DTW is applied to classify ECG frames by time normalizing two frames through a warping function. Since the warping function is complicated, a segment-based registration of ECG frames is proposed to approximate the ECG frame. Partitioning of ECG frames is realized by a windowed-variance technique. Then the block encoding is proposed to compress segments of ECG frames. This compression scheme achieves a very high compression ratio of about 50:1.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. W. Kinsner, not only for providing many of the ideas in this research and giving me his time and patience in the course of research, but also for teaching me the methodology of how to analyze and solve questions in research. I am also indebted for his extensive editing of this thesis document.

I would like to acknowledge everyone in the Delta Research Group, past and present, including Richard Dansereau, Hong Zhang, Luotao Sun, Alexis Denis, Tina Ehtiati, Jonathan Greenberg, Steven Miller, Reza Fazel, Hakim El-Boustani, Martin Stadler, Kalen Brunham, Sharjeel Siddiqui, Michael Potter, Robert Barry, Stephen Dueck, Neil Gadhok, Yongqing Fu, Jay Kraut, Aram Faghfouri, and Leila Sadat Safavian for the useful discussion. The suggestions and encouragement from all of these friendly people played important roles on my progress as a graduate student and researcher.

We would like to thank and acknowledge the financial support of the University of Manitoba, the Shaw Cable of Winnipeg, and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

I would also like to thank my parents-in-law for their unconditional support to my study. They help me take care of my son when I cannot bring him with me.

I dedicate the work done towards to my parents. Their constant support over the years has made many of my achievements possible. Finally, I dedicate the work to my wife, Jin, and my son, Yujing.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

1D	one dimensional
2D	two dimensional
3D	three dimensional
ALZ77	approximate Ziv-Lempel algorithm
AV	atrioventricular
AZTEC	amplitude zone time epoch coding algorithm [CNFO68]
BPN	backpropagation network
bps	bits per sample
CC	cycle-to-cycle
CCSP	cardinality constrained shortest path
CELP	code excited linear prediction
CORTES	coordinate reduction time encoding system algorithm
CPBC	cycle-pool-based compression algorithm
CWT	continuous wavelet transform
DC	direct current
DCT	discrete cosine transform
DPCM	differential pulse code modulation
DWPT	discrete wavelet packet transform
DWT	discrete wavelet transform
DTW	dynamic time warping
ECG	electrocardiogram or electrocardiography
EEG	electroencephalography
EHG	uterine electrohysterography
EMG	electromyogram or electromyography
EOG	electrooculography
FBC	fractal block coding
FFT	fast Fourier transform

FNN	false nearest neighbours
GWAVQ	gold washing adaptive vector quantization
ICWT	inverse continuous wavelet transform
IFS	iterated function systems [Barn88], [Jacq90]
IFSM	IFS map
IFSP	IFS probability
iid	independent and identically distributed
ISODATA	iterative self-organizing data analysis techniques A [HaBa65]
KLT	Karhunen-Loève transform
LA	left arm
LPT	Legendre polynomial transform
MAD	mean absolute difference
MI	moment-invariant
MLFN	multiple layer feedforward network
MPRD	modified percent root-mean-square difference [NyMK00]
MRA	multiresolution analyses [Mall98]
MSD	multiresolution signal decomposition
MSE	mean squared error
MSVQ	mean-shape vector quantization [CBLG99]
NED	normalized Euclidean distance
NIFS	nonlinear iterated function systems
NN	neural network
NPRD	normalized percent root-mean-square difference [HuKi99]
NRF	noise reduction factor
OWPT	optimal wavelet packet transform
PCD	principal component denoising
PDF	probability density function
PNN	probabilistic neural network [Meis72], [Spec88]

PRD	percent root-mean-square difference [JHSC90]
PSAFM	piecewise self-affine fractal model [MaHa92]
RA	right arm
SA	sinoatrial
SAFM	self-affine fractal model [MaHa92]
SAPA	fan and scan-along polygonal approximation algorithm
sps	samples per second
SF	sampling frequency
SNR	signal-to-noise ratio
SR	sinus rhythm
TP	turning point data reduction algorithm [Muel78]
vel	volume element
VF	ventricular fibrillation
VFD	variance fractal dimension
VFDT	variance fractal dimension trajectory
VQ	vector quantization
VT	ventricular tachycardia
WPT	wavelet packet transform
WV	windowed-variance

LIST OF SYMBOLS

$C(\tau)$	autocorrelation function
$c_j(i)$	approximation of the signal at scale j in wavelet transform
$C_q(r)$	extended pair-correlation function
$CWT_x(a, \tau)$	continuous wavelet transform of x
D_B	box counting dimension
D_C	correlation dimension
D_E	topological (or Euclidean) dimension
D_F	dimension
d_h	Hausdorff distance
D_{HM}	Hausdorff dimension
D_I	information dimension
$d_j(i)$	detail of the signal at scale j in wavelet transform
D_{Ly}	Lyapunov dimension
$\boldsymbol{dp}$	dipole
D_q	Rényi dimension spectrum
D_S	self-similar dimension
D_σ	variance dimension
$DWT_x(i, n)$	discrete wavelet transform of x
G and **H**	quadrature mirror filters
$g(n)$	highpass filter in wavelet transform
$h(n)$	lowpass filter in wavelet transform
H	Hausdorff space
H^*	Hurst exponent
H_q	Rényi entropy
Hz	Hertz
$\eta(n)$	noise
$\boldsymbol{J}$	current density

λ_j	the Lyapunov exponent
$N(0, 1)$	Gaussian distribution with zero mean and variance of 1
σ^2	variance
P	probability density
p	probability
$\mathbf{R}$	set of all real numbers
R_{cr}	compression ratio [Kins98]
SNR_G	SNR gain
T	sampling period
τ	time lag
$\phi(t)$	scaling function
φ_1 to φ_7	moment-invariant
$\psi(t)$	mother wavelet
$\Psi(\omega)$	Fourier transform of $\psi(t)$.
$\psi_{a,\tau}(t)$	wavelet
Ω	conductivity of the tissue
V	scalar potential or voltage
$(\mathbf{X}, d)$	metric space
$x(n)$	generic notion for a time series (signal)
$x_c(n)$	contaminated (measured) signal
$x_{cp}(n)$	compressed signal
$x_d(n)$	noise reduced signal
$x_e(n)$	estimated signal
$x_o(n)$	original signal
$x_r(n)$	reconstructed signal
$x_u(n)$	uncontaminated (pure) signal
$x(t)$	continuous time signal
$\mathbf{Z}$	set of all integers

CHAPTER I

INTRODUCTION

1.1 Background and Motivation

Classical signal processing has been used in the biomedical field such as electrocardiography (ECG), electroencephalography (EEG), electromyography (EMG), electrooculography (EOG), and uterine electrohysterography (EHG), for analysis, monitoring, and diagnostic purposes. Well-known examples of such techniques include the use of the Fourier transform and autoregressive models for analyses of various biomedical signals. However, these classical methods are not always applicable to the nonstationary ECG signal. For example, prediction of ventricular tachycardia (VT) is still difficult to achieve. There has been no satisfactory method for suppression of interfering signals.

Linearity is defined through the superposition principle which states that a function f has the property

$$f(ax + by) = af(x) + bf(y) \tag{1.1}$$

where x and y are variables, and a and b are arbitrary constants. If the superposition principle is not satisfied, the system is nonlinear.

A signal is a physical phenomenon which conveys information and depends on time and space.

A random signal is stationary if its statistical characteristics such as the mean and

the variance are invariant under time or space shifts; *i.e.*, if they remain the same at t and $t + \Delta$, where t is a real and Δ is arbitrary displacement. For such a stationary signal, the probability densities, together with the moment and correlation functions, do not depend on the absolute position of the point on the time/space axis, but only on their relative configuration [Berg99]. Nonstationarity can be associated with regimes of different drifts in the mean value of a given signal, or with changes in its variance which may be gradual or abrupt.

Many biomedical signals are nonstationary as they originate from very complicated nonlinear systems. The analysis of such signals is difficult. In recent years, new theories such as time-frequency wavelet analysis [Daub92], chaos theory [PeJS92], fractal analysis [Kins95], and neural networks [Mast95], have been introduced in nonlinear systems. Since these new techniques do not require the strong hypotheses such as linearity and stationarity, they now find applications in various areas of biomedical engineering. For example, wavelet analysis allows tracking the onset of transient phenomena, while chaos theory can help explain the behaviour of certain physiological systems and model them further, and neural networks also help model nonlinear systems. There is now strong incentive to apply these innovative techniques to the monitoring of cardiovascular signals. Indeed, if parameters extracted are really relevant to the biological mechanism under scrutiny, reliable diagnoses can be posed, and medical intervention can be made efficiently.

The ECG signal processed in this thesis is a nonstationary signal from a nonlinear system. Its discrete form is also called the ECG time series. Figure 1.1 illustrates an ideal ECG signal from Lead I. Such an ECG signal shows the variation of bioelectric potential

with respect to time during a single beat of a human heart. The practical ECG signal consists of a succession of beats which reflect the pulsation of the heart. Each normal beat of the ECG signal should contain the P wave, the QRS complex, and the T wave. Any change in these components is usually related to an abnormality of the heart. An experienced cardiologist can use minute features of such a signal to obtain important information about the heart function of a patient, including arrhythmias, myocardial infarctions, atrial enlargements, ventricular hypertrophies, and bundle branch blocks [Yano00]. In addition, the measurement of the ECG signal is non-invasive and simple. Therefore, the ECG has been used widely in medical care for monitoring, diagnosis, and treatment of patients. However, such a diagnosis is not the focus of this thesis.

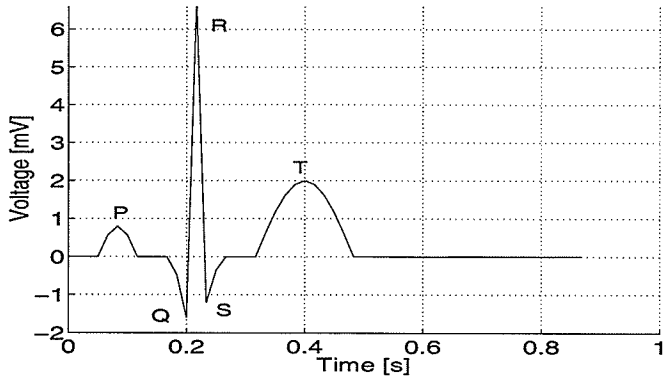

Fig. 1.1. An ideal ECG frame with typical features.

Instead, this thesis concentrates on another problem: it is difficult for a short-time ECG recorder to capture the exact moment when heart disorder occurs since the heart is in a normal state most of the time. Consequently, a Holter monitoring system has been used to record the ECG signal over a period of 24 hours or longer. In order to get good clinical interpretation, data are acquired at high sampling rate. Feature extraction, classification,

and data compression are necessary for the nonstationary ECG signal since the recordings must often be stored, transmitted, and analyzed. The results of this research could be used in the diagnoses.

1.2 Thesis Statement and Objectives

The purpose of this thesis is to research and develop nonlinear analysis methods to characterize the nonstationary ECG signal, with particular interest in the efficient representation of the signal. The ECG data compression must be performed in such a way that important clinical information is maintained. An ECG compression technique is required to reduce (i) the storage requirements of hospital database and ambulatory ECGs, (ii) the time to transmit data over telephone lines, when remote interpretation is required, and (iii) the time of browsing the ECG recordings. The research primarily addresses the aspects as how to:

1) condition the signal for feature extraction;
2) extract statistical features from the ECG signal;
3) classify the ECG beats according to the features;
4) classify the ECG beats by time registering two frames;
5) compress the ECG data involved in the long-term recordings of ECGs;
6) browse the ECG recording quickly;
7) determine the strange attractor of the ECG, if there is one;
8) denoise the ECG signal;
9) partition the ECG signal; and
10) reduce the computational complexity in fractal compression.

This thesis uses modern signal processing techniques such as multifractals, chaos theory, wavelet transform, and neural networks, to characterize, classify, and compress the ECG signal. The important quasiperiodic nature of the ECG signal is employed in the processing. The ECG signal is compressed by removing redundancy existing both among beats and between points.

The ECG beat-based compression techniques developed in this thesis are based on operational characteristics of a Holter monitor to process regular ECG recordings contaminated by noise. Since the heart is in its normal state most of the time, these techniques can achieve a high compression ratio. The instantaneous abnormal state of the heart will be monitored and detected, and lossless compression techniques will be applied to them. Thus, in this thesis, we do not select an ECG database with different clinical conditions and abnormal ECG beats.

Since an ECG signal from any actual recording is a mixture of the pure ECG signal and different types of noise due to tissue propagation, skin, EMG, and EEG, direct evaluation of denoising techniques applied to such a signal is not possible because neither the pure ECG signal nor the noise is known. Consequently, we use a pure ECG signal (obtained from simulation of a mathematical model of the heart) mixed with pure noise (generated by fractional models). The only remaining noise in the mixture is the numerical noise. However, we can model it as a white noise whose mean is zero and variance is σ^2. Although the simulated ECG signal may change with parameters of the body, it is considered to be appropriate if its power spectrum is broadband with chaotic characteristics, and the spectrum distribution is similar to that of the real ECG signal.

1.3 Organization of This Thesis

This thesis consists of nine chapters. Chapter 1 states the problems in ECG signal processing and outlines motivation and objectives of the thesis.

Chapter 2 provides background information about the ECG signal and its compression. Mathematical models of the heart are evaluated for ECG simulation. The compression efficiency and fidelity measure in the ECG signal are involved. A review of current ECG data compression techniques is also given in this chapter.

Chapter 3 provides background on wavelet transform. This chapter describes the wavelet transform from the continuous form to the discrete form and the packet decomposition. It also deals with the application of the wavelet transform to signal processing.

Multifractal characteristics of the ECG signal are investigated in Chapter 4. First the Lorenz system is used as a chaos example to illustrate the complexity of a chaotic system. Then the strange attractor of a practical ECG time series is reconstructed according to the Takens theorem. To characterize the strange attractor, fractal analysis is included in this chapter. Single-fractal dimensions are presented, together with multifractals which have the capability of characterizing complex objects through a series of complexity measures such as the Rényi dimension spectrum.

Chapter 5 applies denoising techniques to improve the convergence of the Rényi dimension spectrum of the ECG signal since Chapter 4 shows an incomplete convergence. One such a technique is threshold denoising in the wavelet domain. Principal component analysis in the phase space is another denoising technique. Both of them are discussed and

applied to the ECG signal in this chapter. The performance of the denoising techniques is evaluated by SNR gain and the improvement of the convergence degree.

Chapter 6 presents the fractal compression technique for time series signals. It begins from basic fractal encoding and decoding based on iterated function systems (IFS), collage theorem, and fractal block coding. The piecewise self-affine fractal model developed by Mazel and Hayes is presented to encode the time series. Then, a nonlinear IFS is proposed to compress the ECG signal. Furthermore, signal partitioning technique based on compositional complexity measure is proposed to reduce computational complexity in fractal compression.

Chapter 7 develops an ECG beat classification technique based on the feature extraction of moment-invariants and neural network classification. The moment-invariant contains statistical shape information of the ECG waveform. A probabilistic neural network is used to classify the ECG beat for data compression and browsability purposes. This chapter also describes how to cluster the training set without supervision.

Chapter 8 applies a modified dynamic time warping technique to classify ECG beats based on time registering two frames through a complicated warping function. To make use of classification information to compress the ECG signal, a windowed-variance technique is used to partition the ECG frame into segments which can be encoded according to blocks.

Chapter 9 gives conclusions and contributions for this thesis and recommendations for future research.

CHAPTER II

ECG CHARACTERISTICS AND ITS COMPRESSION

Before discussing electrocardiogram (ECG) signal processing, it is necessary to know where the signal comes from, how the human heart generates the ECG, what the meaning of the ECG signal is, and how to acquire it. All of the background knowledge shows different aspect of the characteristics of the ECG signal and is the foundation to do further research.

This chapter also reviews research in ECG compression and the performance evaluation of ECG compression algorithms.

2.1 Electrical Activity of Heart

A human heart is an electrically active organ. The ECG comes from the electrical activity of the human heart. Chemical reactions cause an electrical chain of events within cardiac muscle. Electrical activity (depolarization) precedes mechanical activity (contraction). Electrical activation describes the events that result in the contraction and relaxation of cardiac muscle, thus pumping blood to the whole body.

The heart may be thought of as an electrical conduction system. Figure 2.1 illustrates such a system, which consists of the sinoatrial (SA) node, the atrioventricular (AV) junction, the bundle branch system, and the Purkinje fibers. The electrical activity begins at the SA node, the principal pacemaker of the heart. Following discharge of the SA node, the electrical impulse is carried by specialized conduction fibers to the left atrium, and

through the right atrium to the AV node. The impulse is momentarily delayed at the AV node. It then enters the ventricular conduction system, which is made up of the bundle of His and the right and left bundle branches. From the bundle branches, the electrical activity is carried to the ventricular muscle by the network of specialized Purkinje fibers [Rawl91].

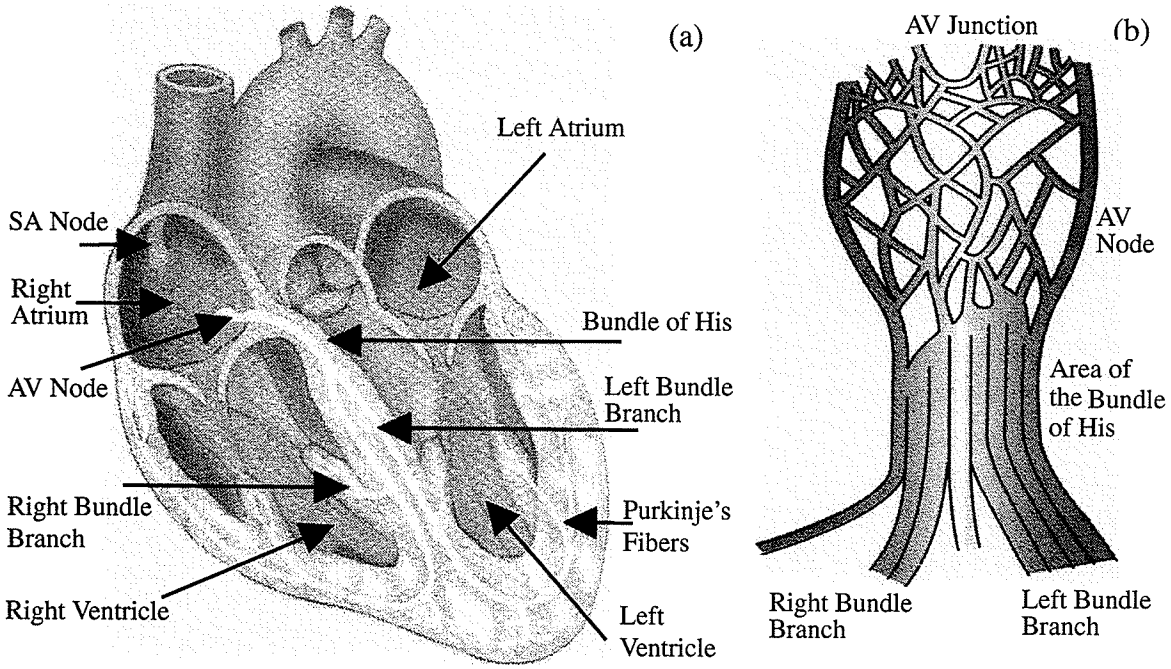

Fig. 2.1. (a) The structure of the heart's electrical conduction system. (b) Schematic representation of the AV junction, demonstrating the entrance fibers into the AV node, orientation of the AV node to the bundle of His, and the entrance fibers into the interventricular septum (from [LeHa00]).

The electrical activity of the heart results from the chemical reaction in the heart cells. The human heart has hundreds of thousands of cells, each acting like a kind of bag in which some chemicals exist in water. Each cell is a complex system with characteristics and performances that contribute to cardiac function. The foremost function is the coordi-

nated contraction, or the beating, of the heart [Rawl91].

Chemical reactions inside and outside the cell provide mobile ions, and a small number of them move through the membrane. The ease of passage, called permeability, varies for different ions. An imbalance of ions across the membrane of a cell causes voltage, which changes with the movement of ions. The term action potential describes the electrolyte exchanges that occur across the cell membrane of the heart during depolarization and repolarization. Depolarization corresponds to the electrical activation of myocardial cells due to the spread of an electrical impulse. It is the process where the inside of the cells becomes less negative. Repolarization means the process by which the cells return to the resting level. Repolarization is fast at first, then followed a longer slower surge until the resting state is achieved.

The action potential change and electrolyte movement in the cardiac period may be divided into five phases, which is shown in Fig. 2.2: (i) Phase 0 is initial phase of the cardiac cycle, which consists of rapid depolarization. As a change in cellular permeability occurs, sodium rushes into the cells making them more positive. This action produces the characteristic upstroke in the action potential. (ii) During Phase 1, initial repolarization, a rapid influx of chloride inactivates the inward pumping of sodium. This serves to make the internal charge nearly equal to the external charge. (iii) In Phase 2, the plateau, a slow inward flow of calcium occurs, while the flow of potassium is slowed considerably. The electrical charge remains nearly equal at this stage and contraction of the cardiac muscles begins. (iv) Phase 3 consists of final rapid depolarization. During this stage there is a sudden acceleration of the repolarization rate as the slow calcium current is inactivated and

the outward flow of potassium is accelerated. Cells begin to regain their negative electrical charge at this stage. (v) Phase 4 includes diastolic depolarization. There is a difference in activities of nonpacemaker (working) cells and pacemaker cells. Nonpacemaker cells remain in the steady state until their membranes are acted on by another stimulus [LeHa00].

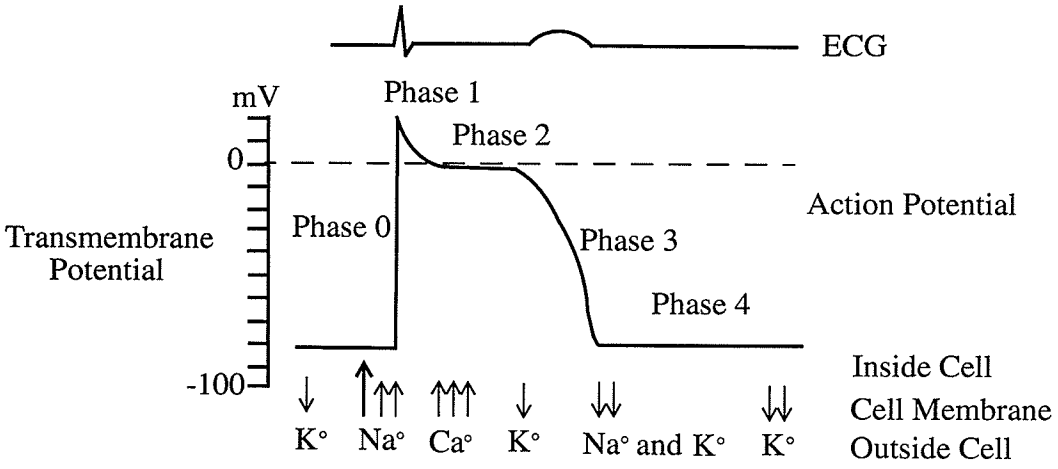

Fig. 2.2. The relationship between action potential of myocardial working cells of the heart and the ECG (after [LeHa00]).

2.2 Meaning and Use of ECG

Section 2.1 explained the electrical activity of the heart and the corresponding ECG. We must now clarify what the ECG means exactly in this thesis because it may have different meanings. The ECG is an abbreviation of electrocardiogram or electrocardiography. It may be a medical device capable of recording the electrical activity of the heart. Or the ECG means the signal or graph of the recording from the heart. This thesis defines the ECG as an abbreviation for "electrocardiogram" and "the signal recorded from the heart". An example of such a signal is shown in Fig. 2.3. We will explain the important

features contained in the ECG in the next section.

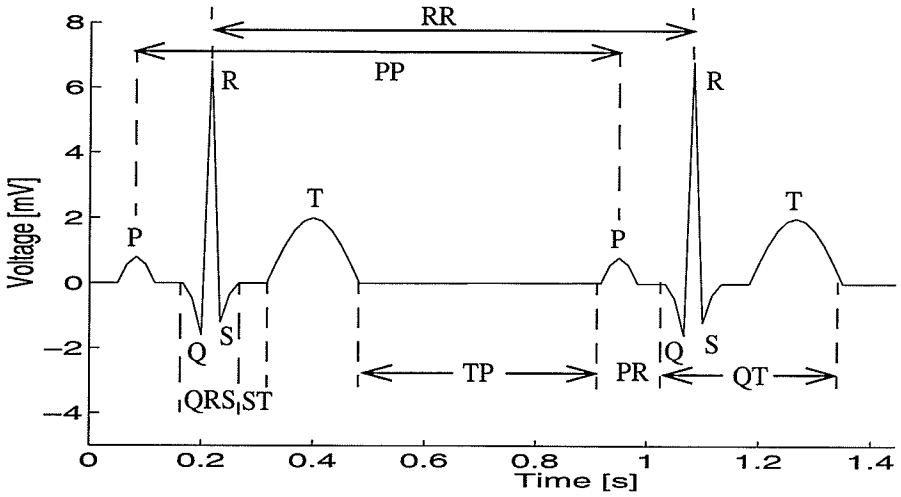

Fig. 2.3. An ideal ECG signal and its features.

The ECG can be recorded from the wave passage of the depolarization and repolarization processes in the heart. The potential in the heart tissues is conducted to the body surface where it is measured using electrodes. By strategically positioning the electrodes on the body surface as shown in Fig. 2.4, the direction and strength of the potential can be observed.

The ECG signal is detected by electrodes, which may include unipolar or bipolar leads. The unipolar leads record the difference between an active electrode and zero potential. The zero potential is generated approximately by placing an electrode on each arm and the left leg and connecting them via a common terminal to form a triangle with the heart at its centre [Marr72] (see Fig. 2.4). Depolarization moving towards an active electrode (V_1 to V_6) produces a positive deflection in the ECG recording while depolarization moving away produces a negative deflection [Gano85]. Bipolar leads record the

difference in potential between two active electrodes which are generally located on the limbs.

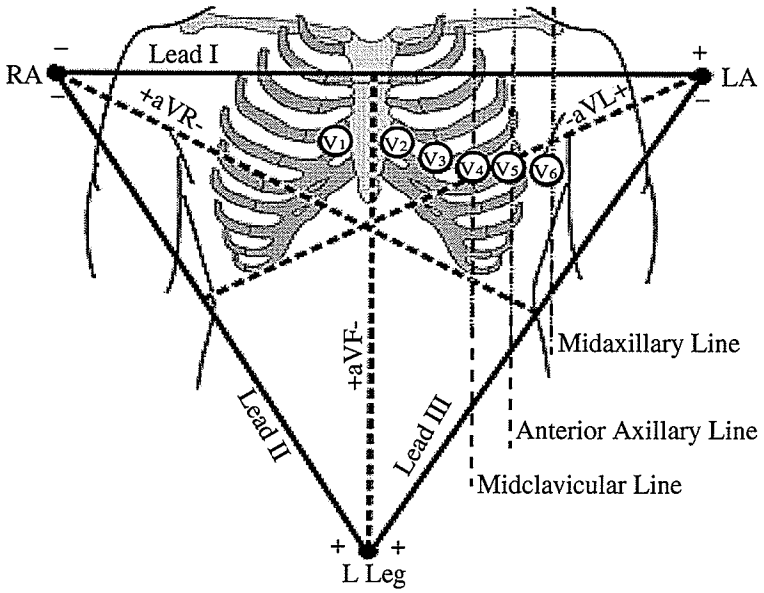

Fig. 2.4. Electrode positions on the body for the ECG detection (from [Yano00]).

The voltage signal detected by the electrodes can be amplified, displayed, digitized, recorded, and analyzed in various ways, depending on the applications. The signal may come from either single-channel or multi-channel. The latter situation occurs when multiple electrodes are used at various parts of the body simultaneously. Each channel represents a different "view" of the heart rhythm. Modern ECGs usually utilize 3 to 12 channels to measure the electrical activity of the heart. We take the frequently used ECG from V_5 as the signal in our research.

Figure 2.5 shows the relationship between the ECG and the heart activity. A morphological change in the ECG waveform is a visible sign of a heart muscle illness. An experienced cardiologist can observe small features of ECG to extract important informa-

tion about the heart function of a patient, including arrhythmias, myocardial infarctions, atrial enlargements, ventricular hypertrophies, and bundle branch blocks. The measurement of the ECG signal is non-invasive and simple. The ECG has been widely used in medical care, for monitoring, diagnosis, and treatment of patients possibly suffering from heart diseases.

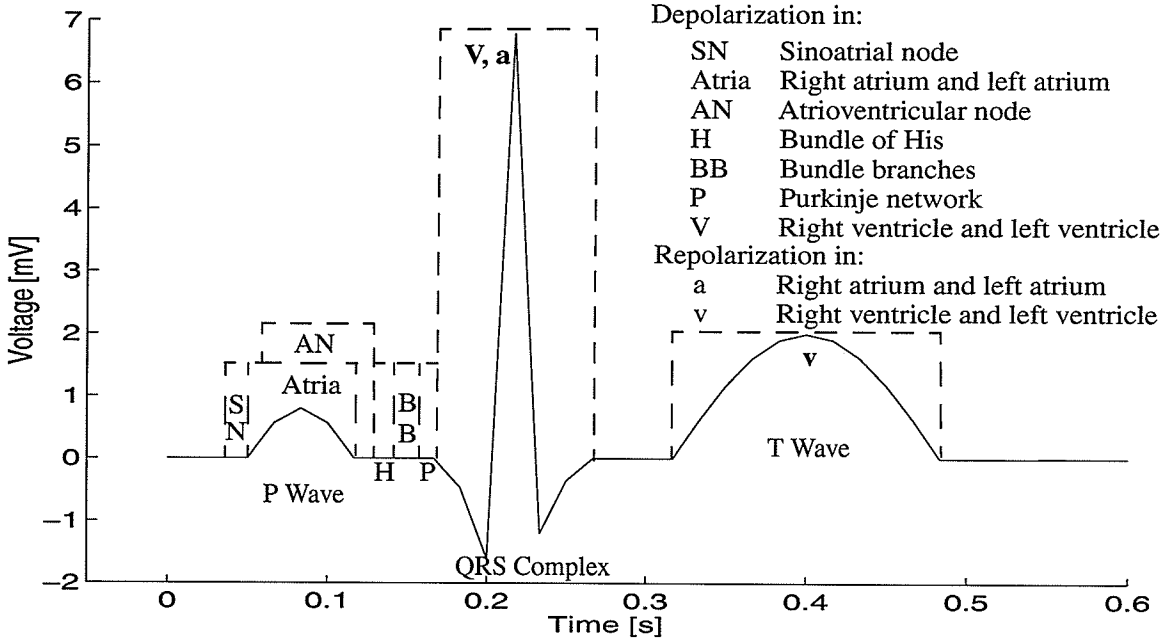

Fig. 2.5. The relationship between the ECG and the heart activity (from [Rawl91]).

2.3 Characteristics of ECG

From the relationship between the ECG and the heart activity shown in Fig. 2.5, it is clear that the peaked area in the ECG beat, commonly called the QRS complex, together with its neighboring P wave and T wave (shown in Fig. 2.3), is the portion of the signal thought to contain most of the diagnostically important information. Other important information includes the elevation of the ST segment and heartbeat rate, the RR or PP. Therefore, it is of the utmost importance that these parts of the signal can be rendered with

good precision after any attempt at compression as discussed in this thesis.

The ECG signal repeats beat by beat, but the heartbeat rate of a recorded ECG changes with time, as shown in Fig. 2.6. The mean and variance of the beat vary with time. Therefore, the ECG signal is considered to be quasiperiodic and nonstationary.

A typical feature of such a nonstationary signal is the presence of "patchy" patterns which change over time. These patterns cannot be thought of as noise simply, though they are noise-like or even have some noise characteristics. The ECG is a very complicated signal from a nonlinear system.

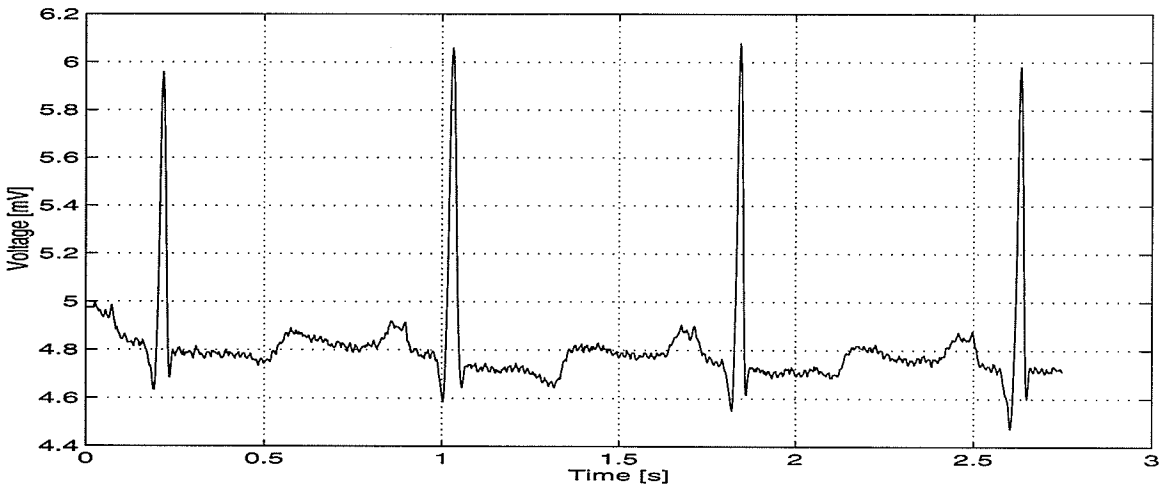

Fig. 2.6. A recorded ECG waveform.

Recall that in Fig. 2.2 (which reveals the relationship between the ECG and the electrical activity of the heart) the action potential of the heart jumps at Phase 0, then varies slowly until settling down to a resting state. It means that the action potential signal has many frequency components. It is clear that the ECG signal is not the same as the potential of the heart. It only reflects the change of the action potential in the heart. Compared to the original potential, the ECG loses some frequency components, especially high frequen-

cies. The high frequency loss is due to the heart potential passing through an equivalent lowpass filter [PlBa88]. Since the ECG acquired on the surface of the body has fewer frequency components than the potential of the heart, such a loss may affect the chaotic characteristics of the ECG.

Rawlings [Rawl91] argued that the bandwidth from 0.05 to 5000 Hz is proper to get a clean ECG signal considering the noise and artifact influences. Jalaleddine *et al.* [JHSC90] estimated that the minimal sampling rate should be 457 Hz with 10 bit sampling precision to avoid spectral foldover. Most of the ECG components with diagnostic significance are included in the bandwidth of 0.05 to 100 Hz [Rawl91]. The following figures show the broadband spectrum distribution of a recorded ECG signal. Figure 2.7(a) is a semi-log plot of the power density vs. linear frequency. Three peaks occur at about 60, 120, and 180 Hz. To observe the distribution clearly, a log-log plot of the spectrum is given in Fig. 2.7(b). Its spectrum distributions are piecewise-linearly close to white noise, f^{-5}, black noise, and pink noise, respectively.

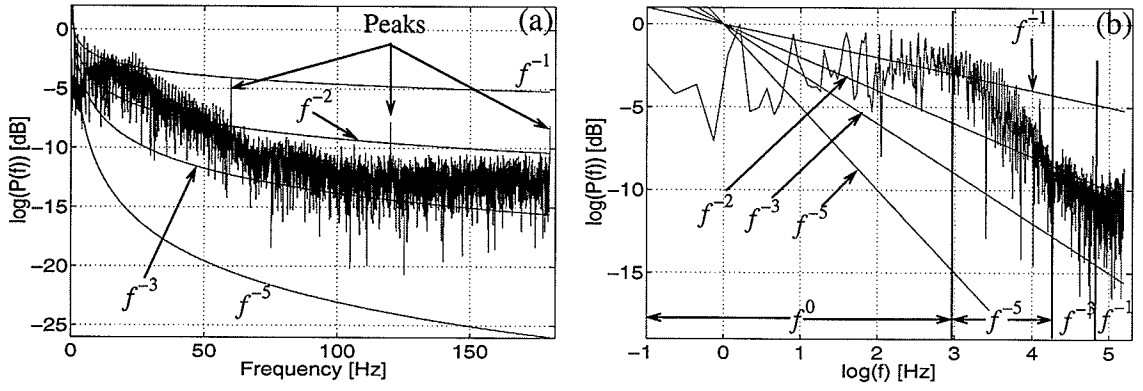

Fig. 2.7. The spectrum distribution of an ECG signal. (a) Semi-log plot with 8000 sample points. (b) Log-log plot with 2000 sample points.

There is no standard about what sampling frequency and sampling resolution are suitable to acquire the ECG signal. In a typical ECG monitoring device, the sampling rate varies from 125 to 500 Hz, and each data sample may have 8, 12, or 16 bit resolution [NyMK00].

2.4 Modelling of ECG

As described in Sec. 1.2, the pure ECG signal is required for denoising algorithm evaluation in this thesis. The pure signal, a modelled ECG, is related to the so-called "forward" problem which entails the calculation of the body-surface potentials, starting usually from either equivalent current dipoles that represent the heart's electrical activity or from known potentials on the heart's outer surface. The forward problem is one of two topics in the ECG calculation. The other topic is the inverse problem. The calculation of the ECG is a branch in cardiology [Gulr98].

As discussed already, electrocardiographic potentials on the torso surface arise from electrical generators within the heart, and are conducted to the torso surface through the intervening medium, thus reflecting the properties of both the cardiac sources and the surrounding thoracic tissue. Important in the ECG is the consideration of the effects of the thoracic inhomogeneities, anisotropies, and geometrical shape on potential distributions within and on the torso surface [PiPl84]. The ECG literature contains many numerical techniques for potential field calculations, and they can be divided into two broad categories: surface methods and volume methods. Surface methods for potential calculations are used for piecewise-homogeneous, isotropic ECG models. For such models, potentials throughout the entire volume can be determined by the solution of surface integral equa-

tions representing either sources directly or potentials on the heart surface, the torso surface, and intervening conductivity interfaces. Surface methods can be used to examine the anisotropic effects of the skeletal muscle layer by applying the boundary extension approximation suggested by Rush and Nelson [RuNe76] [CMNB83]. However, this transform can only be utilized for regions of constant anisotropy and is only accurate for flat or slowly changing surfaces [StPI86].

The second numerical volume method can be used to solve for potential distributions in an ECG model with any type of inhomogeneity or anisotropy. Volume methods allow direct representation of conductivity in any number of piecewise-homogeneous or anisotropic regions without a resulting increase in complexity of solution. Potentials throughout the volume are determined through the solution of one or more volume equations by either the finite difference, or finite element, or finite volume methods [Gulr98].

The torso may be treated as a volume conductor. Such a volume conductor has three important features: quasi-static, linear, and resistive [Gese63]. Quasi-static means that currents and potentials throughout the body are instantaneously related to sources in the heart. Linearity means that the potential arising from several sources is equal to the sum of the potentials contributed by each of the sources acting alone. The resistance of the tissue means that the capacitive component of the electric impedance of body tissues is negligible to a good approximation. Experimental verification of the resistive and linear nature of the volume conductor is obtained by comparing the voltage pulse response observed at the skin with the current pulse delivered to the heart by an artificial pacemaker [Bril66].

The above conditions are summarized in mathematical form as follows. Let $\boldsymbol{J}$ be the vectorial current density and V be the electric scalar potential (voltage) at a point in the volume conductor. According to Ohm's law, we have

$$\boldsymbol{J} = -\Omega \nabla V \tag{2.1}$$

where Ω is the conductivity of the tissue at the point and ∇V is the gradient of the potential V. In the absence of an external source of current, the divergence $\nabla \bullet$ of the current $\boldsymbol{J}$ must be

$$\nabla \bullet \boldsymbol{J} = 0 \tag{2.2}$$

Bioelectric sources are associated with the movement of ions across the cell membrane, and involve the conversion of chemical energy to electric energy. The sources can be represented as an "impressed" current source density $\boldsymbol{J}^i$ [Gese67]. Hence (2.1) becomes

$$\boldsymbol{J} = -\Omega \nabla V + \boldsymbol{J}^i \tag{2.3}$$

where $\boldsymbol{J}^i$ vanishes outside the heart. This concept has been widely adopted. From (2.2) and (2.3), we have

$$\nabla \bullet \Omega \nabla V = \nabla \bullet \boldsymbol{J}^i \tag{2.4}$$

which in an unbounded medium of conductivity Ω has a solution

$$V = -\frac{1}{4\pi\Omega}\int \frac{\nabla \bullet \boldsymbol{J}^i}{r} dv \tag{2.5}$$

where r is the distance from the source to the observation point.

With the use of the divergence theorem, the scale source $\nabla \bullet \boldsymbol{J}^i$ can be treated alternatively as a vector source $\boldsymbol{J}^i$. Consider an arbitrary volume containing the sources

$$\int \nabla \bullet (\boldsymbol{J}^i \varphi) dv = \int \boldsymbol{J}^i \varphi \bullet dS = \int (\varphi \nabla \bullet \boldsymbol{J}^i + \boldsymbol{J}^i \bullet \nabla \varphi) dv \tag{2.6}$$

where φ is an arbitrary scalar function, and S is the closed surface surrounding the region of integration. If this surface lies outside the source region, the second integral in (2.6) vanishes, and

$$\int \nabla \varphi \bullet \boldsymbol{J}^i dv = -\int (\nabla \bullet \boldsymbol{J}^i) \varphi dv \tag{2.7}$$

If φ is taken to be $1/r$, then from (2.5)

$$V = \frac{1}{4\pi\Omega} \int \boldsymbol{J}^i \bullet \nabla \left(\frac{1}{r}\right) dv + \gamma \tag{2.8}$$

where γ is a constant.

This expression is precisely of the form of the potential of a current dipole. Hence the impressed current density is equivalent to a source current dipole moment per unit volume. Henceforth, we will assume that V is measured with respect to a defined reference potential and omit the constant.

The torso as a volume conductor is linear, inhomogeneous, resistive, anisotropic, and bounded. It is reasonable to divide it into discrete regions (such as heart, lung, bone,

blood, and muscle), with an appropriate conductivity assigned to each region. The heart contains the sources of electricity, or electromotive forces $\boldsymbol{J}^i$. Before a mathematical framework for the solution to the problem is developed, we can write down the form of the potential. Strictly as a consequence of linearity, the potential V at an arbitrary point on the surface of the body (or anywhere in the volume conductor) is related to the distribution of current sources $\boldsymbol{J}^i$ throughout the heart as follows

$$V = \int \nabla Z \bullet \boldsymbol{J}^i dv \tag{2.9}$$

This equation is a statement of superposition. The term ∇Z is a transfer impedance which relates the source $\boldsymbol{J}^i$ in the element of heart volume dv to the potential V at the observation point. Z depends on the location of the source, the location of the observation point, the reference point, and the shape and conductivity of the torso and its internal structures. Volume conductor properties may vary with time; for example, as a consequence of respiration or the beating of the heart.

The transfer impedance ∇Z relates a vector source to a scalar potential, and is itself a vector function of position throughout the source region. The fact that it is the gradient of a scalar follows from considerations of reciprocity. From (2.8), an unbounded homogeneous volume conductor of conductivity Ω is

$$Z = \frac{1}{4\pi\Omega r} \tag{2.10}$$

It is sometimes convenient to divide the heart into a number of regions, and to assign a lumped dipole $\boldsymbol{dp}$ to the centroid of each region. Then for region i

$$\boldsymbol{dp}_i = \int \boldsymbol{J}^i dv_i \tag{2.11}$$

where the integration is over the *i*th region. In this case the cardiac electric sources are represented by a finite number of lumped current dipoles. Such a source is a multiple dipole source [FiBa63]. The surface potential is then given by

$$V = \sum \boldsymbol{dp}_i \bullet \nabla Z_i \tag{2.12}$$

where ∇Z_i is evaluated at the centroid of region *i*, or is an appropriate average over the volume.

To calculate the ECG, five essential steps should be implemented:

1) Creation of 3D anatomical computer models of the heart and body (applying computer tomography, magnetic resonance tomography, and different methods of digital image processing);
2) Simulation of the electrical excitation propagation in the anatomical heart model (using a cellular automaton or more sophisticated tools);
3) Calculation of the impressed source current densities (using the bidomain model);
4) Calculation of the potential distribution by solving Poisson's equation with the impressed source current densities in the anatomical body model (using a finite difference or finite element solver in a volume conductor); and
5) Calculation of the ECG by taking the leads at the body model.

Werner *et. al.* calculated an ECG shown in Fig. 2.8 according to the above five steps [WeSD98]. To model the heart and torso anatomically, they used the Visible Man

dataset provided by the National Library of Medicine, Bethesda, Maryland, USA [Acke91]. The anatomical model of the heart of the Visible Man contains about 30 million voxels of the size of 1/3 mm × 1/3 mm × 1 mm. For performance reasons the model resolution for this simulation is reduced to 350,000 cardiac cells (voxels). The knowledge of the orientation of the muscle fibres related to anisotropy is determined for each voxel by texture analysis. The excitation is simulated by applying a cellular automaton [WKDE89], [WOHH95].

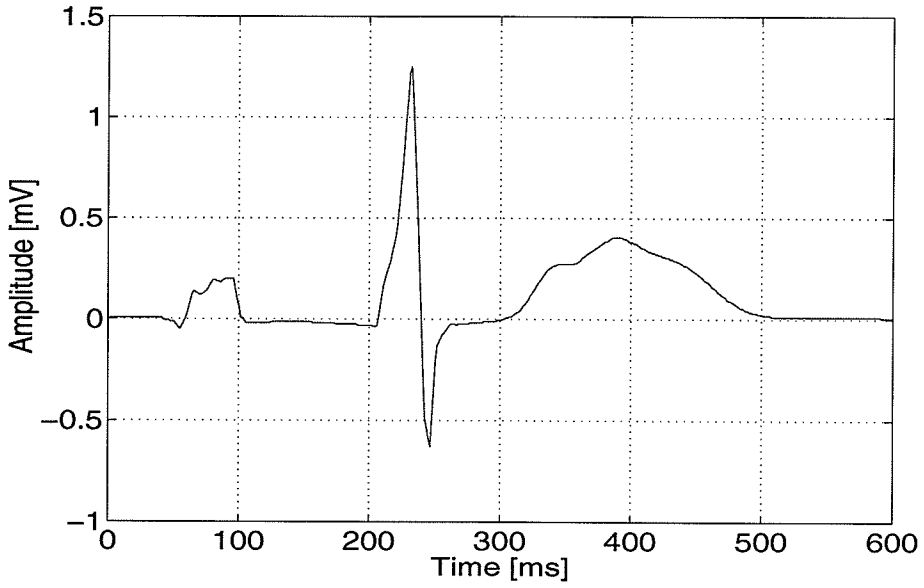

Fig. 2.8. Calculated ECG of the Visible Man. The electric source distribution to calculate the ECG is taken from the transmembrane potentials (after [WeSD98]).

It is seen that the calculated ECG has no small notches as compared to the measured ECG. Therefore, such an ECG can be taken as a **pure** signal from a mathematical point of view in our research. However, the simulated ECG still has some artifacts which are due to the limited bioelectric modelling of the heart and the computing limitation of the computer. More research is necessary to obtain a more realistic ECG. However, since

the procedure of the ECG simulation is very complicated, this thesis uses the ECG signal generated by Werner *et. al* [WeSD98] as the pure ECG signal. As discussed in Sec. 1.2, it is appropriate to use the simulated ECG in denoising algorithm evaluation.

2.5 Evaluation of ECG Data Compression Techniques

It is difficult to capture the on-set of a heart disorder because the heart is in a normal state most of the time. A Holter monitoring system has been used to record the ECG signal over a period of 24 hours or longer. This system is designed to capture the instantaneous abnormal state of the heart. In order to get accurate clinical interpretation, data are acquired at high sampling rate and high sampling resolution. Since a large amount of data is generated by the long-term Holter system, and the recording must often be stored, transmitted (telemedicine), and analyzed, ECG signal compression is necessary.

Signal compression refers to the reduction in the number of bits required to represent a signal compared to the raw uncompressed representation. Usually the signal is represented as a data set that a computer can process conveniently. The process of data compression is to detect and eliminate redundancy in a given data set. The inverse procedure from the compressed data may restore the raw signal precisely or lose some information. Therefore, data compression is divided into two kinds: lossy and lossless. Lossless compression requires that the decompression can restore the original data exactly. Lossy compression allows certain information loss to achieve greater data reduction, while preserving the significant signal features upon reconstruction.

Lossy compression is widely applied to various signals from nature such as image,

speech, and biomedical signals. It usually achieves a higher compression ratio than the lossless method. We may use lossy techniques to compress ECG data because the ECG diagnosis does not need precise ECG waveforms. The performance of lossy compression techniques can be evaluated through compression efficiency, reconstruction error, and computational complexity based on the same ECG data.

2.5.1 ECG Database

To compare performance of ECG compression algorithms, experiments should be performed on the same ECG data. Unfortunately, there is no standard ECG database yet, which leads to a problem to evaluate various ECG compression schemes.

There are some ECG databases such as MIT-BIH Arrhythmia Database, MIT-BIH Supraventricular Arrhythmia Database, MIT-BIH ST Change Database, European ST-T Database, QT Database, Long-Term ST Database, and Creighton University Ventricular Tachyarrhythmia Database [Phys01]. Among them, the MIT-BIH Arrhythmia Database is used frequently by researchers to evaluate their algorithms [Mood99].

The MIT-BIH Arrhythmia Database is used in the thesis. This database gives the ECG signal sampled at 360 samples per second (sps), with 11 bits per sample (bps). We compress the ECG data contained in the file, x_100.txt, which is a 10-minute recording (758 beats) over a 10 mV range from V_5. Record x_100 is relatively clean.

2.5.2 Compression Efficiency Measure

In ECG data compression, compression efficiency is measured by the compression

ratio, R_{cr}. R_{cr} is defined as the ratio between the number of bits in the original signal and that in the compressed version [Kins98]

$$Rcr = \frac{\sum_{j=1}^{N} b[x_o(j)]}{\sum_{j=1}^{N_1} b[x_{cp}(j)]} \tag{2.13}$$

where x_o represents the original signal with length N and x_{cp} the compressed signal with length N_1. $b[x(j)]$ is defined as the function to find the bit number of $x(j)$.

This compression ratio is suitable for comparing different compression algorithms. However, the R_{cr} does not consider the sampling rate and sampling resolution of the time series for various ECG databases. Since the R_{cr} does not show how many bits are required for the resultant output, a bit rate (bits/s) of output data is also used in some papers.

2.5.3 Fidelity Measure

If a signal is compressed by a lossy technique, the reconstructed signal will be a distorted version of the raw signal. The quality of reconstructed signal is evaluated from two aspects: subjective and objective. Both of them are significant in ECG compression. Subjective evaluation refers to the perceptual review of the signal and its reconstruction, usually by experienced cardiologists.

Objective evaluation is based on the measure of the difference between the original signal and the reconstructed signal. The percent root-mean-square difference (PRD) is one

of the objective evaluation criteria defined as [JHSC90]

$$PRD = \sqrt{\frac{\sum_{j=1}^{N} [x_o(j) - x_r(j)]^2}{\sum_{j=1}^{N} x_o^2(j)}} \times 100\,\% \tag{2.14}$$

where x_o and x_r are the original and reconstructed signals, respectively, and N is the number of samples in the signal.

The ECG signal should have a zero mean after it passes through a band pass filter. When the signal is amplified and sampled, the offset voltage of the amplifier and capacitor coupling of the analog-to-digital converter (A/D) may induce the D.C. to the signal. The distortion measure given by (2.14) varies with the D.C. components (mean) of the signal, which is improper for being used in algorithm evaluation with different ECG databases. To solve this problem, Huang and Kinsner proposed another distortion measure, the normalized percent root-mean-square difference [HuKi99]

$$NPRD = \frac{1}{max(x_o) - min(x_o)} \sqrt{\frac{1}{N} \sum_{j=1}^{N} [x_o(j) - x_r(j)]^2} \times 100\,\% \tag{2.15}$$

Nygaard *et al.* [NyMK00] proposed still another modified measure. In their approach, the mean of the signal, $\bar{x}_o$, is removed from the original signal

$$MPRD = \sqrt{\frac{\sum_{j=1}^{N} [x_o(j) - x_r(j)]^2}{\sum_{j=1}^{N} [x_o(j) - \bar{x}_o]^2}} \times 100\% \qquad (2.16)$$

Hilton performed experiments to investigate the relationship between subjective evaluation and objective measurement based on wavelet compression of the ECG signal [Hilt97]. Two conclusions can be obtained from his results: (i) high sampling frequency benefits subjective evaluation in lossy compression techniques, and (ii) compression ratio of 8:1 may give a 100% correct diagnosis for the MIT-BIH ST Change Database with sampling frequency of 360 sps and 12 bps. Since Chen and Itoh achieved compression ratio of 6.8:1 with PRD of 7.0% by applying wavelet transform to the ECG signal from the MIT-BIH Arrhythmia Database [ChIt98], the PRD of 7.0% should be a reasonable index for ECG data compression. Under such a threshold, a reliable subjective diagnosis based on the signal reconstructed from lossy ECG compression schemes can be made.

2.5.4 Problems with Evaluating ECG Compression

As there is no standard for ECG data acquisition, various time sequences are used in ECG compression algorithms. The performance of these algorithms changes with different conditions and constraints, which poses a problem in evaluating ECG data compression algorithms using above criteria. On the other hand, the reported compression ratio and error measure do not take into account factors such as bandwidth, sampling rate, precision of the original data, wordlength of compression parameters, reconstruction error threshold, database size, lead selection, and noise level [JHSC90]. It is still an open ques-

threshold, database size, lead selection, and noise level [JHSC90]. It is still an open question about what are the optimal criteria on evaluation of the ECG compression algorithms.

2.6 Current ECG Compression Techniques

During the last several decades, a large number of schemes for a bit-efficient representation of the ECG signal have been presented. Roughly, they can be classified into three categories: direct data compression, transform encoding, and other methods [JHSC90] [CBLG99].

The direct data compression includes time domain techniques based on adaptively retaining the ECG samples which represent some key features of the ECG [HBHe94]. This kind of techniques mainly uses piecewise-linear segment compression technique, including prediction, interpolation, and approximation. The compression ratio of the direct data compression is low and is related to the sampling rate, the sampling precision, and the preset error threshold [JHSC90]. Channel noise also affects compression performance negatively. However, the implementation of direct compression algorithms is fast.

More recently, modern data compression methods such as vector quantization, transform coding, fractal technique, and subband coding, have also been applied not only to the ECG signal, but also to other biomedical signals [HBHe94]. They achieve better compression performance, though at a high computational cost.

2.6.1 Direct Data Compression

The direct data compression methods attempt to reduce redundancy in a data

sequence by examining a successive number of neighboring samples. These techniques generally eliminate samples that can be predicted by examining a preceding and succeeding sample. Examples of such techniques include *amplitude zone time epoch coding* (AZTEC) [CNFO68], *Fan* [Gard64], *scan-along polygonal approximation* (SAPA) [ISHS83], *coordinate reduction time encoding system* (CORTES) [AbTo82], *delta coding* algorithm [WoSR72], and *Slope* [Tai91].

The AZTEC was originally proposed by Cox *et al.* to preprocess the real-time ECG signal for rhythm analysis [CNFO68]. It fits raw ECG sample points with plateau regions and slopes under a preset error. The AZTEC plateau regions (horizontal lines) are segments with certain length, which are produced by utilizing the zero-order interpolation. The production of a slope starts when the number of samples needed to form a plateau of three samples or more can be formed. The stored values for the slope are the duration and the amplitude of last sample. Signal reconstruction is achieved by expanding the AZTEC plateau regions and slopes into a discrete sequence of data points. Although the AZTEC achieves a high compression ratio of about 10:1, the fidelity is not acceptable to the cardiologist because of the discontinuity that occurs in the reconstructed signal. Jalaleddine *et al.* employed the Fan technique (discussed later) to alleviate the discontinuity problem in AZTEC. Compared to AZTEC, their modified algorithm achieves 50% improvement in signal fidelity [JHCS88].

The *turning point* (TP) data reduction algorithm was developed by Mueller to reduce the sampling rate of the ECG signal [Muel78]. The TP processes three sample points at a time, a reference point and two consecutive points. The decision on which of

the two consecutive points is retained depends on which point preserves the slope of the original three points better. The TP algorithm produces a compression ratio of 2:1. The major problem with the TP algorithm is that the compressed data set cannot represent equal time intervals.

Abenstein and Tompkins suggested the CORTES algorithm, which is a hybrid of the AZTEC and TP techniques [AbTo82]. The CORTES applies the TP algorithm to the QRS complex and the AZTEC to the isoelectric regions of the ECG signal. If the segment obtained by the AZTEC is shorter than an empirical length, it is replaced by the TP data. There is no slope in the CORTES signal. The CORTES signal is reconstructed by expanding the AZTEC plateaus and interpolating points between each pair of the TP data. Parabolic smoothing is applied to the AZTEC portions of the reconstructed CORTES signal to reduce distortion. The CORTES maintains the high compression ratio of AZTEC, while reducing distortion sharply.

Gardenhire first suggested and tested the Fan algorithm on the ECG signal [Gard64] [Gard65]. The algorithm starts by accepting the first data point as a permanent point. Two slopes are drawn between the permanent point and the next sample plus a preset error threshold ($\pm\varepsilon$) to form a fan area. Another fan area of slopes is formed between the permanent point and the third sample point. If there is an intersection between the two fan areas, it is taken as a new fan. If the third point falls within the new fan area, the second point and its fan area are replaced by the third point and the new fan area, respectively. The procedure is repeated for the future samples until a point falls outside the new fan area. The preceding sample is saved as the next permanent sample and ready for a new

process. During signal reconstruction, the permanent samples are connected with straight lines.

Ishijima *et al.* presented three algorithms based on the SAPA technique to represent the ECG signal by a series of segments [ISHS83], [SkGo80]. In the three SAPA algorithms, the SAPA-2 gives the best results. The SAPA-2 is the same as the Fan technique, but with one difference: in addition to the two slopes obtained in the Fan algorithm, the SAPA-2 computes a third slope between the permanent point and the third sample. Unlike the Fan which uses a point criterion, the SAPA-2 uses the third slope as the criterion: if the third slope falls between the two slope values of the new fan, the second point is redundant.

One of the *differential pulse code modulation* (DPCM) techniques, delta coding was proposed by Wolf *et al.* [WoSR72], which encodes the first-difference of the ECG signal. Ruttimann *et al.* studied the performance of the DPCM with linear prediction as a function of the order of the predictor [RuBP76]. They concluded that a DPCM system with linear predictors of order higher than two would not result in a substantial performance improvement in ECG data compression. A compression ratio of 7.8:1 is achieved through the DPCM [RuPi79].

Imai *et al.* presented the application of a *peak-picking* compression technique to ECG signals [ImKY85]. This algorithm extracts the significant points which correspond to maxima, minima, and large curvature of the signal by using the second-order difference. The ECG signal is reconstructed by spline function interpolation of these significant points [ImKY85], [LEBS86]. Straight line fitting technique was also used to restore the

signal by Giakoumakis and Papakonstantinou [GiPa86]. Under the same compression ratio, experiments show that the PRD error of the spline method is about half of that of the AZTEC.

Jalaleddine *et al.* suggested a *cycle-to-cycle* (CC) compression technique for the ECG signal with normal beat [JHCS88]. This algorithm assumes that there is a QRS template. The normal QRS is replaced by the difference between itself and the template. Then the Fan algorithm is applied to the resulting signal for compression. The CC compression technique requires QRS detection. It has not shown improvement over the Fan algorithm.

Nygaard *et al.* developed a *cardinality constrained shortest path* (CCSP) algorithm for ECG compression [NyMK00]. This method is based on a rigorous mathematical model of the entire sample selection process. By modelling the signal samples as nodes in a graph, optimization theory is applied in order to achieve the best compression possible under the given constraints. The goal in [HaHH97] is to minimize the reconstruction error, given a bound on the number of samples to be extracted. The ECG signal is compressed by extracting the samples that, after interpolation, will best represent the original signal, given an upper bound on their number.

In addition, lossless entropy encoding may also be applied to any of the above techniques. Entropy coding employs a nonuniform probability distribution of data to encode the data with variable-length codewords, and may be applied to compress the data further at the final stage of various compression techniques. For example, Ruttimann *et al.* employed the variable-length Huffman coding to compress the ECG signal [RuPi79], [PaBW79], [StBD79], [CoRi73], [WhWo80]. A disadvantage of Huffman encoding is the

possibility of a serious decoding error that may occur due to a transmission error. Such a problem can be tackled by the employment of data block coding with known error control techniques [BuSu72], accompanied by an encoding overhead induced. The encoding overhead is introduced to recognize the transmission error and correct the transmitted signal.

2.6.2 Transform Coding Compression

Transform coding methods apply a linear transform to the signal and then compress via redundancy reduction in the transform domain. Typically, the transform produces a sequence of coefficients that exhibit compaction, thus leading to a reduction of the amount of data needed to represent the original signal adequately. Many transforms such as *Karhunen-Loève* (KLT), Fourier, cosine, Walsh, Legendre, the optimally warping, subband coding, and wavelet, are employed in ECG data compression.

The KLT finds the optimal basis function (eigenvectors) from the covariance matrix of the time series and retains only a certain number of the large eigenvalues and corresponding eigenvectors. Hambley *et al.* used part of the KLT coefficients to express the ECG signal [HaMF74], [AhMH75], [WHML77], [ZiWo79], [WoZi80]. Olmos *et al.* suggested a windowed KLT to improve the performance of ECG data compression [OMGL96]. Although the KLT is an optimal orthogonal transform and achieves a compression ratio of 12:1, it is very computationally intensive [WHML77].

The *fast Fourier transform* (FFT) and *discrete cosine transform* (DCT) are orthogonal transforms. Unlike the KLT, they have a fixed basis function. They map signals into the frequency domain. The spectra of the ECG signal are finite, and most of the spectrum

energy locates at the low frequency end. Therefore, the ECG signal may be represented by the most significant coefficients in the frequency domain and compressed. Reddy *et al.* used the FFT to compress ECG data [ReMu86]. Ahmed *et al.* proposed a simpler transform, the DCT, to represent the ECG signal [AhMH75] [AlBe92]. Both algorithms compress the ECG by preserving the most significant part of the transform coefficients. The DCT achieves a compression ration of 7.4:1 and can be implemented in real time [ReMu86].

Like the FFT, the *Walsh transform* can concentrate the energy of signals on some range. Based on this nature, Kuklinski compressed the ECG signal in the Walsh domain by retaining a fraction of the Walsh spectrum and allocating low bits to Walsh coefficients [Kukl83]. The Walsh transform has a computationally simpler basis function than the FFT.

Philips and Jonghe transformed the ECG signal by the *Legendre polynomial* (LPT) basis function, and used part of the transform coefficients to approximate the signal [PhJo92]. They showed that this approach achieves higher compression ratio and lower PRD error compared to the DCT. Philips further improved the performance of the LPT by time-warping the polynomial basis to the signal being compressed [Phil93]. The location detection of RR intervals in the ECG is an important issue in the LPT.

The *subband coding* splits a signal into different frequency bands by filters and then employs some form of quantization and bit allocation strategy [JaNo84]. It has been applied to ECG data compression [AyÇK91], [Tai92], [Haug95]. Tai used a six-octave-band coder on the ECG waveform to compress the ECG data [Tai92]. Different bands are allocated with different bits to trade off the compression ratio and fidelity of the signal. A

compression ratio of 6.5:1 was achieved for the ECG signal by the subband coding [Haug95].

The *wavelet transform* is a time-scale representation of signals in the wavelet domain. A signal may be represented compactly by wavelet coefficients with an optimal mother wavelet. The *discrete wavelet transform* (DWT) has been applied to compress ECG data [ChIH93], [AnDR95], [Brad96], [Hilt97], [ChIt98], [MiYe00]. Chen *et al.* proposed an ECG data decorrelation method based on the DWT and a compression ratio of 6.8:1 is achieved [ChIt98]. Hilton compared the compression performance by applying various wavelets to the ECG signal [Hilt97]. He also used the *wavelet packet transform* (WPT) to compress ECG data and found that this technique did not achieve improvement compared to the DWT. Bradie applied the WPT to the beat of the ECG signal and achieved higher compression ratio than the KLT [Brad96].

2.6.3 Other Methods

There are many other techniques in ECG data compression that cannot be classified under the direct method or transform method. They are categorized simply into other methods. This other methods category includes: *iterated function systems* (IFS), *vector quantization* (VQ), *cycle-pool-based compression* (CPBC) algorithm, linear prediction, and neural networks.

Bansley *et al.* developed the IFS to compress signals by finding the attractor of the system [Barn88], [Jacq90]. Øien and Nårstad [Fish98] applied the IFS to ECG signal compression by modelling range blocks through domain blocks, quantizing the model, and

retrieving the attractor from the quantized coding. A compression ratio of 6.0:1 is achieved [Fish98]. The experimental result of the IFS outperforms that of the subband coding algorithm.

Several VQ schemes were applied to ECG signal compression [MaRa90], [AnDR95], [WaYu97], [CBLG99]. Anant *et al.* used the VQ on ECG wavelet coefficients and a compression ratio of 7.3:1 is achieved, which outperforms the ECG wavelet compression [AnDR95]. Miaou and Yen proposed a *gold washing adaptive vector quantization* (GWAVQ) technique with automatic distortion threshold adjustment to encode the wavelet coefficients of the ECG signal. Cárdenas-Barrera and Lorenzo-Ginori suggested a *direct waveform mean-shape vector quantization* (MSVQ) as an alternative for ECG signal compression [CBLG99]. In this method, an ECG time series is cut into short segments. Each short segment has it mean subtracted off. Then the mean is quantized as a scalar and the short segment is encoded through a vector quantizer.

Cassen and English suggested a *variable bit allocation* (VBA) scheme on ECG compression because different parts in a beat of the ECG signal have different contributions to the clinical diagnosis [CaEn97]. This three-stage technique includes: (i) first order differencing of the ECG time series; (ii) nonlinear quantization (more bits are allocated to the QRS complex of the ECG signal); and (iii) entropy coding.

Hamilton and Tompkins presented a technique, *average beat subtraction and first differencing of residual data*, to compress the ambulatory ECG [HaTo91]. In this technique, the ECG signal is segmented beat-by-beat and subtracted by a time-aligned average beat. The residual ECG is first-order differenced, quantized, and Huffman encoded.

The CPBC algorithm for the ECG signal was proposed by Barlas and Skordalakis [BaSk96]. The basic idea is that a pool of past-seen cycles is maintained and cycles to be encoded can be stored as transformed versions of those residing in the pool. A cycle transform technique is introduced in order to render the pattern matching process and enable cycle substitution. Pattern matching is implemented based on critical points of ECG cycles.

Cohen and Zigel established a *long-term prediction model* for the ECG signal based on beat segment with a beat codebook. Instead of discarding the prediction error, the residual is scalar encoded with some bits [CoZi98]. A bit rate of 166 bps per channel is given by this technique. Unlike Cohen's method, Ramakrishnan and Saha normalized the ECG beat in period and amplitude and then modelled the wavelet coefficients of the ECG signal by long-term prediction [RaSa97]. It seems that Cohen's approach achieves a better result than Ramakrishnan's.

Iwata *et al.* proposed a *dual three-layered neural network* (NN) structure to process the ECG signal [IwNS90]. The three-layered NN includes an input layer of 70 units, a hidden layer of 2 units, and an output layer of 70 units. In the structure, one NN is used for learning and another for compressing the ECG data so that the system can always evolve with the ECG time series. The backpropagation technique is used as the learning algorithm [IwNS90].

To utilize inter-beat and intra-beat correlation of the ECG signal, Lee and Buckley cut and beat-aligned the ECG time series to form a 2D data array [LeBa99]. Then the DCT is applied to the array to compress ECG data. The results show that this 2D method is

much better than the 1D DCT and the DPCM techniques.

2.6.4 Results of Current ECG Compression Techniques

As ECG data compression research has been carried out on a bewildering multitude of databases, sampling frequencies, sampling resolution, and reconstruction error measure, it is quite hard to make qualified judgements about the preference of one technique over the other. However, Table 2.1 is an attempt to sum up some important results from the previous work.

From this table, one can see that ECG data used in the compression techniques have sampling frequency from 200 to 500 Hz, and sampling resolution from 8 to 12 bits. The reconstruction error also changes from 3.5% to 28%. Since the compression ratio usually changes with the reconstruction error and is influenced by sampling frequency and sampling resolution for most compression techniques, Table 2.1 cannot give a precise performance comparison for all ECG data compression schemes. However, we can say that the subband coding technique [Haug95] achieves a better performance than the fractal-based compression [Fish98], and the VQ scheme on wavelet coefficients [AnDR95] improves compression performance for wavelet transform [ChIt98]. These several techniques use the ECG data with the same sampling frequency as our data and have close reconstruction error to our technique.

Table 2.1: Results of various ECG data compression schemes. SF means sampling frequency.

Technique	Rcr	PRD [%]	SF [Hz]	Precision [bits]
Fan/SAPA [Gard64]	3.0	4.0	250	--
AZTEC [CNFO68]	10.0	28.0	500	12
Dual appl. of KLT [WHML77]	12.0	--	250	12
DPCM-linear pred., interp. and entropy coding [RuPi79]	7.8	3.5	500	8
TP [Muel78]	2.0	5.3	200	12
CORTES [AbTo82]	4.8	7.0	200	12
Peak-picking with entropy encoding [ImKY85]	10.0	14.0	500	8
Fourier descriptors [ReMu86]	7.4	7.0	250	12
Classified VQ [MaRa90]	8.6	24.5	200	12
BP and NN/PCA neural networks [NaIw93]	20.0	13.0	360	11
Long-term prediction [NaCo93]	28.2	10.0	250	8
Subband coding [Haug95]	6.5	5.1	360	12
VQ of wavelet coefficients [AnDR95]	7.3	6.3	360	11
Fractal-based compression [Fish98]	6.0	5.8	360	12
Wavelet transform [ChIt98]	6.8	7.0	360	11

2.7 Summary

Chapter 2 introduces background information about the electrical conduction system of the heart, and characteristics and computation methods of the ECG. A paper review of ECG data compression is given.

The electrical conduction system is the source of the ECG. The ECG is a potential transferred from the action potential of the heart to the surface of the body through a series of cascaded equivalent filters. This thesis defines that the ECG stands for electrocardiogram and the signal recorded from the heart. The ECG detection is non-invasive and simple. It is a quasiperiodic and nonstationary signal from a nonlinear system. This signal has important characteristics such as the QRS complex, the P wave, the T wave, and heartbeat rate, which correspond to the activity of the heart. The ECG has been widely applied to heart disorder diagnoses.

The ECG signal can be not only measured, but also modelled, though its simulation is complicated. The ECG is simulated by modelling the heart as a multiple dipole source.

The compression of the ECG signal is necessary because of a large amount of data generated by the long-term Holter system. The compression efficiency and fidelity measure in ECG data compression are involved. It is still an open question of what constitutes the optimal criteria to evaluate ECG compression algorithms. A review of previous ECG data compression techniques is also included in this chapter.

The next chapter addresses wavelets, wavelet transform, and multiresolution decomposition, as well as their application to signal analysis.

CHAPTER III

WAVELETS AND WAVELET TRANSFORM

This chapter provides a summary of the theoretical foundation of wavelets and wavelet transform, as well as their application to signal processing.

The Fourier transform is not applicable to describe the nonstationarity of biomedical signals. Wavelets and wavelet transform have the ability to represent not only stationary signals, but also nonstationary signals. Signal processing can be performed through the corresponding wavelet coefficients since the original signal or function can be represented in terms of wavelet expansions (coefficients in a linear combination of the wavelet functions). If one further chooses optimal wavelets adapted to the data, or truncates the coefficients below a threshold, the data can be represented very compactly. This compact coding makes wavelets an excellent tool in the field of data compression. The truncated coefficients may also be used directly as features for classification. The wavelet coefficients contain the smoothness information of the signal. Irregular noise leads to small amplitude coefficients. Therefore, the threshold method can be adopted to denoise the signal. Wavelet analysis has found wide applications in signal processing including image, speech, biomedicine, earthquake prediction, and radar.

This chapter begins with wavelets and *continuous wavelet transform* (CWT). The time-frequency location is one of the most important properties of the wavelet transform. Then we discuss the discrete wavelet transform (DWT) and *multiresolution signal decom-*

position (MSD) technique for the fast implementation of the transform. The optimal signal decomposition can be done through wavelet packet transform (WPT).

3.1 Continuous Wavelet Transform

Wavelets are functions that satisfy certain mathematical requirements. They can be used to represent stationary or nonstationary signals or functions. The wavelet analysis procedure is to adopt a wavelet prototype function, called an *analyzing wavelet* or *mother wavelet* denoted as $\psi(t)$, to find the correlation between the signal and the dilated and shifted $\psi(t)$. A set of basis functions used in wavelet transform is the scaled and translated version of the $\psi(t)$ as [Daub92] [Rive91]

$$\psi_{a,\tau}(x) = \frac{1}{\sqrt{a}}\psi\left(\frac{x-\tau}{a}\right) \qquad \tau \in \mathbf{R} \tag{3.1}$$

where τ is a shift position and a is a positive scaling factor. $a > 1$ corresponds to a dilation, while $0 < a < 1$ to a contraction of $\psi(t)$, and $\mathbf{R}$ denotes the set of real numbers. It should be noted that although the wavelets are not only functions of time, but also of space.

Equation 3.1 indicates that the wavelets with different scaling factor a keep the same shapes and have changeable sizes. Such a dilation or contraction property is used to represent a nonstationary function through wavelet transform [Meye93]. Taking $\{\psi_{a,\tau}(t) \in L^2(\mathbf{R})\}$ as the basis functions, the CWT of a real valued function $x(t)$ is simply an inner product of the function and a wavelet [Rive91]

$$CWT_x(a,\tau) = \langle x, \psi_{a,\tau} \rangle = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t)\psi\left(\frac{t-\tau}{a}\right)dt \tag{3.2}$$

Now let us discuss the transform function with variables of the shift τ and scale a and its significance in the above equation. For a given shift τ, the CWT is the result of the local analysis of the signal $x(t)$ at the given position τ with a prototype analyzing function whose width depends on the scale factor a. This function achieves maxima at a position τ where the scaled prototype best matches the original function. Therefore, the CWT allows to find specific patterns in nonstationary signals by selecting appropriate prototypes.

For a given scale a, the CWT is the output of the signal $x(t)$ through a filter whose impulse response $\psi_a(t) = \psi(-t/a)/\sqrt{a}$ is the scaled version of the (real valued) $\psi(t)$ with the scale factor a. Taking into account that $\psi(t)$ is bandpass around a given radian frequency ω_0, it is easy to show that the transform can be seen as the signal $x(t)$ after filtering in the neighbourhood of the radian frequency ω_0/a with a filter whose bandwidth is $\Delta\omega_0/a$. It follows that the CWT is equivalent to a filter bank with constant-Q factor ($\Delta\omega_0/\omega_0 = Q$).

The CWT is reversible. The inverse continuous wavelet transform (ICWT) can be defined as [Rive91]

$$x(t) = \frac{1}{\gamma}\int_0^{\infty}\frac{1}{a^2}\int_{-\infty}^{\infty} CWT_x(a,\tau)\psi_{a,\tau}(t)d\tau da \tag{3.3}$$

if the constant γ satisfies the following *admissibility condition* [Mall98]

$$\gamma = \int_0^{\infty} \frac{|\Psi(\omega)|^2}{|\omega|} d\omega < \infty \tag{3.4}$$

which guarantees perfect reconstruction of the signal. Here, $\Psi(\omega)$ denotes the Fourier transform of $\psi(t)$.

The admissibility condition requires that the mother wavelet $\psi(t)$ must satisfy the following conditions: (i) The square of the Fourier transform, $\Psi(\omega)$, must decay faster than $|\omega|$ at $\pm\infty$, which means the mother wavelet $\psi(t)$ is band-limited in the frequency domain; (ii) $\psi(t)$ must be a zero mean function; and (iii) $\psi(t)$ is a function with finite energy.

3.2 Discrete Wavelet Transform

In the previous section, we established that the CWT is equivalent to a filter bank with constant-Q factor, which can be employed to discretize the parameters a and τ in $\psi_{a,\tau}(t)$ according to Shannon's sampling theorem.

3.2.1 Discrete Scale Factor

For filters having a constant-Q factor and whose impulse response have the same shape, the semi-logarithmic representation of frequency yields transfer functions with the same shape, the same width, but different amplitudes and translated position in $\log\omega$. It

means that the samples in $\log\omega_i$ or equivalently in $\log a_i$, must be equally spaced in order to get the same discretization steps for each filter, that is the quantity

$$\log a_i - \log a_{i-1} = \gamma_1, \quad \forall i \in \mathbf{Z} \tag{3.5}$$

where γ_1 is a constant, and $\mathbf{Z}$ denotes the set of integer numbers. From (3.5), the following equation holds with a_0 a constant

$$a_i = a_0 {\gamma_1}^i, \quad \forall i \in \mathbf{Z} \tag{3.6}$$

Equation (3.6) means that the scale factor is a function of the scale level *i*.

3.2.2 Discrete Shift Factor

If $T = 2\pi/\omega_T$ is the correct sampling period to sample $\psi(t)$ according to the Nyquist rule, $T_a = aT$ is the correct sampling period to sample $\psi_a(t) = \psi(t/a)/\sqrt{a}$, and also to sample $CWT(a, \tau)$ for a given scale factor *a*. We can write the discrete shifts τ_n for the corresponding discrete scale a_i as

$$\tau_n = n(a_i T) \tag{3.7}$$

Selecting $a_0 = 1$, rewriting (3.2) in its discrete form with (3.6) and (3.7), the DWT is

$$DWT_x(a_i, \tau_n) = DWT_x(i, n) = {\gamma_1}^{-i/2} \int_{-\infty}^{\infty} x(t)\psi({\gamma_1}^{-i}t - nT)dt, \ \forall (i, n) \in \mathbf{Z}^2 \tag{3.8}$$

When the discretization for (3.8) is done on a dyadic grid; *i.e.*, when $\gamma_1 = 2$, the DWT is interpreted as a constant-Q filtering with a bank of octave-band filters, followed by sampling at the respective Nyquist frequencies (corresponding to the bandwidth of the particular octave band). We can also assume that $T = 1$, which is the same as scaling t by the factor $1/T$. Then (3.8) becomes

$$DWT_x(a_i, \tau_n) = DWT_x(i, n) = 2^{-i/2} \int_{-\infty}^{\infty} x(t)\psi(2^{-i}t - n)dt, \quad \forall (i, n) \in \mathbf{Z}^2 \quad (3.9)$$

The discrete wavelets with a dyadic grid are

$$\psi_{i,n}(t) = 2^{-i/2}\psi(2^{-i}t - n), \qquad \forall (i, n) \in \mathbf{Z}^2 \quad (3.10)$$

From now on, the discussion of the DWT and multiresolution approximation is based on (3.10).

3.3 Multiresolution Analysis

The partition of the frequency axis into octave bands allows the introduction of the multiresolution concept. Mallat described an algorithm to implement the DWT through *multiresolution analysis* using filter banks [Mall89].

The general procedure behind Mallat's DWT algorithm [Mall89] is to decompose the discrete signal into an approximation signal Λ_i and a detail signal Φ_i, where i represents scale level in the multiresolution analysis. The approximation signal Λ_i, or lowpass signal, is formed by the projection of the original signal onto the space formed by the basis

$\{\phi_{i,n} = 2^{-i/2}\phi(2^{-i}t - n), (i,n) \in \mathbf{Z}^2\}$ where $\phi(t)$ is a scaling function. The detail signal Φ_i, or highpass signal, is similarly formed with the basis $\{\psi_{i,n} = 2^{-i/2}\psi(2^{-i}t - n), (i,n) \in \mathbf{Z}^2\}$ where $\psi(t)$ is a mother wavelet following all of the previously discussed properties required by the DWT. Both of these bases are effectively scaled versions of $\phi(t)$ and $\psi(t)$ that have been downsampled by a 2^i-point decimation to a lower resolution.

An important property of $\phi(t)$ is that it must be selected so that approximations are subspaces of higher resolution approximations. These subspaces are associated with scale-independent signal representations known as multiresolution analysis (MRA). The most succinct definition of the MRA has been provided by Mallat [Mall98]: it is a sequence $\{\Lambda_i\}_{i=-\infty}^{\infty}$ of closed subspaces of $L^2(\mathbf{R})$ which satisfies the following requirements [Mall98]:

$$\forall (i,n) \in \boldsymbol{Z}^2,\ x(t) \in \Lambda_i \Leftrightarrow x(t - 2^i n) \in \Lambda_i, \tag{3.11}$$

$$\forall i \in \boldsymbol{Z},\ \Lambda_{i+1} \subset \Lambda_i, \tag{3.12}$$

$$\forall i \in \boldsymbol{Z},\ x(t) \in \Lambda_i \Leftrightarrow x(t/2) \in \Lambda_{i+1}, \tag{3.13}$$

$$\lim_{i \to +\infty} \Lambda_i = \bigcap_{i=-\infty}^{\infty} \Lambda_i = \{0\}, \tag{3.14}$$

$$\lim_{i \to -\infty} \Lambda_i = \text{Closure}\left(\bigcup_{i=-\infty}^{\infty} \Lambda_i\right) = L^2(\mathbf{R}), \text{ and} \tag{3.15}$$

There exists $\phi(t)$ such that $\{\phi(t-n), n \in \mathbf{Z}\}$ constitute an orthonormal basis of Λ_0.

Roughly speaking, Λ_i represents the space of all functions in $L^2(\mathbf{R})$ whose details of resolution less than 2^{-i} have been removed. We can thus construct a sequence of sets Φ_i which contains the difference between functions projected on Λ_i and Λ_{i-1}. This difference is referred to as a detail signal given by

$$\langle(\phi_{i-1,n} - \phi_{i,n}), x\rangle \in \Phi_i, \quad \forall(i,n) \in \mathbf{Z}^2 \tag{3.16}$$

where Φ_i is the orthogonal complement of Λ_i in Λ_{i-1} so that

$$\Lambda_{i-1} = \Lambda_i \oplus \Phi_i \quad \text{and} \quad \Lambda_i \cap \Phi_i = \{0\}, \quad \forall i \in \mathbf{Z} \tag{3.17}$$

where $\oplus$ denotes a direct sum of function spaces.

For $j < J$, Eq. (3.17) can be extended as

$$\Lambda_j = \Phi_{j+1} \oplus \Lambda_{i+1} = \Phi_{j+1} \oplus \Phi_{j+2} \oplus \Lambda_{i+2}$$

$$= \Phi_{j+1} \oplus \Phi_{j+2} \oplus \cdots \oplus \Phi_J \oplus \Lambda_J \tag{3.18}$$

To implement (3.18), Mallat suggested filter banks $\mathbf{G}$ and $\mathbf{H}$ for $\phi(t)$ and $\psi(t)$, respectively, to filter the signal at each scale. To satisfy (3.17), the following impulse response relationship between the two filter banks holds

$$g(n) = (-1)^{1-n} h(1-n), \ (n = 1-M, 2-M, \cdots, 0) \tag{3.19}$$

where M is the vanishing moment of the mother wavelet. Equation (3.19) shows that **G** is the mirror of **H** and thus the pair are known as *quadrature mirror filters* [Mall89]. Given any one, the mother wavelet $\psi(t)$ or the scaling function $\phi(t)$, we can derive the other.

Mallat proposed a fast orthogonal wavelet transform based on the multiresolution decomposition and filter banks [Mall98]. The decomposition is

$$c_j(i) = \frac{1}{2}\sum_n h(n-2i)c_{j-1}(i) \tag{3.20}$$

$$d_j(i) = \frac{1}{2}\sum_n g(n-2i)c_{j-1}(i) \tag{3.21}$$

where $c_0(i)$ is the original signal, $x(i)$. $c_j(i)$ and $d_j(i)$ represent the approximation and detail of the signal at scale j, respectively.

At the restoration, the algorithm is performed with

$$c_j(i) = 2\sum_n h(i-2n)c_{j+1}(i) + 2\sum_n g(k-2n)d_{j+1}(i) \tag{3.22}$$

Figure 3.1 shows the filter bank scheme of decomposing and reconstructing a signal. The left of Fig. 3.1 shows the implementation procedure of the multiresolution decomposition of the signal by filter banks **H** and **G**. The signal is first decomposed into detail part by **G** and approximation by **H**, then down-sampled by 2, respectively. The decomposition and down-sampling for approximation are repeated again and again until a chosen scale is met or only one sample is left in the resulting approximation.

Reconstruction of the signal is done by reversing the process depicted in the above. The right part of Fig. 3.1 shows the procedure.

The computation complexity of the DWT is $O(N)$.

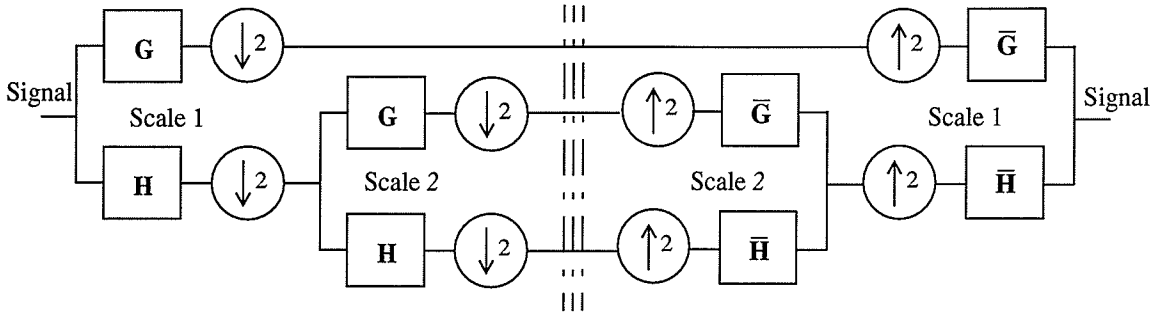

Fig. 3.1. The decomposition and reconstruction of an orthogonal wavelet transform are implemented by cascaded filtering and down/up-sampling operations.

3.4 Wavelet Packet Transform

A generalized decomposition in wavelets is wavelet packet transform (WPT). Wavelet packets were introduced by Coifman *et al.* by generalizing the link between multiresolution approximation and wavelets [CoMW92]. Like the DWT, the WPT decomposes signals according to wavelet basis functions. The difference is that the WPT decomposes not only the approximate pass Λ_i, but also the detail pass Φ_i, in each scale of the decomposition. This procedure is repeated for each approximation and detail signal to form the WPT.

The time-frequency characteristic of the wavelet transform concentrates the signal energy on some locations, which may allow for better approximations of the signal by selecting proper decomposition in wavelet transform. We call this an *optimal wavelet packet transform* (OWPT).

Wavelet decomposition by the DWT, the WPT, and the OWPT is shown in Fig. 3.2. It is apparent that the WPT decomposes signals more completely than the wavelet transform. The DWT is only a specific case of the WPT. Among the three decomposition schemes, the OWPT is the most useful in specific applications theoretically and is generally dependent on characteristics of the signal in the decomposition.

By selecting the basis function and using the OWPT properly, it may only need to decompose certain approximation and detail signals to a certain level. This selective decomposition is illustrated in Fig. 3.2(c) where some approximation and detail decompositions are no longer decomposed further at later levels. The stopping criteria for deciding where to end a further decomposition depend on the signal and the application, but can include factors such as entropy measures, mean squared error measures after quantization, or multifractal measures [CoWi92] [Wick94] [Dans00].

The computational complexity of the WPT is $O(N\log_2 N)$.

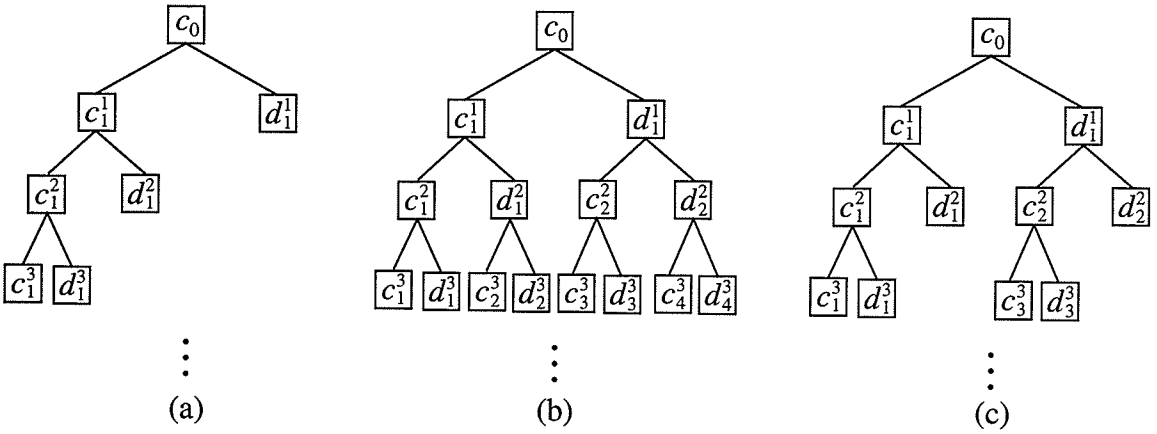

Fig. 3.2. Wavelet decomposition with various schemes: (a) the DWT, (b) the WPT, and (c) the OWPT.

3.5 Wavelet Transform Applications in Signal Processing

The essence of the wavelet transform is to efficiently approximate signals with proper wavelets. The larger the wavelet coefficients, the more similar between the wavelets and the signal. In signal processing, the wavelet transform may be used to: (i) compress data, (ii) reduce or remove noise, and (iii) extract features. The algorithms have low computational complexity and can be implemented in real time.

3.5.1 Data Compression in Wavelet Domain

One of the signal compression techniques is an orthogonal transform. The transform method usually achieves high compression ratio and is insensitive to the noise contained in original ECG signals [JHSC90]. A signal may be approximated efficiently with few non-zero wavelet coefficients through wavelet transform. The wavelet transform decorrelates a signal and concentrates its information into a relatively small number of coefficients with large magnitude. These large coefficients contain more energy than the small coefficients, and thus are more important in reconstructing the signal than the small coefficients.

The wavelet transform can achieve higher performance by selecting a proper wavelet basis for different signals.

Figure 3.3 gives a decomposition result of an ECG signal according to the Mallat's fast wavelet transform with the Daubechies 4 wavelet. Figure 3.3(a) is a raw ECG signal recording with 1024 points. Figures 3.3(b)-(e) show how the original signal is transformed into detailed signals of scale 1 to scale 4. Figure 3.3(f) shows a smoothed signal at scale 4.

After decomposing the ECG into the detailed signals that contain pulses (transitions) at the QRS complex, one observes that the number of wavelet coefficients with high amplitude is quite few in each high frequency pass of the decomposition. When a threshold is set up to make most of the coefficients zero, the data are compressed. Although a high threshold leads to high compression ratio, it also results in high distortion rate. Therefore, a suitable threshold selection is a trade-off between compression ratio and fidelity.

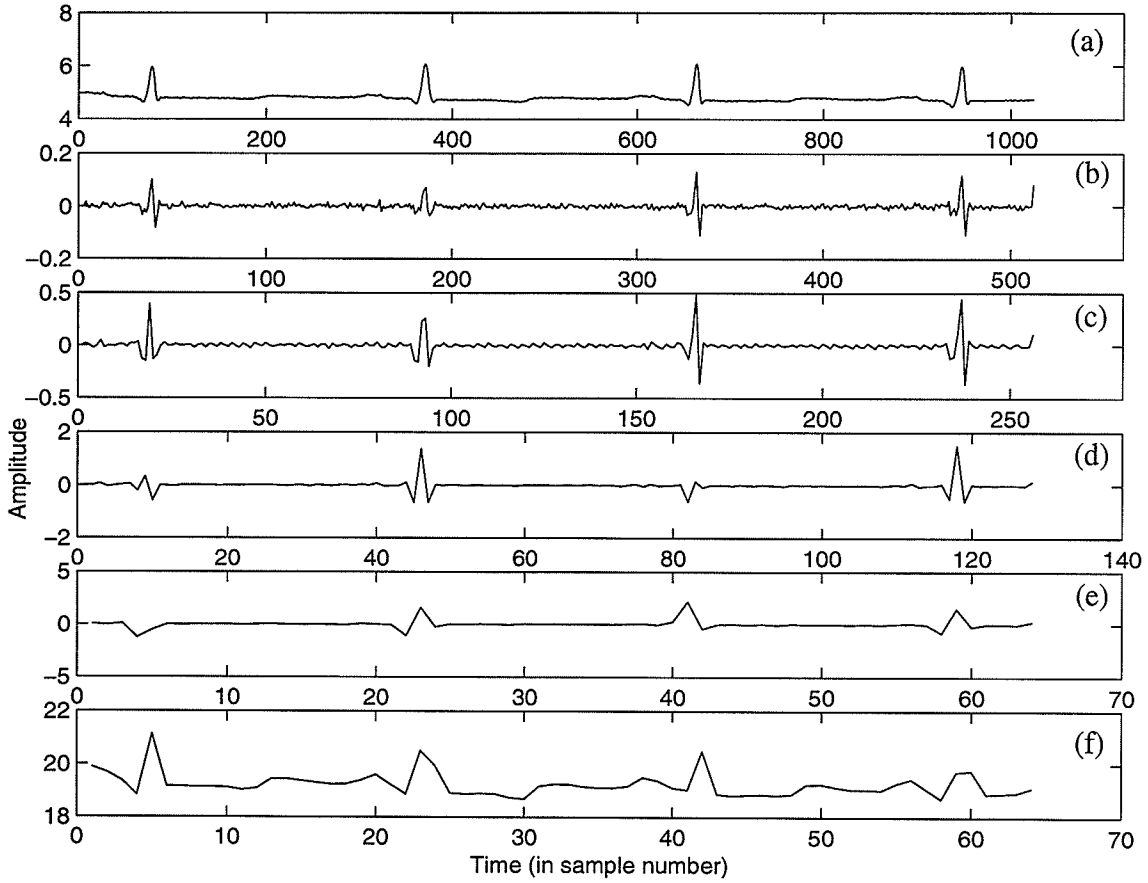

Fig. 3.3. The multiresolution decomposition of an ECG signal with the Daubechies 4: (a) the original ECG signal, (b) the detail at scale 1, (c) the detail at scale 2, (d) the detail at scale 3, (e) the detail at scale 4, and (f) the approximation at scale 4.

Chen and Itoh compressed ECG signals by the Daubechies 10 wavelet transform. They used the MIT-BIT ECG database with 360 sps and 11 bps. They got a compression ratio of about 16.5:1 at a distortion rate (PRD) of 9%.

Hilton [Hilt97] compressed ECG signals with the DWT and the WPT. It is verified that the compression performance of wavelet transform on ECG signals changes with wavelet bases. Although the WPT should achieve higher compression performance than the DWT theoretically, Hilton's experiments gave almost the same results on ECG signal compression for both techniques.

3.5.2 Wavelet Denoising

In Chapter 5, we will use wavelet transform as one of the techniques to denoise the ECG signal. The idea of wavelet denoising is to use the "smoothness" of wavelet basis to measure and extract the "smoothness" of the signal, by which the signal is distinguished from noise.

The smoothness of wavelet basis is determined by a vanishing moment and regularity. A vanishing moment M for a wavelet function $\psi(t)$ is defined as

$$\begin{cases} \int_{\mathbf{R}} t^m \psi(t)dt = 0 \\ \int_{\mathbf{R}} t^{M+1} \psi(t)dt \neq 0 \end{cases}, \quad 0 \leq m < M \tag{3.23}$$

where m and M are nonnegative integer.

One can interpret the degree M as the ability of the wavelet to make a polynomial with order M vanish during a time integration. A high order polynomial is considered a highly smooth signal because it has many derivatives, dependent directly on its order. Therefore, smooth signals can be efficiently represented by wavelets with high degree of vanishing moment because many resulting wavelet coefficients are zero.

The smoothness of wavelets is related to their regularity. The regularity of a wavelet function $\psi(t)$ is defined as the Hölder exponent $0 < \alpha < 1$ [Riou93] with $r > 0$ and a constant γ such that

$$|\psi(t+r) - \psi(t)| < \gamma r^{\alpha}, \quad t \in \mathbf{R} \tag{3.24}$$

The variable r is known as the radius of a disk centred on location t where the exponent is defined. Equation (3.24) controls the growth of the wavelet basis with respect to infinitesimal time change. We often want to characterize the regularity of the nth derivative of the wavelet. The higher order regularity is defined as $n + \alpha$.

Rioul showed that a wavelet with a regularity of more than n possesses n continuous derivatives [Riou93]. Hence, a highly regular wavelet is highly smooth, and suitable for representing smooth signals.

3.5.3 Wavelet Feature Extraction

The time-frequency characteristic makes it easy for the wavelets to decompose high frequency and low frequency components from a signal. The detail version of the signal in the wavelet domain can capture transitions of the signal. This property has been

applied to detect edges of images and to extract features from transients in power systems [Lang96] [ChKH02].

The wavelet coefficients may be used as features directly for signal classification because the wavelet transform decorrelates a signal and concentrates its information into a relatively small number of coefficients with large magnitude.

3.6 Summary

Background on wavelets and wavelet transform is provided in this chapter. The wavelet transform in a continuous form with a set of shifted and dilated basis functions is described first. Then the discrete form is deduced by equally spaced sampling on a log axis of frequency. The filter bank and multiresolution signal decomposition were discussed to perform the wavelet transform. Important properties of the wavelet transform include: (i) time-frequency location, (ii) compact representation of the signal, and (iii) smoothness characteristic. Wavelet analysis has found wide applications in signal processing including: data compression, denoising, feature extraction, detection. Based on different applications, the MSD, WPT, and OWPT can be employed to decompose a signal quickly and effectively. The optimal signal decomposition can be achieved through the OWPT.

The purpose of discussing wavelet transform is to derive the concept of signal denoising based on the smoothness of signals which will be presented in Chapter 5. In the next chapter, the measure of the roughness of signals through fractal analysis will be discussed.

CHAPTER IV

CHAOS AND MULTIFRACTALS IN ECG

4.1 Introduction

Linear methods show limitations in analysis of biomedical signals that originate from very complicated nonlinear living systems. In fact, the majority of natural phenomena can be modelled as nonlinear dynamical systems. The heart is an electrically active organ, and its activity is controlled by a nonlinear dynamical system. That means the change of the heart function corresponds to different dynamical characteristics. Therefore, we can recognize the essence of the heart and extract underlying features by studying the dynamics of the heart.

The behaviour of a nonlinear dynamical system can fall into three classes: stable, unstable, and chaotic [JoSm87]. Stable behaviour means that after some transient periods such a system settles in a periodic or a steady state motion. Unstable behaviour means that the trajectory of the system is aperiodic and unbounded. The chaos that we will study is a particular class about how something changes over time. In fact, change and time are two fundamental subjects that together make up the foundation of chaos. Although the ECG signal is bounded and has some deterministic features such as the QRS complex, the T wave, and the P wave, it is not strictly periodic.

A nonlinear dynamical system can be reconstructed through the 1D time series according to the Takens embedding theorem [Take81]. Therefore, if the heart has dynami-

cal characteristics, they should be contained in the recorded ECG signal and revealed through its reconstructed strange attractor. In this chapter, we will try to find the strange attractor of the heart from the ECG recording. The strange attractor should be a fundamental feature of the ECG signal and may be employed for classification purpose.

The geometry of a strange attractor in the chaotic dynamical system is complicated and difficult to be used in classification directly. In order to model a nonlinear chaotic natural phenomenon, many experiments need to be conducted to determine the nonlinear differential equations which govern the system. Sometimes it is almost impossible to find such equations. Another approach is to characterize the complicated geometrical object by measuring the complexity of its strange attractor.

The prerequisite of characterizing the strange attractor through its complexity measure is first to prove the existence of an underlying strange attractor. It has been found that a complex system with an underlying deterministic model exhibits chaotic dynamics in some cases. Quantitative measures for analyzing such a system have helped gain better insight into the dynamics of the system. Ravelli and Antolini [RaAn92] demonstrated that the degree of complexity measured by the fractal *correlation dimension* increases as the ECG evolves from *sinus rhythm* (SR) to *ventricular fibrillation* (VF) via intermediate rhythms. Since different nonlinear physiological processes of the heart possess different complexities, the measure of the fractal correlation dimension is applied to identify ventricular tachycardia (VT) and VF from other rhythms. Zhang *et al.* used another complexity measure suggested by Lempel and Ziv to analyze the ECG signal [ZZTW99] [LeZi76].

It is not sufficient to characterize a complicated object just using one or two com-

plexity measures. Chen showed that the morphological dimension of two nonintersecting objects is dominated by the object with higher fractal dimension, which means the single complexity measure is incomplete [Chen97]. Instead, the Rényi dimension, a spectrum of multifractal measures, can characterize the complexity of objects completely [Kins94a]. To calculate the multifractal *Rényi dimension spectrum* of the ECG signal, first it must be determined whether the underlying system is deterministic or not, and whether the measured data are random noise or chaotic data. After knowing all the independent variables of trajectories of the system, the strange attractor of the ECG can be characterized by calculating the spectrum of the Rényi dimension. The only problem is that generally there is not enough information about all the variables. The system is often speculated through the trajectory of one measured variable.

In light of the Takens embedding theorem, the strange attractor in a dynamical system can be reconstructed by lagging and embedding the time series in the phase space [Take81]. The time lag τ is estimated by the autocorrelation function of the time series. The minimal embedding dimension is estimated by checking the convergence of the false nearest neighbours and the Rényi dimension spectrum. In this method, we treat the nonlinear dynamical system of the heart as a "black box". Only the time series of the one-channel ECG is used to characterize its dynamics through multifractal measures of its corresponding strange attractor.

4.2 Chaos and Strange Attractor

The majority of natural phenomena can only be modelled as nonlinear systems. The behaviour of the heart should not be an exception. Since nonlinear systems are very

difficult to analyze mathematically, linear systems are usually preferred for modelling purpose. However, only nonlinear systems are capable of exhibiting chaotic behaviours which present more accurate models of the natural phenomena.

A chaotic behaviour of a dissipative system with differential equations in two dimensions or more has a bounded trajectory which converges neither to an equilibrium point nor to a periodic or quasiperiodic orbit and can be represented by an object with complicated structure. Such an object attracts the neighbour points but has some inherent instability along it. This attracting set of points (*i.e.* a strange attractor) is not a simple geometrical object and cannot be characterized well as an integer-dimensional object. Instead, such a strange attractor is a multifractal or fractal. A nonlinear system can exhibit stable, unstable, or chaotic behaviour depending on both the range of the parameters involved in the modelling equations and the value of initial conditions.

A dynamical system is anything that moves, changes, or evolves with time. The dynamical system is chaotic if it satisfies the following three conditions [Deva92]: (i) it has sensitive dependence on initial conditions; (ii) periodic points for the system are topologically dense (like the set of rational or irrational numbers, but not the set of integers); and (iii) the dynamical system is topologically transitive (*i.e.*, given two points, we can find an orbit that comes arbitrarily close to both points).

It is important to note that although the trajectory of a chaotic dynamical system is not periodic, and the Fourier transform of it yields a broadband spectrum, this system is still deterministic and not stochastic. In other words, deterministic chaos includes: (i) it follows a rule (such as some law, equation, or fixed procedure) that determines or specifies

the observable results; (ii) future results are predictable for given constants and input; (iii) although the chaotic system appears to be disorderly, its behaviour has a sense of order and pattern; (iv) chaos can occur even in the complete absence of noise. However, long-term prediction of the behaviour of the chaotic dynamical system is impossible because it is extremely sensitive to the initial conditions. The same initial conditions always generate the same trajectory, but since the numerical tools used for the calculations do not possess infinite resolution, any error in specifying the initial conditions or during the iterative calculation of the trajectory can give a result which may not be accurate for prediction purpose [PeJS92].

One of the methods to study a nonlinear dynamical system is to solve the differential equations of the system and analyze the solutions. The Lorenz system gives an example for such an analysis. The Lorenz system with three ordinary nonlinear differential equations is as [Lore63]

$$\begin{cases} \dfrac{dx}{dt} = \alpha(y - x) \\ \dfrac{dy}{dt} = \beta x - y - xz \\ \dfrac{dz}{dt} = xy - \gamma z \end{cases} \tag{4.1}$$

When $\alpha = 10$, $\beta = 28$, and $\gamma = 8/3$, the Lorenz system shows chaotic characteristics. Figures 4.1(a)-(c) give chaotic solutions of the Lorenz system for the components x, y, and z, respectively. The solutions are random-like curves. They show aperiodicity and bifurcation. The broadband spectrum characteristic of the aperiodic signal is shown in Fig.

4.1(d). Its power spectrum distribution is close to pink noise (f^{-1}) except that in the middle region of the spectrum the distribution is close to that of f^{-7}. Figure 4.2 shows the sensitivity of the Lorenz chaos to initial conditions. The 0.001% change of initial values leads to completely different solutions for the Lorenz chaos. The parameter β is critical for the Lorenz system. It determines whether the system is chaotic, as shown in Fig. 4.3(b), or not. Figure 4.3(a) shows stable solutions for the Lorenz system when $\beta = 16$.

Figures 4.1 and 4.2 show difficulty in analyzing the chaotic solutions. Instead of investigating the solutions directly in the time domain, we can do that in the phase space. The plot composed of components of solutions is called phase space or state space. Figure 4.3(a) shows a stable solution and Fig. 4.3(b) shows a chaotic solution of the Lorenz system in the phase space for different γ values. The trajectory of the solutions evolves with time in the phase space. From Fig. 4.3(b) one sees that the chaotic attractor is dense, bounded, nonintersectable, and of a fixed pattern. This attractor is called "strange" for these remarkable features. Any chaotic system has its characteristic strange attractor in the phase space. It is also important to note that the existence of a strange attractor is a strong indicator of chaos in a system, as described next. The code for solving the Lorenz system and reconstructing its strange attractor is provided in Appendix B.1.1.

Notice that Figs. 4.1(a) and (b) are very similar except for a scaling factor. A considerable difference between waveforms can be found by rescaling them.

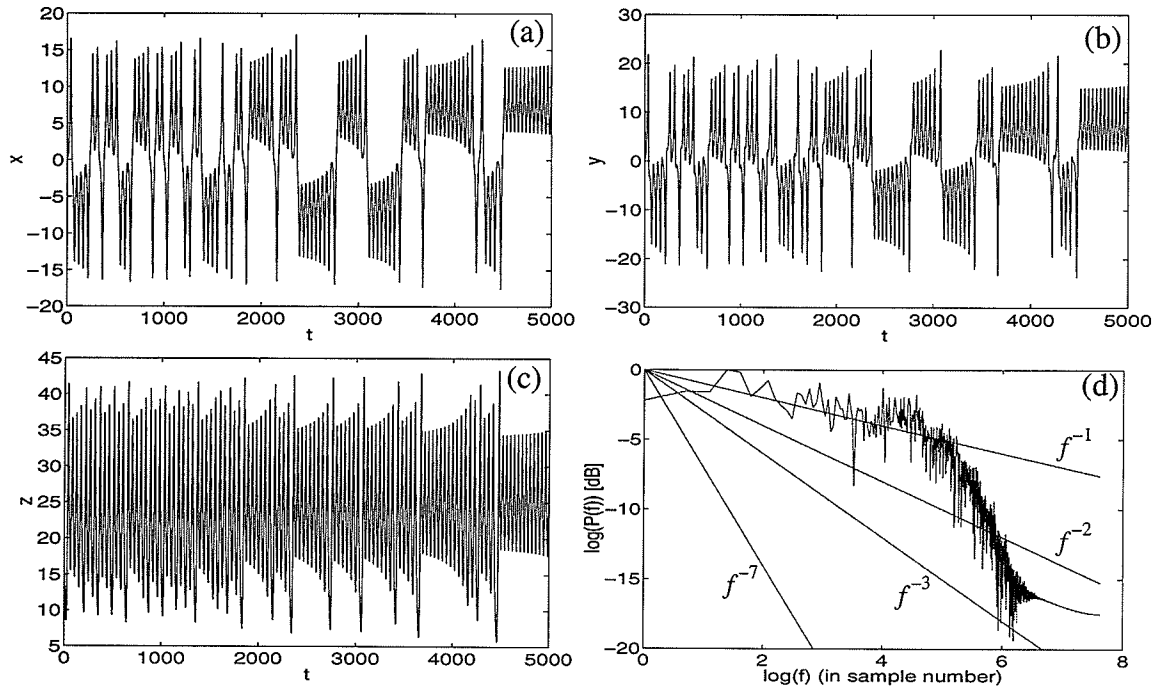

Fig. 4.1. When $\alpha = 10$, $\beta = 28$, and $\gamma = 8/3$, the Lorenz system gives chaotic solutions for its components of (a) x, (b) y, and (c) z. The spectrum distribution of the x component is shown in (d).

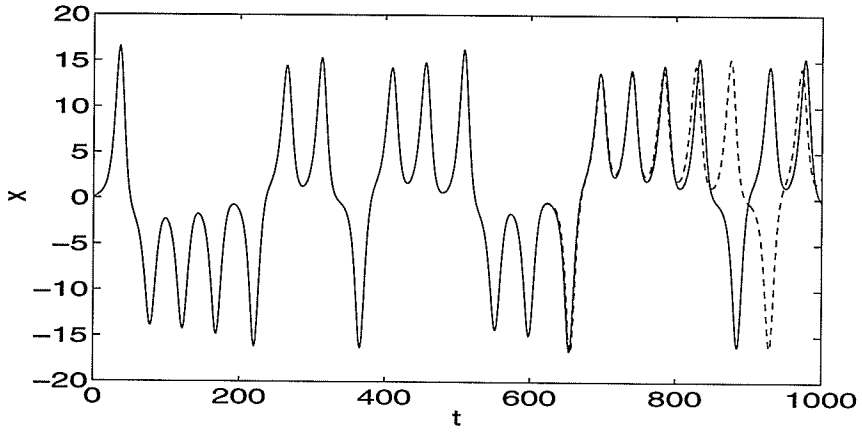

Fig. 4.2. The sensitivity of the Lorenz chaos to initial conditions. The solid line is the solution of the x component with initial values of $x = 0$, $y = 0.5$, and $z = 20$. The dashed line is the another solution with initial values of $x = 0$, $y = 0.500005$, and $z = 20$.

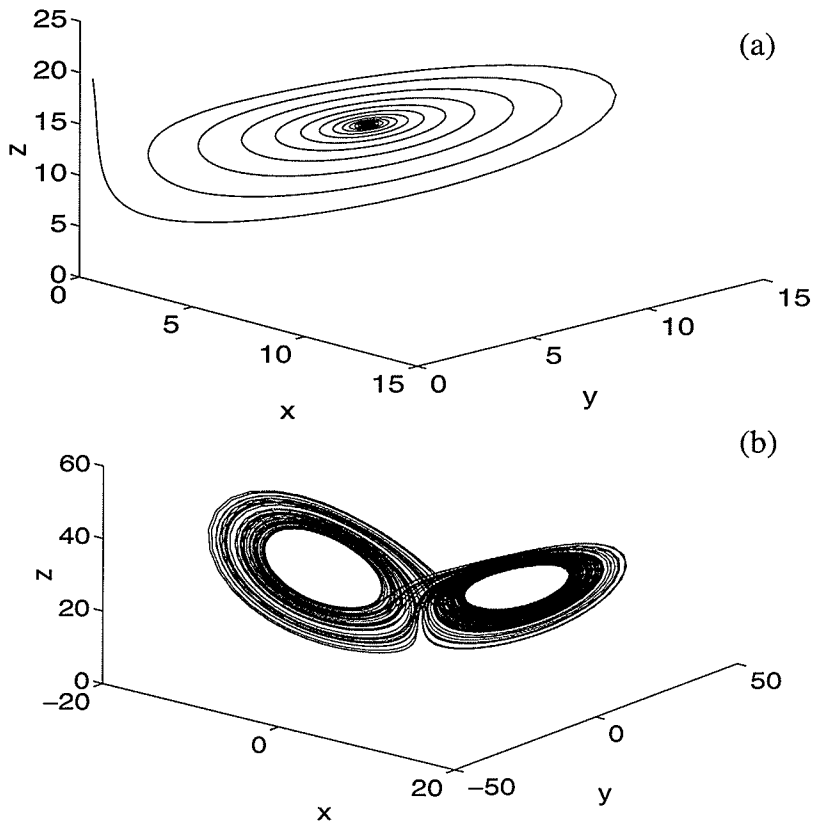

Fig. 4.3. The phase space of the Lorenz system with parameters $\alpha = 10$, $\gamma = 8/3$, and different β. Initial values are that $x(0) = 0$, $y(0) = 0.5$, and $z(0) = 20$. (a) Stable solutions with $\beta = 16$. (b) Chaotic solutions with $\beta = 28$.

4.3 Reconstruction of Strange Attractor

The strange attractor has a special place in pattern recognition because its structure corresponds to the nonlinear dynamics of the corresponding system. Such an attractor is the essence of the dynamical system. Frequently, we do not know the nonlinear dynamical differential equations or how many variables it has when we analyze a system. We can just take some measure; *i.e.* the time series of the signal from the system. How do we reconstruct the chaotic attractor from such a single measurement?

4.3.1 Takens Embedding Theorem

Packard *et. al.* first addressed the question of reconstructing an attractor from the trajectory of the time series [PCFS80]. Here the trajectory means the path followed by the system as it evolves with time. Assume $x(n)$ is the time series obtained by sampling a single coordinate of a dynamical system. From this measurement, it was found that one can obtain a variety of m independent quantities which appear to yield a phase space representation of the dynamics in the original space [PCFS80]. After proving the important observation made by Packard *et. al.*, Takens gave an embedding theorem [Take81]. The embedding theorem states that when there is a single measured quantity from a dynamical system, it is possible to reconstruct a state space that is equivalent to the original one composed of all dynamical variables. The embedding theorem states that if the system produces an orbit in the original state space that lies on a geometric object of dimension D_F (which need not be an integer), then the object can be unambiguously seen without any spurious intersections of the orbit in another space with dimension $m > 2D_F$, and comprised of coordinates that are nonlinear transforms of the original state space coordinates.

According to the Takens theorem, if the time series $x(n)$ is measured from a system, we can lag and embed the time series in an m-dimensional space by taking m-coordinates

$$\boldsymbol{u}(n) = (x(n), x(n+\tau), x(n+2\tau), \cdots, x(n+(m-1)\tau)) \ , \ 1 \le n \le N-(m-1)\tau \quad (4.2)$$

where τ is the lag of the time series, m is the embedding dimension of the reconstructed attractor, and N is the length of the time series. Corresponding to the phase space, we call

the above reconstruction a pseudo phase space since the reconstruction is not exactly the same as the original.

The remarkable consequence of the Takens theorem is that the strange attractor of the chaotic system can be reconstructed from the measured time series of a single coordinate. As an illustration, we take the x component of the solutions of the Lorenz system to reconstruct the Lorenz attractor. Figure 4.4(b) is the pseudo phase space of the Lorenz attractor with $\tau = 5$ and $m = 3$. We shall show how to get the lag τ in Sec. 4.3.2 and the embedding dimension m in Sec. 4.3.3 from the time series. The reconstructed attractor in Fig. 4.4(b) is not exactly the same as the original one in Fig. 4.4(a). It is a squashed, rotated, and projected version of the original attractor. The fact that the pseudo phase space is a distorted view of the real attractor is not important for our purpose. The important feature is that the two versions of the strange attractor are topologically equivalent. It means that both of them have the same dynamical properties. In particular, they have approximately the same values of the key indicators of chaos, such as Lyapunov exponents, correlation dimension, *Kolmogorov-Sinai entropy*, and *mutual information* [Will97]. Therefore, such properties are invariant, or unchanged, by making the pseudo phase space. In other words, when two geometric figures are topologically equivalent, a particular dynamical measure is unchanged regardless of its phase space or its pseudo phase space.

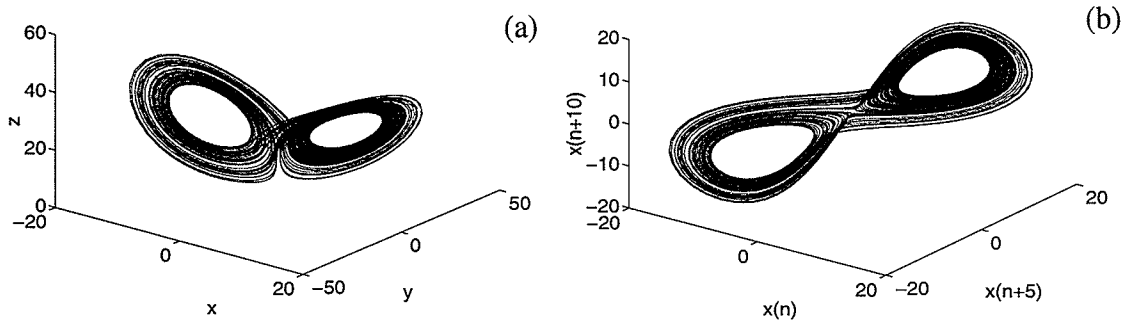

Fig. 4.4. The strange attractor of the Lorenz system and its reconstruction. (a) The original strange attractor. (b) The reconstruction of the Lorenz attractor through its x component with lag $\tau = 5$ in 3D phase space, $m = 3$.

4.3.2 Autocorrelation Function for Lag

The important issue in attractor reconstruction is how to choose the time lag τ and the embedding dimension m from the time series properly. First, let us discuss the choice of the lag. The lag is related to correlation among data. If the lag is too small, then each sample is plotted against itself because of high correlation as shown in Fig. 4.5(a). If the lag is too large, the dynamical relation between points becomes obscure and eventually random (see Fig. 4.5(c)).

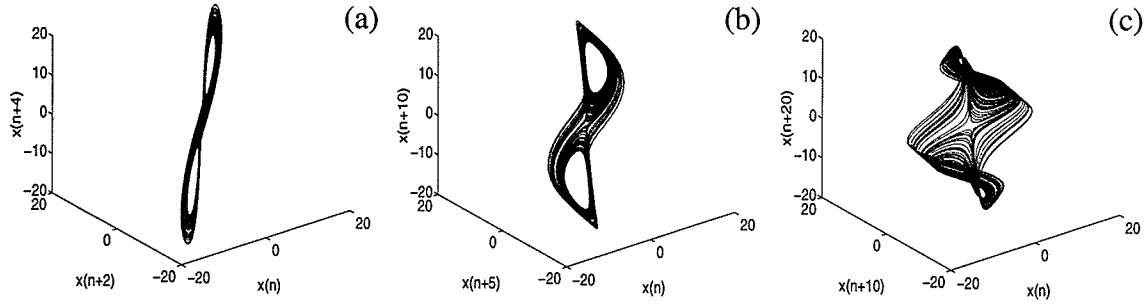

Fig. 4.5. The reconstruction of Lorenz attractor with lag of: (a) 2, (b) 5, and (c) 10.

Several techniques such as *autocorrelation function* indicators, *mutual information*, redundancy, *correlation integral*, and *space-filling*, have been proposed to find the optimal lag [Will97]. No single approach seems to apply to all conceivable conditions. We use autocorrelation function indicators to find the lag of the ECG time series. Autocorrelation measures the degree of correlation of a variable at one time with itself at another time. At a lag of zero, the coordinates of each plotted point are equal, and autocorrelation is a maximum of 1. It decreases as the lag increases. The optimal lag corresponds to the so-called autocorrelation time (the time required for the autocorrelation function to drop to some value). The autocorrelation function of the time series $x(n)$ is calculated as

$$C(\tau) = \frac{\sum_{j=1}^{N-\tau} x'(j)x'(j+\tau)}{\sum_{j=1}^{N-\tau} (x'(j))^2} \tag{4.3}$$

$$x'(j) = x(j) - \bar{x} \tag{4.4}$$

where $\bar{x}$ is the mean of the temporal signal and N is the length of the signal. We take the τ as the lag of the time series when $C(\tau)$ first drops below the value $(1 - 1/e)$ since such a τ is the attenuation constant of time of the 1st-order linear constant differential equation system. Figure 4.6(a) is an autocorrelation function plot of an ECG signal. From this figure we get an estimated lag of four sample points for the ECG signal. It is equivalent to about 11.11 ms. The reconstructed strange attractor of the ECG signal in Fig. 4.6(b) shows a bounded and deterministic pattern under this lag. The code for the autocorrelation function and reconstruction is provided in Appendix B.1.2.

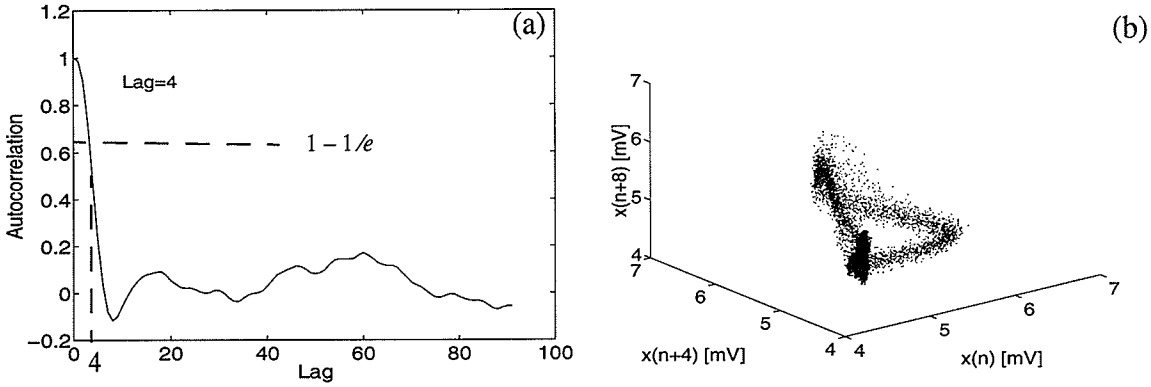

Fig. 4.6. (a) Autocorrelation function of an ECG signal. (b) Strange attractor reconstructed from the ECG time series with 60,000 samples by lag of 4 in 3D phase space.

4.3.3 False Nearest Neighbours

Once the lag τ is determined, the next step is to find the embedding dimension. Although many techniques, such as correlation dimension, *false nearest neighbours* (FNN), *principal component analysis*, and minimum mutual information, are proposed to find the minimum embedding dimension for attractor reconstruction, none of them is yet accepted widely [Will97]. A good idea in practice is to use more than one technique on the same data to give independent checks on results. We determine the minimum embedding dimension of the ECG attractor by checking the convergence of the Rényi dimension spectrum and the FNN [Kins94a] [KeBA92]. The Rényi dimension spectrum will be discussed in Sec. 4.6.

The FNN proposed by Kennel *et al.* deals with the minimal embedding dimension of the measured time series [KeBA92]. Nearest neighbour points closest to any chosen data point in pseudo phase space are called true or false. True nearest neighbours lie at

their true phase space distance from the chosen central data point. False nearest neighbours, in contrast, merely seem to be closer because the embedding space is too small. A point is identified as false if its distance from the central point continues increasing as the number of embedding dimensions increases. It means that the distance of the FNN continues to increase as long as the embedding dimension is too small. The correct embedding dimension is found when the number (or percentage) of the FNN decreases to approximately zero.

Assume we are in a j-dimensional phase space, working with points of an attractor reconstructed from the time series $x(n)$. The rth nearest neighbour of each point $\boldsymbol{u}(n)$ of the reconstructed attractor is denoted by $\boldsymbol{u}^{(r)}(n)$. Then the Euclidean distance between point $\boldsymbol{u}(n)$ and its rth neighbour is

$$d_j^2(\boldsymbol{u}(n), \boldsymbol{u}^{(r)}(n)) = \sum_{k=0}^{j-1} [x(n+k\tau) - x^{(r)}(n+k\tau)]^2 \tag{4.5}$$

where τ is the lag determined from the previous experiment (Sec. 4.3.2).

A j+1 coordinate is added to $d_j^2(\boldsymbol{u}(n), \boldsymbol{u}^{(r)}(n))$ when the pseudo phase space is extended from dimension j to j+1.

$$d_{j+1}^2(\boldsymbol{u}(n), \boldsymbol{u}^{(r)}(n)) = d_j^2(\boldsymbol{u}(n), \boldsymbol{u}^{(r)}(n)) + [x(n+j\tau) - x^{(r)}(n+j\tau)]^2 \tag{4.6}$$

A natural criterion for false neighbours is that the increase in relative distance between $\boldsymbol{u}(n)$ and $\boldsymbol{u}^{(r)}(n)$ is greater than a threshold when going from dimension j to j+1. The criterion is defined as

$$\left[\frac{d_{j+1}^2(\boldsymbol{u}(n),\boldsymbol{u}^{(r)}(n))-d_j^2(\boldsymbol{u}(n),\boldsymbol{u}^{(r)}(n))}{d_j^2(\boldsymbol{u}(n),\boldsymbol{u}^{(r)}(n))}\right]^{\frac{1}{2}} = \frac{\left|x(n+j\tau)-x^{(r)}(n+j\tau)\right|}{d_j(\boldsymbol{u}(n),\boldsymbol{u}^{(r)}(n))} > T_d \quad (4.7)$$

where T_d is the threshold found by numerical experiment (*i.e.*, by fixing the embedding dimension to test the sensitivity of different values of T_d to the relative distance). It is sufficient to consider only the nearest neighbours $r = 1$ and interrogate every point on the attractor to check how many nearest neighbours are false.

Equation (4.7) is not sufficient, however, to determine the embedding dimension m in attractor reconstruction. It is found that even though $\boldsymbol{u}^{(1)}(n)$ is the nearest neighbour of $\boldsymbol{u}(n)$ it is not necessarily close to $\boldsymbol{u}(n)$ because of the influence of noise. Kennel *et al.* [KeBA92] proposed an additional criterion as

$$\frac{d_j(\boldsymbol{u}(n),\boldsymbol{u}^{(r)}(n))}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}[x(i)-\bar{x}]^2}} > Ta \quad (4.8)$$

where $\bar{x}$ is the mean of the time series and Ta is another threshold found by experiment.

Figure 4.7 gives the ratio of the FNN versus embedding dimension under different T_d. The curves converge when $T_d \geq 6$. From this experiment, a minimal embedding dimension of 7 is obtained when $Ta = 1$, $T_d \geq 6$, and the percentage of the FNN is about 0.6%. The code for the FNN is provided in Appendix B.1.2.

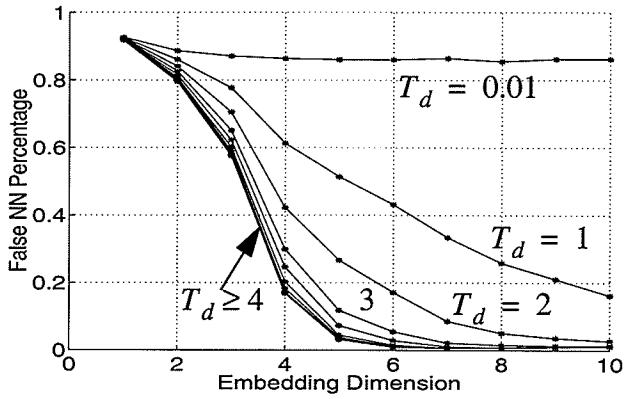

Fig. 4.7. The ratio of the FNN with different T_d in the reconstructed attractor of the ECG time series for different embedding dimensions. The length of the time series is 5000 sample points.

After the strange attractor of the dynamic system is reconstructed, its chaotic characteristics need to be investigated further. The complexity features of chaotic systems may be characterized by fractal technique. In the next section of this chapter, we shall discuss the following points: (i) what is fractal; (ii) how to use Lyapunov exponents to determine the chaos of a system; and (iii) how to use the Rényi dimension spectrum to determine the embedding dimension of a chaotic system and measure the complexity of the strange attractor.

4.4 Fractal Sets

In general, people are used to thinking about objects in topological space more than other spaces. *Topological* (or *Euclidean*) *dimension* D_E is the ordinary integer dimension of abstract objects (set S) such as a point ($D_E = 0$), line ($D_E = 1$), surface ($D_E = 2$), and volume ($D_E = 3$). A set S has topological dimension D_E if each point in S has an arbitrarily small neighbourhood ε whose boundaries intersect S in a set of

dimension $D_E - 1$, where D_E is the least non-negative integer for which this holds. Although topological dimension finds perfect applications to mathematical smooth objects, it loses capability of describing complex natural objects such as trees, rivers, coastlines, mountains, and lightning.

Fractal geometry, which was popularized by Mandelbrot in 1960, provides a tool for describing complex objects. Mandelbrot [Mand82] defines a fractal as a set for which the Hausdorff-Besicovitch (fractal) dimension strictly exceeds the topological dimension. A more accurate definition by Devaney [Deva92] is that a fractal is a subset in $\mathbf{R}^n$ which is self-similar and whose fractal dimension exceeds its topological dimension. Unlike topological geometry with the integer dimension for all objects, fractal geometry uses a fractional dimension to depict the roughness of the complex objects, which should be an important feature of objects in nature. Another feature of a fractal is its self-similarity or self-affinity. A strictly self-similar object is an object which is constructed from exactly the same segments, under various degrees of magnification. That is to say that each small part replicates the whole structure exactly. One of the strictly self-similar examples is the Minkowski curve [PeJS92].

An initiator and a generator are necessary for the generation of a strict fractal object. The Minkowski curve is produced according to the following four steps:

1) Let the initiator be a unit straight line (it may be anything).
2) The generator contains eight initiators with a scaling of 1/4 each. The structure of the generator shows at Step 1 in Fig. 4.8.
3) At Step k, the object at Step $k-1$ is scaled by 1/4 and then used to replace every initi-

ator in the generator.

4) The Minkowski curve is a fractal object when k is infinite.

The generation of the Minkowski curve shows that a complicated fractal may be produced by simple rules. When we zoom on the curve, we always get the same structure as the entire curve. Although the Minkowski curve seems simple, its mathematical characteristics are complicated. In the limit, when $k \rightarrow \infty$, it is nowhere differentiable, it is irregular everywhere, and the length of the curve is infinite. Such irregular characteristics are owned by all fractals. Another example of fractal is the Koch curve shown in Fig. 4.9.

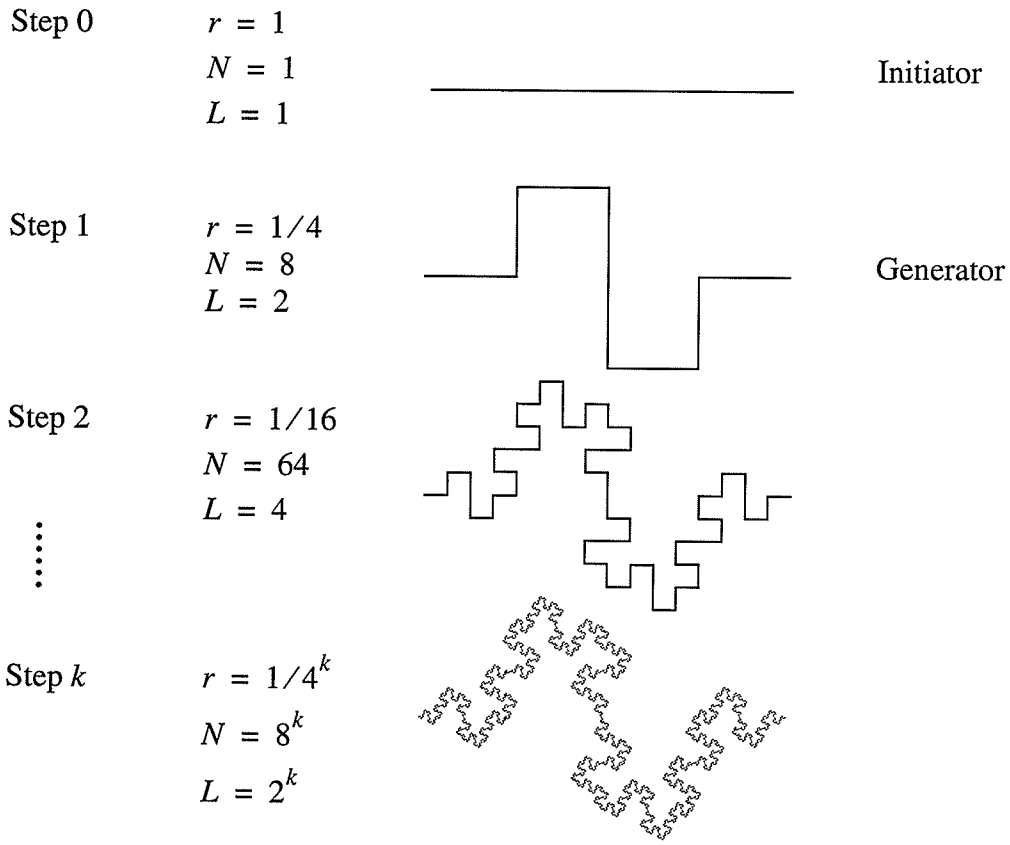

Fig. 4.8. Generation of the Minkowski curve, where r is the size of a segment, N is the number of segments, and L is the total length of the curve (after [Kins94a]).

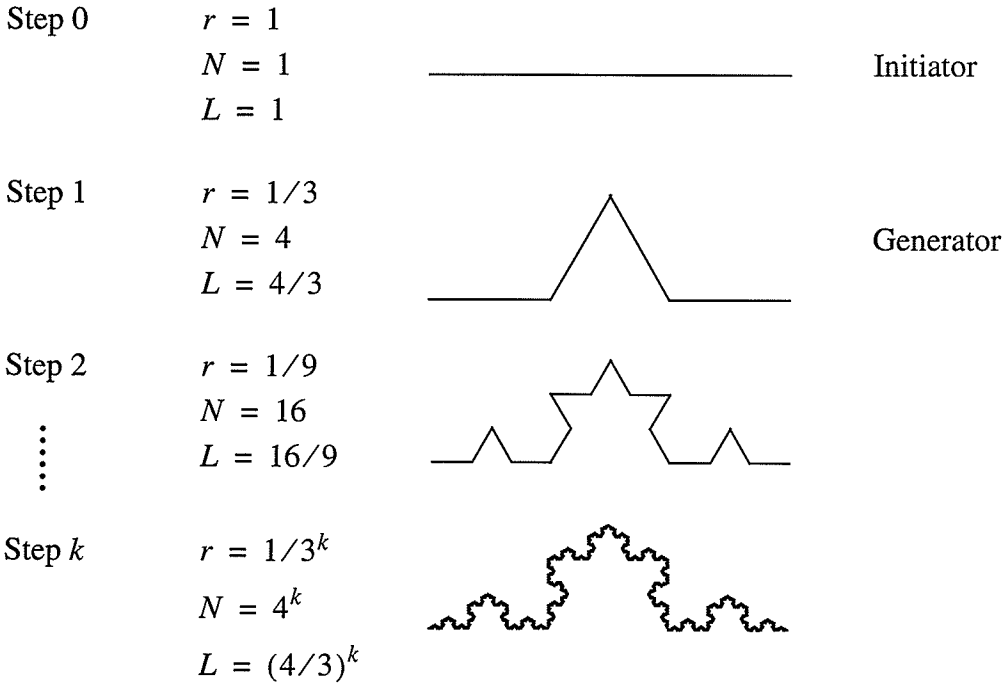

Fig. 4.9. Generation of the Koch curve, where r is the size of a segment, N is the number of segments, and L is the total length of the curve (after [Kins94a]).

Unlike the Minkowski curve and the Koch curve which have the same scaling factor in any direction (x and y), other fractal objects may have different scales along different directions (*e.g.*, the time of a time series may scale by 2, but its amplitude may scale by $\sqrt{2}$). Such objects are called self-affine [Kins94a]. However, most objects in nature are neither strictly self-similar nor self-affine, but have a finite range of self-similarity/affinity. Some fractals have scale-invariance with respect to their statistics and are called statistically self-similar or self-affine [Kins94a].

A fractal object can be characterized by a scalar called fractal dimension. The fractal dimension characterizes the irregularity of the fractal. If the object is smooth, the fractal dimension is equal to its topological dimension. The fractal dimension increases with

the roughness or irregularity. Therefore, since the complexity of an object can be measured by the fractal dimension, it may be employed as a feature for object classification.

A power law is fundamental in fractal dimension estimation. Different power laws result in different dimensions. There are many fractal dimensions. Kinsner has presented more than 21 fractal dimensions in a unified framework [Kins94a], where the dimensions are classified into morphological, entropy, variance, and spectral dimensions. We shall discuss the *Hausdorff mesh dimension*, the Rényi dimension spectrum, the variance dimension, and the Lyapunov exponent based on morphological, entropy, variance, and spectral measures on the object.

4.5 Single Fractals and Their Limitations

This section focuses on several typical single fractals and their limitations in complexity measure, and it follows the presentation in [Kins94a].

4.5.1 Self-Similarity Dimension

Consider a bounded set S in the m-dimensional Euclidean space. The set S is said to be self-similar when it is the union of $N(r)$ distinct (non-overlapping) copies of itself, each of which has been scaled down by a ratio r in all coordinates. The *similarity dimension* of S is given by the following power-law relation

$$N(r) = r^{-D_s} \text{ for } r \to 0 \tag{4.9}$$

or

$$D_S = \frac{\log N(r)}{\log(1/r)} \tag{4.10}$$

Although the Minkowski curve is continuous and spans a finite distance, it is not a line. The length of the segment consisting of the curve is zero. From Fig. 4.8, we have $r = 1/4^k$ and $N(r) = 8^k$ at Step k for the curve. Therefore, the self-similarity dimension of the Minkowski curve is

$$D_S = \frac{\log 8^k}{\log 4^k} = 1.5 \tag{4.11}$$

while the self-similarity dimension of the Koch curve shown in Fig. 4.9 is

$$D_S = \frac{\log 4^k}{\log 3^k} = 1.2619 \tag{4.12}$$

The similarity dimension measure is only suitable for strictly self-similar fractal sets. However, the majority of fractals are not exactly self-similar. They may even be random. Therefore, other measure techniques are needed to estimate such fractal objects.

4.5.2 Hausdorff Mesh Dimension

The Hausdorff mesh dimension provides a basic and practical measure for fractal dimensions. It characterizes geometrical complexity of fractal objects. For multi-dimensional objects, one must consider a hypervolume element set with size r, called vel for short, for fractal measure. One way of measuring dimension of an object is to use the vel set to cover the object and then count the minimum number of the covering set, $N(r)$,

intersected by the object [Kins91], [CaRe94]. The Hausdorff mesh dimension is defined by the following power-law relation

$$N(r) \sim r^{-D_{HM}} \tag{4.13}$$

The dimension is found by repeated measurements of $N(r)$ on reduced mesh sizes according to

$$D_{HM} = \lim_{r \to 0} \frac{\log N(r)}{\log(1/r)} \tag{4.14}$$

The Hausdorff dimension is often referred to as the box counting dimension (D_B) when the vels are square and not overlapping. The Hausdorff dimension can be considered as a morphological dimension [Kins94]. The morphological dimension uses the number of geometrical covering vels as a basic measure. The morphological-based fractal dimensions can be used if the distribution of a measure (such as probability) is uniform (*i.e.*, if the fractal is homogeneous) or the information about the distribution is not available. Since a geometrical coverage is involved, these dimensions are purely a geometrical concept.

If a fractal is nonuniform or its probability density is nonuniform, the fractal dimension must be estimated based on entropy measure. This class includes: *information dimension*, correlation dimension, Rényi dimension spectrum, and multifractal (Mandelbrot) dimension spectrum. As an example, information dimension is defined in terms of the relative frequency of visitation of a typical trajectory (in temporal fractals) or the distribution measure (in spatial fractals), so it uses either information about the time behav-

iour of a dynamical system, or the measure describing the inhomogeneity of a spatial fractal [Kins94b].

4.5.3 Information Dimension

The Shannon entropy is used widely to reflect the complexity of objects. It play a central role in information theory as measures of information, choice and uncertainty. Let us consider covering a fractal object with $N(r)$ vels. The size of the vel is r, where r is either radius or diameter or some other estimate of the size. The frequency in the fractal object may be distributed nonuniformly. The Shannon entropy is defined as

$$H_r = -\sum_{j=1}^{N(r)} p_j \log p_j \tag{4.15}$$

where p_j is defined by

$$p_j = \lim_{\substack{N_T \to \infty \\ r \to 0}} \frac{n_j}{N_T} \text{ and } N_T = \sum_{j=1}^{N(r)} n_j \tag{4.16}$$

where n_j is the number of times the fractal intersects the jth vel of the covering, and N_T is the total number of intersections of the fractal with all the vels.

Based on the Shannon entropy, the information dimension is defined as

$$D_I = \lim_{r \to 0} \frac{H_r}{\log(1/r)} \tag{4.17}$$

4.5.4 Correlation Dimension

The correlation dimension is another entropy fractal dimension. Assume the following power law holds between the sum of squared probabilities over all the vels with size r

$$\sum_{j=1}^{N(r)} p_j^2 \sim r^{D_C} \tag{4.18}$$

where the probability p_j has the same meaning with the probability defined in Sec. 4.5.3.

The correlation dimension is

$$D_C = \lim_{r \to 0} \frac{\log \sum_{j=1}^{N(r)} p_j^2}{\log(r)} \tag{4.19}$$

4.5.5 Limitations of Single Fractal Measures

Usually the objects in nature are not single fractal objects. They are multiple fractal objects. To investigate the objects with multiple fractals, we put the Minkowski curve and the Koch curve together without intersecting as shown in Fig. 4.10. The box counting dimension and the correlation dimension of the Minkowski curve, the Koch curve, and the composite of them are obtained from the log-log plot of the curves (the detail about this technique will be presented in Sec. 4.8.2). The similarity dimension can be obtained directly from (4.10). The experimental conditions and results are given in Tables 4.1 and 4.2, respectively.

The experimental results reveal some important observation. For the single fractal such as the Koch curve, its similarity dimension (D_S), box counting dimension (D_B), and correlation dimension (D_C) are very close. The same conclusion can be made from the Minkowski curve. In the nonintersecting composite of the two curves, its D_S equals to the bigger fractal of the two single fractal sets. The D_B and D_C are also dominated by the fractal with a higher complexity. It means that these three dimensions only describe the most complicated features of the multifractal objects. Therefore, using a single fractal dimension or any other single value complexity to depict complex objects is incomplete.

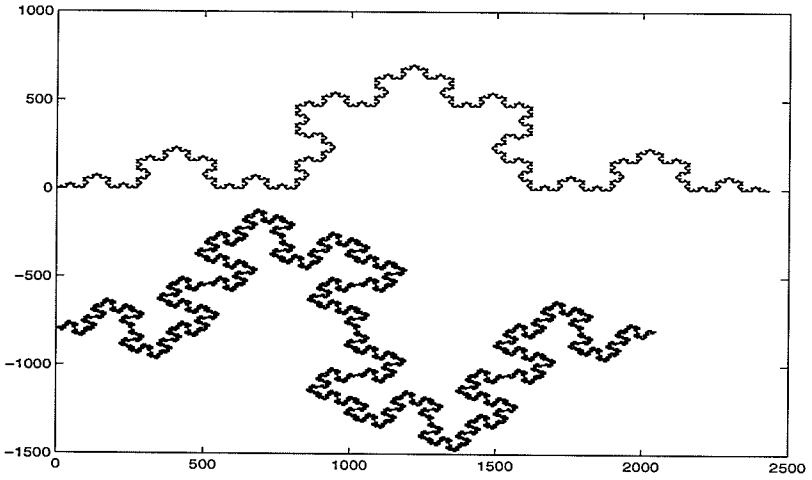

Fig. 4.10. The nonintersecting composite of the Koch and the Minkowski curves.

Table 4.1: Conditions for generating the Koch curve, the Minkowski curve, and the nonintersecting composite.

Dimension	Iterations/number of points in each segment		
	Koch Curve	Minkowski Curve	Nonintersecting Composite
Similarity	∞	∞	∞
Box Counting	7/2	4/3	6/2 (Koch) and 4/3 (Minkowski)
Correlation	6/2	4/3	6/2 (Koch) and 4/3 (Minkowski)

Table 4.2: Fractal dimensions for the Koch curve, the Minkowski curve, and the nonintersecting composite.

Dimension	Koch Curve	Minkowski Curve	Nonintersecting Composite
Similarity	1.2619	1.5000	1.5000
Box Counting	1.2663	1.5446	1.4850
Correlation	1.2594	1.4984	1.4660

4.6 Multifractal Dimensions

4.6.1 The Rényi Dimension Spectrum

Instead of measuring a single fractal, the Rényi dimension spectrum (D_q) measures a series of complexities of complicated objects based on the Rényi entropy. The Rényi entropy H_q is [Kins94a]

$$H_q = \frac{1}{q-1}\log \sum_{j=1}^{N(r)} p_j^q \qquad -\infty < q < \infty \tag{4.20}$$

where p_j is the probability defined by (4.16) and $N(r)$ is the number of non-empty vels intersected by the fractal object. The vel has the side length of r. The Rényi dimension spectrum is defined as

$$D_q = \lim_{r \to 0} \frac{1}{q-1} \frac{\log \sum_{j=1}^{N(r)} p_j^q}{\log r} \tag{4.21}$$

Kinsner shows that [Kins94a]

$$D_{-\infty} = \lim_{r \to 0} \frac{\log(p_{min})}{\log(r)} \tag{4.22}$$

$$D_{\infty} = \lim_{r \to 0} \frac{\log(p_{max})}{\log(r)} \tag{4.23}$$

$$D_0 \equiv D_{HM},\ D_1 \equiv D_I,\ D_2 \equiv D_C, \text{ and} \tag{4.24}$$

$$D_{\infty} \leq D_{HM} \leq D_I \leq D_C \leq D_{-\infty} \tag{4.25}$$

Therefore, the Rényi dimension spectrum includes the Hausdorff mesh dimension D_{HM}, the information dimension D_I, and the correlation dimension D_C as special cases. It is a spectrum of complexity measure of multifractals. Figure 4.11 shows an example of the Rényi dimension spectrum from a strange attractor of an ECG signal with 5000 samples. As explained in Sec. 4.3, the strange attractor of the ECG is reconstructed with embedding dimension of 7 and lag of 4.

The Rényi dimension spectrum has four important properties: (i) it gives the multifractal measure of complex objects; (ii) it is monotonically decreasing with q; (iii) it is bounded; and (iv) it is always normalized.

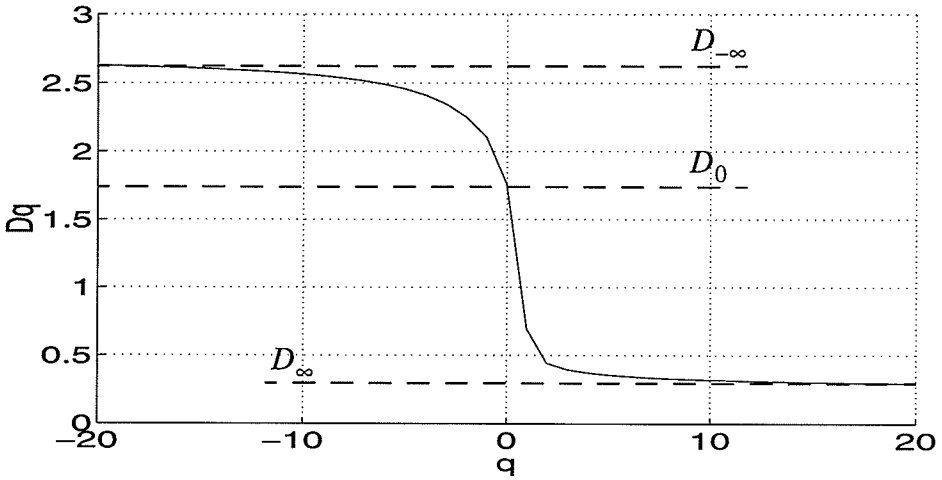

Fig. 4.11. The Rényi dimension spectrum of the ECG with 5000 samples, in which the strange attractor of the ECG is reconstructed with embedding dimension of 7 and lag of 4.

4.6.2 The Rényi Dimension Spectrum Estimate

Given a time series of a fractal object, its Rényi dimension spectrum can be estimated from the log-log plot (see Sec. 4.8.2), if the probability distribution of the strange attractor is known. One of the techniques to estimate the Rényi dimension spectrum is the box counting algorithm [Kins94b], which estimates the probability distribution of the attractor by dividing the phase space into vels without overlapping. Equation (4.16) shows how to estimate the probability distribution. Probability estimation through the box counting technique is simple and easily implementable. A drawback of this technique is the large size of memory required by the algorithm. It is not suitable for the objects with high embedding dimensions.

Another technique used to estimate the probability distribution in fractal dimension is the pair-correlation function (integral), which was suggested by Grassberger and Procaccia [GrPr83]. Pawelzik and Schuster [PaSc87] extended the pair-correlation func-

tion to any order in the other form. We switch from p_j, *i.e.*, the probability to find the trajectory in one of the homogeneously distributed boxes introduced in (4.16), to $\tilde{p}_j$ which denotes the probability to find the trajectory within a ball around one of the inhomogeneously distributed points of the trajectory. The latter can be written as

$$\tilde{p}_j(r) = \frac{1}{N}\sum_{i=1}^{N}\theta(r - \|\boldsymbol{u}(i) - \boldsymbol{u}(j)\|) \tag{4.26}$$

where $\theta(x)$ is the Heaviside step function and $\boldsymbol{u}(i)$ is defined as (4.2). In this case, the Rényi dimension spectrum is given as

$$D_q = \lim_{r\to 0}\frac{\log C_q(r)}{\log r} \tag{4.27}$$

where the extended pair-correlation function is

$$\begin{aligned} C_q(r) &= \left\{\frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{N}\sum_{j=1}^{N}\theta(r - \|\boldsymbol{u}(i) - \boldsymbol{u}(j)\|)\right]^{q-1}\right\}^{\frac{1}{q-1}} \\ &= \left\{\frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{N}\sum_{j=1}^{N}\theta\left(r - \left(\sum_{k=0}^{m-1}(x(i+k\tau) - x(j+k\tau))^2\right)^{\frac{1}{2}}\right)\right]^{q-1}\right\}^{\frac{1}{q-1}} \end{aligned} \tag{4.28}$$

where m is the embedding dimension and τ is the time lag of the time series $x(n)$. This is the technique we use to estimate the Rényi dimension spectrum of the reconstructed attractor of the ECG signal.

4.6.3 Multifractal Characteristics of ECG Strange Attractor

We have two reasons for using the Rényi dimension spectrum to analyze the ECG strange attractor: (i) to find the embedding dimension of the ECG strange attractor through the convergence investigation of the Rényi dimension spectrum of the ECG time series, and (ii) to measure the complexity spectra of the strange attractor.

Experiments have been done using the data from the MIT BIH ECG database. The sampling rate of the data is 360 sps. The Rényi dimension spectrum is calculated under different embedding dimensions for an ECG time series with 5000 sample points since we do not know how many variables there are in the dynamical system of the heart. The embedding dimension is set from 1 to 13. For each embedding dimension, let $\log(r)$ change linearly within the interval [-7,1], where r is the vel size. The extended pair correlation, $C_q(r)$, of the attractor is estimated for each r. The $\log(C_q(r))$ vs. $\log(r)$ plots are shown in Fig. 4.12(a). The Rényi dimension spectrum, D_q, cannot be calculated directly from (4.27) because it is impossible for r tending to zero in real world. In general, D_q may be obtained from the linear region of the slope of the $\log(C_q(r))$ vs. $\log(r)$ plots.

We take [-3, 1] as an approximately linear region of the log-log plots to calculate the Rényi dimension spectrum of the ECG signal through the least-square fitting technique discussed in Sec. 4.8.2. The code is provided in Appendix B.1.2. The Rényi dimension spectrum with various embedding dimensions shown in Fig. 4.12(b) is calculated from that region. It shows that the Rényi dimension spectrum varies with embedding dimension. The Rényi dimension spectrum curves for $q < 0$ are convergent when the embedding dimension is greater than 4, while the curves for $q > 0$ do not converge. We propose that

the incomplete convergence of the Rényi dimension spectrum is due to the low-sampling frequency and noise in the ECG data. The low-sampling frequency limits the broadband spectrum characteristics of chaos in the signal, while noise breaks down the criterion that unfolds the chaos in embedding space.

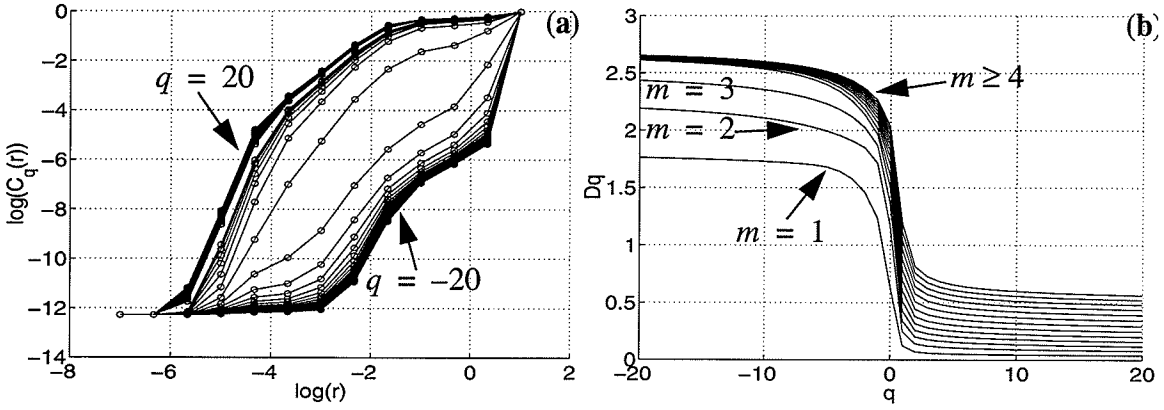

Fig. 4.12. The convergence investigation of the attractor reconstructed through the ECG time series with different embedding dimensions. The length of the time series is 5000 sample points. (a) log-log plots of the extended correlation function with approximately linear region at [-3, 1], and (b) the Rényi dimension spectrum curves.

It may be argued that the sampling rate is certainly adequate for the real ECG signal and the noise level is very low. Since most of the ECG energy is concentrated between 0 and 100 Hz, the sampling rate of 360 sps is adequate for some ECG applications in which the variability of the ECG waveform is not important. If a signal is periodic and has finite spectrum, then its Rényi dimension spectrum is convergent. However, the sampling rate and noise are related to the system dynamics of the heart. The ECG signal is actually not periodic and has broadband spectrum. Accurate exposition of notches in an ECG requires a bandwidth having an upper frequency of several thousand Hertz [Rawl90]. Consequently, an ECG signal with high sampling rate is required for chaos study.

The real ECG signal appears to be clean since its noise components over 180 Hz have been filtered out. This signal contains low-frequency noise that can be reduced by denoising techniques. Characteristics of noise and chaos may be studied in the phase space. It has been demonstrated that there is no convergence in phase space reconstruction of white noise [Tina99]. Since the convergence of the Rényi dimension spectrum of the ECG is improved by the denoising experiments as described in the next chapter, this ECG signal is not noise free.

The convergence of the Rényi dimension spectrum is used as another criterion to determine the embedding dimension m in the attractor reconstruction. Here the convergence is related to the deterministic pattern of the ECG attractor. It means that the attractor is unfolded completely in that space. The D_q curves of the white noise do not converge when the embedding dimension increases since a fixed pattern or attractor does not exist for noise [Ehti99].

From the convergence investigation of the Rényi dimension spectrum and the false nearest neighbours of the ECG time series, the minimal embedding dimension of 7 is determined for the attractor reconstruction purpose. Under this embedding dimension we calculate the Rényi dimension spectrum of the ECG shown in Fig. 4.11. The curve is strictly monotonic decreasing and bounded in (0, 3). The Hausdorff dimension D_{HM} is about 1.75. Thus, the ECG signal has multifractal characteristics.

4.7 Lyapunov Dimension

The *Lyapunov fractal dimension*, D_{Ly}, is useful in the study of chaotic systems,

and particularly in the prediction of the dimension of a strange attractor from the knowledge of Lyapunov exponents of the corresponding transform [KaYo78].

The Lyapunov exponents are a quantitative measure of the sensitive dependence of trajectories on initial conditions (which is a characteristic of chaotic behaviour). They are averaged rates of divergence or convergence of nearby orbits (*i.e.*, the evolution of neighbouring phase trajectories), which can be used to distinguish between chaotic and nonchaotic processes. Actually there is a spectrum of Lyapunov exponents. Their number is equal to the dimension m of the phase space. This approach can be viewed as another multifractal representation of the strange attractor.

To obtain the Lyapunov fractal dimension spectra, imagine an infinitesimal small ball with radius r_0 sitting on the initial state of a trajectory. The flow will deform this ball into an ellipsoid. That is, after a finite time t all orbits which start in that ball will be in the ellipsoid. The ith Lyapunov exponent is defined by

$$\lambda_i = \lim_{r_0 \to 0} \lim_{t \to \infty} \frac{1}{t} \ln\left(\frac{r_i(t)}{r_0}\right) \tag{4.29}$$

where $r_i(t)$ is the radius of the ellipsoid along its ith principal axis as shown in Fig. 4.13.

The Lyapunov dimension is defined as [Ott93]

$$D_{Ly} = K + \frac{1}{|\lambda_{K+1}|} \sum_{j=1}^{K} \lambda_j \tag{4.30}$$

where λ_j denotes the Lyapunov exponents and is ordered as $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$, and K is

the maximum number of the Lyapunov exponents which makes the sum of the exponents nonnegative

$$\sum_{j=1}^{K} \lambda_j \geq 0 \text{ and } \sum_{j=1}^{K+1} \lambda_j < 0 \tag{4.31}$$

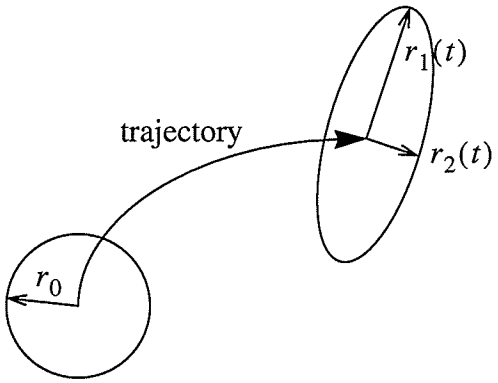

Fig. 4.13. An illustration of how trajectories diverge and converge in 2D phase space.

4.8 Variance Dimension and Log-Log Plot

4.8.1 Variance Dimension

Unlike the Rényi dimension spectrum which analyzes fractal objects based on entropy measure, the variance dimension analyzes the spread of the increments in the signal amplitude (variance, σ^2) directly in time. The variance dimension is usually applied to characterize the fractal components of a nonstationary time series. Although the spectral dimension may reveal the multifractal nature of the nonstationary signal for different short-time (windowed) Fourier analysis, the choice of the window size is difficult, and may introduce artifacts. Since small numerical errors may affect the solution very much, adding artifacts may become catastrophic. The advantage of the variance dimension is that

it does not require a window in the Fourier sense, and therefore does not introduce the window artifact. Another important advantage of the variance approach is that it can be formulated either for a batch or real-time computation [Kins94b].

Now we illustrate how to calculate the *variance fractal dimension* (VFD) from a given time series. Let us assume that the signal $x(n)$ is discrete in time n. The variance, σ^2, of its amplitude increments over a time increment is proportional to the time increment according to the following power law

$$Var[x(n_2) - x(n_1)] \sim |n_2 - n_1|^{2H^*} \tag{4.32}$$

where H^* is the Hurst exponent. By setting $\Delta n = |n_2 - n_1|$ and $(\Delta x)_{\Delta n} = x(n_2) - x(n_1)$, the exponent H^* can be calculated from

$$H^* = \lim_{\Delta n \to 0} \frac{1}{2} \frac{\log[Var(\Delta x)_{\Delta n}]}{\log(\Delta n)} \tag{4.33}$$

For the embedding Euclidean dimension D_E (*i.e.*, the number of independent variables in the observed signal), the variance dimension can be computed from

$$D_\sigma = D_E + 1 - H^* \tag{4.34}$$

4.8.2 Log-Log Plot Estimate

For measured data, it is not practical to set $\Delta n \to 0$. Instead of using (4.33), H^* can be obtained from a log-log plot in which it is related to the slope. To spread the point on the log-log plot equally, a finite sequence of time increments, $\{\Delta n_1, \Delta n_2, \cdots, \Delta n_m\}$,

should follow a b-adic sequence (such as dyadic), as shown in Fig. 4.14. First, we calculate the following two log values for the plot

$$X_m = \log_b \Delta n_m \tag{4.35}$$

$$Y_m = \log_b (Var(\Delta x)_m) \tag{4.36}$$

where Δn_m is the discrete time increment at the mth order scale. Then the slope s is determined from these points by a polynomial line fitting technique, with careful attention given to outliers [Kins94a].

$$s = \frac{(i_2 - i_1 + 1)\sum_{j=i_1}^{i_2} X_j Y_j - \sum_{j=i_1}^{i_2} X_j \sum_{j=i_1}^{i_2} Y_j}{(i_2 - i_1 + 1)\sum_{j=i_1}^{i_2} X_j^2 - \left(\sum_{j=i_1}^{i_2} X_j\right)^2} \tag{4.37}$$

where the range $[i_1, i_2]$ corresponds to the optimal linear range of the log-log plot. For example, the linear range is [2, 5] for Fig. 4.14.

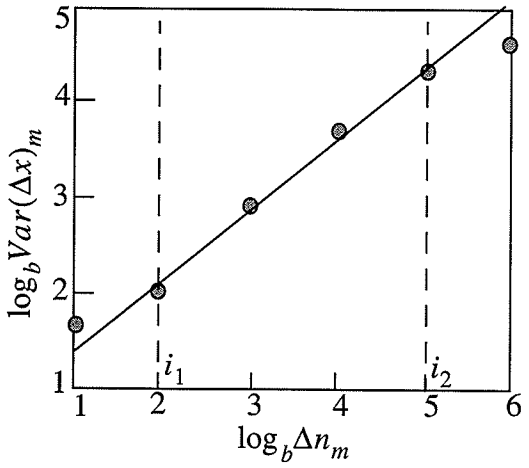

Fig. 4.14. The log-log plot and its line fitting.

The Hurst exponent is obtained from

$$H^* = s/2 \tag{4.38}$$

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

In the above processing, if Δn_m is replaced by r_k and $Var(\Delta x)_m$ is replaced by the measurement such as entropy or morphology, we may obtain other fractal dimensions such as the Rényi dimension spectrum.

4.9 Summary

In this chapter, we examine the chaotic and multifractal characteristics of the ECG through its time series. The majority of natural phenomena can be modelled as nonlinear dynamical systems. The ECG signal should not be an exception. The fact that the ECG signal has the same spectral distribution as pink noise makes us consider that it has chaotic characteristics. After the lag of 11.11 ms is obtained for the ECG time series from the autocorrelation function, a low dimensional attractor for perceptual purpose is reconstructed in light of the Takens theorem. One may see the bounded and deterministic pattern in the strange attractor.

To reconstruct the strange attractor of the ECG, a minimal embedding dimension needs to be found out. The convergences of the Rényi dimension spectrum and the FNN are investigated to determine the embedding dimension. The percentage of the FNN decreases to almost zero and its curves converge when the embedding dimension of the reconstructed attractor is equal to or greater than 7. The extended correlation function is

also convergent. The incomplete convergence of the Rényi dimension is due to the low-sampling frequency and noise in the ECG data. The low-sampling frequency limits the broadband spectrum characteristics of chaos in the signal, while noise breaks down the criterion that unfolds the chaos in the embedding space.

To characterize the ECG attractor, we use multifractals to extract complexity features of the object. Multifractals can give more information than single fractal when characterizing a complex object. The minimal embedding dimension of 7 is determined for the ECG attractor reconstruction purpose. Under this embedding dimension, we calculate the Rényi dimension spectrum of the ECG. The curve is strictly monotonic decrease. Its Hausdorff dimension, D_{HM}, is about 1.75. We say that the ECG signal has multifractal characteristics.

The Rényi dimension spectrum of the ECG attractor is bounded between 0 and 3. This is a unique property of the Rényi multifractal dimension that the feature normalization is accomplished automatically during the feature extraction procedures. It has a special importance in signal classification.

Consequently, an ECG data acquisition system with a high sampling rate needs to be developed for the study of chaos and fractal feature extraction. Denoising techniques, which will be discussed in the next chapter, are necessary to reduce the noise, while at the same time preserving the chaotic characteristics of the ECG [ScKa96]. The denoised ECG signal with a high sampling frequency will preserve the broadband spectrum and give sufficient amount of data for multifractal analysis.

CHAPTER V

IMPACT OF DENOISING ON MULTIFRACTAL CHARACTERISTICS OF ECG

5.1 Introduction

The previous chapter has shown the incomplete convergence of the Rényi dimension spectrum due to the low-sampling frequency and noise in the recorded ECG data. The low-sampling frequency limits the broadband spectrum characteristics of chaos in the signal, while noise breaks down the criterion that unfolds the chaos in the embedding space. Since the available ECG recordings at 360 sps cannot be changed, we address the second problem, the noise. Consequently, this chapter presents an investigation of the performance of denoising techniques and the influence of them on the Rényi dimension spectrum of ECG signals.

Before discussing denoising techniques, it is necessary to know what types of noise can be found in the ECG signal. Noise in the ECG signal may include: (i) the electromyogram (EMG) signal; (ii) relative movement between electrodes and skin; (iii) external electromagnetic field interference; (iv) component noise in data acquisition system; and (v) power supply interference. Often, more than one type of noise appears in the recorded ECG signal. Since such a noise has broadband characteristics, signal filtering cannot be used.

Traditional filtering techniques usually assume that the power spectra of the signal

and noise are separable and thus noise can be removed through a linear filter. It is true if the signal is dominated by periodic oscillations. The power spectrum of periodic signals shows sharp peaks at the harmonics of the oscillation frequency and any continuous component should originate from noise. Or if we know that the signal spectrum extends to an upper cutoff frequency, the noise with higher frequency can be removed by a linear low-pass filter. If the system is nonlinearly dynamical, it may go through a chaotic evolution which leads to a continuous spectrum distribution. In such a case, the noise spectrum could not be separated from the spectrum of the time series without affecting the signal. Even if we could identify the spectrum distribution of the noise, a spectral filter could not remove it effectively. Unfortunately, the ECG signal has characteristics of broadband spectrum. Thus, denoising techniques are necessary.

Since linear filters are not suitable to process the time series containing broadband noise, denoising has been developed within the last decade. Denoising means noise removal or reduction from a signal. Donoho and Johnstone proposed a soft-threshold denoising technique implemented in the wavelet domain [Dono92]. The main idea of this algorithm is that time-frequency features of the wavelet transform can concentrate the signal energy and separate the signal and noise through wavelet shrinkage. Carré *et al.* further developed this idea and denoised signals by an undecimated wavelet transform [CLFM98]. This wavelet denoising is applied to the ECG signal, as described in Sec. 5.2. Due to its limitations, we discuss in Sec. 5.3 a better denoising technique, chaos denoising in embedding space, as developed by Grassberger [GHKS93].

Linear filtering and denoising techniques are applied to an ideal ECG signal super-

imposed by coloured noise to evaluate the performance of the algorithms by examining the SNR gain. The impact of denoising on convergence of the Rényi dimension spectrum of the recorded ECG signal is also investigated. We consider denoising techniques as a lossless compression technique in the sense that the reconstruction error is at the same level as quantization error. Denoising suppresses the insignificant components in ECG and reduces the dynamic range of the signal. Therefore, the entropy of the denoised signal is lowered.

5.2 ECG Denoising in Wavelet Domain

Let us consider $x(n)$ as a generic time series. Let us further consider four other notations: (i) $x_u(n)$ as an uncontaminated (pure) signal, (ii) $x_c(n)$ as a contaminated (measured) signal, (iii) $x_d(n)$ as a noise reduced signal, and (iv) $x_e(n)$ as an estimated signal.

Suppose we wish to recover an unknown signal $x_u(n)$ from the measured ECG time series $x_c(n)$

$$x_c(n) = x_u(n) + \sigma\eta(n) \qquad n = 1, 2, \cdots, N \tag{5.1}$$

where $\eta(n)$ is an independent and identically distributed white Gaussian noise with zero mean and variance of 1, and σ is the noise level.

Let $x_e(n)$ be the estimate of $x_u(n)$. Donoho suggested that the goal of denoising is to optimize the following mean squared error, MSE,

$$MSE = \frac{1}{N}\sum_{n=1}^{N}[x_e(n) - x_u(n)]^2 \tag{5.2}$$

subject to the side condition that with high probability, the denoised signal, x_e, is at least as smooth as x_u with any of a wide range of smoothness measures [Dono92].

In this section, a denoising technique based on the wavelet transform is applied to the ECG signal. Multiresolution decomposition technique is used to implement the wavelet transform of the ECG signal. The details of the wavelet transform have been discussed in Chapter 3.

The essence of the wavelet transform is to find the similarity between the signal and the shifted and dilated mother wavelet. The wavelet coefficients represent how the signal is similar to the wavelets at various scales and locations. A large coefficient means that the signal is very similar to a certain wavelet at a certain location. The similarity also includes the smoothness of the signal. The smoothness is decided by the vanishing moment or regularity of the mother wavelet. The wavelet transform with orthogonal bases makes the signal concentrate on some parts. On the other hand, the white noise remains a white noise of the same magnitude and is uncorrelated on all scales. If the discrete wavelet transform (DWT) is applied to the composite signal expressed by (5.1), we get

$$DWT_{x_c}(i,n) = DWT_{x_u}(i,n) + \sigma DWT_{\eta}(i,n) \tag{5.3}$$

where $DWT_x(i,n)$ denotes the nth wavelet coefficient of x at scale i. The wavelet coefficient DWT_η of the white Gaussian noise $\eta(n)$ has the same distribution as the original noise [Dono92].

By applying the DWT to the smoothed versions of the signal iteratively, we get the multiresolution decomposition of the signal [Mall89]. Figure 3.3 shows a general decomposition procedure of the DWT for a signal from scale 1 to scale 4; *i.e.*, in different frequency bands. It may also be considered as the noise separation procedure of the noisy signal. Based on the separation property of wavelet transform, Donoho and Johnstone proposed two types of shrinkage functions, a new soft threshold and well-known hard threshold, to denoise a measured signal in wavelet domain [Dono92].

1) Soft threshold is defined as

$$d_{sj}(n) = \begin{cases} d_j(n) - T_d & d_j(n) > T_d \\ 0 & -T_d \le d_j(n) \le T_d \\ d_j(n) + T_d & d_j(n) < -T_d \end{cases} \tag{5.4}$$

2) Hard threshold is defined as

$$d_{hj}(n) = \begin{cases} d_j(n) & |d_j(n)| \ge T_d \\ 0 & |d_j(n)| < T_d \end{cases} \tag{5.5}$$

where T_d is an estimated threshold of noise and $d_j(n)$ is the detail of the signal at level j (refer to (3.20) and (3.21)). Donoho gave a universal threshold expressed by

$$T_d = \sigma\sqrt{2\log(N)} \text{ with } \sigma = Median(|d_1(n)|)/0.6745 \quad (5.6)$$

where the factor 0.6745 is chosen as a calibration for a Gaussian distribution. Since T_d is related to a good visual quality of reconstruction obtained by the "shrinkage" of wavelet coefficients, he named it VisuShrink. The noise dispersion is estimated on the first scale which mainly contains noise coefficients.

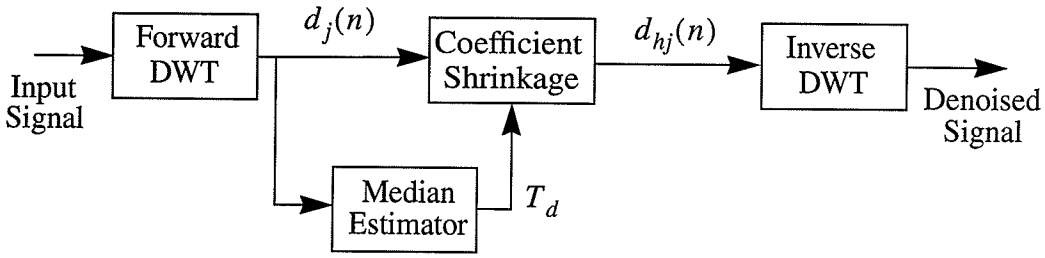

Fig. 5.1. Donoho's wavelet denoising scheme.

The Donoho's wavelet denoising scheme is illustrated in Fig. 5.1. The procedure includes:

1) Decompose the signal into various scales by the DWT;
2) Estimate noise threshold from the first detail of the decomposition;
3) Shrink the wavelet coefficients in each scale; and
4) Reconstruct the signal by the inverse DWT.

We use the Daubechies 4 (Daub4) as the mother wavelet, and the soft-thresholding technique to denoise an ECG signal. First, the ECG signal with 8192 sample points is decomposed into five scales by the multiresolution decomposition technique. Then, the soft threshold is estimated from the first scale of the transform. Then, the soft-thresholding

technique is applied to each scale to remove the noise. Figure. 5.2 shows the experimental result of wavelet denoising. It is noticed that the denoised output is smoother than the original signal, except in the QRS area, where the denoising algorithm has no obvious affect on the smoothness of the signal from a perceptual point of view. We will use the denoised ECG to calculate the Rényi dimension spectrum in Sec. 5.4.

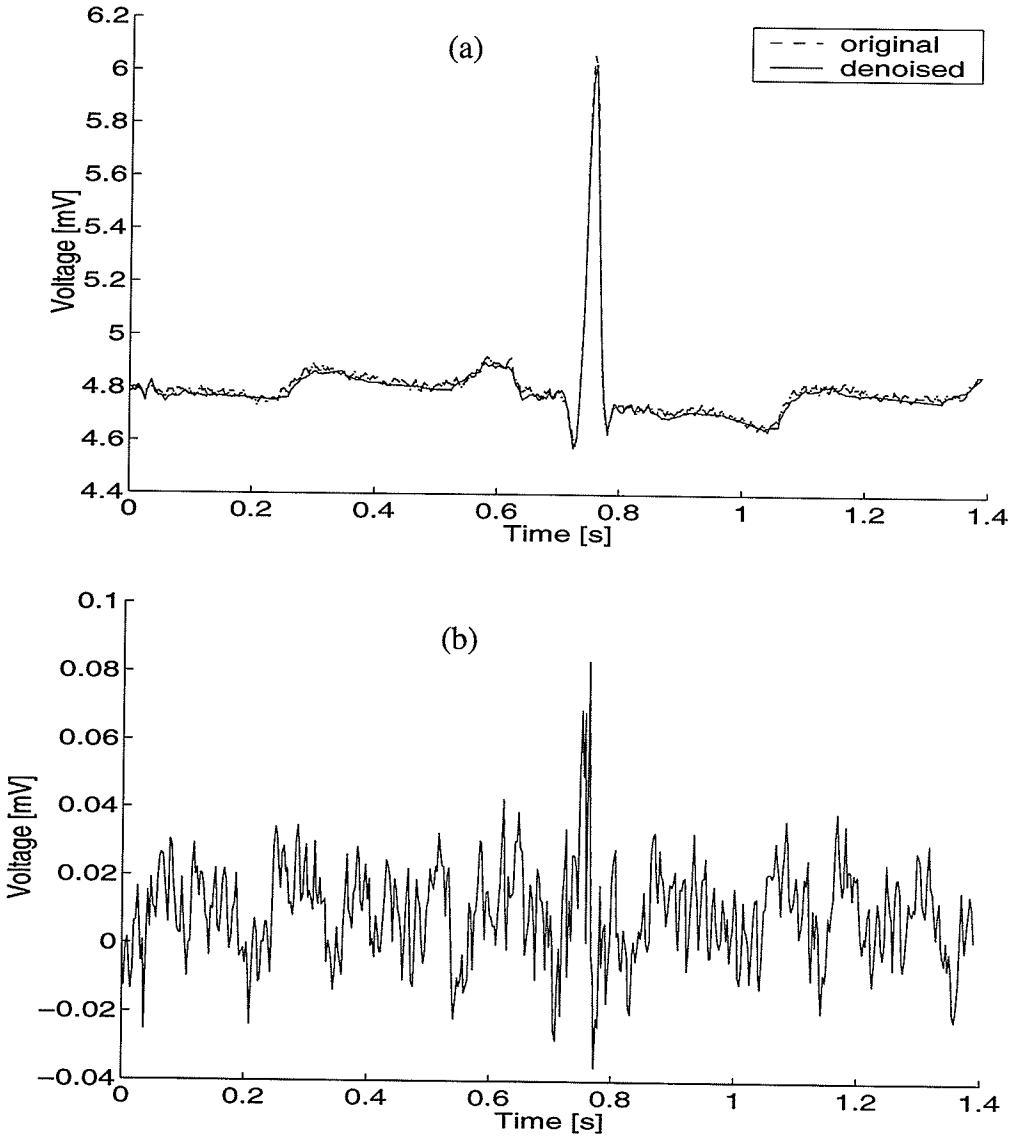

Fig. 5.2. (a) A real ECG and its denoised waveform through the DWT with the Daub 4. (b) The difference between the real signal and its denoised version.

Experiments with Donoho's wavelet denoising on ECG signals and images [Lang96] show that the separation of the noise from the signal is not complete, and the signal can be altered to some degree. Consequently, an alternative denoising technique has been developed, as described next.

5.3 ECG Chaos Denoising in Embedding Space

There is another kind of denoising technique, *chaos denoising*, which is completely different from wavelet denoising in alleviating some of its problems. Chaos denoising separates the signal and noise based on the observation that, in the embedding space, the chaotic signal is deterministic, while noise is not.

To denoise the signal, we use the knowledge developed in Chapter 4 to reconstruct the strange attractor of the signal and to obtain the trajectory in an m-dimensional embedding space. Unlike the scalar time series, the point of the trajectory in the phase space is a vector. In this phase space, principal components are computed in the small neighbouring subset of a given point that will be corrected. The assumption is that the clean signal "lives" in a smooth manifold with dimension $D < m$, where D is the Euclidean dimension and m is the embedding dimension, and that the variance of the noise is smaller than that of the signal. Then, for the noisy data, the large eigenvalues (*i.e.* principal components) correspond to the dominant directions of the attractor and small eigenvalues in all other directions. Therefore, we project the vector (the given point) under consideration onto the subspace related to the large eigenvectors to minimize the noise components. Thus, the fit of the assumed deterministic dynamics is local and linear, being contained implicitly in the construction of the linear subspace.

The procedure of chaos denoising technique is as follows:

1) Reconstruct the strange attractor for the time series $\{x_c(n), 1 \le n \le N\}$ according to (4.2) in an m-dimensional embedding space;
2) Around each given point $x_c(n)$ on the trajectory of the strange attractor, find the subset ω_n^r in a small neighbourhood of the point with radius r for which

$$max\left|x_c(j+k\tau) - x_c(n+k\tau)\right| < r\,, \quad k = 0, 1, \cdots, m-1 \text{ and } 1 \le j \le N \tag{5.7}$$

where τ is the time lag in the embedding space and r is related to the amplitude of the noise;
3) Compute the principal components of the subset;
4) Project the given point from the m-dimensional space to the D-dimensional space, where $D < m$; and
5) Repeat from Step 1 to Step 4 for the smaller r until r is almost zero (where r approaching to zero means that almost no noise can be removed from the signal anymore).

The realization of the above procedure is time-consuming. Schreiber [Schr93] proposed an approximate and fast noise reduction technique in embedding space. This technique simply replaces the present coordinate $x_c(n)$ by the mean value of the central coordinates of the neighbourhood,

$$x_d(n) = \frac{1}{N_n^r} \sum_{j \in \omega_n^r} x_c\left(j + \left\lfloor \frac{m}{2} \right\rfloor \tau\right) \tag{5.8}$$

where N_n^r is the number of points in the neighbourhood. The central coordinate, $x_d(n)$, in the delayed window is taken as the correction because it is controlled optimally from the past to the future on the trajectory.

Grassberger *et al.* suggested a *principal component denoising* (PCD) technique based on chaos theory [GHKS93]. The PCD computes the principal components of the neighbourhood of each given point and then projects the given point to some important components of the phase space to denoise signals. It achieves better performance than Schreiber's technique.

To denoise a given point $x_c(n)$ with the subset ω_n^r, the PCD first calculates the centre of ω_n^r

$$\mu(i) = \frac{1}{N_n^r} \sum_{j \in \omega_n^r} x_c(j + i\tau) \qquad i = 0, \cdots, m-1 \tag{5.9}$$

and the $m \times m$ covariance matrix

$$\sigma(i, j) = \frac{1}{N_n^r} \sum_{k \in \omega_n^r} x_c(k + i\tau) x_c(k + j\tau) - \mu(i)\mu(j) \tag{5.10}$$

with $i = 0, \cdots, m-1$ and $j = 0, \cdots, m-1$

The N_v orthonormal eigenvectors of the matrix σ with smaller eigenvalues are called $v_e(k)$, $k = 0, \cdots, N_v$. Then, the projector onto the subspace spanned by these vectors is

$$\varepsilon(i, j) = \sum_{k=0}^{N_v} v_e(k, i) v_e(k, j) \tag{5.11}$$

Finally, the *i*th component of the correction is given by

$$x_d(n, i) = x_c(n + i\tau) + \sum_{j=0}^{m-1} \varepsilon(i, j)(\mu(j) - x_c(n + j\tau)) \quad i = 0, \cdots, m-1 \tag{5.12}$$

Such a correction is done for each component of every point (vector) along the trajectory, such that we get a corrected trajectory in the embedding space. Since each element of the scalar time series occurs in m different points on the trajectory, we finally obtain m different suggested corrections for each element, of which we take the average of the corresponding corrections to restore the time series. Therefore, the corrected vectors do not lie on the correct trajectory precisely but only move toward it in embedding space. The procedure can be iterated to improve the performance of denoising.

Unlike wavelet denoising, chaos denoising needs to set the noise threshold manually. Figure 5.3(a) shows a recorded ECG signal denoised by the PCD technique with the maximal threshold of 0.04 mV. An embedding dimension of 20 and a time lag of 4 are used for the reconstruction of the strange attractor. The maximal number of points in the neighbourhood is limited to 300. Like wavelet denoising, chaos denoising makes the signal smoother, except in the region of the QRS complex. However, the denoised signal by chaos denoising follows the original signal better than that by wavelet denoising, as shown in Fig. 5.3(b). The effect of chaos denoising will also be investigated through the convergence improvement of the Rényi dimension spectrum of the ECG signal.

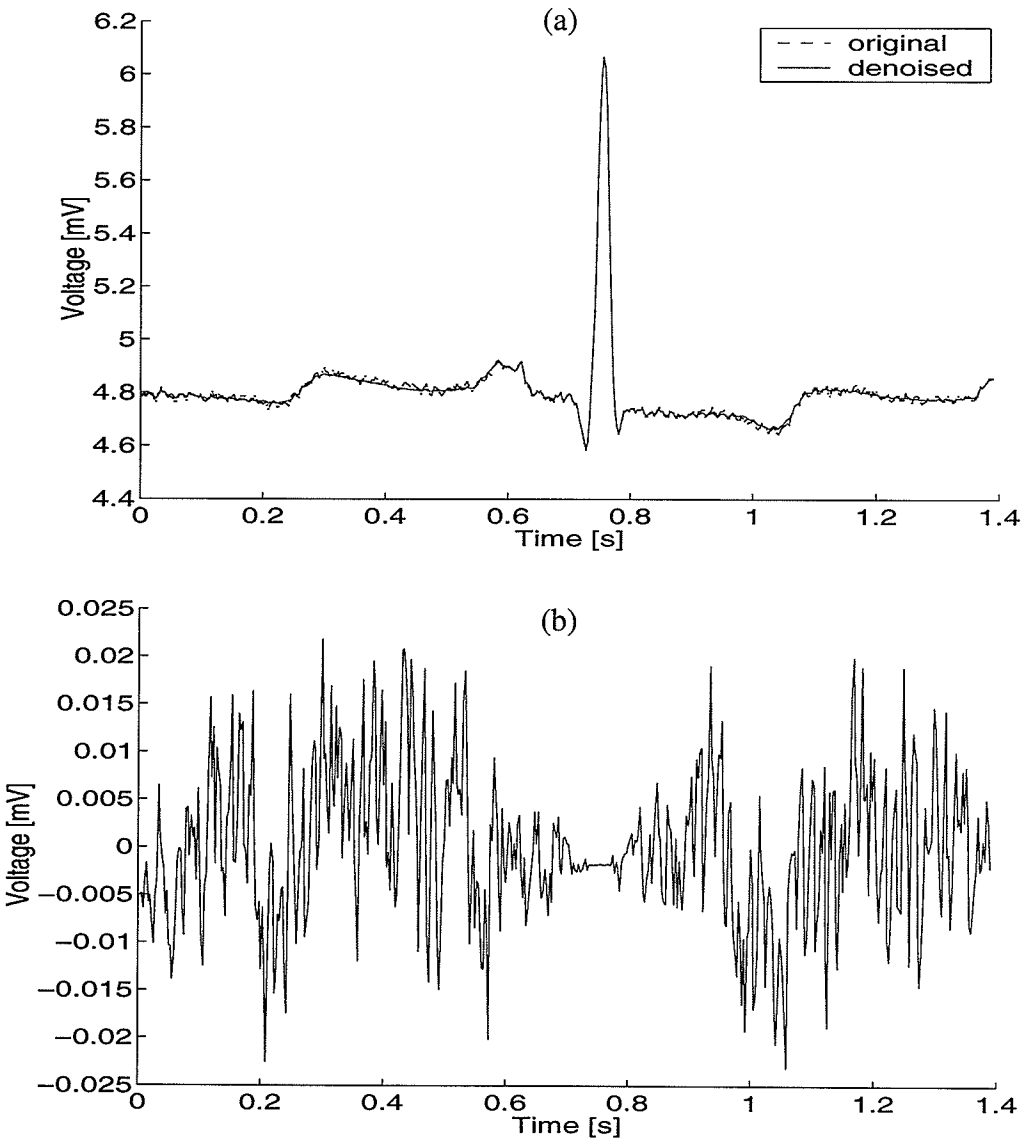

Fig. 5.3. (a) A real ECG and its denoised waveform through chaos denoising. (b) The difference between the real signal and its denoised version.

5.4 Performance Evaluation of Denoising Techniques

In this section, the pure ECG signal contaminated by coloured noise and the recorded ECG signal are used to investigate the performance of denoising techniques. The purpose of denoising the contaminated signal is to: (i) verify correctness of the algorithms,

and (ii) show quantitatively that denoising techniques are more effective than the linear filter technique for the ECG signal. Then, denoising techniques are applied to the recorded ECG signal to improve the convergence of the Rényi dimension spectrum, which is related to the reconstruction of the strange attractor of the signal. Here the ideal signal refers to the computed ECG signal discussed in Sec. 2.4. The recorded ECG signal is taken from the MIT-BIH ECG database [Mood99].

5.4.1 Denoising the ECG Corrupted with Coloured Noise

The ideal signal $x_u(n)$ is considered as the signal without noise $\eta(n)$. Coloured noise is superimposed on the ideal ECG signal, which can be used for quantitative analysis in signal denoising experiments. Denoising techniques are applied to the contaminated signal to evaluate the performance of the techniques by a *signal-to-noise ratio* (SNR) gain and a noise reduction factor.

Before implementing various denoising algorithms, let us discuss coloured noise. Noise is divided into different classes according to its power spectrum density, $1/f^{\beta}$. The noise can be classified as: (i) white, with a flat power spectrum $1/f^0 = 1$, (ii) pink, with $\beta = 1$, (iii) brown, with $\beta = 2$, and (iv) black, with $\beta \geq 3$. In general, the exponent β does not have to be an integer, thus leading to fractional noise [Kins94c].

Coloured noise is used in the experiments because there are at least two kinds of noise in the ECG signal. Noise coming from the data acquisition system is probably white. The interference from other bioelectrical signals is predominantly a pink noise, though other higher correlated noise can also be found. Therefore, we include the above men-

tioned four classes of coloured noise in the experiments. Figure 5.4 shows the four type coloured noise. The coloured noise has been generated by a spectral technique [PeJS92] [Kins94c] in which (i) white noise is generated in the time domain, (ii) Fourier transform of the white noise produces random amplitude and phase, (iii) filtering is performed in the frequency domain on the amplitude to achieve the desired slope $1/f^{\beta}$ of the power spectrum density, and (iv) the inverse Fourier transform produces the corresponding noise. The code is provided in Appendix B.2. It is seen that the predictability of coloured noise increases from white noise to black noise gradually.

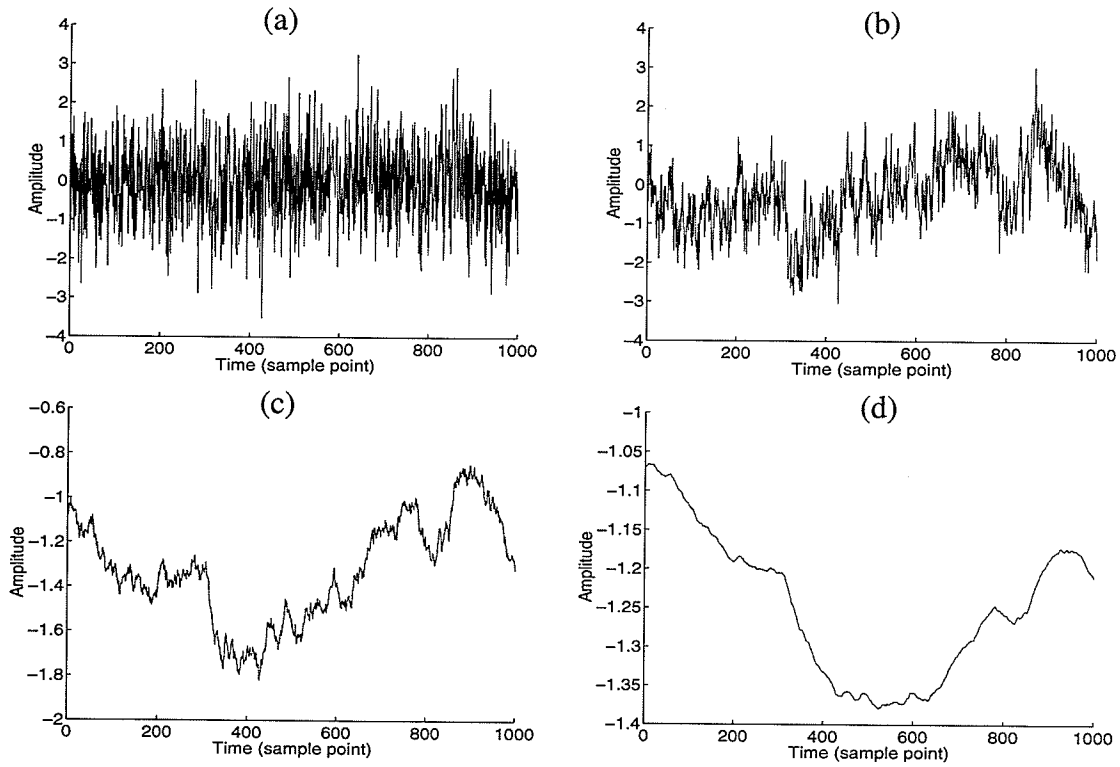

Fig. 5.4. Examples of coloured noise: (a) white, (b) pink, (c) brown, and (d) black.

The pure ECG signal is shown in Fig. 5.5. To investigate the denoising techniques, coloured noise with 5 dB, 10 dB, and 20 dB SNR has been superimposed with the pure signal, respectively. Figure 5.6 shows the contaminated ECG signal by coloured noise

with 5 dB SNR.

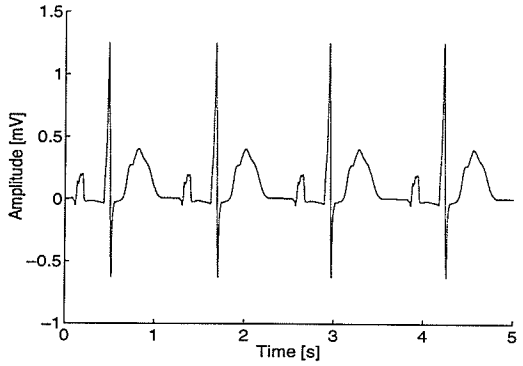

Fig. 5.5. A pure ECG signal.

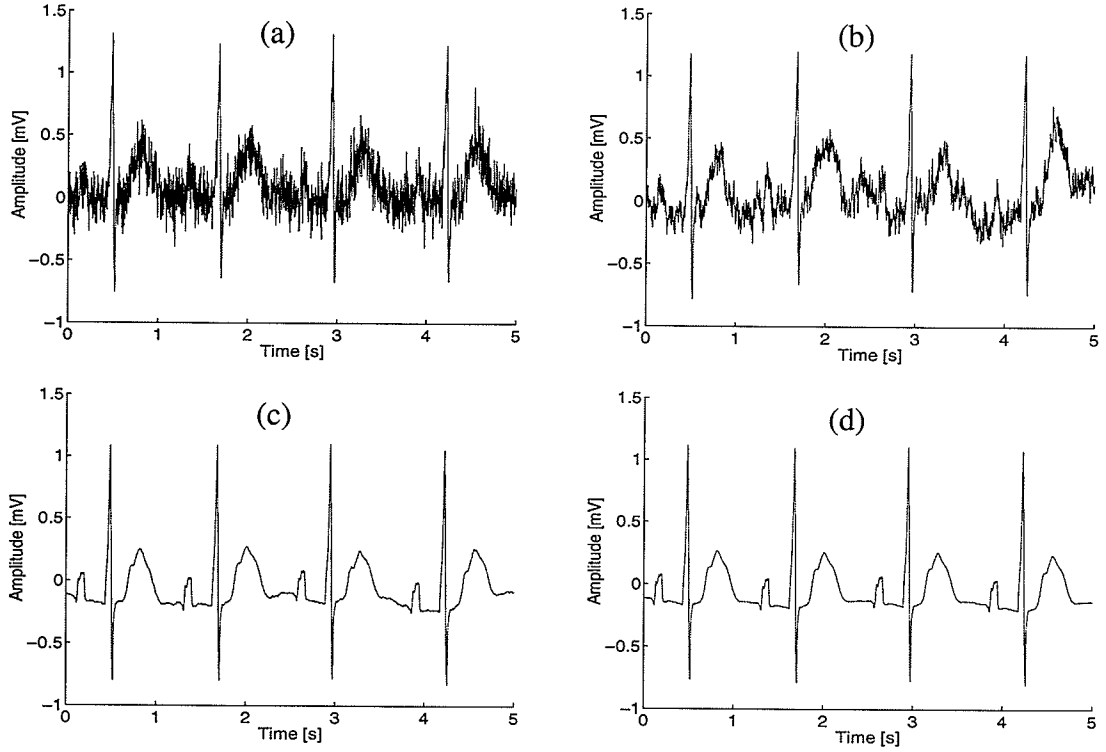

Fig. 5.6. A pure ECG signal is superposed by coloured noise with 5 dB SNR: (a) white, (b) pink, (c) brown, and (d) black.

To reduce noise, wavelet denoising and chaos denoising are used. In addition, a linear lowpass filter is applied to the contaminated ECG signal as a reference for denoising

techniques. The code is also provided in Appendix B.2.

The *noise reduction factor* (NRF) and the *SNR gain* ($\mathrm{SNR_G}$) are used to evaluate the performance of the various algorithms for the ECG signal. The NRF is defined as [ScKa96]

$$NRF = \sqrt{\frac{\sum_{j=1}^{N} [x_c(j) - x_u(j)]^2}{\sum_{j=1}^{N} [x_d(j) - x_u(j)]^2}} \tag{5.13}$$

where x_u is the pure signal, x_c is the contaminated signal, x_d is the noise-reduced signal, and N is the length of the signal. If $NRF > 1$, the quality of the signal is improved. Otherwise, the signal is degraded for $NRF < 1$.

The SNR gain is defined as

$$SNR_G = SNR_r - SNR_c \tag{5.14}$$

where $\mathrm{SNR_c}$ is the SNR of the contaminated ECG signal, and $\mathrm{SNR_r}$ is the SNR of the ECG signal after noise reduction. $SNR_G > 0$ means improvement of the signal quality. The signal is degraded if $SNR_G < 0$.

To evaluate denoising techniques, a linear filter is applied to the contaminated ECG to give a reference for performance comparison. Since a Wiener lowpass filter is optimal under a minimal squared error sense and quite stable, such a filter with 53-order and a cutoff frequency of 180 Hz is used in the experiment to filter coloured noise from the

ECG signal with 5 dB, 10 dB, and 20 dB SNR.

Experimental results are listed in Table 5.1. It is seen that the linear filter is effective for white noise, especially for the signal with low SNR (≤ 10dB). This filter has almost no improvement for pink noise and degrades the signal with brown noise and black noise. It can be explained from the spectrum distribution of the ECG signal. The ECG signal has a spectrum distribution similar to that of the pink noise. Therefore, the lowpass filter may remove some white noise from the signal since the energy of white noise is distributed over the spectrum domain evenly. On the other hand, since the signal and pink noise have similar spectrum distributions, the linear filter cannot filter such a noise. The energy of brown and black noise is spread over the spectrum of the ECG signal. If the lowpass filter is used to remove such noise, it will also affect the signal. This experiment shows the limitation of linear filtering when the spectra of the signal and noise overlap.

Table 5.1: Performance of the Wiener lowpass filter with 53-order and 180 Hz bandwidth applied on the ECG signal contaminated by coloured noise.

SNR	White Noise		Pink Noise		Brown Noise		Black Noise	
	SNR_G	NRF	SNR_G	NRF	SNR_G	NRF	SNR_G	NRF
5 dB	6.9 dB	2.2	0.73 dB	1.1	-0.12 dB	0.98	-0.14 dB	0.98
10 dB	6.3 dB	2.0	0.58 dB	1.1	-0.25 dB	0.96	-0.28 dB	0.96
20 dB	1.9 dB	1.2	-1.7 dB	0.88	-1.7 dB	0.81	-1.8 dB	0.81

Unlike the linear filter, denoising techniques do not reduce noise based on the spectrum distribution. Wavelet denoising is based on the smoothness of the signal, and the correlation between the signal and wavelets to remove noise. Table 5.2 gives experimental

results for wavelet denoising on the ECG signal which contains the same noise as the above experiment. Here the used mother wavelet is a biorthogonal wavelet [CoDF92] since it achieves better performance than other mother wavelets. The signal is decomposed into four scales. A hard-thresholding technique is used in the denoising. The table shows that the effect of hard-threshold wavelet denoising is similar to that of the linear filter for different kinds of noise, except for two points: (i) the performance of this denoising technique is better than that of the Wiener filter for white noise, and (ii) this wavelet denoising does not degrade the signal when reducing noise according to the measurement of the SNR_G and the NRF.

Performance of wavelet denoising increases when noise changes from black, brown, pink to white. Mother wavelets with various vanishing moments are also used to test the algorithm. It was found that the higher the order of the vanishing moment, the better the performance of the denoising.

Table 5.2: Performance of wavelet denoising on the ECG signal contaminated by coloured noise. The mother wavelet is bior6.8. Hard threshold is used in the denoising.

SNR	White Noise		Pink Noise		Brown Noise		Black Noise	
	SNR_G	NRF	SNR_G	NRF	SNR_G	NRF	SNR_G	NRF
5 dB	7.7 dB	2.4	1.1 dB	1.1	0.026 dB	1.0	0.022 dB	1.0
10 dB	6.9 dB	2.2	0.98 dB	1.1	0.025 dB	1.0	0.021 dB	1.0
20 dB	3.7 dB	1.5	0.72 dB	1.1	0.015 dB	1.0	0.013 dB	1.0

In addition to Daubechies wavelets, we have also tried other mother wavelets including Coiflets [Daub88] [Math01], discrete Meyer [Meye86] [Math01], biorthogonal

[CoDF92] [Math01], and reverse biorthogonal [Math01] wavelets for wavelet denoising. The performance improvement between the mother wavelets is not significant. Biorthogonal mother wavelet achieves somewhat higher performance than the others.

On the other hand, chaos denoising makes use of the determinacy of the strange attractor of the signal in the phase space to reduce noise. Table 5.3 shows experimental results by applying the PCD technique to the ECG signal contaminated by coloured noise. Here, the ECG signal is the same as the linear filtering experiment used. In the experiment, embedding dimension is set to 20, time lag is 4, and the maximal number of neighbour points is limited to 300. From these experimental results, chaos denoising is effective for all the four kinds of coloured noise, although its performance increases when noise changes from black, brown, pink to white.

Table 5.3: Performance of chaos denoising on the ECG signal contaminated by coloured noise. Embedding dimension is set to 20, time lag is 4, and the maximal number of points in the neighbourhood is limited to 300.

SNR	White Noise		Pink Noise		Brown Noise		Black Noise	
	SNR_G	NRF	SNR_G	NRF	SNR_G	NRF	SNR_G	NRF
5 dB	9.1 dB	2.9	3.5 dB	1.5	0.87 dB	1.1	0.73 dB	1.1
10 dB	10 dB	3.3	4.9 dB	1.8	3.8 dB	1.5	3.6 dB	1.5
20 dB	11 dB	3.5	8.7 dB	2.7	6.5 dB	2.1	6.1 dB	2.0

Compared to Tables 5.1 and 5.2, Table 5.3 shows that chaos denoising achieves the best performance of the three techniques. The same conclusion is made from Fig. 5.7. As Fig. 5.7(c) shows, the waveform after chaos denoising is very close to the original ECG signal. Figure 5.7(d) demonstrates noise at the TP region of the signal after wavelet

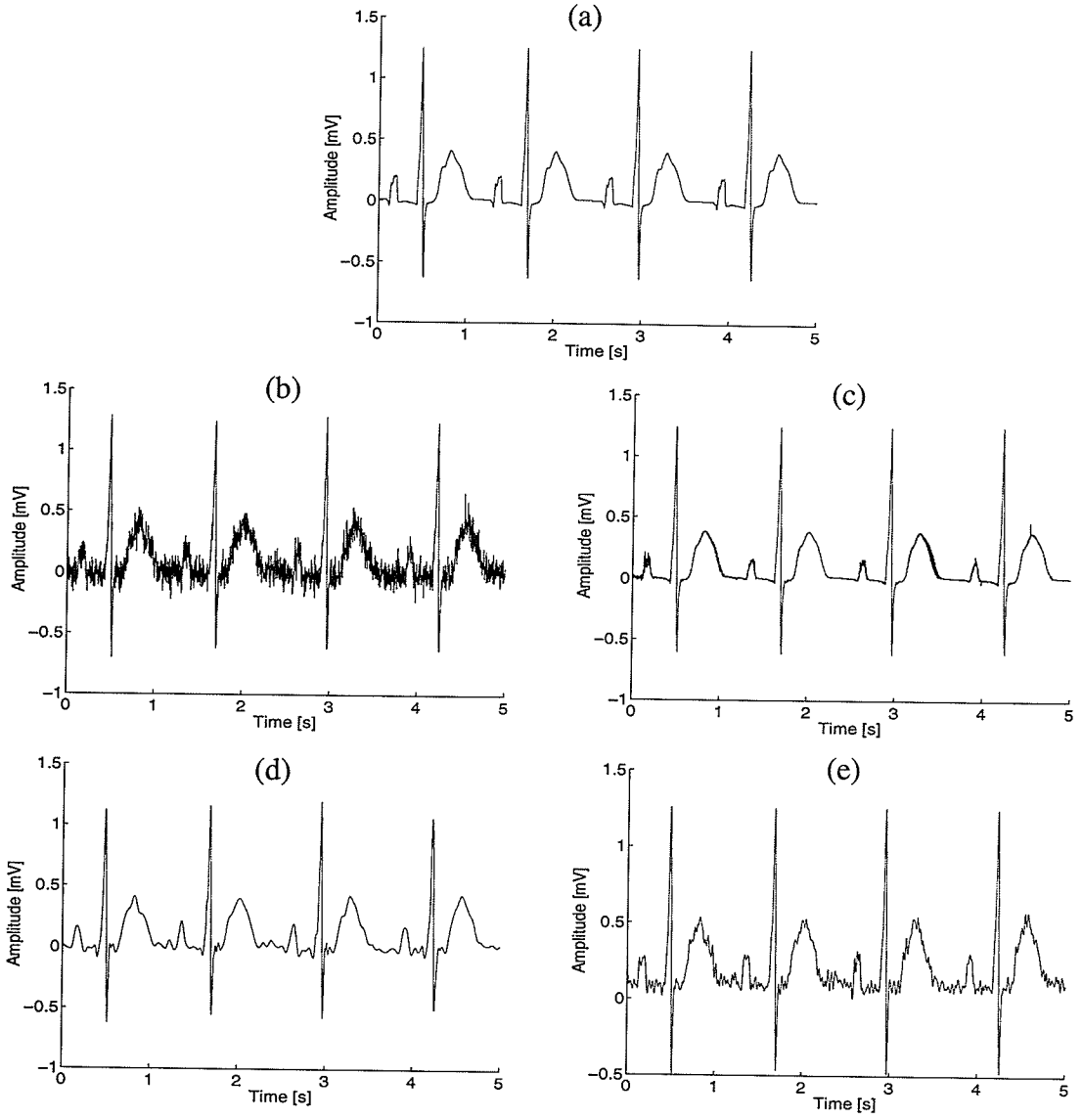

Fig. 5.7. Examples of various noise reduction techniques applied to the ECG signal. (a) a pure ECG signal, (b) the ECG signal with 10 dB white noise, (c) chaos denoising, (d) wavelet denoising, and (e) Wiener filtering.

denoising. It can be seen in Fig. 7.5(e) that noise after Wiener filtering is more obvious.

Unlike wavelet denoising and linear filtering, the performance of chaos denoising increases with SNR of the signal. It is explained that the reconstruction of the strange attractor is more precise in low noise environment. Thus, the signal and noise can be sepa-

rated better.

5.4.2 Impact of Denoising on Convergence of Rényi Dimension Spectrum

The main purpose of denoising in this chapter is to investigate the impact of ECG denoising on its Rényi dimension spectrum. The Rényi dimension spectrum is a complexity measure of the strange attractor of the object in the phase space [Kins94a]. With the increase of the embedding dimension, the strange attractor is unfolded in the phase space and becomes deterministic if the embedding dimension exceeds the dimension of the attractor. Then the Rényi dimension spectrum will not change any more with the increase of the embedding dimension. However, noise is nondeterministic in the phase space. The convergence of the Rényi dimension spectrum will be affected if the signal contains noise. Signal denoising is required to remove noise and keep the chaotic component unaffected. Therefore, the convergence of the Rényi dimension spectrum of the ECG should be an important indicator to evaluate the performance of denoising techniques in the reconstruction of the strange attractor.

Experiments have been performed to calculate the Rényi dimension spectrum by using three kinds of ECG signals: (i) the recorded ECG signal, (ii) the ECG signal after wavelet denoising, and (iii) the ECG signal after chaos denoising. Figure 5.8(a) shows the Rényi dimension spectrum of the ECG signal with embedding dimension of 1 to 13 under no denoising, Donoho's soft-threshold denoising with the Daub 4 (Fig. 5.8(b)), and the PCD denoising with lag of 4 and embedding dimension of 20 (Fig. 5.8(c)). Figure 5.8(a) shows the convergence of the Rényi dimension spectrum when $m \geq 4$ and $q < 0$ for the original ECG signal. However, it is not convergent when $q > 0$. Figure 5.8(b) gives the

Rényi dimension spectrum for the signal with wavelet denoising. There is no convergence improvement for $q > 0$. When $q < 0$, wavelet denoising degrades the convergence of the Rényi dimension spectrum. An improvement of the convergence is observed in Fig. 5.8(c) for both $q < 0$ and $q > 0$, compared to Fig. 5.8(a).

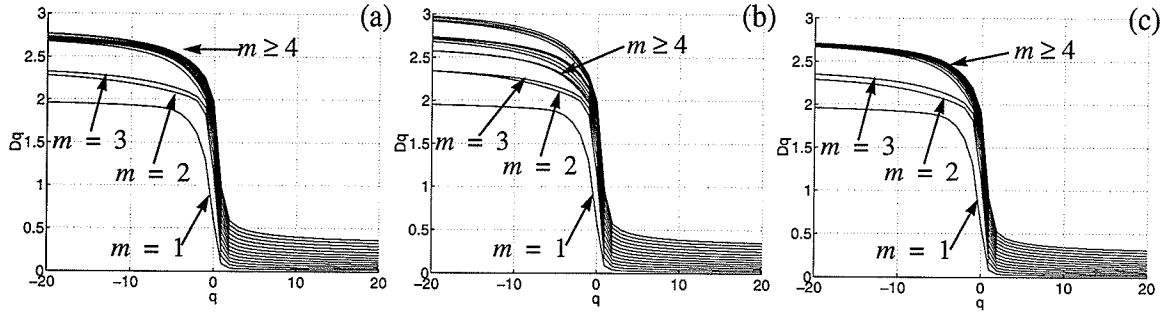

Fig. 5.8. The Rényi dimension spectrum of the ECG changing with embedding dimension m by (a) no denoising, (b) Donoho's soft-threshold denoising with the Daub 4, and (c) the PCD denoising with time lag of $\tau = 4$ and embedding dimension of $m = 20$.

To investigate the improvement of the convergence quantitatively, we propose a *mean absolute difference* (MAD) between the Rényi dimension spectra with embedding dimensions m, D_q^m, and $m-1$, D_q^{m-1}, to measure the level of convergence. A small MAD indicates a good convergence. The MAD is defined as

$$MAD = \frac{1}{N_R} \sum_{q=-Q}^{Q} \left| D_q^m - D_q^{m-1} \right| \tag{5.16}$$

where Q is the maximum range of q considered in the calculation of the Rényi dimension spectrum (here $Q = 20$), and N_R is the number of points considered in the calculation ($N_R = 2Q+1$).

Table 5.4: The MAD of the Rényi dimension spectrum of the ECG signal.

$m = 13$	No Denoising	Wavelet Denoising	Chaos Denoising
MAD	2.49×10^{-4}	4.46×10^{-4}	5.73×10^{-5}

Table 5.4 gives the MAD of the Rényi dimension spectrum of the ECG signal under different situations. We see that wavelet denoising degrades the convergence about two times. The reason is that the threshold technique affects the ECG signal components when it reduces noise. On the other hand, chaos denoising improves the convergence about five times. This technique only discards the noise components in the phase space. However, it is still inadequate to impart sufficient convergence for the multifractal extraction of features from the signal.

5.4.3 Other Techniques for Denoising Evaluation

In addition to the SNR gain and the convergence of the Rényi dimension spectrum, there are also some other indicators to evaluate the performance of denoising techniques when applied to nonstationary signals [KoSc93] [Kins95].

1) The denoised signal $x_d(n)$ is more predictable. This means that one sample has relationships with its neighbouring samples and they are not independent.
2) The power spectrum of $x_d(n)$ is still similar to that of the original signal, at least in the frequency range and spectrum shape. This means that there is no frequency filtering taking place, as in the traditional filtering. Furthermore, this ensures that fractal dimensions of signals are preserved.

3) The $x_d(n)$ keeps the same autocorrelation as the original signal. On the other hand, the autocorrelation of the removed signal should be low. The cross-correlation between the removed signal and the $x_d(n)$ should also be low. This is a logical consequence of removing additive noise.
4) The removed signal should behave like a random process whose statistics match those of the assumed noise model.
5) The multifractal measure of the removed signal represents a flat line, which approaches the dimension of the noise.

5.5 Summary

Chapter 4 has revealed the incomplete convergence of the Rényi dimension spectrum of the ECG signal for $q > 0$. One of the reasons was that the sampling frequency was insufficient to preserve the chaotic behaviour of the signal. The other reason was that the noise affects the reconstruction of the ECG attractor. We cannot use linear filtering techniques to remove noise from the ECG signal because the signal has broadband characteristics. Consequently, this chapter described alternative denoising techniques applied to the ECG signal. Denoising reduces noise in the signal.

One denoising technique is Donoho's wavelet shrinkage. In this technique, a wavelet transform is applied to the ECG signal. The wavelet coefficients represent the correlation between the signal and the wavelets. They also contain the smoothness information of the signal. The coefficients of highly irregular noise are low. Based on this idea, the threshold technique is applied to the wavelet coefficients of the ECG signal for noise reduction.

Another denoising technique is chaos denoising. A chaotic signal is deterministic in the phase space, while noise is not. It means that we may recognize noise through the attractor reconstruction in the phase space. Noise is reduced by principal component transform.

A linear filter and denoising techniques have been applied to an ideal ECG signal superimposed with coloured noise to evaluate the performance of the algorithms by examining the SNR gain. Chaos denoising achieves the highest SNR gain out of the three techniques. The PCD is effective on the pink, brown, and black noise, while the other two techniques are not. We have also implemented experiments by applying wavelet denoising and chaos denoising to the recorded ECG signal. The denoising effect is investigated through the convergence improvement of the Rényi dimension spectrum of the ECG. The convergence degree has been evaluated through the MAD, under the conditions of no denoising, wavelet denoising, and chaos denoising. It was found that chaos denoising improves the convergence degree about five times under the MAD, while wavelet denoising degrades it about two times.

CHAPTER VI

SIGNAL COMPRESSION BY IFS AND DOMAIN BLOCK PARTITIONING

6.1 Introduction

Chapter 4 presented fractal and multifractal measures of complicated objects. Based on self-similarity or self-affinity of fractal objects, a new signal compression technique was developed about 15 years ago [Barn88]. More specifically, the *iterated function systems* (IFS) approach to signal compression has been developed based on the fundamental property of fractal objects. Thus, our discussion begins with a description of a fractal signal, as this approach will be used in a new compression of ECG signals.

Definition 6.1: A random discrete process $x(n)$ defined for all integers, $-\infty < n < \infty$, is said to be statistically self-similar if its statistics are invariant to its dilations and contractions. Thus, $x(n)$ is statistically self-similar with a parameter D_F if for any real $\alpha > 0$, it obeys the following scaling relation [Worn96]

$$x(n) \overset{p}{=} \alpha^{-D_F} x(\alpha n) \tag{6.1}$$

where $\overset{p}{=}$ denotes equality in a statistical sense.

To employ the multiplicative (rather than additive) invariance of the dilation and contraction of the fractal signal, Barnsley proposed a fractal compression technique, the IFS, which uses an affine transform to map the process or signal onto itself [Barn88].

The IFS compression technique partitions the signal into domain blocks and range blocks, and finds a set of affine transforms between the domain blocks and range blocks. Computational complexity, compression ratio, reconstruction quality of the signal, and convergence are important issues in the IFS approach. All these issues are related to the quality of the matching between the blocks which results in the affine transform.

Barnsley's IFS uses a linear affine transform. A major problem with a linear affine transform is that its matching capability is very limited for signals such as image, speech, and the ECG, which have nonlinear characteristics. Another problem is that an exhaustive and time-consuming search in the domain pool is required to find an optimal matching. Consequently, we propose to solve the matching problem by finding (i) a more flexible transform and (ii) a more effective search scheme with low computational complexity.

From the above description, one sees that fractal coding is different from vector quantization and transform coding. Fractal coding tries to find a function expression for a data set and is a lossy data compression technique. There is no codebook or basis function in fractal coding.

Since the theory behind the IFS is quite extensive, it is described in Appendix A.2.

6.2 The Collage Coding

A fractal object described by (6.1) has statistical invariance to its dilations and contractions. Such invariance means that a small part of the fractal object can be similar to another small part of the object, under the affine mapping relationship. If we can find such relationships for every part of the object, a union collage of the mappings consists of a

transform from the fractal object to itself.

Fractal compression techniques are based on a corollary of contractive transform theory called the *collage theorem*, which was proposed by Barnsley [Barn88]. The collage theorem tells us that to find an IFS whose attractor is close to a given object, one must try to find a set of transforms, contraction mappings on a suitable space within which the given object lies, such that the union, or collage, of the given object under the transforms is close to the original object. The degree to which two objects look alike is measured using the Euclidean metric or Hausdorff metric.

Fractal coding maps a fractal object to itself. The primary difficulty of fractal coding associated with the collage theorem is what kind of mapping functions should be used. Barnsley again proposed the IFS, which makes the collage theorem of practical significance in signal compression [Barn88].

6.2.1 Iterated Function Systems

Originally, the central goal of fractal compression was to find resolution independent models, defined by finite length (and hopefully short) strings of zeros and ones, for real world images. One can achieve fractal image compression via the IFS compression algorithm, which is an interactive image modelling method based on the collage theorem [BJMR88].

"Iterated function systems" is the name introduced by Barnsley and Demko to denote a system of contraction mappings in a complete metric space [BaDe85]. The idea was developed earlier by Hutchinson who shows how typically self-similar fractal sets can

be generated by the "parallel" action of such systems of contraction mappings [Hutc81]. The IFS maps (IFSM) plus a set of associated probabilities (IFSP) define operators which act on probability measures. As a result, early IFS research work focused on the representation of image by measures and the approximation of these IFSP measures. However, it is more convenient to represent spatial and temporal signals by the functions generated by iterating an IFS-type operator. The IFSM is an example of the IFS or fractal affine transform method over an appropriate space of function which represents signals. The IFS is defined as [BaHu85]

Definition 6.2: A (hyperbolic) iterated function system consists of a complete metric space $(\mathbf{X}, d)$ together with a finite set of contraction mappings $w_i: \mathbf{X} \rightarrow \mathbf{X}$, for $i = 1, 2, \cdots, N$ such that

$$d(w_i(B), w_i(C)) \leq s_i d(B, C), \quad \forall (B, C) \in \mathbf{X}^2 \tag{6.2}$$

with respective contractivity factors $0 \leq s_i < 1$, where d is a metric on a space $\mathbf{X}$, as described in Appendix A.2.

The notation for this IFS is $\{\mathbf{X}; w_i, i = 1, 2, \cdots, N\}$ and its contractivity factor is $s = max\{s_i: i = 1, 2, \cdots, N\}$. The contractivity requirement of the affine mappings in the IFS is to ensure convergence of the IFS, which makes the IFS useful. Based on the above definition, we have the following theorem [BaHu93]

Theorem 6.1 (The IFS Theorem): Let $\{\mathbf{X}; w_i, i = 1, 2, \cdots, N\}$ be a hyperbolic iterated function system with contractivity factor s. Then the transform $\boldsymbol{W}$:

$H(\mathbf{X}) \rightarrow H(\mathbf{X})$ defined by

$$\boldsymbol{W}(B) = \bigcup_{i=1}^{N} w_i(B), \quad \forall B \in H(\mathbf{X}) \tag{6.3}$$

is a contraction mapping on the complete metric space $(H(\mathbf{X}), d_h)$ with contractivity factor s, that is

$$d_h(\boldsymbol{W}(B), \boldsymbol{W}(C)) \leq s d_h(B, C), \quad \forall (B, C) \in H^2(\mathbf{X}) \tag{6.4}$$

where $H(\mathbf{X})$ and d_h are the Hausdorff space and the Hausdorff distance, respectively, and are defined in Appendix A.2.

It should be observed that $\boldsymbol{W}$ has a unique fixed point, $A \in H(\mathbf{X})$, which obeys

$$A = \boldsymbol{W}(A) = \bigcup_{i=1}^{N} w_i(A) \tag{6.5}$$

where the A is given by

$$A = \lim_{n \rightarrow \infty} \overbrace{\boldsymbol{W}(W(\cdots W(B)\cdots))}^{n}$$

$$= \lim_{n \rightarrow \infty} \boldsymbol{W}^{\circ n}(B), \quad \forall B \in H(\mathbf{X}) \tag{6.6}$$

The fixed point $A \in H(\mathbf{X})$ described in the IFS theorem is called the attractor of the IFS. According to the IFS theorem, the sequence of a fractal object generated by

iteratively affine transforms is convergent. No matter what the initial set is, the sequence always converges to the same set, the attractor.

6.2.2 The Collage Theorem

Although the IFS gives a method of how to find the attractor of an object, it does not show how to find the affine transform set $\boldsymbol{W}$ from a given object. For a given object, finding the $\boldsymbol{W}$ is called the inverse problem of the IFS. The collage theorem provides a way of assessing how well the attractor of an IFS approximates a given object, which indicates a direction to seek the $\boldsymbol{W}$.

Theorem 6.2 (The Collage Theorem): Let (**X**, d) be a complete metric space. Let $\varepsilon \geq 0$ be a given scalar quantity. Choose an IFS $\{\mathbf{X}; w_i, i = 1, 2, \cdots, N\}$ with contractivity factor $0 \leq s < 1$, such that

$$d_h\left(B, \bigcup_{i=1}^{N} w_i(B)\right) \leq \varepsilon, \; B \in H(\mathbf{X}) \tag{6.7}$$

where $d_h(\bullet, \bullet)$ is the Hausdorff metric. Then

$$d_h(B, A) \leq \frac{\varepsilon}{1-s}$$

or

$$d_h(B, A) \leq \frac{d_h\left(B, \bigcup_{i=1}^{N} w_i(B)\right)}{1 - s} \tag{6.8}$$

where A is the attractor of the IFS. A proof for the collage theorem is given in [Barn88].

The collage theorem tells us how to find the required IFS for a given fractal object. To find the IFS whose attractor is close to a given object, one must try to find a set of transforms; *i.e.* contraction mappings on a suitable space within which the given object lies, such that the union, or collage, of the given object under the transforms is close to the original given object. The degree to which two fractal objects look alike may be measured by the Hausdorff metric or Euclidean metric.

6.3 IFS Model of Time Sequence

Barnsley's collage coding is very time consuming to solve the inverse problem. Its computational complexity for an image of $N \times N$ size is $O(N^6)$ [Barn88]. Many researchers have proposed methods to reduce the computational complexity. One such a technique was proposed by Jacqin [Jacq92] for grey scale images and is referred to as *fractal block coding* (FBC). The FBC splits an image into two kinds of small blocks. One kind is the subimage we want to encode, and is called range block. The other is the sample blocks from which we try to find what is a similar component to the range block under an affine mapping. These sample blocks are called domain blocks. The FBC requires that usually the domain block is larger than the range block for contraction mapping purpose. This technique reduces the complexity to $O(N^4)$. Another improvement was introduced

by Kinsner and Wall [WaKi93] to reduce the complexity to $O(N^3)$ by reducing the search space through a frequency-sensitive neural network. The technique is called the reduced-search FBC.

The idea of the FBC can be employed to process a 1D signal. The 1D signal can be cut into segments for fractal coding. For convenience, the segments are also called blocks. The examples of the fractal coding of the 1D signal include the speech signal and the ECG signal [Vera99] [Fish98]. Now we discuss how to use the IFS technique to encode the ECG or other time series signal.

A time series is defined as

$$\{(t_n, x_n) : n = 1, 2, \cdots, N\}, \quad t_n \in \mathbf{R} \text{ and } t_1 < t_2 < \cdots < t_N \tag{6.9}$$

where N is the length of the time series.

Now the question is how to find a set of mappings that best fits the given data sequence over the interval $[t_1, t_N]$. Mazel and Hayes suggested two IFS models, self-affine fractal model (SAFM) and piecewise self-affine fractal model (PSAFM), to approximate the given time series [MaHa92].

The SAFM is an IFS whose attractor is self-affine over the interval $[t_1, t_N]$, which means that range blocks may be described by applying affine mappings to the entire interval. Vera shows that the SAFM applied to speech signals does not achieve good compression performance [Vera99]. The poor performance of the SAFM results from taking the entire interval of a time series which may not be similar to many of range blocks. Such a

shortcoming may be overcome by the PSAFM.

The PSAFM splits the interval $[t_1, t_N]$ of the time series into domain blocks and range blocks. Each of the blocks is a subset of the time series. Usually the domain block is larger than the range block. For each range block, we search all the domain blocks (called domain pool) and find the most similar block under an affine transform with the Euclidean metric. The set of the affine transforms consists of the IFS of the PSAFM.

Definition 6.3: A domain pool, which is a set of domain blocks, is composed of

$$\{[t_{n+1}, t_{n+l_d}]: n = 0, l_s, 2l_s, \cdots, (\lfloor (N+1-l_d)/l_s \rfloor - 1)l_s\} \tag{6.10}$$

where l_d is the length of the domain blocks, l_s is the displacement between the domain blocks selected, and $\lfloor x \rfloor$ is the floor function which means taking the maximal integer which is smaller than x.

If $l_s < l_d$ then the domain pool consists of overlapped blocks. Notice that the position t_{n+i} in time axis may be rewritten as

$$t_{n+i} = t_n + i, \quad i \in \mathbf{Z} \tag{6.11}$$

Figure 6.1(a) illustrates a general idea of domain block partitioning for the ECG signal. One may notice that the SAFM is a special case of the PSAFM when $n = 0$ and $l_d = N$.

Definition 6.4: A range pool, which is a set of range blocks, is composed of

$$\{[t_{n+1}, t_{n+l_r}] : n = 0, l_r, 2l_r, \cdots, N/l_r - 1\} \tag{6.12}$$

where l_r is the length of the range blocks. The N and l_r are selected such that N/l_r is an integer.

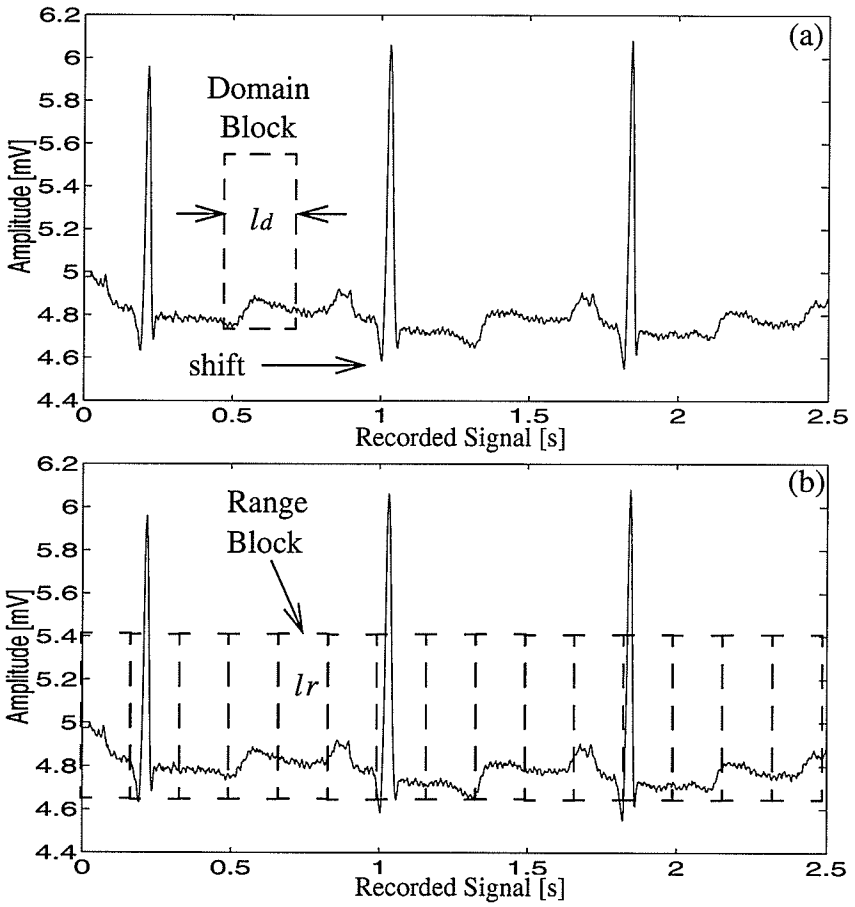

Fig. 6.1. The ECG signal partitioning for (a) domain blocks and (b) range blocks.

Figure 6.1(b) shows range block partitioning for the ECG signal. There is no overlapping between range blocks.

After defining the domain pool and range pool, we may use the affine transform to map a domain block to a range block.

Definition 6.5: A transform w: $\mathbf{R}^2 \to \mathbf{R}^2$ in the Euclidean plane

$$w\begin{pmatrix} t \\ x \end{pmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} t \\ x \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{6.13}$$

where a_1, a_2, a_3, a_4, b_1, and b_2 are real numbers, is called a (2D) affine transform.

Figure 6.2 illustrates that the elements a_1, a_2, a_3 and a_4 in the matrix perform four transforms like linear operators: scaling, rotating, reflecting, and shearing. The four coefficients decide the degree to which any of these transforms is done. The elements b_1 and b_2 give a translation along the abscissa and ordinate.

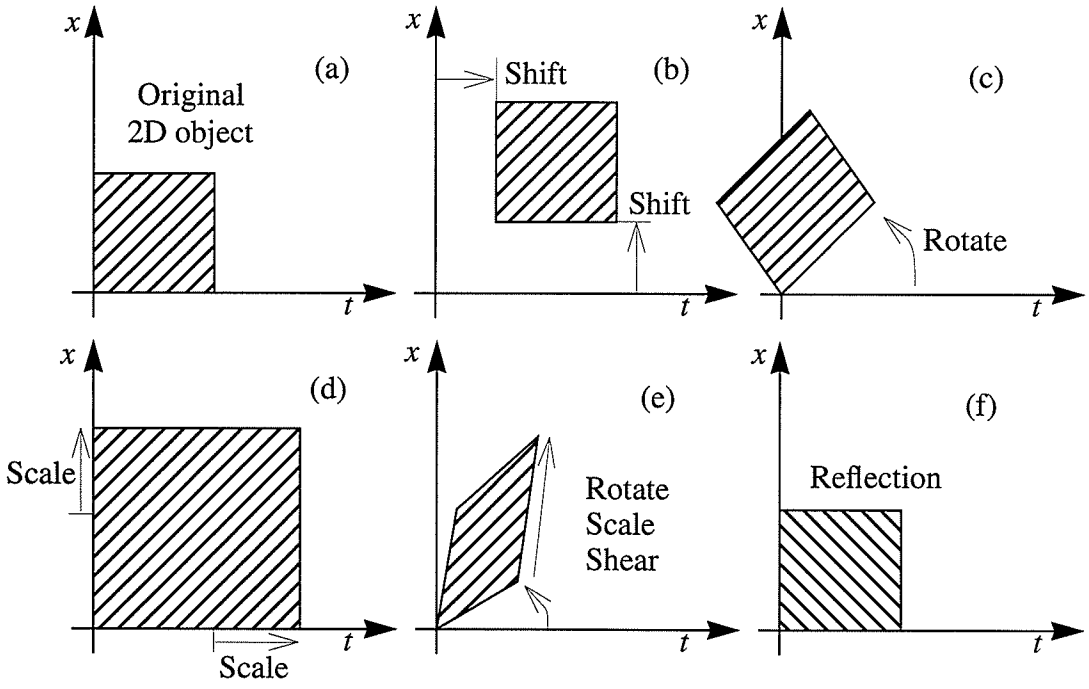

Fig. 6.2. (a) An object can be (b) shifted, (c) rotated, (d) scaled, (e) sheared, and (f) reflected by the affine transform.

A simplified form of the affine transform can be applied to the time series signal.

Time series data are single valued. The coefficient a_2 in the affine transform should be set to zero to ensure that the IFS only generates single valued data [Vine93] [ViHa93]. Thus, the affine transform for the 1D signal is rewritten as

$$w\begin{pmatrix} t \\ x \end{pmatrix} = \begin{bmatrix} a_1 & 0 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} t \\ x \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{6.14}$$

The boundary of the transformed domain block $[t_{d+1}, t_{d+l_d}]$ and the range block $[t_{r+1}, t_{r+l_r}]$ can be constrained by

$$w_i\begin{pmatrix} t_{d+1} \\ x_{d+1} \end{pmatrix} = \begin{bmatrix} t_{r+1} \\ x_{r+1} \end{bmatrix} \text{ and } w_i\begin{pmatrix} t_{d+l_d} \\ x_{d+l_d} \end{pmatrix} = \begin{bmatrix} t_{r+l_r} \\ x_{r+l_r} \end{bmatrix} \tag{6.15}$$

which gives

$$a_{1i} = \frac{t_{r+l_r} - t_{r+1}}{t_{d+l_d} - t_{d+1}} = \frac{l_r - 1}{l_d - 1} \tag{6.16}$$

$$b_{1i} = \frac{t_{d+l_d} t_{r+1} - t_{d+1} t_{r+l_r}}{t_{d+l_d} - t_{d+1}} = t_{r+1} - a_{1i} t_{d+1} \tag{6.17}$$

$$a_{3i} = \frac{(x_{r+l_r} - x_{r+1}) - a_{4i}(x_{d+l_d} - x_{d+1})}{l_d - 1} \tag{6.18}$$

$$b_{2i} = \frac{(x_{r+1} t_{d+l_d} - x_{r+l_r} t_{d+1}) - a_{4i}(x_{d+1} t_{d+l_d} - x_{d+l_d} t_{d+1})}{l_d - 1} \tag{6.19}$$

Since a_{1i} and b_{1i} are known, we can map every point t_{d+k} in the domain block into the range block which corresponds to

$$\begin{aligned} t_{r+j} &= \lfloor a_{1i} t_{d+k} + b_{1i} \rfloor \\ &= \lfloor a_{1i}(t_{d+1} + k - 1) + t_{r+1} - a_{1i} t_{d+1} \rfloor \\ &= t_{r+1} + \lfloor a_{1i}(k-1) \rfloor \end{aligned} \tag{6.20}$$

There is more than one point t_{d+k} mapped to the same point t_{r+j} since the transform from interval $[t_{d+1}, t_{d+l_d}]$ to $[t_{r+1}, t_{r+l_r}]$ is contractive. An x^*_{r+j} is defined as the average amplitude of all the points mapped to t_{r+j}. Thus

$$\begin{aligned} x^*_{r+j} &= \frac{1}{N_k} \sum_{j = \lfloor a_i(k-1) \rfloor} (a_{3i} t_{d+i} + a_{4i} x_{d+i} + b_{2i}) \\ &= a_{3i} t^*_{d+k} + (a_{4i} x^*_{d+k} + b_{2i}) \end{aligned} \tag{6.21}$$

where N_k is the number of the points k in the domain block mapped to the same point in the range block under the condition $j = \lfloor a_{1i}(k-1) \rfloor$, and

$$t^*_{d+k} = \frac{1}{N_k} \sum_{j = \lfloor a_i(k-1) \rfloor} t_{d+k} \tag{6.22}$$

$$x^*_{d+k} = \frac{1}{N_k} \sum_{j = \lfloor a_i(k-1) \rfloor} x_{d+k} \tag{6.23}$$

Consider (6.18) and (6.19) and let

$$\alpha_j = [(x_{r+l_r} - x_{r+1})t^*_{d+k} + (x_{r+1}t_{d+l_d} - x_{r+l_r}t_{d+1})]/(l_d - 1) \tag{6.24}$$

$$\beta_j = [(l_d - 1)x^*_{d+i} - (x_{d+l_d} - x_{d+1})t^*_{d+k} - (x_{d+1}t_{d+l_d} - x_{d+l_d}t_{d+1})]/(l_d - 1) \tag{6.25}$$

Equation (6.21) can be rewritten as

$$x^*_{r+j} = \alpha_j + \beta_j a_{4i} \tag{6.26}$$

Then we can use the Euclidean or a related metric such as the mean squared error (MSE), to measure the distance (error) between the transformed domain block and the range block

$$MSE \equiv d_i = \frac{1}{l_r}\sum_{j=1}^{l_r}(x^*_{r+j} - x_{r+j})^2$$

$$= \frac{1}{l_r}\sum_{j=1}^{l_r}(\alpha_j + \beta_j a_{4i} - x_{r+j})^2 \tag{6.27}$$

The minimal error in the above equation can be obtained by letting

$$\frac{\partial d_i}{\partial a_{4i}} = \frac{2}{l_r}\sum_{j=1}^{l_r}(\alpha_j + \beta_j a_{4i} - x_{r+j})\beta_j = 0 \tag{6.28}$$

This equation gives coefficient a_{4i} in the affine transform as

$$a_{4i} = \frac{\sum_{j=1}^{l_r} (x_{r+j} - \alpha_j)\beta_j}{\sum_{j=1}^{l_r} \beta_j^2} \tag{6.29}$$

Equations (6.16) to (6.19), and (6.29) give the affine transform from the domain block to the range block with a minimal error. The corresponding measure error between the domain block and the range block can be obtained by (6.27). To find a similar domain block, we will compare the given range block with all the domain blocks in the domain pool and get a set of measure error. The most similar domain block corresponds to the domain block which has the minimal measure error in the set. Then the attractor of the IFS of the time series interval at $[t_1, t_N]$ is obtained by searching the entire range pool.

To encode the affine transform, the coefficients need to be quantized. A quantized encoding set of the IFS is called the fractal coding. Each kind of the coefficients has different contribution to the measure error. This contribution also varies with different objects processed. Usually the bit distribution to each kind of coefficients is decided through experiment, as described in Sec. 6.5.

6.4 IFS Reconstruction of Time Sequence

In the previous section, we presented an algorithm of how to find the attractor of the IFS of the given time series. With such an IFS, the time series can be reconstructed by iteratively mapping the signal to itself.

To reconstruct the original fractal object from the fractal coding, an iterative

To reconstruct the original fractal object from the fractal coding, an iterative decoding algorithm is used according to the collage theorem. This decoding algorithm begins with a nonempty set A_0. The affine transform of the IFS on the A_0 is

$$A_1 = \bigcup_{i=0}^{N/l_r - 1} w_i(A_0) \tag{6.30}$$

If the IFS is applied iteratively to the set, we get a sequence of the sets $A_0, A_1, A_2, \cdots$. The contractibility of the IFS guarantees that the sequence is convergent and converges to the attractor of the IFS. We say that A_n is the attractor of the IFS if there exists a threshold ε such that

$$d(A_n, A_{n+1}) \leq \varepsilon \tag{6.31}$$

Then the A_n is the reconstruction of the time series corresponding to the IFS.

Figure 6.3 shows an example of how an ECG signal is reconstructed from fractal coding. Since a nonempty set is the only requirement for the initial signal, the reconstruction begins iterations from a straight line. When the IFS with the fractal coding is applied to the signal we see a convergence procedure of the ECG with the iteration. The ECG reconstruction is finished after 11 iterations when the iteration causes almost no change for the ECG waveform under (6.31). What is remarkable with this technique is that the main characteristic features of the signal emerge after 2 to 6 iterations.

From the reconstruction procedure, one sees that fractal coding is different from vector quantization and transform coding. Fractal coding uses a functional expression of a

data set. There is neither a codebook nor a basis function in fractal coding. Since the reconstruction is an approximate of the original signal, fractal coding is a lossy signal compression approach.

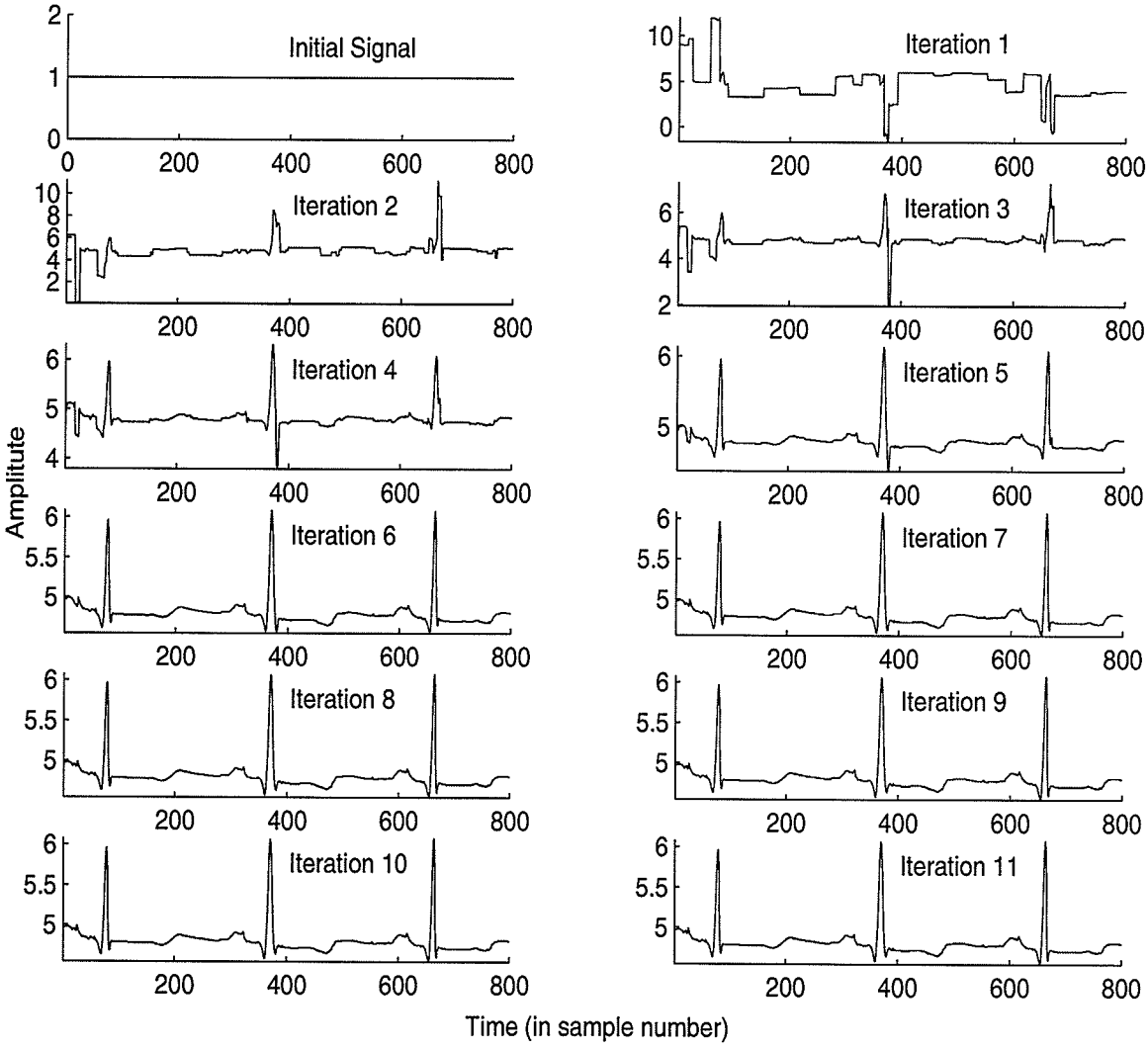

Fig. 6.3. A reconstruction procedure of an ECG signal by the IFS through 11 iterations, with the initial signal being a straight line.

6.5 Quantization of Coefficients in the IFS

The IFS produces a set of affine transforms, mapping an object to itself. Such

transforms are composed of real transform coefficients. Like in the Fourier and wavelet transforms applied to data compression, the coefficients in the affine transform need to be quantized for storage and transmission purposes.

The process of representing a large, possibly infinite, set of values with a much smaller set is called quantization. For example, any real number x can be rounded off to the nearest integer, say $Q(x) = \text{round}(x)$. Then the real line in a continuous space is mapped into a discrete space.

Many of the fundamental ideas of quantization and compression are most easily introduced in the simple context of scalar quantization. Let us pose the quantization design problem in precise terms. Suppose we have an input modelled by a random variable X with *probability density function* (PDF) $P(x)$. In general, a (scalar) quantizer Q can be described by (i) a set of M disjoint intervals $y = \{y_i; i \in \boldsymbol{Z}\}$ and $M+1$ endpoints and (ii) a set (codebook) of reproduction values or points or levels $C = \{q_i; i \in \boldsymbol{Z}\}$ for each of the M intervals such that the quantizer is defined by

$$Q(x) = q_i \text{ for } x \in y_i \tag{6.32}$$

In the rounding off example, a left semi-closed interval is used, $y_i = [i - 1/2, i + 1/2)$ and $q_i = i$ for all integers i. More generally, $y_i = [r_{i-1}, r_i)$ where the r_i (called thresholds) forms an increasing sequence. The width of a cell y_i is its length, $r_i - r_{i-1}$.

Since the source cannot be reconstructed exactly from the quantization output, quantization process leads to an error, $\varepsilon = Q(x) - x$. Therefore,

$$Q(x) = x + \varepsilon \tag{6.33}$$

The above equation implies additive noise model of a quantizer, as shown in the following figure. Unlike signal sampling, data quantization loses information.

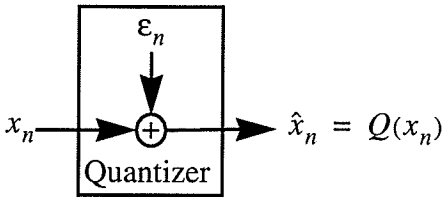

Fig. 6.4. Additive noise model of a quantizer.

Quality of the quantizer is measured by the goodness of the resulting reproduction in comparison to the original. A distortion measure, $d(x, \hat{x})$, is defined to quantify cost or distortion resulting from reproducing x as $\hat{x}$. The most common distortion measure is a squared error, $d(x, \hat{x}) = |x - \hat{x}|^2$. An average distortion under the squared error is given as

$$\begin{aligned} \sigma(Q) &= E[d(X, Q(X))] \\ &= \int_{-\infty}^{\infty} (x - Q(x))^2 P(x) dx \\ &= \sum_{i=1}^{M} \int_{y_{i-1}}^{y_i} (x - q_i)^2 P(x) dx \end{aligned} \tag{6.34}$$

A PDF-based optimal quantizer can be obtained by minimizing the above error function. Setting the derivative of $\sigma(Q)$ with respective to q_i to zero and solving for q_i, we have [Sayo00] [Sayo96]

$$q_i = \frac{\int_{y_{i-1}}^{y_i} xP(x)dx}{\int_{y_{i-1}}^{y_i} P(x)dx} \qquad 1 \le i \le M \tag{6.35}$$

The output point for each quantization interval is the centroid of the probability mass in that interval. Taking the derivative with respect to y_i and setting it equal to zero, we get an expression for y_i [Sayo96]

$$y_i = \frac{q_i + q_{i+1}}{2} \qquad 1 \le i \le M-1 \tag{6.36}$$

The endpoints are simply the midpoint of the two neighbouring reconstruction levels. Solving these two equations will give us the values for the reconstruction levels and intervals that minimize the average quantization distribution. Unfortunately, to solve for q_i we need the values of y_i and y_{i-1}, and to solve for y_i we need the values of q_i and q_{i+1}.

Lloyd and Max proposed an iterative trial-and-error algorithm to solve the problems [Lloy82] [Max60]. The procedure of the Lloyd-Max algorithm is as follows:

1) Choose a trial value q_1 satisfying

$$q_1 < \int_{-\infty}^{\infty} xP(x)dx \tag{6.37}$$

2) The condition that q_1 is the centre of mass on $[y_0, y_1)$ determines y_1 as the unique solution of

$$q_1 = \frac{\int_{-\infty}^{y_1} xP(x)dx}{\int_{-\infty}^{y_1} P(x)dx} \tag{6.38}$$

3) After the quantities q_1 and y_1 are known, q_2 is obtained according to (6.36)

$$q_2 = 2y_1 - q_1 \tag{6.39}$$

If this q_2 lies to the right of the centre of mass of the interval (y_1, ∞) then the trial chain terminates, and we go to Step (1) and start over again with a different trial value q_1. Otherwise, y_1 and q_2 are known and serve to determine y_2 according to (6.35)

$$q_2 = \frac{\int_{y_1}^{y_2} xP(x)dx}{\int_{y_1}^{y_2} P(x)dx} \tag{6.40}$$

4) Then we get q_3 by

$$q_3 = 2y_2 - q_2 \tag{6.41}$$

5) We continue in this way, obtaining successively $q_1, y_1, \cdots, q_{M-1}, y_{M-1}$.

6) The last step is to determine q_M according to

$$q_M = 2y_{M-1} - q_{M-1} \tag{6.42}$$

At the same time, q_M can be obtained from

$$q_M = \frac{\int_{y_{M-1}}^{\infty} xP(x)dx}{\int_{y_{M-1}}^{\infty} P(x)dx} \tag{6.43}$$

The q_M obtained from (6.42) will not satisfy (6.43) in general. The discrepancy between the right numbers of (6.42) and (6.43) will vary continuously with the starting value q_1, and the technique consists of running through such chain using various starting values until the discrepancy is reduced almost to zero.

Now we see that the quantizer designed by the Lloyd-Max algorithm is determined uniquely by the PDF of the source. Common distributions concerned in quantization design are uniform, Gaussian, and Laplacian, as given in (6.44), (6.45), and (6.46), respectively.

$$P(x) = \frac{1}{y_M - y_0} \tag{6.44}$$

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{6.45}$$

$$P(x) = \frac{1}{\sqrt{2}\sigma} e^{-\sqrt{\frac{2}{\sigma}}|x-\mu|} \tag{6.46}$$

where μ and σ are the mean and variance of the source, respectively.

The code for a nonuniform quantizer design is provided in Appendix B.3.2.

For the source with uniform PDF, we may get an optimal quantizer with equal spaced intervals and reproduction levels. Such a quantizer is called a uniform quantizer and widely applied in analog to digital converter.

Fractal compression focuses on encoding transforms and coefficients. In fractal compression for images, the FBC separates the affine transform into three distinct transforms performed sequentially as shown in the following figure. These transforms are (i) spatial contraction, (ii) isometric block transform, and (iii) grey level scaling and translation, respectively.

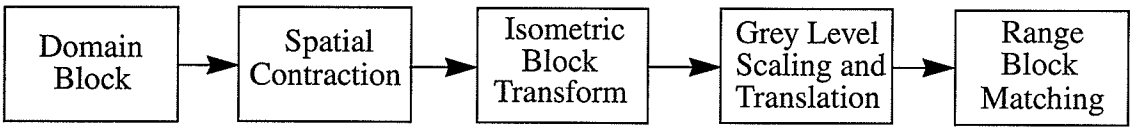

Fig. 6.5. The fractal block transform in terms of its sequential component transforms: spatial contraction, isometric block transform, and grey level scaling and translation.

For square domain and range blocks the following eight isometric block transforms are used:

1) identity,
2) reflection about mid-vertical axis,
3) reflection about mid-horizontal axis,
4) reflection about 45° diagonal,
5) reflection about 135° diagonal,
6) +90° rotation about centre,
7) +180° rotation about centre, and

8) $-90°$ rotation about centre.

Then three bits are allocated to encode isometric block transform and appropriate bits are used to encode the address of the domain block, spatial contraction, and grey level scaling and translation. The number of encoding bits used in the FBC is independent of the range block size.

This quantization scheme cannot give an optimal design of the quantizer and minimal error for the affine transform. It only tries to achieve a low compression ratio under a certain reconstruction error through a time-consuming search.

Øien and Nårstad applied an orthogonal fractal compression to the ECG signal. Instead of the affine transform, they used orthogonalization transforms in the FBC and found that the transform coefficients are close to Laplacian distribution [Fish98]. Then a nonuniform quantizer based on such a distribution was designed to encode the coefficients. Still the number of the encoding bits is independent on the range block size.

We confirmed the near Laplacian distribution of the fractal coefficients of the ECG signals in our experiments. Thus, a nonuniform quantizer with a Laplacian distribution is designed by the Lloyd-Max approach. Unlike Øien and Nårstad, however, we use an optimal number of bits to encode each coefficient in the transforms and do not allocate a special code to the isometric block transform any more. The benefit of such an encoding scheme is to maintain, not to reduce, the ability of the transforms in modelling the fractal objects.

6.6 The Nonlinear IFS (NIFS)

6.6.1 Limitation of Affine Transform

The IFS data compression is realized through finding a set of affine transforms between the range pool and the domain pool. The role of the affine transform is to fit specific range blocks with the domain blocks optimally. The compression performance is directly decided by the capability of the affine transform that models the signal. The current transform used in the IFS is a linear affine transform. It may model perfectly some signals which have strict inherent linear relationship (such as the Cantor set, Koch curve, Sierpinski carpet, and Julia set) [PeJS92]. Unfortunately, many signals in real life have relationships that are more complicated than linear. To model such signals, the linear affine transform is inadequate. Image compression by the IFS fractal method is an example of how difficult it is to find a suitable mapping between a range block and the domain blocks. A complicated scheme that requires a point-by-point search, combined with a variable size of the domain blocks is necessary to find a reasonable mapping. Even then, it may fail occasionally. This linear transform causes two problems: (i) poor compression performance, and (ii) a very time-consuming search.

6.6.2 Nonlinear Transform for the IFS

Instead, a generalized transform which may represent an arbitrary function is needed to model such complicated signals. We propose that a more flexible way to model complicated signals is to use a nonlinear transform. This can be accomplished by modelling the ECG signal by a polynomial expansion as it is an efficient way to model arbitrary

functions. Convergence is a critical issue in signal reconstruction by the IFS using the proposed transform. Although we have not proved the convergence theoretically, our experiments show that the new extended affine transform gives convergent results. This nonlinear model also demonstrates the flexibility in modelling various complexities in the ECG signal. We shall show that it can give a much better reconstruction quality under the same compression ratio, as compared to a corresponding linear model.

For a 1D signal $x(n)$, the generalized transform is defined by

$$w\begin{pmatrix} n \\ x \end{pmatrix} = \begin{bmatrix} a_1 n + a_2 \\ f(x, n) \end{bmatrix} \tag{6.47}$$

where the coefficient a_1 is limited to the range $[0, 1)$ to guarantee a contraction transform of signals in time. If $f(x, n)$ is also a contractive transform, then (6.47) satisfies the convergence condition of the IFS, as required by (6.2).

To find an appropriate $f(x, n)$, it is not sufficient to use the convergence condition only. According to Taylor's remainder formula, a function may be expanded as a kth-order polynomial format if its first $k-1$ derivatives are continuous and its kth derivative exists [Trim89]. Thus, a kth-order polynomial is used to approximate the $f(x, n)$

$$f(x, n) \approx \sum_{m=0}^{k} \sum_{q=0}^{m} b_{mq} x^q n^{m-q} \tag{6.48}$$

where k, m, and q are nonnegative integer. Coefficients b_{mq} are decided by the function $f(x, n)$ according to Taylor's remainder formula. Since the $f(x, n)$ is unknown, it may be

determined through the inverse procedure, *i.e.*, finding b_{mq}. To obtain an attractor of the object, it is reasonable to assume [Barn88]

$$f(x(n_1), n_1) = x(n_2) \tag{6.49}$$

where $x(n_1)$ is mapped to $x(n_2)$ under the condition

$$a_1 n_1 + a_2 = n_2 \tag{6.50}$$

Equations (6.48) to (6.50) constitute our nonlinear contraction transform for finding the attractor of the object. When the expansion order k is equal to 1, the nonlinear transform becomes the traditional affine transform. Therefore, one sees that the nonlinear transform leads to more flexible mappings between the range blocks and the domain blocks than the linear affine transform. Such flexibility may result in two advantages: (i) improving compression performance, and (ii) speeding up the search.

We shall now discuss how to use the least squared error technique to determine the coefficients in (6.48) and (6.50). For a lossless fractal compression technique, a transform must be found to satisfy (6.49) with a measured signal exactly. In practice, it is reasonable to use an approximation to replace the strict equality condition between $f(x, n)$ and $x(n)$ in the lossy fractal compression. Then we use the MSE to measure the distance between the transformed domain block and the range block

$$MSE = \frac{1}{r_l} \sum_{n_2 = 1}^{r_l} [f(x(n_1), n_1) - x(n_2)]^2 \tag{6.51}$$

where r_l is the length of the range block.

A set of optimal transform coefficients can be obtained by applying the least squared error technique to (6.51). A similar procedure has been described in Sec. 6.3.

6.6.3 Application of the NIFS to ECG [HuKi01a]

The new nonlinear IFS (NIFS) transform is applied to compress a 1D ECG signal. To compare compression performance, the traditional IFS technique is also implemented. The code for the NIFS is provided in Appendix B.3.1. The performance of a compression algorithm is dependent mainly on two parameters: compression ratio and reconstruction (distortion) error.

Prior to the calculation of the compression ratio and reconstruction error, the coefficients in the $f(x, n)$ must first be quantized. Figure 6.6 shows two distributions of coefficients b_{10} and b_{20} with the ECG signal from the MIT-BIH Arrhythmia Database [Mood99]. We conducted an experiment using the ECG data contained in the file, x_100.txt, which has a 10-minute recording of the ECG signal. The Lloyd-Max nonuniform quantizer discussed in Sec. 6.5 [Lloy82] was used to quantize the coefficients because their distribution is close to Laplacian.

To improve compression performance, the coefficient a_1 is set to 2^{-m} where $m = 1, 2, 3$, to increase the chance for finding the optimal mapping. The length of range blocks is also a variable, changing according to 2^l with the integer l varying from 2 to 6. Figures 6.7(a) and (b) give the size distributions of range blocks for the traditional and the NIFS technique, respectively. The NIFS can use the most frequent range block size of 32

sample points to model the ECG signal compared to 4 sample points used by the linear affine transform. The size distributions of range blocks show that the NIFS finds more range blocks with large size than the traditional IFS, thus leading to an improved compression ratio. In our experiment, the mean range size is about 30.0 for the NIFS and 26.9 for the IFS. It demonstrates that the nonlinear transform can model signals more flexibly.

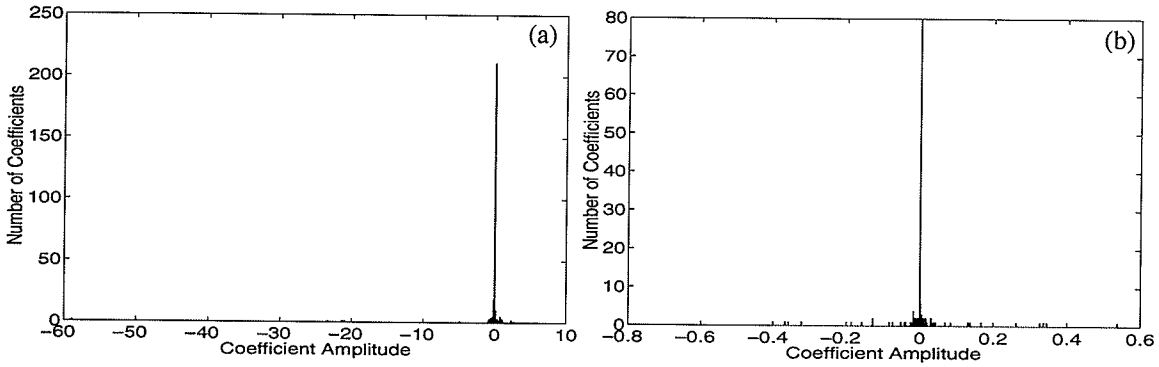

Fig. 6.6. Distribution of coefficients (a) b_{10} and (b) b_{20}.

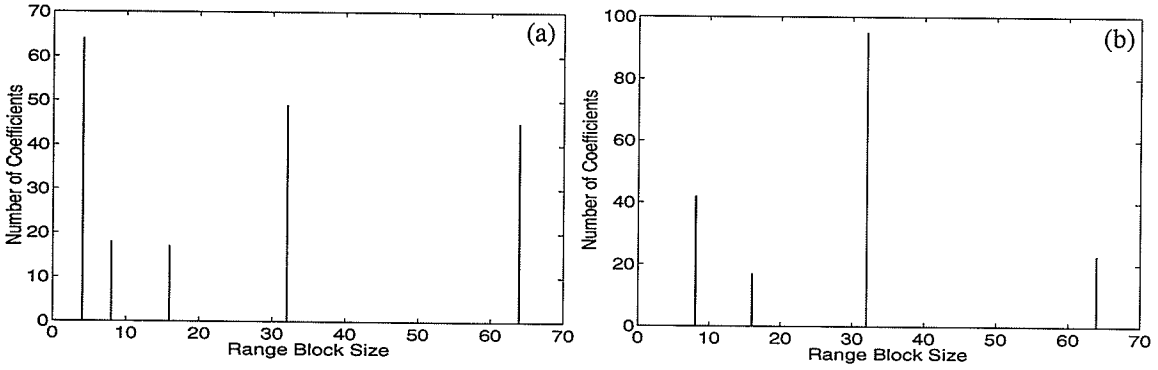

Fig. 6.7. Range block size distributions for (a) the traditional affine transform and (b) the nonlinear transform.

Figures 6.8(a) and (b) give the address distribution of the optimally mapped domain blocks found by the traditional IFS and the NIFS techniques, respectively. It is seen that the address distribution of the mapped blocks given by the NIFS is more concen-

trated around the first point of the search. Compared to full distribution in the segment of 1024 points of the IFS, the NIFS gives a distribution of domain block addresses within 110 point range. It shows that the optimal mapping can be found by searching fewer domain blocks. This is useful in developing a fast search algorithm as done in the next section.

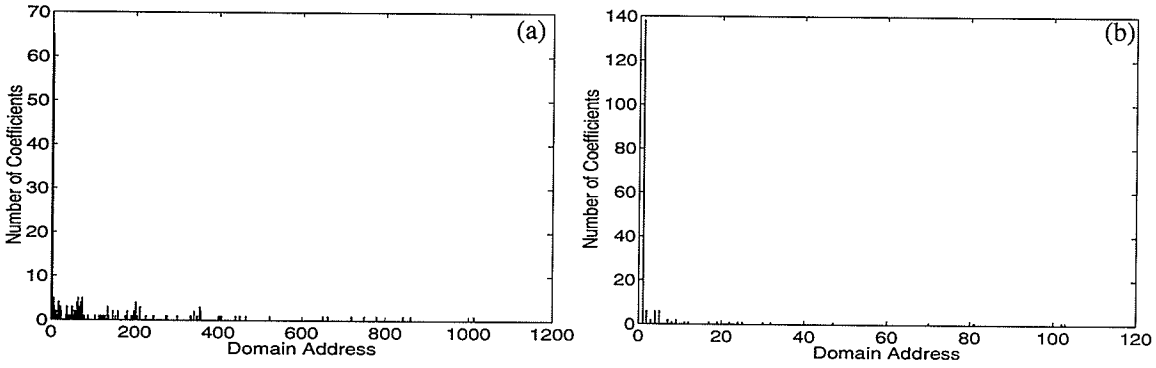

Fig. 6.8. Address distribution of the optimally mapped domain blocks found by (a) the traditional IFS and (b) the NIFS approaches.

Figures 6.9(a), (b), and (c) show the original ECG signal, its reconstruction, and the reconstruction error, respectively. The first important issue in the NIFS technique is the convergence of the strange attractor reconstruction. From these figures, one can see that the NIFS compresses and reconstructs the ECG signal extremely well. The new technique has the remarkable property of modelling the QRS complex of the ECG signal almost perfectly, which is a difficult issue for the linear affine transform and the orthogonal transform [Fish98]. The reconstruction error is limited to $[-33\mu V, 32\mu V]$. It is mainly due to the noise in the signal.

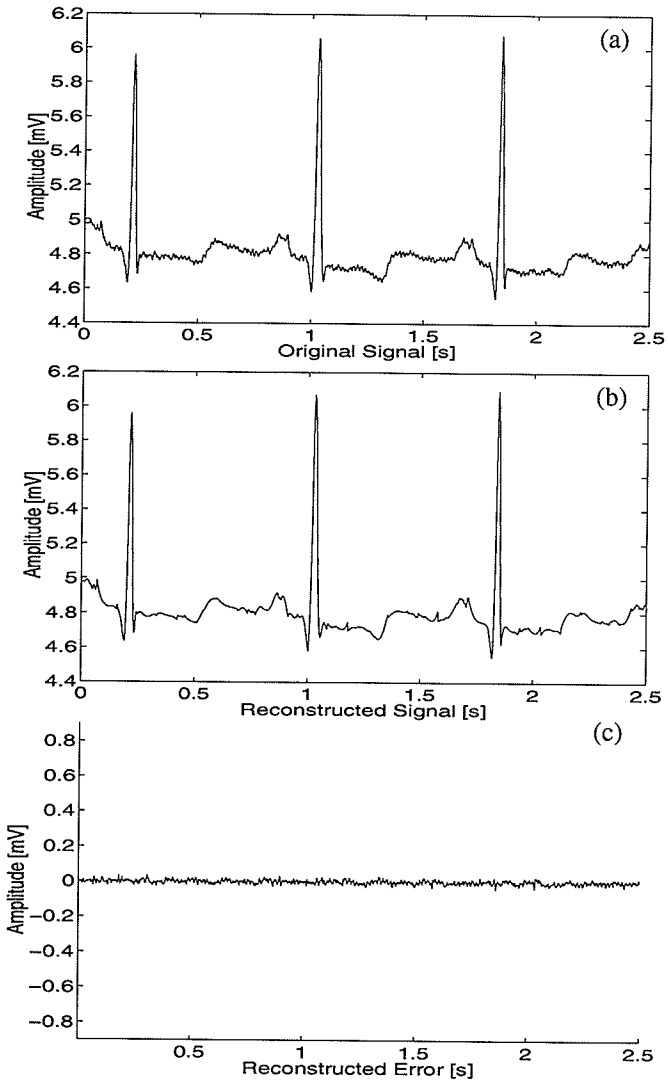

Fig. 6.9. The ECG signal compression: (a) original ECG signal, (b) reconstructed signal, and (c) reconstruction error.

Table 6.1 shows the influence of the order k of the Taylor series expansion and quantization resolution on the compression performance of the ECG signal. By changing the expansion order from 1st to 4th, as well as the resolution of the quantizer from 7 to 14 bits, various MPRD and R_{cr} are obtained. The maximal and minimal MPRDs are 6.1%

and 5.6%, respectively. The maximal and minimal compression ratios are 7.3:1 and 5.5:1, respectively. The MPRD of 100% at a 9 bit resolution and $k = 1$ is considered as an outlier. Such a reconstruction failure is probably due to the sensitivity of the reconstruction to the quantization resolution of the transform coefficients. By monitoring the change of the MPRD and R_{cr}, we observe that (i) the NIFS outperforms the traditional IFS and (ii) optimal parameters for the NIFS can be found.

For each polynomial expansion, Table 6.1 shows that when the resolution increases the R_{cr} increases first and then decreases. The MPRD is controlled by the error threshold. A quantizer with 8 bit resolution gives optimal performance for the NIFS with various orders.

Table 6.1: Compression performance investigation of the NIFS with quantization resolution and polynomial expansion change on the ECG signal (x_100.txt).

Resolution (bit)	k = 1		k = 2		k = 3		k = 4	
	MPRD	R_{cr}	MPRD	R_{cr}	MPRD	R_{cr}	MPRD	R_{cr}
7	5.6%	6.3	5.8%	7.0	6.0%	6.7	6.1%	5.5
8	5.7%	6.4	5.9%	7.3	6.1%	7.0	6.1%	6.0
9	100%	6.4	5.8%	7.1	6.0%	7.0	6.1%	6.1
10	5.8%	6.4	5.8%	6.9	5.9%	6.7	6.1%	6.2
11	5.9%	6.1	5.8%	6.5	5.9%	6.4	6.0%	5.9
12	5.9%	6.1	5.8%	6.5	5.9%	6.3	6.0%	5.9
13	5.9%	6.0	5.8%	6.3	5.9%	6.2	5.9%	5.8
14	5.9%	5.9	5.8%	6.3	5.9%	6.1	5.9%	5.8
Threshold	0.0115		0.0118		0.0123		0.0130	

One may predict that the increasing order of the expansion will result in the MPRD

decrease because of the more powerful modelling ability of the NIFS. It is validated by the error threshold used in Table 6.1. To keep the MPRD the same for comparison purpose, higher error threshold is required for the higher order NIFS. However, a higher order results in more coefficients, thus leading to a lower compression ratio. Consequently, there must be a trade-off between the MPRD and the R_{cr} for choosing the optimal order k. Although Table 6.1 demonstrates that the change of expansion order has almost no influence on the MPRD (because the MPRD is controlled by the error threshold), it results in different R_{cr}. The R_{cr} increases with the order first, and then decreases. The expansion order of 2 achieves optimal performance for all the orders.

It was found that the NIFS improves compression performance compared to the traditional IFS in the ECG signal compression. When $PRD = 5.8\,\%$, Øien and Nårstad achieved 6.0:1 with their IFS ECG compression by their orthogonal transform [Fish98]. With the optimal choice of order 2 and 8 bit nonuniform Laplacian quantizer, we get the compression ratio about 6.4:1 for the affine transform and about 7.3:1 for the NIFS when $MPRD = 5.8\,\%$. It should be noted that the MPRD gives a better reconstruction quality than the PRD under the same error value.

6.7 Domain Block Partitioning Based on Complexity Measure

As we have discussed already, a fractal object may be expressed in terms of a series of affine mapping transforms of itself. In data compression of the traditional linear IFS approach, a time-consuming search needs to be performed on the entire signal to obtain an optimal match between a small range and some other part of the signal because the linear affine transform is not flexible to model complicated signals locally. For exam-

ple, the computational complexity of Barnsley's collage coding for an image of $N \times N$ size is $O(N^6)$ [Barn88]. In order to reduce the computational complexity, Jacqin proposed a fractal block coding (FBC) technique $O(N^4)$ [Jacq92]. The FBC splits the image into two kinds of small blocks: domain blocks and range blocks. Each range block is compared to affine-transformed versions of the domain blocks. The most similar pair gives the optimal affine transform for that range block. Kinsner *et al.* suggested a reduced FBC $O(N^3)$ in which a neural-network based classification technique is used to find an optimal match between the range and domain blocks [WaKi93]. This approach was applied first to images [WaKi93], and later to speech through the residual in the code excited linear prediction (CELP) technique [Vera99]. Still a more efficient search approach is necessary to compress the ECG signal by fractal technique in real time. We will make use of fractal properties to develop a fast IFS approach in ECG data compression.

Equation (6.1) reveals two important properties in the fractal object: (i) self-similarity which signifies scale invariance and (ii) fractal dimension which signifies the structural and informational (compositional) complexity. The real α in (6.1) signifies the invariance to dilations and contractions, which is the foundation of the IFS. The IFS uses a set of contractive mappings from the signal to itself to represent the fractal object. However, the complexity of the fractal object is not employed by the IFS to find the mappings.

We conjecture that a strong self-similarity of the fractal object exists in the area where the signal has the same compositional complexity, and the self-similarity is much weaker between regions with different complexities. If that is true, then it is not necessary to search domain blocks which have different complexities with the range block for the

multifractal object. Therefore, if we can measure the complexity of the signal and only search domain blocks which have the same complexity with the range block, the search can be very efficient.

It has been shown that the ECG signal is multifractal, with its singularity varying with time [KiHC00]. Thus, we could segment the signal into ranges with similar fractal dimensions. Based on this idea, a new scheme is proposed to segment the ECG signal for the construction of an optimal domain pool. Instead of taking the domain blocks from almost every point of the signal, our pool is only composed of the partitioned segments, which reduces the search problem to $O(N)$ if we take the search size to be N. The application of a nonlinear affine transform to such a domain pool can reduce the reconstruction error, while maintaining fast search [HuKi01a]. Experimental results show that our approach can achieve lower reconstruction error and faster implementation speed under the same compression ratio than the traditional IFS.

6.7.1 Signal Partitioning By the VFDT

Since we know that the ECG signal has multifractal characteristics, it is possible to segment the signal based on a local fractal dimension estimate [KiHC00]. Fractal dimensions such as the Rényi dimension spectrum, Mandelbrot dimension spectrum, correlation dimension, and Hausdorff-Besicovitch dimension, require a large number of sample points to obtain a statistical measure [Kins94a]. Usually, such a measure is a global estimate for the fractal object. Instead of a single-scale statistical measure, the variance fractal dimension (VFD) discussed in Sec. 4.8 is estimated by computing the spread of the increments in the signal amplitude, based on a much smaller number of points within a window

of interest, and at several scales. A time series representing a chaotic or nonchaotic process can be analyzed directly in time through its VFD [Kins94b].

Fractal analysis of nonstationary signals is usually conducted in terms of fractal dimensions as a function of time, thus resulting in the *VFD trajectory* (VFDT) which reflects the change of the compositional complexity of the signal with time. In fact, the VFDT represents the multifractal characteristics of signals with time. The VFDT is generated by calculating local VFDs for a rectangular sliding window that is displaced along the entire signal.

Section 4.8 illustrates how to calculate the variance dimension from a given time series. The *b*-adic rule is not followed to accommodate the small window for the ECG signal. To achieve a good segmentation for the signal, the sliding-window size for the VFDT should be chosen properly for the algorithm. We can see that the complexity segments change with the window size. An optimal sliding-window size is chosen experimentally by inspecting the VFDT, computed for different numbers of samples varying from N_{min} to N_{max}, as shown in Fig. 6.10. The histogram of the VFDT of an ECG signal shown in Fig. 6.11 has three peaks. For the VFDT shown in Fig. 6.10, two thresholds are selected experimentally to separate the trajectory into three levels, thus segmenting the ECG into three areas with different complexities. Figures 6.12(a) to (d) illustrate the corresponding segmentations of the ECG signal. Figure 6.12(d) shows that the sliding window with 128 sample points results in intervals that are too wide for the low complexity parts. Figure 6.12(c) with 64 sample point window shows that the region is still too wide. On the other hand, Fig. 6.12(a) shows that the wide segmentation in the high complexity areas is caused

by the window size of 16 sample points being too short. Consequently, Fig. 6.12(b) with window size of 32 samples gives a proper segmentation for the ECG signal because it is partitioned into intervals with (i) high but uniform complexity, (ii) middle but varying complexity and (iii) low complexity representing the QRS complex. In this chapter, the sliding window with 32 sample point width is chosen as an optimal parameter for the complexity measure of the ECG signal. The code for the VFDT is provided in Appendix B.3.3.

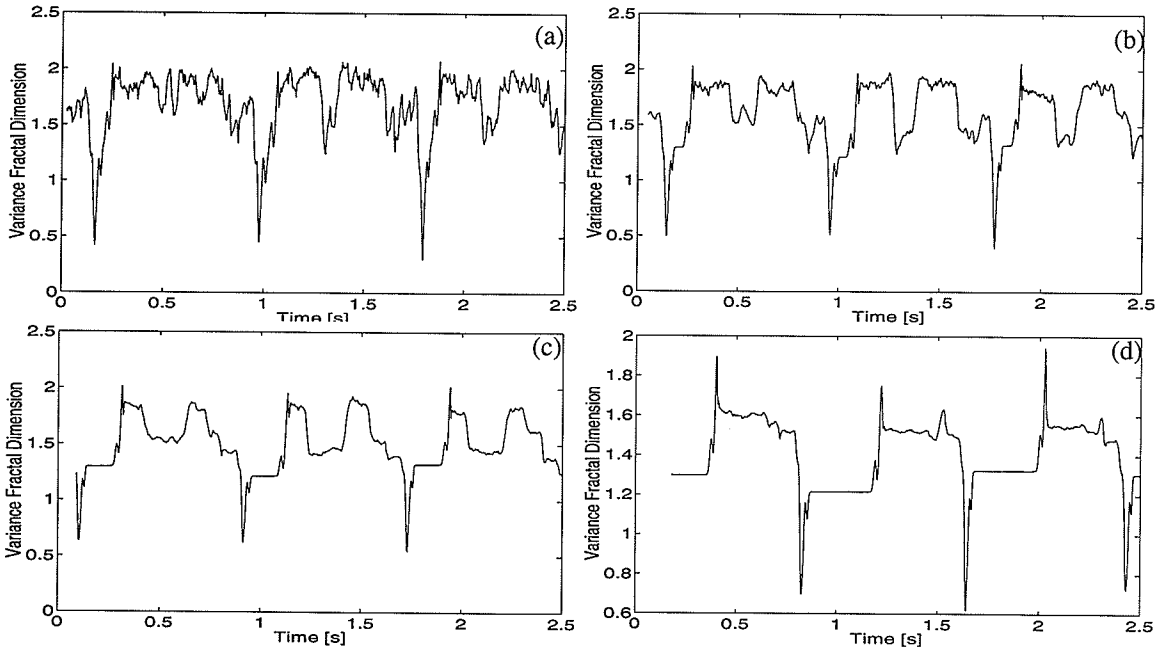

Fig. 6.10. The variance fractal dimension trajectory of an ECG signal with window size of (a) 16, (b) 32, (c) 64, and (d) 128 sample points.

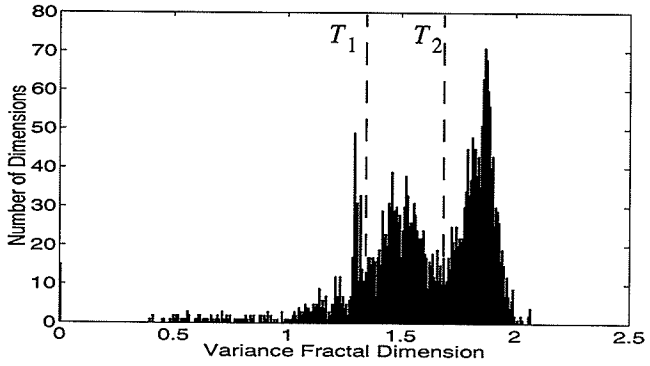

Fig. 6.11. Histogram of the VFDT of an ECG signal with sliding window of 32 sample points.

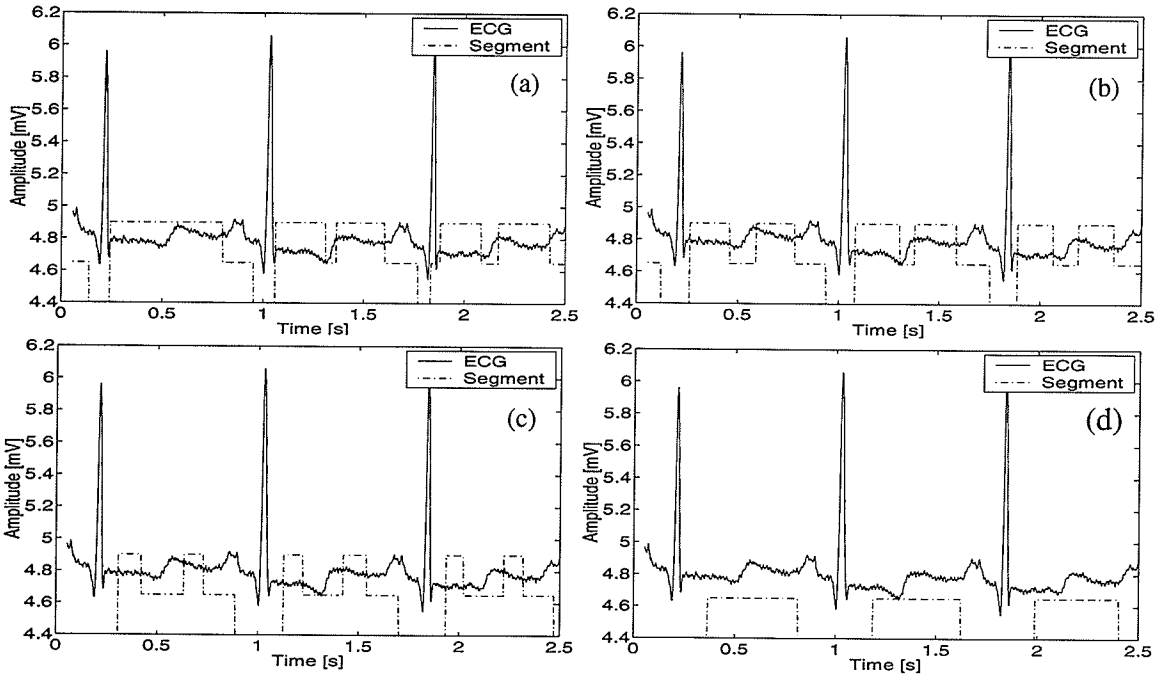

Fig. 6.12. Partitioning of an ECG signal based on complexity estimate with window size of (a) 16, (b) 32, (c) 64, and (d) 128 sample points.

The segmentation of the signal seems to depend on prior knowledge of the signal since the QRS complex, the T wave, and the P wave are isolated. However, this is not our purpose. Signal partitioning is to segment the ECG signal according to its compositional complexity characteristics through a local complexity measure. Thus, such a partitioning is not focused on diagnosis and will not lead to missing important artifacts in abnormal signals.

6.7.2 Application of the NIFS to the Partitioned ECG

Figure 6.13 gives an original ECG signal and its reconstruction by the IFS (with signal partitioning). Although the search speed of finding the transform set is fast, the compression performance suffers from large reconstruction errors, especially in the QRS

area. One of the reasons is that the affine transform defined by (6.14) is not a proper transform for the ECG signal.

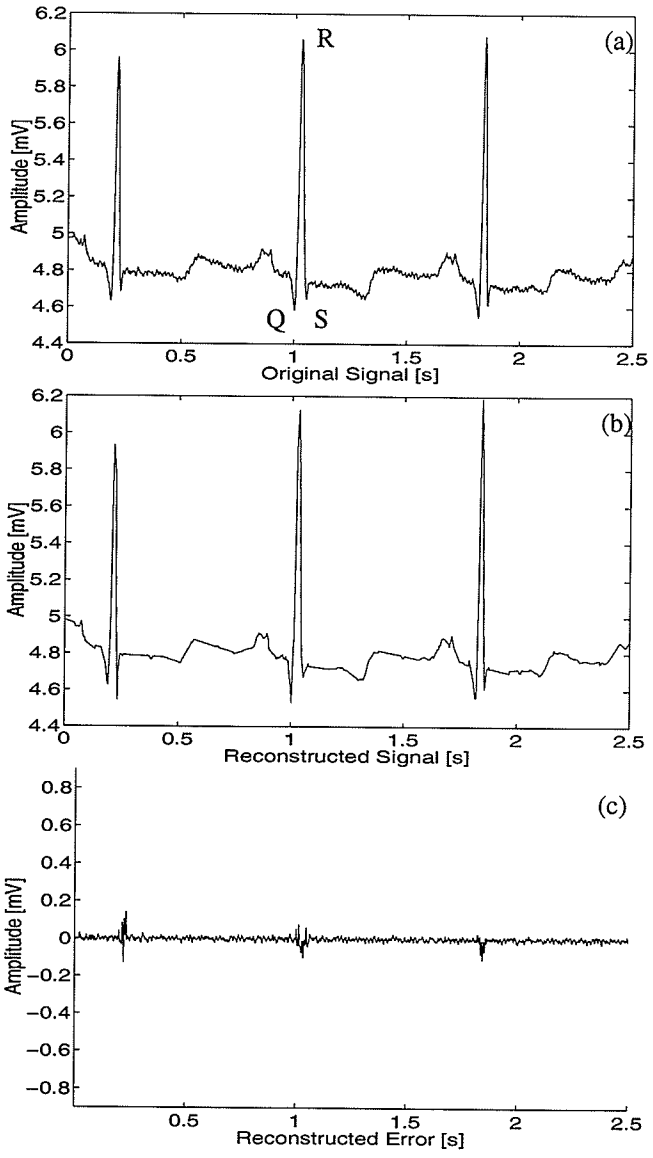

Fig. 6.13. The ECG signal compression with the IFS approach: (a) original ECG signal, (b) reconstructed ECG signal, and (c) reconstruction error.

Since (6.14) defines a linear transform, it should not be applied to signals with nonlinear characteristics. The ECG signal, like speech and images in nature, comes from a nonlinear system and is very complicated. Although there is self-similarity in the ECG

signal in the area with the same compositional complexity, the linear affine transform cannot represent the abrupt change in the QRS complex.

In principle, Eq. (6.48) may fit an arbitrary function. Therefore, we combine the NIFS and the VFDT techniques to develop a fast IFS algorithm for multifractal signal compression. Such a scheme must be validated by experimental results.

In this section, experiments are performed to compress the ECG signal by combining the VFDT and the NIFS together. Again, the ECG signal is taken from the MIT-BIH ECG database [Mood99]. We compress the ECG data contained in the file, x_100.txt, which is a 10-minute recording. Since such a file contains 216,000 samples in 758 periods, the number of the average samples in a period is

$$N_T = 216,000/758 \approx 285 \tag{6.52}$$

Before the NIFS approach is applied to the ECG signal, the domain pool is prepared based on the signal partitioning according to the complexity measure (see Sec. 6.7.1). Instead of constructing the traditional domain pool by shifting a block point-by-point in the time series, the blocks in the new domain pool are bounded by the 5 partitioning regions for each beat (frame), thus leading to an average block with $285/5 = 57$ sample points. Since the NIFS can use the most frequent range block size of 32 sample points to model the ECG signal as shown Fig. 6.7(b), the corresponding domain block size should be at least 64 samples, which is larger than the average size of the domain blocks in the bounded domain pool. Therefore, the matching search is performed by the NIFS on a few partitioned blocks. Often, an optimal mapping can be found by searching just a few

domain blocks (usually fewer than 10 domain blocks) by the NIFS as shown in Fig. 6.8(b). Compared to the complexity $O(N^2)$ by the traditional IFS, it is reasonable to consider that the complexity of the NIFS with signal partitioning is $O(N)$ for the ECG signal.

Now the NIFS is applied to the ECG signal using the reduced domain pool. Table 6.2 shows the influence of the order k of the Taylor series expansion and quantization resolution on the compression performance of an ECG signal. By changing the expansion order from 1st to 4th, as well as the resolution of the quantizer from 7 to 14 bits, various MPRD and R_{cr} are obtained. The maximal and minimal MPRDs are 20% and 5.6%, respectively. The maximal and minimal compression ratios are 6.5:1 and 3.7:1, respectively. By monitoring the change of the MPRD and R_{cr}, optimal parameters for the NIFS on the ECG signal can be found. Table 6.2 shows that when the resolution increases, the MPRD decreases and the R_{cr} increases first, and then decreases. A high resolution quantizer will benefit the MPRD, but not the R_{cr}. An optimal quantization resolution may be found based on the trade-off between the MPRD and the R_{cr}. When the quantizer takes more than 11 bits, the MPRD and R_{cr} change little with the increase of resolution. Thus, an 11-bit nonuniform quantizer with a Laplacian distribution is chosen for each kind of coefficients.

From the experiments in Sec. 6.6.3, we observe that a high expansion order k may decrease the MPRD because of the more powerful modelling ability of the NIFS. It is also observed that a higher order results in more coefficients, thus leading to (i) higher quantization error for the MPRD, and (ii) a lower compression ratio. Consequently, there must be a trade-off between the MPRD and the R_{cr} for choosing the optimal order k. Table 6.2

demonstrates that the increase in k results in the R_{cr} increasing first, and then decreasing. The expansion order of 2 is taken as the optimal parameter because it can achieve the highest compression ratio under about the same MPRD in the experiments.

It was found that the NIFS achieves higher compression ratio than the traditional IFS under the same reconstruction error in the ECG signal compression with the segmentation. When $PRD = 5.8\,\%$, Øien and Nårstad achieved 6.0:1 with their IFS ECG compression by orthogonal transform [Fish98]. We get the compression ratio about 5.5:1 for the linear affine transform with a 10 bit nonuniform quantizer and about 5.7:1 for the NIFS with an 11 bit nonuniform quantizer, both under order 2 and $MPRD = 5.8\,\%$. Thus, the NIFS outperforms the traditional IFS approach under the fast domain block search.

Table 6.2: Compression performance investigation of the combined VFDT and NIFS approach with quantization resolution and polynomial expansion change on an ECG signal (x_100.txt).

Resolution (bit)	k = 1		k = 2		k = 3		k = 4	
	MPRD	R_{cr}	MPRD	R_{cr}	MPRD	R_{cr}	MPRD	R_{cr}
7	12%	5.4	13%	6.1	12%	5.6	20%	3.7
8	6.2%	5.6	8.2%	6.5	7.8%	6.3	12%	4.6
9	6.0%	5.6	8.0%	6.3	7.7%	6.3	11%	4.7
10	5.7%	5.5	6.6%	6.1	6.6%	6.1	7.5%	4.8
11	5.6%	5.3	5.9%	5.7	6.2%	5.7	6.3%	4.9
12	5.6%	5.3	5.9%	5.7	6.2%	5.7	6.4%	4.9
13	5.6%	5.2	5.8%	5.6	6.2%	5.6	6.3%	4.8
14	5.6%	5.1	5.8%	5.5	6.1%	5.5	6.2%	4.8
Threshold	0.0115		0.0118		0.0123		0.0130	

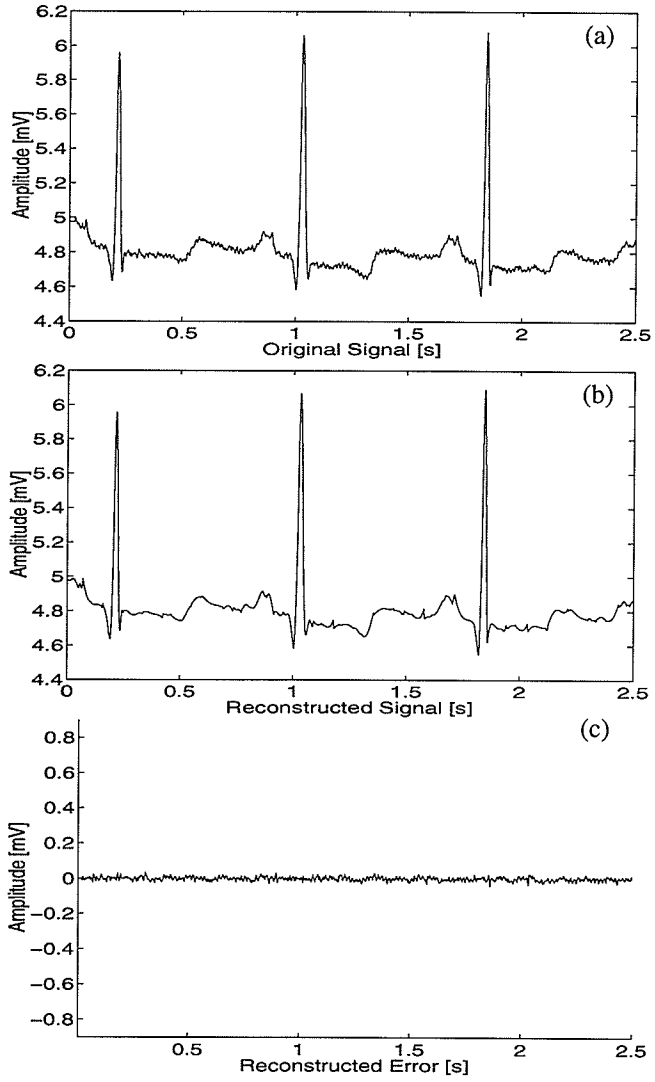

Fig. 6.14. The ECG signal compression with the combined VFDT and NIFS approach under expansion order 2 and 11 bit nonuniform quantizer: (a) original signal, (b) reconstructed signal, and (c) reconstruction error.

Figure 6.14 shows a compression and reconstruction of an ECG signal by the proposed scheme with optimal parameters. Unlike the reconstruction error obtained by the IFS, here the error is controlled to $[-33\mu V, 32\mu V]$, whose range is the same as the NIFS achieves under no signal partitioning. The QRS complex in the ECG signal is reconstructed almost perfectly.

We use a personal computer with Pentium® III of 600 MHz to run Matlab programs of the ECG compression algorithm. The linear IFS requires 3540 s to compress an ECG signal of length of 1024, while the segmented NIFS requires 106 s only, which is over 33 times faster.

6.8 Summary

This chapter presented fractal signal compression, beginning from basic techniques based on the self-similarity or self-affine property of a fractal object. Such a similarity may be represented through the iterated function systems (IFS) [BaHu85]. The collage theorem was introduced by Barnsley to describe the fractal object by collaging a set of contraction mapping transforms together because the practical fractal object is usually not strictly self-similar and may be composed of a series of mapping transforms of itself [Barn88]. To find the set of the mapping transforms, Jacquin proposed the fractal block coding (FBC) technique. The FBC divides the fractal object into domain pool and range pool. Each block in the range pool is compared to the transformed version of the blocks in the domain pool. The fractal coding corresponds to the transform of the most matched pair. The set of such transforms consists of the IFS. Therefore, a fractal object can be compressed and synthesized by a set of functions mapping it to itself. The contractivity of the IFS is necessary to ensure that the decoding procedure would converge to an adequate representation of the original fractal object. The Lloyd-Max algorithm is discussed and used to design a quantizer for our new fractal compression.

Mazel and Hayes developed a self-affine fractal model and a piecewise self-affine fractal model to encode a time series. The piecewise self-affine fractal model was applied

to the ECG signal. Although this approach is very time consuming, the decoding procedure is fast.

We proposed a new nonlinear IFS (NIFS) transform and a signal partitioning technique to improve the compression performance of the traditional IFS. Several important experimental results were obtained: (i) good convergence, (ii) high compression ratio at a small error, (iii) excellent reconstruction of the QRS complex, (iv) few search blocks, and (v) a fast algorithm for compression. The convergence of the strange attractor reconstruction of the NIFS has been achieved in all our experiments. The NIFS approach may compete with the traditional IFS technique in ECG signal compression in that, with an optimal choice of the order k and a nonuniform Laplacian quantizer, the extended transform achieves a maximal compression ratio of 7.3:1 under $MPRD = 5.8\,\%$, which is higher than that of 6.0:1 obtained by Øien and Nårstad under $PRD = 5.8\,\%$. Another advantage of the new extended transform is that it can model the QRS complex of the ECG signal very well, which has been a problem with the affine transform and the orthogonal transform in fractal compression. Still another advantage of the NIFS is that fewer blocks are required to model the ECG signal as compared to the linear IFS, thus providing more flexible modelling. Still another advantage stems from the address distribution of the optimally mapped domain blocks around the first point in the search as shown in Fig. 6.8(b) which can be employed to speed up the search of finding optimal mappings between the range blocks and domain blocks.

A combined scheme of the NIFS and signal partitioning approach based on the compositional complexity measure by the variance fractal dimension trajectory (VFDT) is

applied to compress the ECG signal rapidly. Compared to a much weaker self-similarity between regions with different complexities, a strong self-similarity of fractal object exists in the area with the same complexity. With the domain pool prepared by the VFDT, the compression ratio is 5.5:1 by the traditional IFS under $PRD = 5.8\,\%$, while 5.7:1 by the NIFS under $MPRD = 5.8\,\%$. The R_{cr} of the NIFS is slightly lower than that of 6.0:1 obtained by Øien and Nårstad. The domain search is reduced to $O(N)$ for a time series with length N, compared to the computational complexity of $O(N^2)$ of the original IFS. We conclude that the NIFS can model ECG better than the IFS.

CHAPTER VII

MOMENT-INVARIANT FEATURES AND NEURAL NETWORK CLASSIFICATION OF ECG

7.1 Introduction

The previous chapter presented the nonlinear iterated function systems (NIFS) compression technique and its application to the ECG signal. In that compression, the ECG signal is treated simply as a general 1D signal. The quasiperiodicity of the signal has not been employed. This chapter will consider ECG frame classification and compression since the signal is a beat-repeated signal.

ECG frame classification has important applications in the following three aspects of ECG signal processing: (i) compression, (ii) browsability, and (iii) recognition of cardiac rhythms for defibrillators. *Compression* can be achieved because there is a high correlation among ECG frames. It means that information about the ECG frame morphology may be exploited to compress ECG data. A high compression ratio may be achieved through ECG frame classification [HuKi02b] [HuKi99b]. Browse means to look at information from a source. *Browsability* is desired because manual analysis of the large amount of data recorded by a Holter monitor takes many hours. Through this ECG frame classification, an analysis of a case can be speeded up by browsing an ECG recording automatically. Thirdly, implantable *cardiac defibrillators* have created a new impetus for automatic recognition of cardiac rhythms. At present, most types of defibrillators use only rate criteria for detection of potentially lethal arrhythmias. Sometimes this results in false positive

detections and inappropriate delivery of electric therapy. If the device could distinguish better the unrecoverable from the benign arrhythmias, this would constitute progress. A neural network has already been used to do the arrhythmia classification in the implantable device by extracting features from only the QRS complex of ECG frames [Vouk95]. Our approach is to use the full frame information for the arrhythmia classification.

Some frame-based ECG signal compression techniques have been proposed. Cohen and Zigel establish a long-term prediction model for the ECG signal based on beat segments with a beat codebook. Instead of discarding the prediction error, the residual is encoded scalarly with some bits [CoZi98]. Beat-based classification technique provides another way to compress the ECG signal. As discussed in Sec. 2.6.3, neural network classification approach is used by Iwata *et al.* to compress the ECG signal [IwNS90]. The ECG beats can be classified into different classes and represented by class parameters in a compressed data file.

To classify ECG patterns, a similarity measure is often established through the smallest sum of the absolute differences between ECG beats. However, since the duration of the beats varies with time in practice, the estimates are inaccurate. The computation cost is also high. It is necessary to find an efficient technique to extract features from the ECG beat for classification purposes.

In Sec. 7.2, a *moment-invariant* (MI) technique is proposed to extract statistical morphology features of ECG beats. The MI was used originally in 2D digital image processing to extract the morphology information of an image [GoWo92]. The moment is invariant when the image translates, rotates, and scales. Based on these properties, the MI

is often used to match objects. Object matching technique based on the MI has been applied to image compression by many researchers, including Novak, Götting, and Popescu [Fish98]. It also has been applied to character recognition [TrJT96]. If ECG beats are treated as characters, the morphology information of the signal can be described by the MI. Such moment features have the same dimension for those beats with various durations. The dimension number of the features may be much less than the sample number of the beats, which allows the classification to be fast.

After feature extraction, a training set must be clustered for a classifier. There is no standard training set for the ECG beat because it changes with each individual and with time. Therefore, the training set of the ECG beat with MI features is clustered by the ISODATA algorithm [HaBa65] in an unsupervised mode (Sec. 7.3). Then such a training set is used to establish discriminant criteria for the classifier.

A proper classifier should be chosen for the ECG beat based on extracted features. The Bayes rule supplies a theoretical background for finding an optimal classifier. Currently, the neural network is one of the most suitable techniques to design classifiers. A *probabilistic neural network* (PNN) is used as a classifier in this thesis since it has useful characteristics drawn from both neural networks and statistical analysis as described in Sec. 7.4. The selected moment features constitute the input to the PNN for ECG beat classification.

A classification compression scheme for the ECG signal is shown in Fig. 7.1. The PNN should be trained by a clustered training set of the ECG beats first. Then a new ECG beat can be input to the PNN for classification. Finally, the input ECG beat is compared to

each beat in the class given by the PNN. We shall discuss the compression scheme by considering the MI feature extraction first, then ISODATA clustering, the PNN, its training and classification, and finally experimental results.

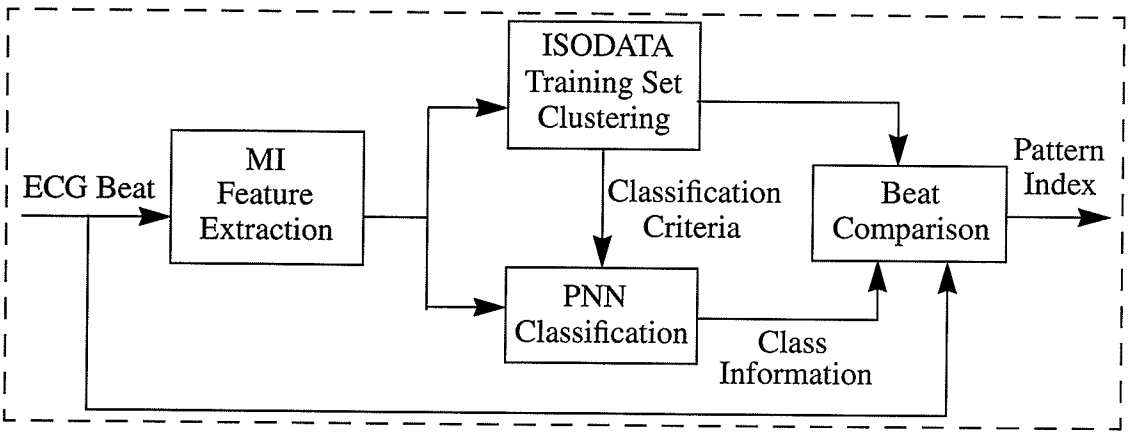

Fig. 7.1. Classification compression scheme for the ECG signal.

7.2 Moment-Invariant Features of ECG

In this section, the MI technique is proposed to extract the statistical morphology features of the ECG beat for classification of the ECG signal.

A beat of the ECG is defined as the interval between two successive R peaks (see Fig. 2.3). In this thesis, the beat is also called a *frame* or pattern. The quasiperiodicity of the ECG signal makes it possible to segment the signal into beats. The beat partitioning can be carried out since the R peak is the dominant point in the ECG signal. The partitioned beat can be grouped into various classes. However, the nonstationarity of the ECG signal poses a problem in the similarity measure between the beats. The duration change of the beats makes it impractical to measure the similarity directly through the smallest sum of the absolute differences of the two ECG beats.

Instead, the MI technique is proposed for ECG frame classification. The MI has been used in the past in 2D signals to extract morphology information of an image in a statistical sense. It is often applied to match objects because moments are invariant when the image translates, rotates, and scales [GoWo92]. It has been also applied to character recognition [TrJT96]. We extract MI features from the ECG beat since such features contain the shape information of the object, which is useful for similarity comparison between the beats. The dimension of MI features will not change with the length of the beat.

Let us first consider moments for 2D objects. For a 2D continuous function $f(x, y)$ shown in Fig. 7.2, the moment m_{ij} with ith order in x axis and jth order in y axis is defined as

$$m_{ij} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^i y^j f(x, y) dx dy \tag{7.1}$$

where the order i and j are nonnegative integers.

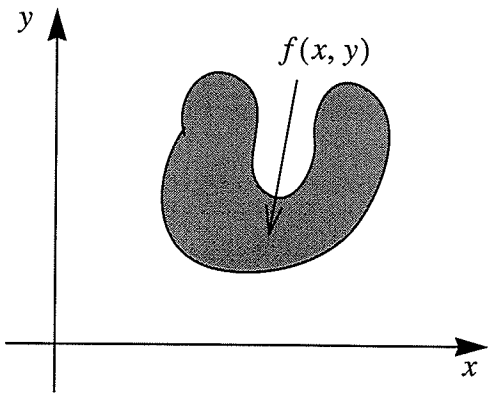

Fig. 7.2. An object in 2D plane.

Papoulis states that if $f(x, y)$ is piecewise continuous and has nonzero values only in a bounded part of the xy plane, moments of all orders exist and the moment sequence $\{m_{ij}\}$ is determined uniquely by $f(x, y)$. Conversely, $\{m_{ij}\}$ uniquely determines $f(x, y)$ [Papo65]. By shifting $f(x, y)$ to the centroid (μ_x, μ_y) of the object, a central moment μ_{ij} is given as

$$\mu_{ij} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x-\mu_x)^i (y-\mu_y)^j f(x, y)\,dx\,dy \tag{7.2}$$

where

$$\mu_x = m_{10}/m_{00}\,, \quad \mu_y = m_{01}/m_{00} \tag{7.3}$$

and m_{00} is the centroid of the object, m_{10} is the 1st order moment in x axis, and m_{01} is the 1st order moment in y axis. Notice that the integral operation in (7.1) and (7.2) should be replaced by a sum operation for digital signals; *i.e.*

$$m_{ij} = \sum_x \sum_y x^i y^j f(x, y) \tag{7.1a}$$

and

$$\mu_{ij} = \sum_x \sum_y (x-\mu_x)^i (y-\mu_y)^j f(x, y) \tag{7.2a}$$

where i and j are nonnegative integers.

The normalized central moment ξ_{ij} is given by

$$\xi_{ij} = \mu_{ij}/\mu_{00}^{\gamma} \tag{7.4}$$

where $\mu_{00} = m_{00}$, $\gamma = \frac{i+j}{2}+1$, and $(i+j) = 2, 3, \cdots$.

From the normalized central moments, a set of seven invariant moments can be derived [Hu62]. They are given by [GoWo92] [GoWo02]

$$\varphi_1 = \xi_{20} + \xi_{02} \tag{7.5}$$

$$\varphi_2 = (\xi_{20} - \xi_{02})^2 + 4\xi_{11}^2 \tag{7.6}$$

$$\varphi_3 = (\xi_{30} - 3\xi_{12})^2 + (3\xi_{21} - \xi_{03})^2 \tag{7.7}$$

$$\varphi_4 = (\xi_{30} + \xi_{12})^2 + (\xi_{21} + \xi_{03})^2 \tag{7.8}$$

$$\begin{aligned}\varphi_5 = {} & (\xi_{30} - 3\xi_{12})(\xi_{30} + \xi_{12})[(\xi_{30} + \xi_{12})^2 - 3(\xi_{21} + \xi_{03})^2] \\ & + (3\xi_{21} - \xi_{03})(\xi_{21} + \xi_{03})[3(\xi_{30} + \xi_{12})^2 - (\xi_{21} + \xi_{03})^2]\end{aligned} \tag{7.9}$$

$$\begin{aligned}\varphi_6 = {} & (\xi_{20} - \xi_{02})[(\xi_{30} + \xi_{12})^2 - (\xi_{21} + \xi_{03})^2] \\ & + 4\xi_{11}(\xi_{30} + \xi_{12})(\xi_{21} + \xi_{03})\end{aligned} \tag{7.10}$$

$$\begin{aligned}\varphi_7 = {} & (3\xi_{21} - \xi_{03})(\xi_{30} + \xi_{12})[(\xi_{30} + \xi_{12})^2 - 3(\xi_{21} + \xi_{03})^2] \\ & + (3\xi_{12} - \xi_{30})(\xi_{21} + \xi_{03})[3(\xi_{30} + \xi_{12})^2 - (\xi_{21} + \xi_{03})^2]\end{aligned} \tag{7.11}$$

To validate that the moments are invariant, we have performed initial experiments on the characters shown in Fig. 7.3 with their pixel values set to 1, and expressions for the

moment taken from [GoWi77]. We found that their φ_7 changes with the rotation of a character. Since the moment of an object ought to be invariant under rotation, we checked both the formula and the program and discovered a mistake in their formula for φ_7. Equation (7.11) is correct [GoWo92] and the wrong one is given as [GoWi77]

$$\varphi_7 = (3\xi_{12} - \eta_{30})(\xi_{30} + \xi_{12})[(\xi_{30} + \xi_{12})^2 - 3(\xi_{21} + \xi_{03})^2]$$

$$+ (3\xi_{21} - \xi_{03})(\xi_{21} + \xi_{03})[3(\xi_{30} + \xi_{12})^2 - (\xi_{21} + \xi_{03})^2] \qquad (7.11a)$$

Based on the correct expressions given by (7.5) to (7.11), an experiment has been implemented to extract the MI of the characters *A* to *D* shown in Fig. 7.3. The code is provided in Appendix B.4.1. Figure 7.3(a) is character *A*. The rotations of *A* are shown in Figs. 7.3(b)-(d), and its enlargement by a factor of 2 in Fig. 7.3(e). The values of the corresponding seven moments are listed in Table 7.1. It is seen that the moments are contained in the interval [-0.0024, 0.5505]. It is also seen that the variations of moments φ_5 to φ_7 are too small to be significant in character recognition. Furthermore, since the values of the moments φ_1 to φ_4 are almost the same for the rotated *A*, but different from *B*, *C*, and *D*, we can say that the moments contain the shape information of objects.

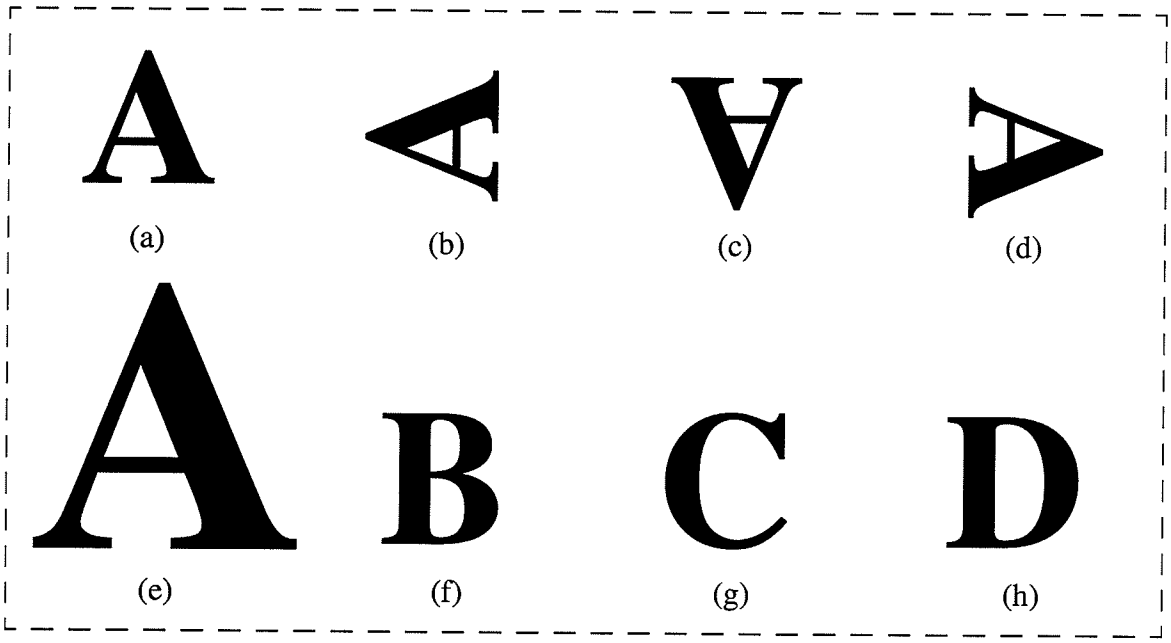

Fig. 7.3. An example of MI calculation for characters with the same size except (e) with double size. (a) is rotated 90°, 180°, and 270° counterclockwise to give (b), (c), and (d), respectively. (f)-(h) characters different from (a), but with the same orientation.

Table 7.1: The MI of characters corresponding to Figs. 7.3(a)-(h).

Moment / (Fig.)	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7
(a)	0.3693	0.0078	0.0356	0.0019	0.0000	-0.0001	0.0000
(b)	0.3693	0.0078	0.0356	0.0019	0.0000	-0.0001	0.0000
(c)	0.3693	0.0078	0.0356	0.0019	0.0000	-0.0001	0.0000
(d)	0.3693	0.0078	0.0356	0.0019	0.0000	-0.0001	0.0000
(e)	0.3685	0.0083	0.0343	0.0018	0.0000	-0.0001	0.0000
(f)	0.2894	0.0029	0.0007	0.0000	0.0000	-0.0000	-0.0000
(g)	0.5505	0.0161	0.0261	0.0318	-0.0004	-0.0024	-0.0008
(h)	0.3405	0.0000	0.0038	0.0000	-0.0000	-0.0000	-0.0000

Since the moment can be a shape descriptor of objects, it may be used to extract morphology information of the ECG waveform. Like characters, we treat the ECG

waveform shown in Fig. 7.4 as a 2D graphics and set value of 1 for the shaded area and 0 outside the shaded area. Notice that each waveform in the figure is the difference between the original ECG signal and its minimal value. Then, Eqs. (7.5)-(7.11) are applied to the waveforms to calculate the moments of the ECG beat. Figure 7.4 show different ECG beats. Their corresponding moments are listed in Table 7.2. It is observed that the seven moments from the same ECG waveform are different. The values of the moments also change with various waveforms. The range of the moments is in the interval $[-0.0029, 1.4524]$.

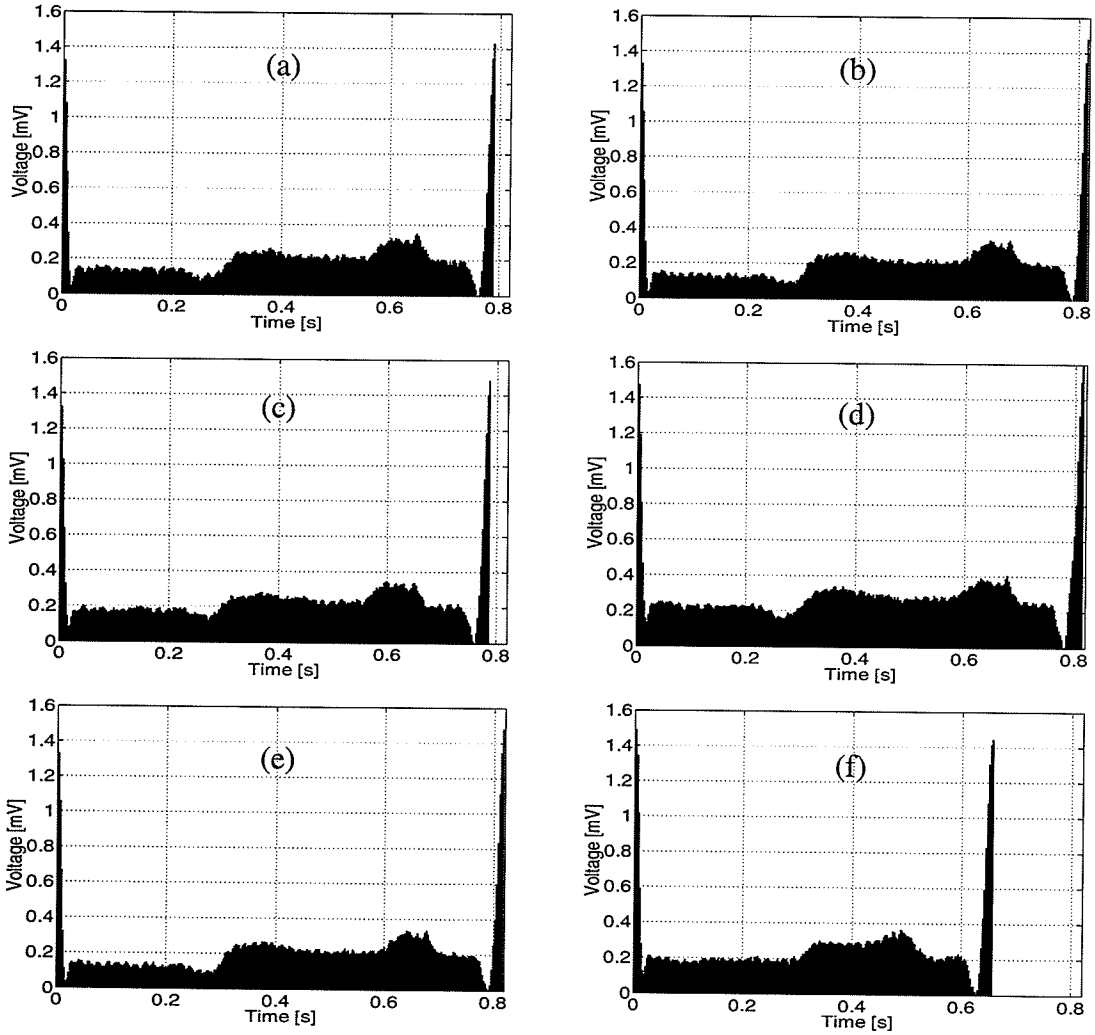

Fig. 7.4. An example of MI calculation for different ECG waveforms. The ECG waveform is treated as a 2D graphics with value of 1 in shaded area and 0 in other place.

Table 7.2: The MI of ECG waveforms corresponding to Figs. 7.4(a)-(f).

Moment / (Fig.)	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7
(a)	1.3334	1.3752	0.4099	0.2278	0.0481	0.1372	0.0503
(b)	1.3611	1.4524	0.4474	0.2492	0.0602	0.1673	0.0574
(c)	1.1551	1.0434	0.1889	0.0712	0.0017	0.0083	0.0081
(d)	1.0131	0.7975	0.1019	0.0376	-0.0001	-0.0029	0.0023
(e)	1.3611	1.4524	0.4474	0.2492	0.0602	0.1673	0.0574
(f)	1.0043	0.6221	0.1368	0.0770	-0.0028	-0.0267	0.0074

The purpose of calculating the MI is to see how similar one object is to another according to an appropriate metric. A *normalized Euclidean distance* (NED) is used to quantify similarity between two vectors, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$

$$NED = \sqrt{\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2}{|\boldsymbol{x}_1 \bullet \boldsymbol{x}_2|}} \times 100\,\% \tag{7.12}$$

To investigate similarity between ECG waveforms, we express the seven moments in Table 7.2 as a vector $\boldsymbol{x}$. A small NED means that the two vectors are similar. Since the waveforms in Figs. 7.4(a) and (b) are similar from a perceptual point of view, their NED is small 4.88%. Since Figs. 7.4(c) and (d) are somewhat different, their NED is 21.0%. The waveforms in Figs. 7.4(e) and (f) are quite different, and their NED is 64.8%. From this experiment, we conclude that the more similar the quasiperiodic ECG waveforms, the closer the values of their moments. Since the MI describes the shape information of ECG frames, we select it as features to classify quasiperiodic ECG patterns.

7.3 ISODATA Clustering Algorithm

Since we have found the MI-based approach suitable to distinguish between different ECG frames, discriminant criteria need to be established through the training set for classification. For this purpose, we must solve two problems: (i) the training set needs to be clustered in order to derive the discriminant criteria, and (ii) optimal discriminant criteria need to be established. The solution for the problems will be discussed in the next two sections.

An ECG signal is very complicated. Different people have different waveforms. Even the beats of the ECG measured from the same person change with time. Therefore, we cannot establish a standard set of ECG waveform patterns, and a training set can only be prepared by clustering observations in an unsupervised clustering technique, in which the clustering process does not depend on any prior information.

There are techniques for unsupervised clustering such as *one-pass clustering, K means, fuzzy C means, minimum distribution angle, ISODATA, self organization,* and *adaptive resonance* [Micr01]. Since the implementation of the ISODATA is more flexible than the simple one-pass clustering, *K* means, and minimum distribution angle, and also simpler than the fuzzy *C* means, self organization, and adaptive resonance, we choose it as our unsupervised algorithm to cluster the training set of ECG patterns.

The ISODATA clusters the training set according to two criteria: (i) minimizing the sum of squared distances of all points within a cluster domain to their cluster centre, trace $[M_W]$, and (ii) maximizing the sum of squared distances of cluster centres to the centre of

the entire data set, trace $[\boldsymbol{M}_B]$ where

$$\boldsymbol{M}_W = \sum_{j=1}^{N_C} p(\omega_j)\boldsymbol{M}_j \tag{7.13}$$

$$\boldsymbol{M}_B = \sum_{i=1}^{N_C} p(\omega_j)(\boldsymbol{u}_j - \boldsymbol{u}_0)(\boldsymbol{u}_j - \boldsymbol{u}_0)^T \tag{7.14}$$

where

$$\boldsymbol{u}_0 = \sum_{j=1}^{N_C} p(\omega_j)\boldsymbol{u}_j \tag{7.15}$$

and where N_C is the number of classes, $\boldsymbol{M}_j$ is covariance matrix of class ω_j, $\boldsymbol{u}_j$ is mean vector of class ω_j, and $p(\omega_j)$ is the *a priori* probability of class ω_j [Pawl99].

The ISODATA uses an iterative technique that incorporates a number of heuristic (trial and error) procedures to compute classes. Such a technique is similar to the *K* means approach, but incorporates procedures for splitting, combining, and discarding trial classes to obtain an optimal set of output classes [Fuku90].

The ISODATA algorithm analyzes samples of the input to determine a specified number of initial class centres. Patterns are assigned to classes by determining the closest class centre (according to the minimum Euclidean distance). After each classification step, the process calculates a new centre for each class by finding the mean vector. At the beginning of each iteration, the process evaluates the set of classes produced by the previous

iteration. Large classes may be split on the basis of a combination of factors, including the maximum standard deviation for the class, the average distance of class patterns from the class centre, and the number of patterns in the class. If the distance between the centres of two classes falls below a user-defined threshold, the classes are combined. If the number of patterns in a class falls below a user-defined threshold, the class is discarded, and its patterns are reassigned to other classes. Iterations continue until there is little change in the location of class centres in successive iterations, or until the maximum allowed number of iterations is reached.

In the ISODATA algorithm, 7 parameters must be initialized. They are: (i) the number of classes expected, N_{ce}; (ii) the minimum number of patterns in a class, N_p; (iii) the maximum standard deviation threshold, T_{sd}; (iv) the minimum distance threshold to combine two classes, T_d; (v) the maximum merging pairs of class centres, C_p; (vi) the number of allowable iterations, N_{ai}; and (vii) the separate coefficient, S_c [Bow92].

By changing the centres of the classes iteratively as well as merging and splitting the classes, the ISODATA gives the approximations of the minimal trace $[\boldsymbol{M}_W]$ and maximal trace $[\boldsymbol{M}_B]$ with clustered classes. The procedure of this algorithm is given as follow [Fuku90]:

1) Choose initial parameters and some initial clustering centres;
2) Assign pattern $\boldsymbol{x}$ to the nearest clustering centre. If

$$\|\boldsymbol{x}-\boldsymbol{u}_j\|_2 < \|\boldsymbol{x}-\boldsymbol{u}_i\|_2, \quad i = 1, 2, \cdots, N_C, \quad i \neq j \tag{7.16}$$

then $\boldsymbol{x} \in \omega_j$, where N_C is the number of classes in training set, and ω_j is a sample

subset with clustering centre $\boldsymbol{u}_j$ and N_j patterns;

3) Check if any cluster does not have enough samples. For any j, if $N_j < N_p$, then discard ω_j and let $N_C = N_C - 1$;

4) Renew each clustering centre by taking the average of the patterns in their domain

$$\boldsymbol{u}_j = \frac{1}{N_j} \sum_{\boldsymbol{x} \in \omega_j} \boldsymbol{x}, \quad j = 1, 2, \cdots, N_C \tag{7.17}$$

5) Compute the within distance d_j for each clustering domain ω_j

$$d_j = \frac{1}{N_j} \sum_{\boldsymbol{x} \in \omega_j} \|\boldsymbol{x} - \boldsymbol{u}_j\|_2, \quad j = 1, 2, \cdots, N_C \tag{7.18}$$

6) Compute the average distance d_{N_C} for the entire training set

$$d_{N_C} = \frac{1}{N} \sum_{j=1}^{N_C} N_j d_j \tag{7.19}$$

where N is the total number of patterns in the training set;

7) a. If $N_{ai} = 0$, set $T_d = 0$ and jump to Step 11,

b. If $N_C \leq N_{ce}/2$, jump to Step 8,

c. If $N_C \geq 2N_{ce}$ or N_{ai} is even, jump to Step 11; otherwise continue;

8) Compute the standard deviation σ_{ij}

$$\sigma_{ij} = \sqrt{\frac{1}{N_j} \sum_{\boldsymbol{x} \in \omega_j} (x_{ik} - u_{ij})^2}, \quad i = 1, 2, \cdots, n, \; j = 1, 2, \cdots, N_C \tag{7.20}$$

where n is the dimension of the feature space, x_{ik} is the ith component of the kth observation in ω_j, u_{ij} is the ith component of $\boldsymbol{u}_j$, and σ_{ij} is the ith component of standard deviation σ_j of ω_j;

9) Find the maximum component of σ_j, $j = 1, 2, \cdots, N_C$, expressed by σ_{jmax};

10) For any $\sigma_{jmax} > T_{sd}$, $j = 1, 2, \cdots, N_C$, if $d_j > d_{N_C}$ and $N_j > 2(N_p + 1)$, or $N_C \leq N_{ce}/2$, then split $\boldsymbol{u}_j$ into two new clustering centres $\boldsymbol{u}_j^1$ and $\boldsymbol{u}_j^2$, delete $\boldsymbol{u}_j$, let $N_C = N_C + 1$, and jump to Step 2; otherwise continue;

11) Compute the pairwise distances among all clustering centres d_{ij}

$$d_{ij} = \|\boldsymbol{u}_i - \boldsymbol{u}_j\|_2, \quad i = 1, 2, \cdots, N_C - 1, \; j = i + 1, \cdots, N_C \tag{7.21}$$

12) Take C_p as merging pairs of clustering centres if it satisfies $d_{ij} < T_d$, $[d_{i_1 j_1}, d_{i_2 j_2}, \cdots, d_{i_{C_p} j_{C_p}}]$, where $d_{i_1 j_1} < d_{i_2 j_2} < \cdots < d_{i_{C_p} j_{C_p}}$;

13) Begin from $d_{i_1 j_1}$ to merge the pair of clustering centres. The new centre is

$$\boldsymbol{u}_l^* = \frac{1}{N_{il} + N_{jl}} [N_{il}\boldsymbol{u}_{il} + N_{jl}\boldsymbol{u}_{jl}], \quad l = 1, 2, \cdots, C_p \tag{7.22}$$

Delete $\boldsymbol{u}_{il}$ and $\boldsymbol{u}_{jl}$, then let $N_C = N_C - 1$; and

14) If $N_{ai} = 0$, end the program; otherwise let $N_{ai} = N_{ai} - 1$, and jump to Step 2.

The ISODATA algorithm has been applied to cluster the set of the moments from an ECG signal with 505 beats. The code is provided in Appendix B.4.2. Figure 7.5 shows the clustering result for ECG patterns. Since we cannot visualize the 7D space, three directions with the larger moments, φ_1, φ_2, and φ_3, are selected to show the clustering. Figure 7.5(a) shows the clustering plot with four classes and Fig. 7.5(b) has twelve classes. Clustering output is decided mainly by the initial parameters including the number of expected classes, the minimum number of patterns in a class, and the maximum standard deviation threshold.

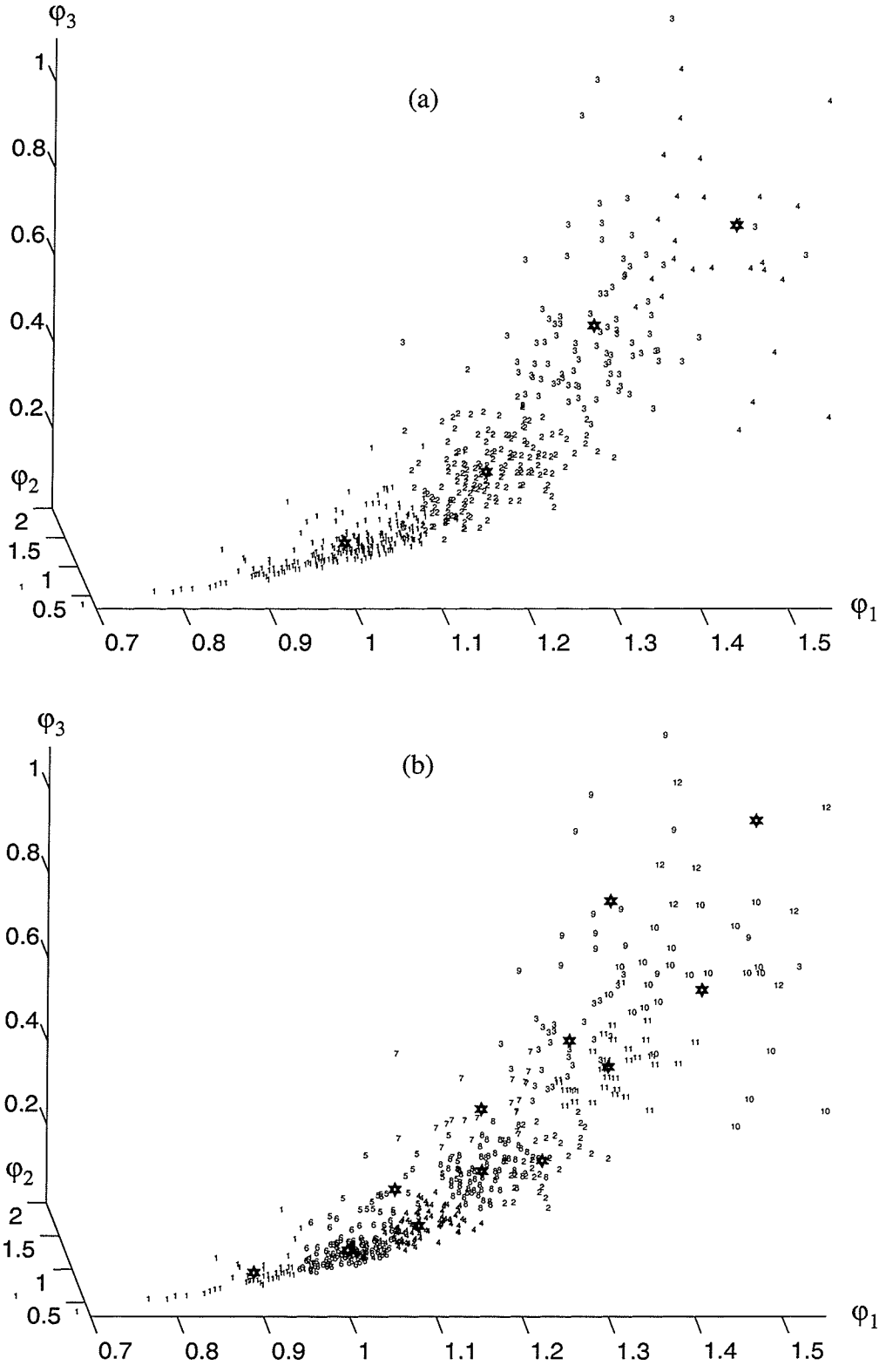

Fig. 7.5. Clustering plot of features for the first three MIs of 505 ECG beats with: (a) 4 classes and (b) 12 classes. The centre of the class is marked by pentagrams.

7.4 Neural Networks as Classifiers

With the clustered training set given by the ISODATA algorithm, a discriminant criterion may be established for the classifier to perform the desired ECG frame classification. The Bayes rule supplies a theoretical background for optimal classification. Statistical methods are important in classifier design. In addition, classification may include neural network computing techniques. Neural networks contain a large number of interconnected processing elements or nodes. Like real neural systems, an artificial neural network can learn by experience and develop criteria for making decisions. A neural network classifier can analyze events and develop classification criteria that do not require assumptions about the distribution statistics of the events.

A neural network consists of a set of nodes and the connections between them. Usually the nodes are grouped in layers, with connections from one layer to a subsequent layer. In a neural network classifier, there are usually at least three layers of nodes: input, hidden, and output layer. Each node connection has a weighting factor (weight) which multiplies the signal traveling along the connection. Ideally, a particular set of input values should cause the output layer to assign a value of 1 to the node corresponding to the correct class, and 0 to all the other nodes. During a learning phase, training patterns are passed through the net in a number of iterations, and various criteria are used to update the connection weights to strengthen a given connection for some patterns, and weaken it for others. This iterative updating eventually converges to a set of weights that can discriminate between patterns in the training set effectively, and then can be used to classify the entire data set.

A typical neural network used for classification is the multiple layer feedforward network (MLFN), called the backpropagation network (BPN). It has been proved to work well in many different applications. Such a network can model any linear or nonlinear deterministic function if sufficient training time and neurons are given. Spending long time for training is a problem for this technique in applications. A more suitable candidate for the classifier in ECG application is the probabilistic neural network (PNN). The PNN is based on a statistical algorithm first proposed in 1972 [Meis72]. Specht showed how the algorithm could be split up into a number of simple processes which could operate in parallel, much like in neural networks [Spec88].

7.4.1 Introduction to PNN

Figure 7.6 shows the architecture of the PNN. The exact number of neurons in each layer is determined by the dimension of a feature space, the number of patterns in each class, and the number of classes in the training set. The number of input neurons in the input layer, which serves no functional purpose other than to distribute input data to the next layer, is equal to the dimension number N_D of the feature space used to describe the objects to be classified. The pattern layer can be treated as a 2D array and contains one neuron for each pattern in the training set. It represents a certain number of training samples $(n_1, \cdots, n_{N_C})$ from each of the N_C different classes for a total number of N in the training set. This is a disadvantage of the PNN that the entire training set must be stored for classification usage, which requires a large amount of memory. Each neuron in the pattern layer measures a distance between an unknown input and the training pattern represented by that neuron. An activation function, known as the Parzen window, is then

applied to the distance measure. It may be another shortcoming of the PNN when the classification speed becomes slow in some cases [Mast93]. In the summation layer, there are N_C neurons. Each of them corresponds to one class of the training set. Such a neuron sums the values of the pattern layer neurons corresponding to that class to obtain an estimated density function of the class, which is desired by Bayes discriminant criterion. The width, σ, of the Parzen window is the only parameter needed to be decided for the PNN. It shows the simplicity of the PNN. The number of neurons in the output layer is also equal to the number of classes, N_C. The output layer is often a simple threshold discriminator which activates a single neuron to represent the projected class of the unknown sample. In more advanced implementations, the neurons in this layer can bias the results to compensate for prior class probabilities and the cost of misclassifying a pattern to a certain class [Mast95].

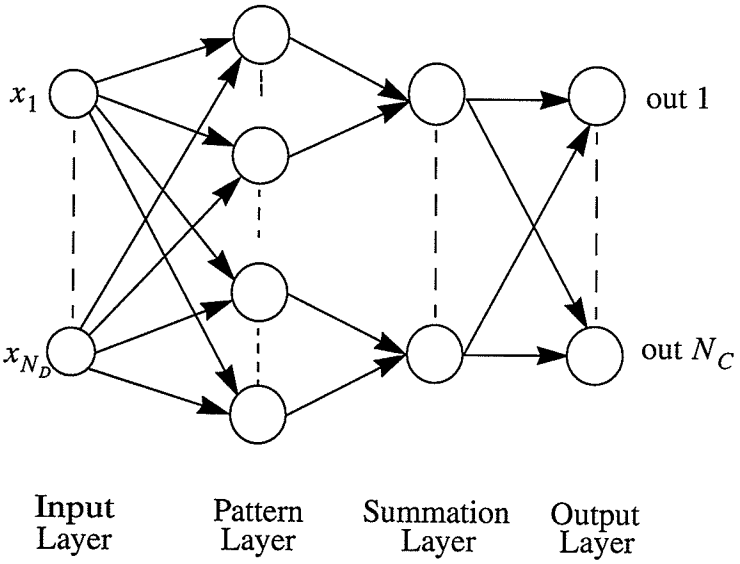

Fig. 7.6. The PNN architecture (from [KiCh00]).

The PNN has useful characteristics drawn from both neural networks and statistical analysis. First of all, the PNN has the neural network elegance of being able to handle even the most complex data distributions effectively. The importance for many applications is that it can often provide mathematically sound confidence levels for its decisions because it is based on established statistical principles. In fact, the classification abilities of the PNN approach the optimal Baysian classifier. Another advantage of the PNN is that it has a fixed architecture which allows its implementation to be relatively simple. Furthermore, in most situations, outliers have no real effect on decisions relative to the more frequent events. The most important advantage of the PNN is that its training speed is faster than that of the MLFNs, while maintaining the quality of classification.

Recall that for a simplified case, the summation layer requires an activation function which can be derived from Parzen's probability density function (PDF). In the summation layer of this simplified case, the activation function $\pi_k(\boldsymbol{x})$ for class ω_k with n_k patterns in the training subset is defined as

$$\pi_k(\boldsymbol{x}) = \frac{1}{n_k} \sum_{j=1}^{n_k} W\left(\frac{d(\boldsymbol{x}, \boldsymbol{x}_j)}{\sigma}\right) \tag{7.23}$$

where $\boldsymbol{x}$ is an unknown input vector, $\boldsymbol{x}_j$ is a training pattern belonging to ω_k, $d(\boldsymbol{x}, \boldsymbol{x}_j)$ is the distance measure between $\boldsymbol{x}$ and $\boldsymbol{x}_j$

$$d(\boldsymbol{x}, \boldsymbol{x}_j) = \|\boldsymbol{x} - \boldsymbol{x}_j\|_2 \tag{7.24}$$

$W(y)$ is the weight function (also known as the potential function or the kernel), most

commonly taken as the unnormalized Gaussian function given by

$$W(y) = \exp(-y^2) \tag{7.25}$$

and σ is the scale of the distance measure which decides the width of Parzen window that surrounds each sample point. The performance of the PNN is dependent on it. For different applications, the scaling parameter σ can be selected as: (i) the same for all classes and features, (ii) the same for all classes, but variant with different features, and (iii) variant for different classes and features. The selection of σ directly affects the training speed and classification speed of the PNN. The simpler the change of σ, the faster the PNN [Mast95].

The Bayesian confidence levels of the PNN can be defined as

$$\kappa_k(\boldsymbol{x}) = \frac{\pi_k(\boldsymbol{x})}{\sum_{j=1}^{N_C} \pi_j(\boldsymbol{x})} \tag{7.26}$$

A continuous classification error can be estimated from the normalized activation function. If a pattern $\boldsymbol{x}_j$ belongs to class ω_k, an ideal classifier should generate activations as $\kappa_k(\boldsymbol{x}_j) = 1$ and that $\kappa_l(\boldsymbol{x}_j) = 0$ for $l \neq k$. In most cases, we cannot achieve this ideal classifier. Thus, there is a classification error for the practical classifier. The error attributed to this observation is [Mast95]

$$e_k(\boldsymbol{x}_j) = [1 - \kappa_k(\boldsymbol{x}_j)]^2 + \sum_{l \neq k} \kappa_l^2(\boldsymbol{x}_j) \tag{7.27}$$

The total error for the given training set is

$$e(\sigma) = \sum_{k=1}^{N_C} \sum_{j=1}^{n_k} \left\{ [1 - \kappa_k(\boldsymbol{x}_j)]^2 + \sum_{l \neq k} \kappa_l^2(\boldsymbol{x}_j) \right\} \tag{7.28}$$

We see that the classification error is related only to the scaling parameter, σ.

7.4.2 PNN Training

In pattern recognition, we know that the higher the classification rate, the better the classifier. A minimal classification error may be achieved by training the neural network under a given training set. Training also means learning. Neural network learning is the process of adapting connection weights in response to sets of input sample values and resulting in sets of output values.

The basic method of training a neural network is trial and error. If the network is not behaving the way it should, the weighting of a random connection is changed by a random amount. If the accuracy of the network declines, undo the change and make a different one. Unfortunately, the number of possible weights rises exponentially as new neurons are added, making it impossible to construct large neural nets using the trial and error method. In the early 1980s, Rumelhart and Parker independently rediscovered an old calculus-based learning algorithm. The backpropagation algorithm compares the obtained result with the expected result. It then uses this information to modify the weights systematically throughout the neural network. It also can be used reliably to train networks on only a portion of the data, since it makes inferences. The resulting networks are often con-

figured correctly to answer problems that they have never been trained on specifically. Although this training takes only a fraction of the time that trial and error method takes, it is still time-consuming.

The PNN is designed to recognize natural groups of patterns in the input data, and to produce the same neural network output (class identification) in response to input of similar patterns. From (7.23) to (7.28), it can be found that the performance of the PNN is determined uniquely by the scaling parameter, σ. The σ (or a group of σ) may be found when minimizing the continuous classification error in (7.28) under a given training set by training the PNN. Since the total classification error $e(\sigma)$ of the PNN can be obtained, the minimal error problem may be converted as to find the global extreme of a function. That means to find the solutions of the following equation

$$\frac{d(e(\sigma))}{d\sigma} = 0 \tag{7.29}$$

Since $e(\sigma)$ contains a lot of exponential terms, it is difficult to solve (7.29) directly. We look for a numerical solution of (7.28).

There are many candidate techniques for finding extremes for a function including the *fixed step, steepest descent, Newton-Raphson, momentum,* and *conjugate gradient.* Mathematically, the neural network training refers to the process of finding the minimum of a function. Some important issues in the PNN training include: (i) the estimated extreme of classification error should be global, and (ii) the training speed should be fast. There is no effective method to search the global minimum for a function $f(\boldsymbol{x})$. Three

main techniques, such as the steepest descent, the conjugate gradient, and the Newton-Raphson, are discussed next.

The steepest descent is the simplest in gradient methods. The choice of direction for updating $\boldsymbol{x}$ is where f decreases most quickly, which is in the direction opposite to the gradient $\nabla f(\boldsymbol{x}_i)$. The search starts at an arbitrary point $\boldsymbol{x}_0$, and then slides down the gradient, until it is close enough to the solution. In other words, the iterative procedure is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \lambda_k \nabla f(\boldsymbol{x}_k) \tag{7.30}$$

where $\nabla f(\boldsymbol{x}_k)$ is the gradient at a given point $\boldsymbol{x}_k$.

The reason why the steepest descent converges slowly is that it has to take a right angle turn after each step, and consequently search in the same direction as earlier steps. The conjugate gradient is an attempt to mend this problem by "learning" from experience.

Conjugacy means that two unequal vectors, $\boldsymbol{d}_i$ and $\boldsymbol{d}_j$, are orthogonal with respect to any symmetric positive definite matrix, for example $\boldsymbol{Q}$, if

$$\boldsymbol{d}_i^T \bullet \boldsymbol{Q} \bullet \boldsymbol{d}_j = 0 \tag{7.31}$$

This can be looked upon as a generalization of orthogonality, for which $\boldsymbol{Q}$ is the unity matrix. The idea is to let each search direction $\boldsymbol{d}_i$ be dependent on all the other directions searched to locate the minimum through (7.31). A set of such search directions is referred to as a $\boldsymbol{Q}$-orthogonal, or conjugate, set.

The algorithm of conjugate gradient is as follows:

1) Initialize

$$\boldsymbol{d}_0 = -\nabla f(\boldsymbol{x}_0) \tag{7.32}$$

2) Determine the step length

$$\min_{\lambda_k > 0} f(\boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k) \tag{7.33}$$

3) Update the point and gradient

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k \tag{7.34}$$

4) Determine the direction of search

$$\boldsymbol{d}_{k+1} = -\nabla f(\boldsymbol{x}_{k+1}) + \beta_k \boldsymbol{d}_k \tag{7.35}$$

where

$$\beta_k = \frac{(\nabla f(\boldsymbol{x}_{k+1}))^T \bullet (\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k))}{(\nabla f(\boldsymbol{x}_k))^T \bullet \nabla f(\boldsymbol{x}_k)} \tag{7.36}$$

The Newton-Raphson technique differs from the steepest descent and conjugate gradient approaches. In the Newton-Raphson technique, the information of the second derivative is used to locate the minimum of the function $f(\boldsymbol{x})$. This results in faster convergence, but not necessarily in less computing time. The computation of the second derivative and the handling of its matrix can be very time-consuming, especially for large systems.

The idea behind the Newton-Raphson method is to approximate the given function $f(\boldsymbol{x})$ in each iteration by a quadratic function, and then move to the minimum of this quadratic. The Newton-Raphson technique uses the following iterative formula

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathbf{H}_k^{-1} \bullet \nabla f(\boldsymbol{x}_k),\ k = 0, 1, 2, \cdots \tag{7.37}$$

where $\mathbf{H}_k^{-1}$ is the Hessian matrix and $-\mathbf{H}_k^{-1} \bullet \nabla f(\boldsymbol{x}_k)$ is often referred to as the Newton direction.

The size of the Hessian matrix is crucial to the effectiveness of the Newton-Raphson technique. For systems with a large number of dimensions; *i.e.* that the function $f(\boldsymbol{x})$ has a large number of variables, both the computation of the matrix and the calculation that includes it will be very time-consuming. This can be mended by either just using the diagonal terms in the Hessian; *i.e.* ignoring the cross terms, or just not recalculating the Hessian at each iteration (which can be done due to slow variation of the second derivative). Another serious disadvantage of the Newton-Raphson is that it is not necessarily globally convergent, meaning that it may not converge from any starting point.

Since the conjugate gradient is more efficient in the above techniques to find a extreme for a function [Mast95], it is suitable for the PNN training.

7.4.3 PNN and Minimal Residual Classification

To our knowledge, the PNN has never been used in ECG data compression before. In this thesis, one of the ECG compression techniques is to apply the PNN to ECG frame classification. Then, the class information is employed to compress the signal. In this tech-

nique, signal reconstruction is an open question. A class consists of a subset of the training set. Since such a subset contains more than one frame, a classified ECG frame cannot be reconstructed through the class information directly. Two techniques are proposed to reconstruct the input ECG frames.

First, we propose a *mean frame* technique to reconstruct the ECG signal [HuKi99b]. There are many frames in a class of the training set. The mean frame is introduced to give a representation for such a class based on the patterns in the class. The mean frame period is calculated from all the patterns belonging to that class. Then each frame in the class is contracted/dilated linearly to the mean period. The mean frame of the class is achieved by averaging the processed frames.

Now we connect the unknown frame to the mean frame through a class index. To improve ECG reconstruction quality, the length, amplitude, and mean of the unknown input are extracted and preserved for reconstruction. Although the compression ratio of over 70:1 can be achieved by using four parameters to represent the ECG frame, this technique cannot control the local error amplitude in the reconstructed signal [HuKi99b].

Second, the *minimal residual classification* technique is proposed to reduce the reconstruction error in ECG frame classification and compression. After PNN classification, the unknown ECG frame is compared further to each pattern in the classified class of the training set. Before the comparison, each pattern is contracted/dilated linearly to the length of the input frame. The minimal residual and corresponding pattern are then determined. Such a pattern is used to reconstruct the ECG signal. Thus, in the minimal residual classification, we use the class index, pattern index, length, amplitude of the input frame,

and the sum of the mean of the input frame and the mean of the residual, to reconstruct the ECG signal. Although this scheme needs five parameters to represent the ECG frame, the reconstruction error is improved significantly.

7.5 Experimental Results and Discussion

ECG frame classification and compression experiments have been conducted. The ECG data sampled at 360 sps with 11 bps are taken from the MIT-BIH ECG database [Mood99]. The experiment uses the ECG signal contained in the file, x_100.txt, which has a 10-minute recording of ECG data. Since this file contains 758 ECG periods, it can be segmented into 758 frames, numbered sequentially from Frame 1 to Frame 758. The seven MIs of ECG frames are extracted from each period of the ECG signal.

We cannot use a supervised approach to evaluate the classifier since: (i) there is no standard set of ECG frames, and (ii) the ISODATA and the PNN use different classification rule. The classification error becomes less important since the class itself cannot be defined strictly. Instead, an important parameter for evaluating the classifier is the reconstruction error in this classification compression scheme.

To use the PNN for classification, a training set and a testing set should be prepared. We use the *hold-out technique* to allocate the 758 ECG frames [Pawl99]. The hold-out technique allocates 2/3 of the sample set to the training set and 1/3 to the testing set. The ISODATA algorithm is used for clustering the training set. By changing the initial values for clustering, we find that the final class distribution is decided mainly by the initial parameters; *i.e.*, the number of expected classes, standard deviation threshold, and dis-

tance threshold between the class centres.

The Matlab V6.1.0 supplies a neural network toolbox that is convenient for PNN simulation. PNN training and classification can be done just by calling functions.

Since the classification is for the purpose of compression, it is necessary to reconstruct waveforms and calculate reconstruction error. Other than the classification error, the reconstruction error is another important parameter to indicate the feasibility of this scheme. The reconstruction error is measured according to (2.14) to (2.16). Five parameters, the class index, pattern index, length, amplitude of the input frame, and the sum of the means of the input frame and the residual, are preserved to reconstruct a frame of the ECG signal. Without an optimal bit allocation, the class index, pattern index, and length are represented by fixed length of 8 bits each. The amplitude of the frame and the sum of the means are allocated 11 bits, the same as the original data. It should be noticed that the patterns in the training set used as a frame dictionary must be known to both the encoder and decoder in the compression. According to (2.13), the compression ratio without considering entropy encoding is

$$R_{cr} = \frac{70204 \times 11}{254 \times (8+8+8+11+11)}$$

$$= 66.1:1$$

where 70204 and 254 are the length and the number of frames of the testing set, respectively. Since the bit allocation of the parameters is not optimized, this R_{cr} is the worst case. An optimal bit allocation for each parameter would require entropy calculations for each frame.

The classification and compression are performed on ECG frames. The code is provided in Appendices B.4.3-4. Table 7.3 gives the classification time, classification error, and reconstruction error in our experiment. By changing the number of classes, the classification time of the 505 ECG frames varies from 84.7 to 384 s and the MPRD varies from 14.65% to 16.35% by a Matlab program running on the Pentium(R) III computer as described in Sec. 6.7.2. When the number of the classes increases, the classification error and reconstruction error of the testing set increase. On the other hand, the classification time decreases with increasing classes. One of the error sources for classification is introduced due to the incompletely consistent criterion used by the ISODATA algorithm and the PNN, as described in Sec. 7.3 and Sec. 7.4.1. Another reason is the somewhat small training set. The reconstruction error increases with classes due to fewer patterns remained in some classes. Considering the trade-off between reconstruction error and classification time, 14 classes are a good choice from the experimental results in our case.

The experimental results have demonstrated that the PNN is a good classifier for compression because of its: (i) low reconstruction error: when C = 14, we get MPRD = 15.94%; (ii) short training time: it takes less than 1 minute for training the PNN for the 505 events in the training set by using the above mentioned Pentium(R) III computer; and (iii) short classification time: the same computer can complete the classification of one observation within a second.

Table 7.3: Classification error and reconstruction error of the hold-out estimate for ECG data, where number of patterns = 758, PRD = the percent root-mean-square difference, NPRD = normalized PRD, and MPRD = mean removed PRD.

No. of Classes	Classification Time [s]	Classification Error [%]	PRD [%]	MPRD [%]	NPRD [%]
4	386	3.56	0.55	14.65	1.71
6	332	8.30	0.56	14.93	1.73
8	187	6.72	0.58	15.61	1.80
10	144	10.3	0.59	15.88	1.83
12	125	13.0	0.60	16.16	1.86
14	126	10.7	0.59	15.94	1.84
16	98.6	13.4	0.61	16.40	1.88
18	88.3	15.0	0.60	16.01	1.85
20	84.7	15.4	0.61	16.35	1.88

A high compression ratio of 66.1:1 is obtained by using five parameters for each beat to reconstruct the ECG signal. Figure 7.7 shows the original and reconstructed ECG signal which is processed through the feature extraction, PNN classification, and minimal residual techniques. The 5 beats are successively taken from a recording. The reconstruction error is measured by the PRD. The PRD ranges from 0.45% to 0.61% in this case. Figure 7.7(a) demonstrates a good reconstruction for an input ECG frame. Its reconstruction error is shown in Fig. 7.7(b). It is observed that the error distribution is not even and the larger error occurs at transition regions of the signal such as the QRS complex, the P wave, and the T wave. The same comments can be made from other reconstruction examples in Fig. 7.7.

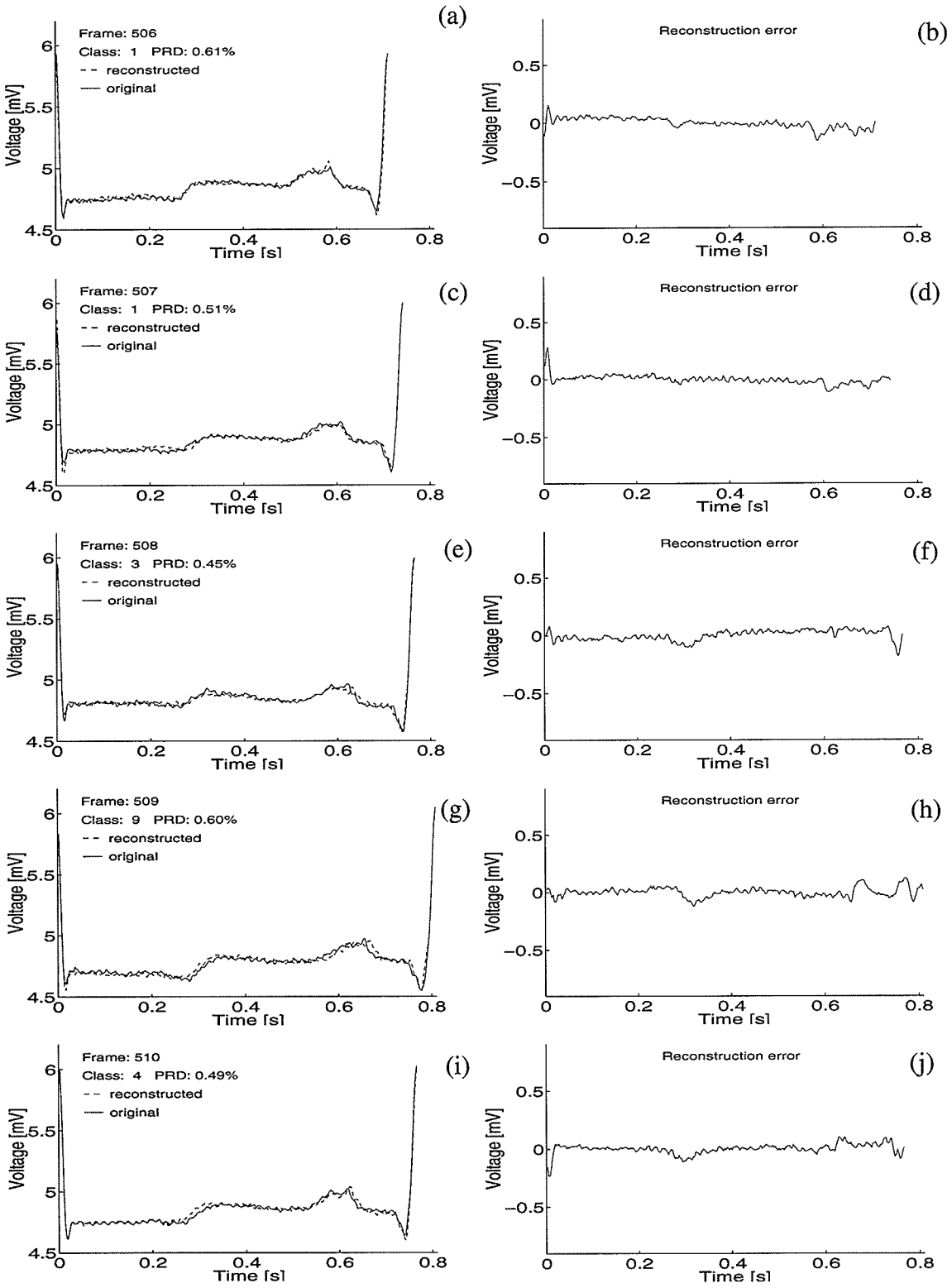

Fig. 7.7. Comparison of the original and reconstructed waveforms for the classification compression of ECG frames with 10 classes. The left figures show 5 sequential ECG frames and their reconstructions. The corresponding reconstruction errors are shown in the right figures.

The ISODATA algorithm can complete clustering automatically, under some initial parameters. It has the ability of merging and splitting. However, it is only suboptimal when using the criteria of minimizing trace $[\boldsymbol{M}_W]$ and maximizing trace $[\boldsymbol{M}_B]$ separately. It is time-consuming to find a global optimization by adjusting the initial parameters manually. Consequently, an automatically unsupervised clustering algorithm is needed in light of the following criterion [Pawl99]

$$\max(\text{trace}\,[\boldsymbol{M}_W^{-1}\,\boldsymbol{M}_B]) \tag{7.38}$$

7.6 Summary

This chapter presents an attempt to use MI features and the PNN to classify, compress, and browse ECG signals. The duration of the ECG beat changes with time because of the nonstationarity of the ECG signal. The MI technique solves the problem by extracting features from the ECG beat with the same dimensionality. The MI features represent the morphology information in the ECG signal. The same number of feature dimension makes the ECG classification possible using neural networks. The dimension number of the features may be much smaller than the sample number of the beats, which allows a fast classification implementation.

Before the ECG classification by the PNN, the training set of the ECG signal needs to be clustered. There is no standard training set for the ECG beat since it changes with each individual and with time. Therefore, the training set of the ECG beat with MI features is clustered by the ISODATA algorithm [HaBa65] in an unsupervised mode. Then such a training set is used to establish discriminant criteria for the neural network classi-

fier.

The PNN is trained by the clustered training set first. Then, it is used to classify the ECG frames. Experiments show that it can classify the ECG beats in terms of the MI features. The training speed of the PNN is fast. The classification results of ECG frames can be employed to compress or browse the ECG signal. A high compression ratio of about 66:1 is achieved with an MPRD of about 16%. The ECG compression based on beat classification can be implemented in real time.

The next chapter presents another classification and compression method in which a dynamic time warping (DTW) is applied to register the ECG frames in time for classification. Then a block encoding is proposed to compress the signal based on segments of the frame. Instead of classifiying the ECG frames based on their statistical characteristics, the DTW technique will be used to study the ECG signal through a nonlinear mapping in the next chapter.

CHAPTER VIII

DYNAMIC TIME WARPING CLASSIFICATION AND BLOCK ENCODING FOR ECG FRAME

8.1 Introduction

The previous chapter presented the MI feature extraction and PNN classification techniques for the classification and compression of the ECG frame. In order to explore further improvements, we now propose another ECG frame classification and compression method based on a *dynamic time warping* (DTW) matching, a *windowed-variance* smoothing, and a *block encoding* techniques.

As we have already discussed, a major problem in ECG frame classification is the nonstationarity of ECG frames, which means the Euclidean distance is not adequate to compare the similarity between two ECG frames due to the inconsistency of the frame length. It is noticed that heartbeat variation causes nonlinear time fluctuation of ECG frames. Elimination of this fluctuation by linear time-normalization has been one of the techniques for ECG classification research. However, such a linear normalization technique in which the timing difference between ECG frames is minimized is inherently insufficient for dealing with the highly nonlinear ECG frame fluctuations. On the other hand, the DTW matching, as discussed in this chapter, is a pattern matching algorithm with a nonlinear time-normalization effect. It has been used successfully in speech recognition [SaCh71]. In this algorithm, the fluctuation in time is modelled approximately by a nonlinear warping function with some carefully specified properties. The timing differ-

ence between any two ECG frames is minimized by warping the time duration of one frame such that the maximum coincidence is attained with the other frame. Then the time-normalized distance is calculated as the individual minimized residual distance between them. This minimization process is carried out efficiently by the DTW technique [SaCh78].

A new classification criterion for the DTW, the mean of absolute residual, is obtained by searching an optimal or suboptimal mapping path between a warped ECG frame and a template from the end of the frame to its start. The ECG frame is placed into the class with which it has minimal residual distance, if the residual is within the range of a certain residual threshold. Otherwise, a new class is generated.

Since the nonstationarity of the ECG signal makes the warping function complicated, compression of the ECG frame by a direct use of the warping function is difficult. Instead of using the warping function directly, the ECG frame can be classified approximately by time-normalizing the QRS complex, the T wave, and the P wave, only. Consequently, an effective partitioning of the ECG frame into the segments is required for such an approximate normalization. A windowed-variance technique is proposed to smooth the ECG signal, which makes the segmentation of the ECG signal simple. Then a block encoding is applied to the segments of the ECG signal to encode and reconstruct the signal.

8.2 DTW Mapping

In this section, we will discuss the DTW matching algorithm for ECG frame clas-

sification. One of the most fundamental concepts in the nonlinear pattern recognition is "time-warping" an input pattern to a reference pattern so as to align the two patterns in time. The DTW proposed by Sakoe and Chiba is one of the most versatile algorithms in speech recognition [SaCh71]. Figure 8.1 illustrates the basic idea about the time warping.

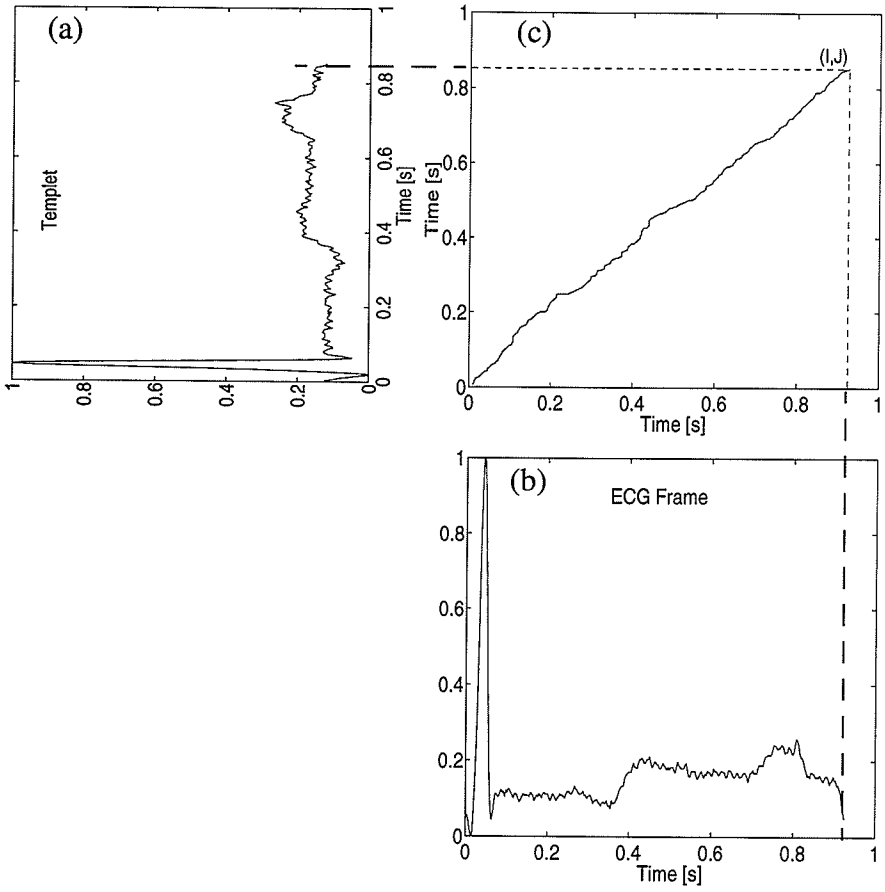

Fig. 8.1. Illustration of time warping. (a) Reference (template) ECG frame. (b) Input ECG frame. (c) Time warping function.

Figure 8.1(a) represents a template (reference) ECG frame obtained through a residual threshold technique, as to be described in Sec. 8.3.1. Figure 8.1(b) is the current input ECG frame which can be aligned (warped) to match the template as closely as possible, according to the warping function shown in Fig. 8.1(c). If the warping function is sim-

pler than the input frame, it can be encoded more compactly than the input frame. Consequently, the frame compression or classification can be done more efficiently. In general, there may be more than one template in a codebook (dictionary).

Let us denote the template of the ECG signal by $\{x_t(j),\ 1 \le j \le J\}$, and an unknown frame of the signal as $\{x(i),\ 1 \le i \le I\}$. The purpose of the time warping is to provide a mapping between the time indices i and j such that a time registration (alignment) between the template and the unknown frame of ECG signals is obtained. We denote the mapping by a sequence of points $v = (i, j)$, between i and j as [SaCh78]

$$M = \{v(k), 1 \le k \le K\} \tag{8.1}$$

where

$$v(k) = (i(k), j(k)) \tag{8.2}$$

This sequence can be considered representing a function which approximately realizes a mapping from the time axis of template $x_t(j)$ onto that of frame $x(i)$. It is called a warping function.

A complete specification of the warping function results from a point-by-point minimal distance measure between the template $x_t(j)$ and the frame $x(i)$. A similarity measure, or a distance function, d, for every pair of points (i, j) within the adjustment window shown in Fig. 8.2 is given by

$$d(v(k)) = d(i(k), j(k))$$

$$= \left\| x(i(k)) - x_t(j(k)) \right\|_2 \tag{8.3}$$

where $\|\bullet\|_2$ is the Euclidean norm.

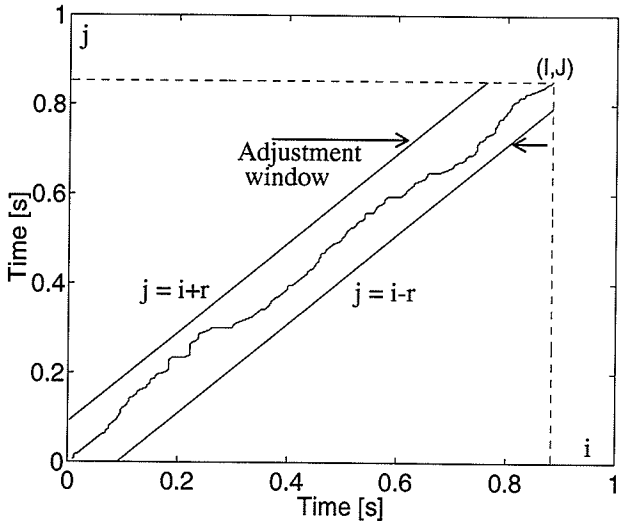

Fig. 8.2. Adjustment window between a template and an input frame of the ECG signal.

The smaller the value of *d*, the higher the similarity between $x(i)$ and $x_t(j)$. Given the distance function *d*, an optimal dynamic path *m* is chosen to minimize an accumulated distance d_a along the path [RaRL78]

$$d_a = \min_{\{M\}} \sum_{k=1}^{K} d(v(k))W(k) \tag{8.4}$$

where $W(k)$ is a nonnegative weighting coefficient, which is introduced intentionally to make the d_a measure flexible.

An effective technique for determining the optimal path is *dynamic programming*

[SaCh71]. Using this technique the accumulated distance to any grid point (i, j) can be determined recursively as [RaRL78]

$$d_a(v(k)) = d(v(k)) + min(d_a(v(k-1))) \tag{8.5}$$

where $d_a(v(k))$ is the minimal accumulated distance to the grid point $(i(k), j(k))$ and has the form

$$d_a(v(k)) = \sum_{n=1}^{k} d(v(n))W(n) \tag{8.6}$$

The time variable of the ECG signal has properties such as continuity, monotonicity, and limitation on the ECG transition. These conditions give the following restrictions on the warping function [SaCh78]:

1) Monotonic condition expressed by

$$i(k-1) \le i(k) \text{ and } j(k-1) \le j(k) \tag{8.7}$$

2) Continuity condition expressed by

$$i(k) - i(k-1) \le 1 \text{ and } j(k) - j(k-1) \le 1 \tag{8.8}$$

Notice that the above two restrictions lead to the following relation between two consecutive points

$$v(k-1) = \begin{cases} (i(k), j(k)-1) \\ (i(k)-1, j(k)-1) \\ (i(k)-1, j(k)) \end{cases} \tag{8.9}$$

3) Boundary conditions expressed by

$$i(1) = 1, j(1) = 1$$

$$i(K) = I, j(K) = J \tag{8.10}$$

4) Adjustment window condition expressed by

$$|i(k) - j(k)| \leq N_s \tag{8.11}$$

where N_s is an appropriate positive integer called *window size*. This condition corresponds to the fact that time-axis fluctuation does not cause an excessive timing difference.

5) Slope constraint condition that could be described as

"Neither too steep nor too gentle a gradient is allowed for the warping function M because such deviation may cause undesirable time-axis warping."

Sakoe and Chiba demonstrated that when the slope is equal to 1 and the search path takes a symmetric form, the DTW algorithm reduces the number of paths to be searched and obtains optimal results [SaCh78]. The weighting coefficient of the symmetric form is

$$W(k) = (i(k) - i(k-1)) + (j(k) - j(k-1)) \tag{8.12}$$

The symmetric DTW equation with slope of 1 is

$$d_a(v(k)) = d(v(k)) +$$

$$min\begin{pmatrix} d_a(i(k-1), j(k-2)) + 2d(i(k), j(k-1)) \\ d_a(i(k-1), j(k-1)) + 2d(v(k)) \\ d_a(i(k-2), j(k-1)) + 2d(i(k-1), j(k)) \end{pmatrix} \tag{8.13}$$

Finally the optimal accumulated distance d_a is normalized by $(I+J)$ for a symmetric form.

8.3 ECG Frame Classification by DTW

Since the DTW technique has been applied in speech recognition successfully, we use it to classify ECG frames. The ECG and speech signals have similar characteristics such as: (i) they are nonstationary and originate from nonlinear systems, and (ii) they can be divided into frames, with the separability of ECG frames much easier than the separability of speech utterances such as sentences, isolated words, or phonemes in connected (continuous) speech.

8.3.1 Modified DTW Classification Rule for ECG

To classify an ECG frame, the DTW is used to calculate an optimal distance between the ECG frame and a template. However, we found that the optimal distance of many frames does not correspond to the final point (I, J). Figure 8.3 shows this case.

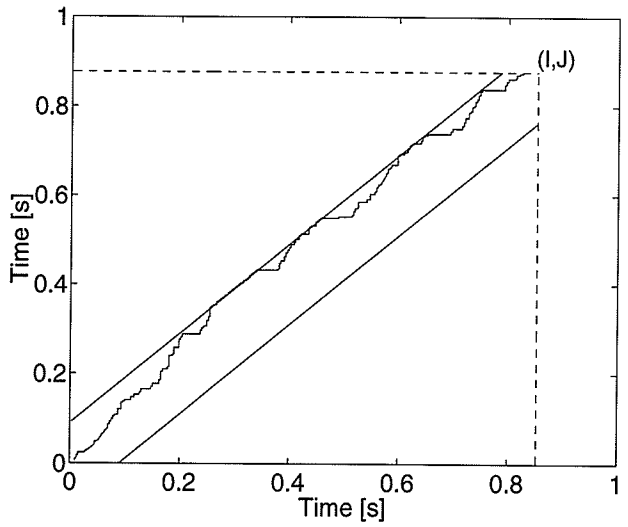

Fig. 8.3. Warping function obtained by searching the mapping path from the start to the end. This is an example of the final point (I, J) that does not correspond to the optimal distance.

It means that sometimes the optimal mapping exists between one frame and only a part of another. Therefore, the warping rule must be modified for ECG frames. Unlike the original DTW, we first calculate the minimal accumulated distance to each grid point in the adjustment window according to (8.6), and then search the mapping path, starting from the final point (I, J), which guarantees that at least a suboptimal mapping path can be found if (I, J) does not correspond to the optimal mapping. We now obtain two new mapping series, $\{j(k),\ 1 \le k \le K\}$ and $\{i(k),\ 1 \le k \le K\}$. The corresponding residual is $\{R(k),\ 1 \le k \le K\}$. The mean of absolute residual, $\overline{R}$, is calculated and taken as the classification criterion for ECG frames.

We have also modified the classification criterion for the DTW, originally used in the speech recognition. The minimal accumulated distance is not used as the classification criterion for the ECG frame since such a distance contains a weighting factor.

There are also pronounced differences between the DTW for speech and ECG. In speech, the boundaries of isolated utterance are imprecise due to the ambient noise and the common detection technique based on the cumulative power of the signal. On the other hand, although the ECG signal is also contaminated by noise, its frame detection technique is very accurate as it looks for the well-defined maxima in the QRS complex. Consequently, the DTW in speech can proceed either from the start of a frame to its end or in the opposite direction with similar penalty, provided the bounds are well defined, while the DTW in ECG must be done from the end of a frame to its start to give a warping for the entire frame.

The frame classification and establishment of a template set are carried out *at the same time*. A *classification threshold* T_c is setup to decide whether an input frame is a new class or can be classified to some class. The T_c is related to the residual error directly. The classification procedure consists of the following four steps:

1) Partition an ECG frame by detecting the R peak;
2) Warp the unknown input ECG frame with a template frame in time;
3) Calculate the mean of absolute residual, $\bar{R}$, between the warped frames; and
4) Put the input frame into template set to add a new class if $\bar{R} > T_c$ for any $\bar{R}$; otherwise, place it into the class which corresponds to the minimal $\bar{R}$.

8.3.2 ECG Classification Results by DTW and Discussion

Experiments have been performed to classify the ECG frame using this modified DTW algorithm. The code is provided in Appendix B.5.1. The ECG data sampled at 360

sps with 11 bps have been taken from the MIT-BIH ECG database [Mood99]. The experiment uses the ECG signal contained in the file, x_100.txt, which has a 10-minute recording of ECG data. As described in Sec. 7.5, the 758 ECG periods in this file is segmented into 758 frames, numbered sequentially from Frame 1 to Frame 758.

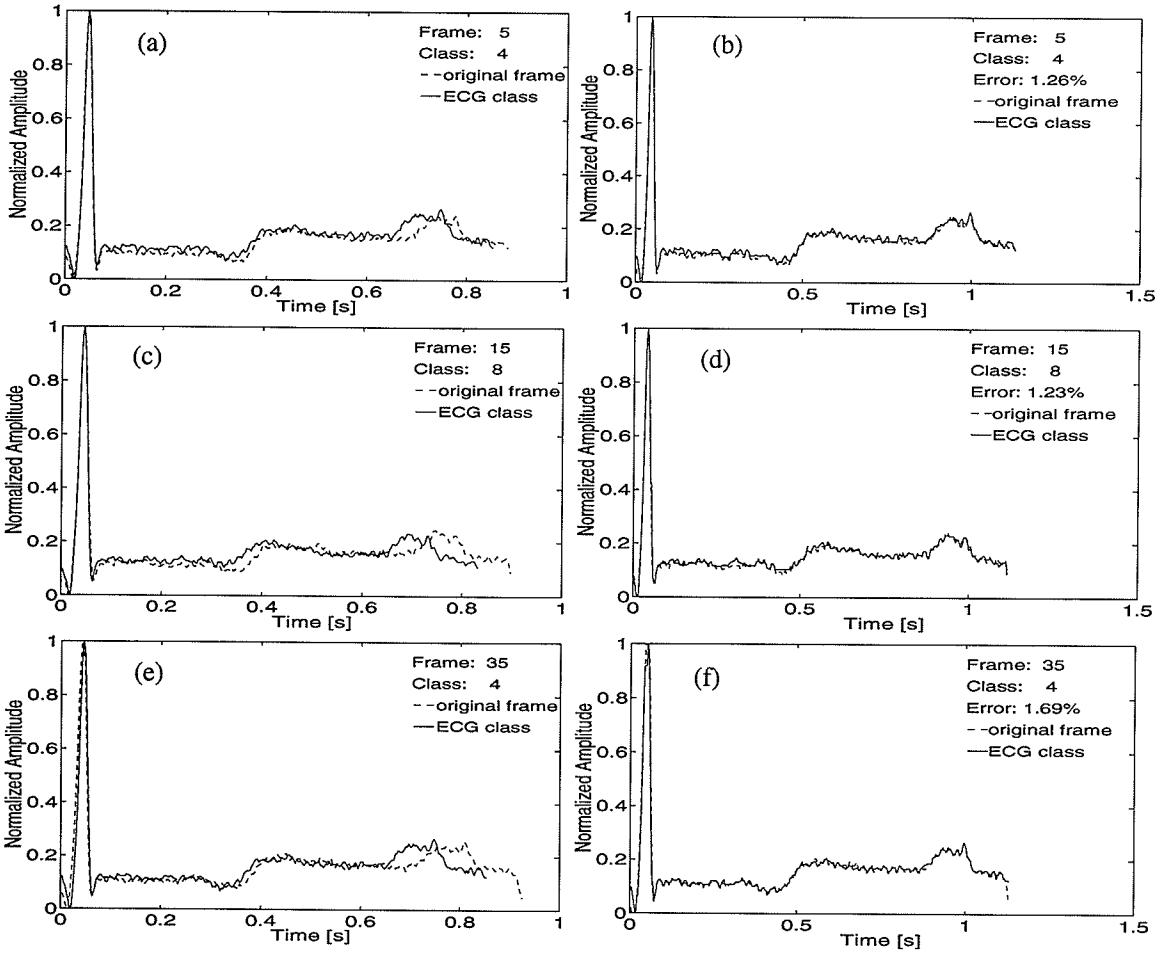

Fig. 8.4. Examples of ECG frames and templates before (left) and after (right) the DTW. The pairs (a) and (b), (c) and (d), (e) and (f) illustrate such a classification, with the NPRD error shown on different frames.

We use the DTW to classify the above recording under a residual threshold of $T_c = 0.01$, which generates 53 classes for a frame dictionary. Some classification examples are given in Fig. 8.4. Figure 8.4(a) shows Frame 5 and Class 4 to be processed. The

position difference between corresponding feature waves such as the QRS complex, the T wave, and the P wave for the two waveforms changes with time. Figure 8.4(b) is the corresponding frame and class after warping. The feature waves between the two frames are registered precisely in time. Only a small difference can be observed. Notice that the warped frame is longer than the original. The warping function is nonlinear. The two waveforms in Fig. 8.4(b) are very similar. Therefore, Frame 5 is classified as Class 4. Another two examples are shown in Figs. 8.4(c)-(d), and Figs. 8.4(e)-(f). Again, a good registration is observed for the feature waves of the ECG frame. Here, all patterns are normalized to [0,1] by: (i) subtracting the minimal value of the frame from the signal, and (ii) then dividing the signal by its maximal value.

Figure 8.5 is the residual between the waveforms in Fig. 8.4(b). Compared to the original frame, the amplitude of the residual is small. The maximal normalized absolute residual is about 0.06. Its NPRD is 1.26%. The small residual error is consistent with the high similarity seen in Figs. 8.4(b), (d), and (f). Although the amplitude of the residual at the QRS complex is greater than the other part of the frame, its magnitude is still smaller than errors produced by other techniques. The errors are also consistently small, as demonstrated by NPRD of 1.33% calculated over the entire 10-minute ECG data containing 758 ECG frames.

Dynamic time warping classification will be used to compress the ECG signal through a lossy compression technique. Such a technique is developed to remove redundancy in the signal to achieve a high compression with minimum affect on the signal, not directly on diagnosis information. Time registration between ECG beats is not for diagno-

sis purpose but for data compression capable of providing a high-quality ECG reconstructed signal that may be used by cardiologists.

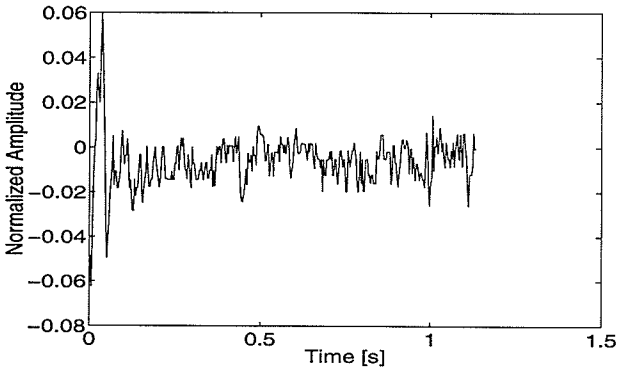

Fig. 8.5. The residual between the ECG frame and the template in Fig. 8.4(b).

8.4 ECG Compression with Block Encoding

One of the applications of the ECG frame classification is to compress the ECG signal. Making use of the classification information contained in the ECG frames to do compression can lead to more efficient techniques. We have described different ECG frame classification techniques and provided some classification results [HuKi02a] [HuKi99b]. Although the DTW can achieve low residual error in the frame classification, the warping function between the input frame and template frame is still too complicated for compression purpose. As explained before, since the inherent nonstationarity of ECG signals poses a major obstacle in signal compression of ECG frames, the classification information cannot be used directly to describe the original signal.

Consequently, another important problem in ECG frame compression is how to describe efficiently the input frame given a template frame. By observing Fig. 8.1 carefully, the complicated warping function may be simplified into several piecewise-linear

segments for frame description. In ECG data compression experiments, we also found that a large reconstruction error often occurs at the transition area of the waveform. Therefore, we propose block encoding to process the complicated nonstationarity of the ECG signal. A windowed-variance technique is proposed to smooth a noisy ECG signal and then partition an ECG frame into five segments (corresponding to feature waves of the ECG) for block encoding purpose.

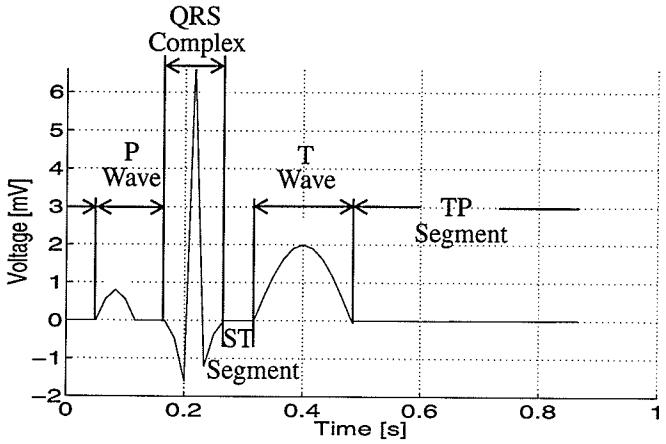

Fig. 8.6. An ideal ECG frame with typical features.

8.4.1 Windowed-Variance of ECG

As we shall see in Sec. 8.4.2, block encoding of an ECG frame involves separate encoding of the QRS complex, the T wave, and the P wave. Therefore, the first step in this scheme is to partition the ECG frame into different parts called blocks. It is difficult to find a general technique to detect the T wave and the P wave directly from the ECG signal since the recorded ECG waveform shown in Fig. 8.7 is much more complicated than the ideal waveform of Fig. 8.6. The noise in the signal and the nonstationarity make ECG waveforms fluctuate with time. Instead of analyzing the signal directly, detecting the transition in the waveform may be done in a transform domain. A windowed-variance (WV)

technique is proposed to detect such features in the ECG frame.

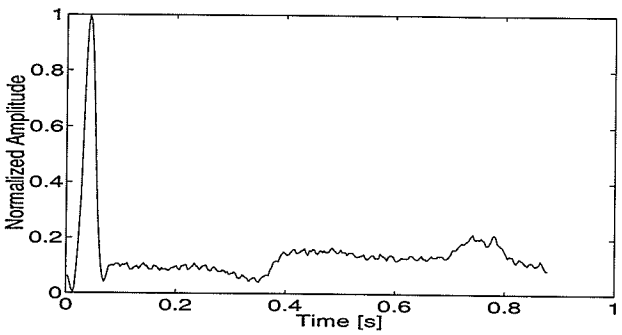

Fig. 8.7. A recorded ECG waveform.

Assume $x(i)$ is a discrete ECG time series with length N, where i is an integer. In general, variance σ_x^2 is used to measure the total fluctuation degree of the signal. If we take $y(i;j)$ as a windowed part of the signal

$$y(i;j) = x(j), \quad i - \frac{L}{2} \le j \le i + \frac{L}{2} \tag{8.14}$$

where L is the length of the window, then the degree of fluctuation of $y(i;j)$ can be described by $\sigma_y^2(i)$, the variance of $y(i;j)$. If we let $y(i;j)$ proceed along $x(i)$, then the function $\{\sigma_y(i), 1 \le i \le N\}$ will indicate the local change in $x(i)$. The curve of $\sigma_y^2(i)$ will be smooth if L is selected properly [Kins94b]. Notice that $\sigma_y^2(i)$ is the conventional variance computed over a window, and is not the variance fractal dimension D_σ computed over a window at multiple scales [Kins94b].

Figure 8.8(a) shows that the WV curve is much smoother than the original ECG waveform, and has elevated values for the desired ECG feature regions. The flat areas in the signal give low values in the variance domain. Only the transitions of the ECG wave-

form lead to elevated envelopes in the WV curve. Consequently, it is much easier to detect the ECG features in the smooth WV curve than in the original ECG waveform. An algorithm based on a threshold T_A, an envelope length L_e, and an envelope distance d_e is proposed to detect the elevated envelopes in the WV curve. The detection procedure is based on the following three steps:

1) If $\sigma_y^2(i) \le T_A$, then $\sigma_y^2(i) = 0$;
2) For any $\sigma_y^2(i) > 0$, $n_1 \le i \le n_2$,
   if $(n_2 - n_1 + 1) \ge L_e$,
   then $\sigma_y^2(i) = \gamma$, $\gamma = \text{const}$,
   otherwise $\sigma_y^2(i) = 0$;
3) Let d be the difference between the end point of one envelope and the initial point of the next envelope. If $d \le d_e$, the two envelopes are merged.

Figure 8.8(b) gives an example of four envelopes. Step 1 detects envelopes by applying a given threshold. Then, a small envelope is removed by Step 2. After Step 3, Fig. 8.8(c) shows that the last two adjacent envelopes are merged into one envelope. It is seen clearly that this detection scheme identifies the main features in the ECG frames.

The code of partitioning an ECG frame is provided in Appendix B.5.2.

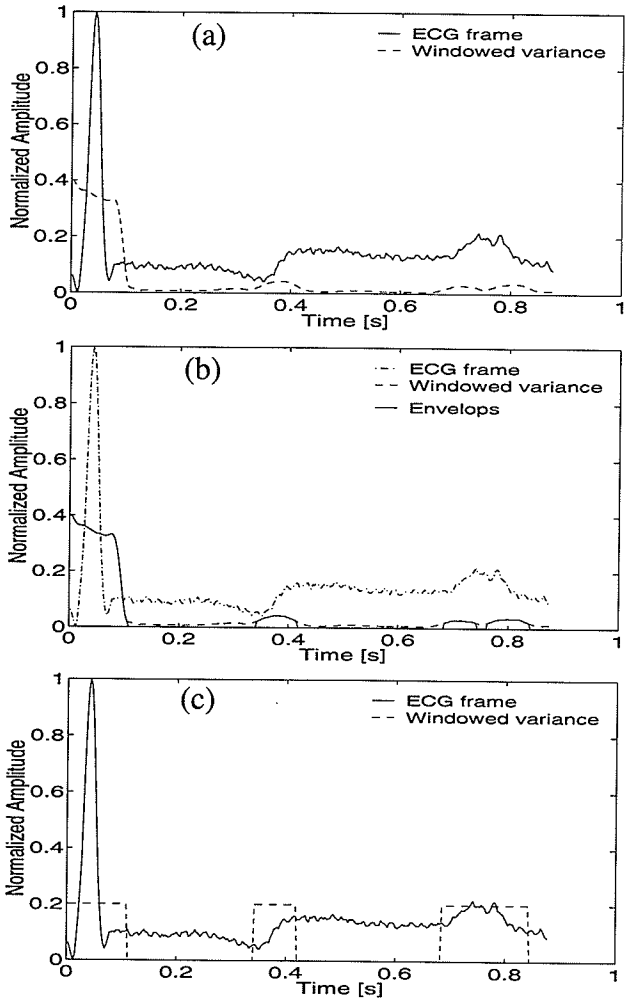

Fig. 8.8. The three stages of the WV algorithm. (a) The WV curve of an ECG frame. (b) The envelopes of the WV curve detected by a threshold (after Steps 1 and 2). (c) ECG features detected by the WV technique (after Step 3).

8.4.2 Block Encoding and Reconstruction

The ECG signal can be compressed based on frames because of the quasiperiodicity of the signal. The ECG frame compression may be realized through frame classification and block encoding. Such a classification is carried out by the DTW technique, which aligns an unknown input frame with a template frame in time. A minimal residual between the time warped frames is used as the criterion for classification [HuKi02a].

Once the ECG frame is classified, an important issue is how to describe the frame by class information for ECG frame compression. A simple technique is to contract/dilate the template frame to the input ECG frame linearly in time. Although this leads to a very high compression ratio, the accompanying reconstruction error can be very large. The large reconstruction error often occurring at the transition areas of ECG signals prompts us how to improve the reconstruction error.

To reduce the reconstruction error, we propose to partition the ECG frame into feature regions, and then treat them as blocks to encode. Generally, transitions in the ECG signal occur at the QRS complex, the T wave, and the P wave. We partition the ECG frame into five such regions: the QRS complex, ST segment, the T wave, TP segment, and the P wave. The ECG frame partitioning is done by the WV technique, which transforms the ECG waveform into a smooth curve in the variance domain. Block encoding means to encode the block of the ECG frame by the corresponding block in the template frame. In general, the block from the current ECG frame cannot maintain a fixed relationship with the corresponding block from the template frame since the ECG signal is nonstationary. A linear contraction/dilation between the corresponding segments is done to reconstruct the signal. Block encoding for the ECG frame based on the frame classification and the WV techniques consists of the following five steps:

1) Extract an ECG frame by detecting the R peaks;
2) Apply the DTW to classify a normalized unknown input ECG frame into a class which contains a normalized template frame;
3) Partition the input frame and the template frame into five blocks by the WV technique;

4) Encode the five blocks of the input frame; and
5) Reconstruct the block of the input ECG frame by linear contracting/dilating of the corresponding segment in the template.

The purpose of block encoding is to reduce the reconstruction error in the ECG frame compression. Although such a technique cannot achieve the low error of the DTW technique, the above model gives an optimal approximation for the ECG frame compression.

Eight parameters are used to encode the ECG frame. The parameters include: (i) the class index, (ii) the amplitude and the mean of the input frame, and (iii) lengths of the five segments. Without an optimal bit allocation, each parameter is represented by 8 bits. Therefore, the compression ratio by the block encoding can be high and is not related to the reconstruction error.

8.5 Experimental Results and Discussion

We have performed three experiments to compress ECG data through the block encoding technique. As described in Sec. 1.2, recorded regular ECG data are used in the ECG beat-based compression techniques. In the experiment, the ECG data have been taken from the same data file as used by the DTW classification in Sec. 8.3.2. The total number of samples is 10 minutes $\times$ 60 s $\times$ 360 sps = 216,000. The 758 ECG periods contained in this file are segmented into 758 frames, numbered sequentially from Frame 1 to Frame 758.

As described in Sec. 8.3.2, the above ECG frames are classified by the DTW under

a residual threshold of $T_c = 0.01$, which generates 53 classes for the frame dictionary. Such a frame dictionary must be known to both the encoder and decoder in the compression.

The WV technique is applied to detect feature segments in the ECG frame for the block encoding. The parameters of the algorithm are set as $T_A = 0.018$, $L = 28$ samples, and $L_e = d_e = 15$ samples to partition the ECG frame.

Since there are 758 frames, each represented by eight parameters and the file contains (216,000 samples $\times$ 11 bps)/8 bit = 297,000 bytes, then according to (2.13), the compression ratio R_{cr} for the block encoding technique is [Kins98]

$$R_{cr} = \frac{297000}{8 \times 758} = 48.98:1 \tag{8.15}$$

Notice that the quantization of the parameters is not optimized. An optimal bit allocation for each parameter would require entropy calculations for each segment.

An objective evaluation of the classifier is based on a measure of the difference between the warped ECG frame and the corresponding template. As discussed in Sec. 2.5.3, the PRD is one of the objective evaluation criteria. It is often employed to measure the error between the original signal and the reconstruction in ECG data compression. Since the distortion measure given by the PRD varies with the DC components (mean) of the signal, it should not be used for algorithm evaluation with different ECG databases. This experiment uses the NPRD to evaluate the performance of the compression scheme and gives a very small NPRD of 2.53% for a 10-minute ECG recording.

In order to see how the compression performs on individual frames, we shall discuss three frames (15, 5, and 35) belonging to different classes (8 and 4). Figure 8.9 illustrates an example of the original and reconstructed waveforms by applying the block encoding to the ECG frame. Figure 8.9(a) shows Frame 15 and Class 8, as classified by the DTW technique. Figure 8.9(b) shows the corresponding frame and its reconstruction through the block encoding by partitioning the frame into five segments. The two waveforms in Fig. 8.9(b) are very close. The block encoding registers the positions of the QRS complex, the P wave, and the T wave, which have important significance in clinical diagnosis. Figure 8.9(d) shows the residual for this reconstruction. The NPRD of the reconstruction of Frame 15 is about 1.79%. The normalized maximal reconstruction error for this frame is 0.09. The large reconstruction error occurs at the area of the QRS complex.

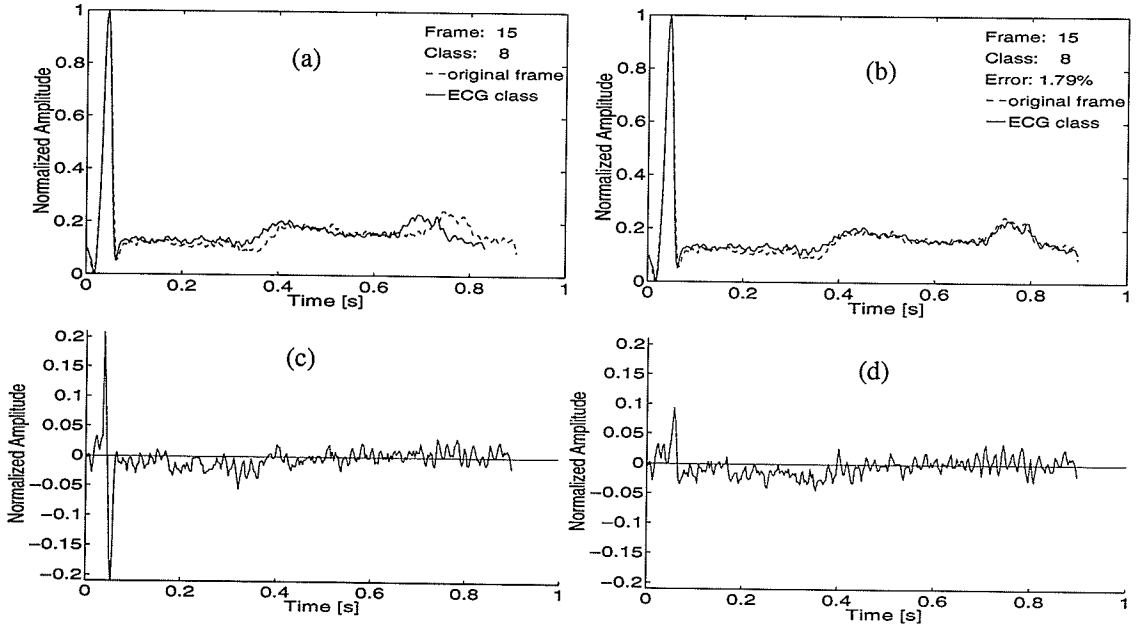

Fig. 8.9. An example of ECG frame compression by the block encoding. (a) Frame 15 is classified into Class 8 by the DTW. (b) Reconstruction of Frame 15 by the block encoding. (c) Residual for the full frame encoding. (d) Residual for the block encoding.

As a comparison, Fig. 8.9(c) gives reconstruction residual of Frame 15 by a full frame encoding; *i.e.*, linearly dilating the length of Class 8 to that of Frame 15. The NPRD of the reconstruction of Frame 15 is now larger, about 2.75%. The normalized maximal reconstruction error for this frame is also larger, 0.21. The largest reconstruction errors occur in the region of the QRS complex.

From Figs. 8.9(c) and (d), the improvement of reconstruction quality for the ECG signal is observed from the NPRD and maximal reconstruction error. The error decreases sharply in the region of the QRS complex.

Another two reconstruction examples for the ECG signal compression by the block encoding are shown in Figs. 8.10(a)-(b) for Frame 5, and Figs. 8.10(c)-(d) for Frame 35, both of Class 4. Again a good registration is observed for the feature segments of the ECG frame. Although a large reconstruction error still occurs in the region of the QRS complex, it decreases sharply compared to the full frame encoding. Here the amplitude of all the frames is normalized to [0, 1], as described in Sec. 8.3.2.

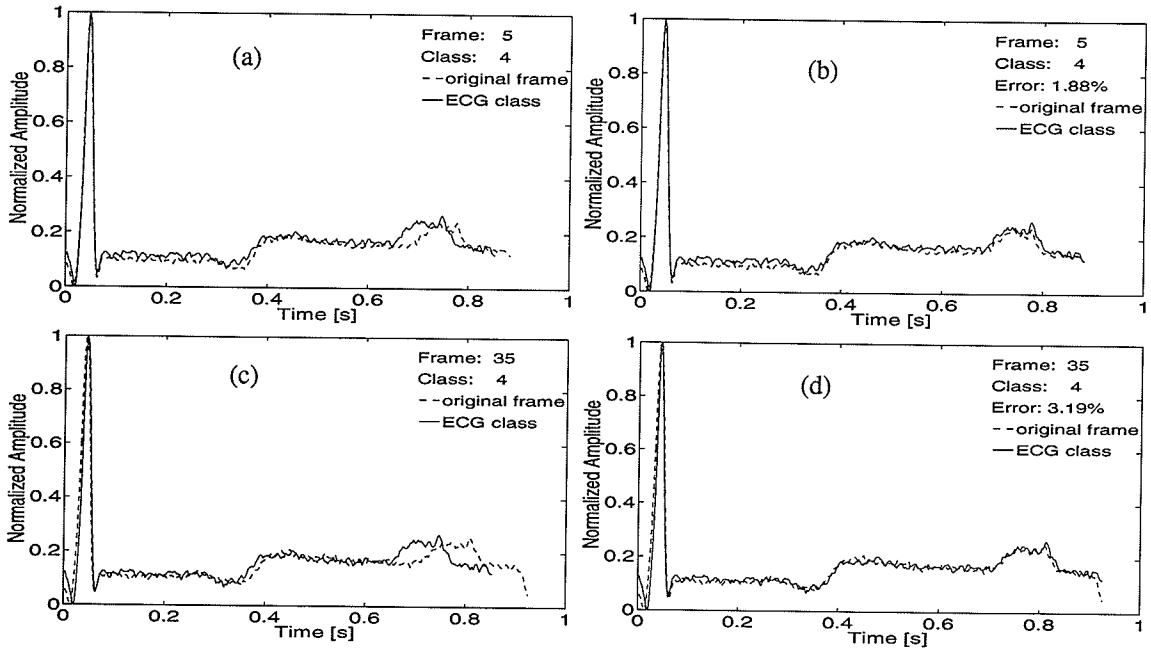

Fig. 8.10. ECG frames before (left) and after (right) reconstruction by the block encoding. The pairs (a) and (b), and (c) and (d) illustrate such a reconstruction on two distinct frames with the NPRD error shown.

8.6 Summary

This chapter presents another attempt to classify, compress, and browse ECG signals. The scheme includes using the DTW for ECG frame classification, and then encoding the blocks in the ECG frame for signal compression. An analysis of a probable heart disease can be speeded up by browsing ECG recording automatically through the classified frames. Although such a scheme is one of the more elaborate techniques described so far, it should be noticed that this browsability can only be used as an analysis tool, not a diagnosis tool, since the classification is based on ECG waveform characteristics, not on disease characteristics.

Since the Euclidean distance is not suitable to compare the similarity between two ECG frames due to the inconsistency of the frame length, the DTW matching is applied to

eliminate this fluctuation by time-normalizing ECG frames for the frame classification research, In this DTW algorithm, the fluctuation in time is modelled approximately by a nonlinear warping function with some carefully specified properties. Timing difference between two ECG frames is minimized by warping the time axis of one frame such that the maximum coincidence is attained with the other. A new classification criterion for the DTW, the mean of absolute residual, is proposed to search an optimal or suboptimal mapping path between a warped ECG frame and a template from the end of the frame to its start. The ECG frame is placed into the class with which it has minimal residual distance, if the residual is within the range of a certain residual threshold. Otherwise, a new class is generated.

It appears that the DTW has been employed to classify ECG frames for the first time in literature. Experiments show that the DTW algorithm with the modified classification criterion is effective to find a nonlinear mapping between two ECG frames. From a perceptual point of view, ECG frame classification using the modified DTW algorithm is successful. The NPRD of 1.33% on a 10-minute recording is small.

There are three problems with the DTW algorithm applied to ECG frame classification: (i) the DTW algorithm is time-consuming, (ii) distribution of the residual amplitude is uneven, and (iii) the warping function is complicated and nonlinear, which leads to a problem for ECG signal compression through frame classification. Consequently, a block encoding technique has been proposed to encode an ECG frame with only five blocks.

In the block encoding, compression of a complicated ECG frame is converted into

a linear time registration problem of blocks. The WV technique makes the detection of features of ECG frames easy. The linear contraction/dilation of segments during reconstruction is also easy to be implemented. The block encoding reconstructs the feature waves of the frame, such as the QRS complex, the T wave, and the P wave well. A compression ratio of 48.98:1 is achieved, and is not related to the reconstruction error. The NPRD is 2.53% for a 10-minute ECG recording. The block encoding model tries to approximate classified ECG frames with a low bit rate. Although its NPRD of 2.53% is greater than that of 1.33% which is obtained by applying the DTW to the ECG frame classification, the reconstruction error is still low.

CHAPTER IX

CONCLUSIONS AND RECOMMENDATIONS

9.1 Conclusions

This thesis has presented a study of nonlinear analysis methods to characterize the nonstationary one-channel ECG signal, with particular interest in the efficient representation of the signal.

The first objective was to develop an improved ECG compression scheme. A scheme of a nonlinear IFS (NIFS) transform combined with signal partitioning based on the compositional complexity measure by the VFDT has been presented to compress the ECG signal rapidly. The NIFS of order 2 achieves the best compression performance as compared to other orders. The domain search is reduced to $O(N)$ for a time series with length N, compared to the computational complexity of $O(N^2)$ of the IFS. With the domain pool prepared by the VFDT, the compression ratio R_{cr} is 5.5:1 by the traditional IFS and 5.7:1 by the NIFS under the MPRD of 5.8% for the ECG signal. Although the R_{cr} of the NIFS is only slightly lower than 6.0:1 obtained by Øien and Nårstad (also an improved IFS), the NIFS has much more modelling flexibility than the IFS. Compared to a much weaker self-similarity between regions with different complexities, a strong self-similarity of fractal object exists in the area with the same complexity.

Another objective was to develop a frame registration scheme suitable for compression and classification. We have presented a study of ECG frame classification and

block encoding based on time warping frames by the modified DTW and frame partitioning by the WV technique. The partitioning of the ECG frame based on feature waves gives optimal approximations for the frame registration which have important significance in clinical diagnosis and can achieve high compression ratio (about 50:1 in our case).

Still another objective was to develop a statistical feature extraction for ECG classification. We have presented a study of ECG frame classes involving statistical feature extraction and neural network classification. The class information is important in the application of browsing long-term ECG recordings. The statistical features of the ECG signal with the same dimension for various beats can be input to the PNN to classify the ECG frame. The moment-invariant feature of the ECG beats is used in the PNN classification. The moment-invariance contains the shape information of the signal waveform. By representing the ECG frame with class parameters, the compression ratio of about 66:1 is achieved with the MPRD of about 16%. The training and ECG beat classification by the PNN can be implemented in real time.

The other statistical features extracted from the ECG signal are multifractal features, which is a spectrum of the complexity measure of the fractal object. The Rényi dimension spectrum is calculated from the strange attractor of the ECG. The strange attractor is reconstructed by lagging and embedding the ECG time series in an m-dimensional phase space. The lag of about 11.11ms is obtained from the autocorrelation function of the ECG signal. A clear deterministic pattern of the ECG time series by using such a lag is observed in 3D phase space. The false nearest neighbourhood technique shows that the embedding dimension for the ECG is 7.

The convergence of the Rényi dimension spectrum is also employed to find the best embedding dimension of phase space in the reconstruction of the strange attractor of the ECG. The convergence of the Rényi dimension spectrum of the ECG time series is incomplete. The incomplete convergence is due to the low-sampling frequency (360 sps) and noise in the ECG data. Consequently, wavelet denoising and chaos denoising are applied to the ECG signal to remove noise. It is found that chaos denoising improves the convergence of the Rényi dimension spectrum of the ECG about five times under the convergence degree measure, while wavelet denoising degrades the convergence about two times. Even with chaos denoising, the convergence is inadequate for the multifractal feature extraction from the signal. An ECG data acquisition system with a higher sampling rate needs to be established for chaos study and multifractal measure. Such an analysis has not been presented in literature before.

It is also found that chaos denoising is the best technique to denoise various colour noises from the ECG signal according to SNR gain, compared to linear filtering and wavelet denoising. Since the ECG signal has a power spectrum distribution that is close to pink noise, we explain that (i) a linear filter cannot filter colour noises from the ECG signal because their spectra overlap; (ii) wavelet denoising loses its capability because the smoothness of the noise increases from white, to pink, to brown, to black noise gradually; and (iii) chaos denoising is based on the determinacy of the chaotic signal and the unique characteristics of the strange attractor of the object itself, which can be distinguished from noise.

9.2 Contributions

To our knowledge, the following contributions and discoveries, as given in this thesis, are novel:

1) Moment-invariance feature is extracted from the ECG signal;
2) The Rényi dimension spectrum features are extracted from the ECG signal;
3) The strange attractor of the ECG is reconstructed in phase space by finding the time lag of 11.11 ms and embedding dimension of 7;
4) The ECG is confirmed to be a multifractal object with a morphological dimension of about 1.75;
5) By applying denoising techniques to the ECG signal, we find that chaos denoising improves the convergence of the Rényi dimension spectrum of the ECG five times under the convergence degree measure, while wavelet denoising degrades the convergence two times;
6) Chaos denoising is found to be the best technique to improve the SNR gain for colour noise, compared to linear filtering and wavelet denoising;
7) The PNN is employed to the ECG frame classification successfully for the first time;
8) The ECG data are compressed based on the beat classification;
9) The DTW is employed to the ECG frame classification;
10) The windowed-variance is applied to partition the ECG frame into segments including feature waves;
11) The ECG frame is compressed based on segments by the block encoding technique;
12) The NIFS is applied to the ECG signal compression;

13) The VFDT is applied to ECG signal partitioning;

14) A combined scheme of the NIFS and signal partitioning is applied to the ECG signal compression. The domain search is reduced to $O(N)$ for a time series with length N, compared to the computational complexity of $O(N^2)$ of the IFS; and

15) Software has been implemented for (i) reconstructing the strange attractor of the ECG signal and the Rényi dimension spectrum calculation, (ii) chaos and wavelet denoising, (iii) NIFS compression with signal partitioning, (iv) PNN classification and minimal residual classification to compress ECG frames, and (v) DTW classification and block encoding of ECG frames.

9.3 Recommendations

Based on the research presented in this thesis, the following work is recommended for the future:

1) An ECG data acquisition system with high sampling rate is required for chaos study and the feature extraction from fractals;
2) The Lyapunov dimension of the ECG time series should be calculated to investigate the chaotic characteristics of the ECG;
3) Geometric patterns of noise changing with embedding dimension in the phase space need to be investigated;
4) More ECG data are necessary in the experiments of the nonlinear analysis methods;
5) The R_{cr}, MPRD, NPRD, and PRD based on the same ECG database should be obtained from some typical ECG compression methods for comparison purpose;

6) Entropy encoding should be applied to the ECG frame classification and compression methods to remove any residual redundancy in the compressed signal; and
7) In NIFS compression, coefficients of the affine transform should be quantized by the optimal quantizers with different resolutions.

REFERENCES

[AbTo82] J. P. Abenstein and W. J. Tompkins, "New data-reduction algorithm for real-time ECG analysis," *IEEE Trans. Biomed. Eng.*, vol. BME-29, pp. 43-48, Jan. 1982.

[Acke91] M. J. Ackerman, "Viewpoint: the visible human project," *Journal Biocommunication*, vol. 18, no. 2, pp. 14, 1991.

[AhMH75] N. Ahmed, P. J. Milne, and S. G. Harris, "Electrocardiographic data compression via orthogonal transforms," *IEEE Trans. Biomed. Eng.*, vol. BME-22, pp. 484-487, Nov. 1975.

[AlBe92] V. A. Allen and J. Belina, "ECG data compression using the discrete cosine transform (DCT)," in *Proc. Computers in Cardiology*, IEEE Computer Society Press, pp. 687-690, 1992.

[AnDR95] K. Anant, F. Dowla, and G. Rodrigue, "Vector quantization of ECG wavelet coefficients," *IEEE Signal Processing Letters*, vol. 2, no. 7, pp. 129-131, July 1995.

[AyÇK91] M. C. Aydin, A. E. Çetin, and H. Köymen, "ECG data compression by subband coding," *Electronics Letter*, vol. 27, pp. 359-360, Feb. 1991.

[BaDe85] M. F. Barnsley and S. Demko, "Iterated function systems and the global construction of fractals," *Pro. Roy. Soc.*, London, vol. A399, pp.243-275, 1985.

[BaHu93] M. F. Barnsley and L. P. Hurd, *Fractal Image Compression*. Wellesley, MA: AK Peters, 1993, 244 pp.

[Barn88] M. F. Barnsley, *Fractal Everywhere*. San Diego, CA: Academic Press, 1988, 394 pp.

[BaSk96] G. D. Barlas and E. S. Skordalakis, "A novel family of compression algorithms for ECG and other semiperiodical, one-dimensional, biomedical signals," *IEEE Trans. Biomed. Eng.*, vol. 43, no. 8, pp. 820-828, Aug. 1996.

[BFVR98] S. Barro, M. Fernandez-Delgado, J. A. Vila-Sabrino, C. V. Regueiro, and E. Sanchez, "Classifying multichannel ECG patterns with an adaptive neural network," *IEEE Eng. Med. Bio. Mag.*, vol. 17, pp. 45-55, Jan.-Feb. 1998.

[BJMR88] M. Barnsley, A. Jacquin, F. Malassenet, L. Reuter, and A.D. Sloan, "Harnessing chaos for image synthesis," *Siggraph Proceedings*, 1988.

[Bow92] S. -T. Bow, *Pattern Recognition and Image Preprocessing*. New York, NY:

Marcel Dekker, pp. 100-113, 1992.

[Brad96] B. Bradie, "Wavelet packet-based compression of single lead ECG," *IEEE Trans. Biomed. Eng.*, vol. 43, no. 5, pp. 493-501, May 1996.

[Bril66] S. A. Briller *et. al.*, "The electrical interaction between artificial pacemakers and patients with applications to electrocardiography," *Amer. Heart J.*, vol. 71, pp. 656-665, 1966.

[BuSu72] H. O. Burton and D. D. Sullivan, "Error and error control," *Proc. IEEE,* vol. 60, pp. 1263-1301, Nov. 1972.

[CaRe94] S. D. Casey and N. F. Reingold, "Self-similar fractal sets: theory and procedure," *IEEE Computer Graphics and Applications Mag.*, IEEE Cat. No. 0272-17-16/94, pp. 73-82, May 1994.

[CaEn97] M. Cassen and M. J. English, "Computationally efficient ECG compression scheme using a non-linear quantizer," in *Proc. Computers in Cardiology,* IEEE Computer Society Press, vol. 24, pp. 283-286, 1997.

[CBLG99] J. L. Cárdenas-Barrera and J. V. Lorenzo-Ginori, "Mean-shape vector quantizer for ECG signal compression," *IEEE Trans. Biomed. Eng.*, vol. 46, no.1, pp. 62-70, Jan. 1999.

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

[ChIH93] J. Chen, S. Itoh, and T. Hashimoto, "ECG data compression wavelet transform," *IEICE Trans. Inform. Syst.*, vol. E76-D, no.12, pp. 1462-1469, 1993.

[ChIt98] J. Chen and S. Itoh, "A wavelet transform-based ECG compression method guaranteeing desired signal quality," *IEEE Trans. Biomed. Eng.*, vol. 45, no.12, pp. 1414-1419, Dec. 1998.

[ChKH02] J. Chen, W. Kinsner, and B. Huang, "Power system transient modelling and classification," *Proc. of IEEE CCECE'02*, Winnipeg, Canada, May 12-15, 2002.

[CMNB83] J. Comelis, F. de Maeyer, E. Nyssen, and P. Block, "Influence of the thoracic volume conductor on the ecg distribution-capacity, inductive, anisotropic and geometrical effects," in *Modeling and Data Analysis in Biotechnology and Medical Engineering*, G. C. Vansteenkiste and P. C. Young, Eds. Amsterdam: North Holland, 1983.

[CNFO68] J. R. Cox, F. M. Nolle, H. A. Fozzard, and G. C. Oliver, "AZTEC, a prepro-

cessing program for real-time ECG rhythm analysis," *IEEE Trans. Biomed. Eng.*, vol. BME-15, pp. 128-129, Apr. 1968.

[CLFM98] P. Carré, H. Leman, C. Fernandez, and C. Marque, "Denoising of the uterine EHG by an undecimated wavelet transform," *IEEE Trans. Biomed. Eng.*, vol. 45, no. 9, pp. 1104-1113, Sept. 1998.

[CoDF92] A. Cohen, I. Daubechies, and J.-C. Feauveau, "Biothogonal bases of compactly supported wavelets," *Commum. on Pure and Appl. Math.*, 45: pp. 485-560, 1992.

[CoMW92] R. R. Coifman, Y. Meyer, and M. V. Wickerhauser, "Wavelet analysis and signal processing," In *Wavelet and their application*, Jones and Barlett. B. Ruskai *at al.* eds, Boston, pp. 153-178, 1992.

[CoRi73] J. R. Cox and K. L. Ripley, "Compact digital coding of electrocardiographic data," in *Proc. VI. Int. Conf. Syst. Sci.*, pp. 333-336, Jan. 1973.

[CoZi98] A. Cohen and Y. Zigel, "Compression of multichannel ECG through multichannel long-term prediction," *IEEE Eng. Med. Biol. Mag.*, pp. 109-115, Jan.-Feb. 1998.

[Dans00] R. M. Dansereau, *Image Quality Measures for Progressive Image Transmission Using Fractals, Multifractals, and Wavelet techniques*. Ph.D. Thesis, Dept. of Electrical and Computer Eng., Uni. of Manitoba, Winnipeg, Canada, 2000.

[Daub88] I. Daubechies, "Orthonormal bases of compactly supported wavelets," *Commum. on Pure and Appl. Math.*, vol. 41, pp. 909-996, Nov. 1988.

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

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

[Dono92] D. Donoho, "Wavelet shrinkage and w.v.d: a 10-minute tour," *Technical Report*, Stanford University, 1992.

[Ehti99] T. Ehtiati, *Multifractal Characterization of Electromyogram Signals*. M. Sc. Thesis, Dept. of Electrical and Computer Eng., Uni. of Manitoba, Winnipeg, Canada, 1999.

[FiBa63] E. S. Fishmann and M. R. Barber, "Aimed electrocardiography, model studies, using a heart consisting of 6 electrically isolated areas," *Amer. Heart J.*, vol. 65, p. 628, 1963.

[Fish98] Y. Fisher, *Fractal Image Encoding and Analysis*. Heidelberg, NY: Springer-Verlag, pp. 201-226, 1998.

[Fuku90] K. Fukunaga, *Introduction to Statistical pattern Recognition.* San Diego, CA: Academic Press, pp. 510-520, 1990.

[Gano85] W. F. Ganong, *Review of Medical Physiology.* Los Altos, CA: Lange Medical Publications, 12th Edition, 1985.

[Gard64] L. W. Gardenhire, "Redundancy reduction the key to adaptive telemetry," in *Proc. 1964 Nat. Telemetry Conf.*, pp. 1-16, 1964.

[Gard65] L. W. Gardenhire, "Data compression for biomedical telemetry," in *Biomedical Telemetry*, C. A. Caceres, Ed., NY: Academic, ch. 11, 1965.

[GeGr92] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression.* Norwell, MA: Kluwer Academic Publishers, 1992, 732 pp.

[Gese63] D. B. Geselowitz, "The concept of an equivalent cardiac generator," *Biomed. Sci. Instrum.*, vol. 1, pp. 325-330, 1963.

[Gese67] D. B. Geselowitz, "On bioelectric potentials in an inhomogeneous volume conductor," *Biophys. J.*, vol. 7, pp. 1-11, 1967.

[GHKS93] P. Grassberger, R. Hegger, H. Kantz, C. Schaffrath, and T. Schreiber, "On noise reduction methods for chaotic data," *Chaos*, vol. 3, pp. 127-141, 1993.

[GiPa86] E. A. Giakoumakis and G. Papakonstantinou, "An ECG data reduction algorithm," *Comput. Cardiol.*, Boston, MA, pp. 675-677, Oct. 1986.

[Gold94] M. Goldburg, *Applications of Wavelets to Quantization and Random Process Representations*. Ph.D. Thesis, Stanford University, 1993, 145 pp.

[GoWo02] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Reading, MA: Addison-Wesley Publishing, 2002, 728 pp.

[GoWi77] R. C. Gonzalez and P. Wintz, *Digital Image Processing*. Reading, MA: Addison-Wesley, pp. 354-358, 1977.

[GoWo92] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Reading, MA: Addison-Wesley Publishing, 1992, 716 pp.

[GrPr83] P. Grassberger and I. Procaccia, "Characterization of strange attractor," *Phys. Rev. Lett.*, vol. 50, no. 5, pp. 346-349, Jan. 1983.

[Gulr98] R. M. Gulrajani, "The forward and inverse problems of electrocardiography," *IEEE Eng. Medicine and Biology*, pp. 84-101, Sept./Oct. 1998.

[HaBa65] D. J. Hall and G. B. Ball, "ISODATA: A novel method of data analysis and pattern classification," *Technical Report,* Stanford Research Institute, Menlo Park, CA, 1965.

[HaHH97] D. Haugland, J. Heber, and J. Husøy, "Optimisation algorithm for ECG data compression," *Medical & Biological Engineering & Computing*, vol. 35, pp. 420-424, July 1997.

[HaMF74] A. R. Hambley, R. L. Moruzzi, and C. L. Feldman, "The use of intrinsic components in an ECG filter," *IEEE Trans. Biomed. Eng.*, vol. BME-21, pp. 469-473, Nov. 1974.

[HaTo91] P. S. Hamilton and W. J. Tompkins, "Compression of the ambulatory ECG by average beat subtraction and residual differencing," *IEEE Trans. Biomed. Eng.*, vol. 38, no. 3, pp. 253-259, Mar. 1991.

[Haug95] T. Haugan, *Compression of ECG signals by means of optimized fir filterbanks and entropy allocation.* M. Sc. Thesis, Rogaland University Center, Norwegian, 1995.

[HBHe94] J. H. Husy, M. Be, and J. G. Heber, "An image coding approach to ECG compression," *Proc. BIOSIGNAL'94,* pp. 98-100, June 1994.

[Hilt97] M. L. Hilton, "Wavelet and wavelet packet compression of electrocardiograms," *IEEE Trans. Biomed. Eng.*, vol. 44, no. 5, May 1997.

[Huan00] B. Huang, "Multifractal characterization of ECG and impact of denoising," *Proc. of GRADCON2000*, Winnipeg, MB, Canada: Dept. of Electrical and Computer Engineering, University of Manitoba, Oct. 20, 2000 (extended abstract).

[Huan01] B. Huang, "IFS compression of ECG based on combination of nonlinear transform and domain block partitioning," *Proc. of GRADCON2001*, Winnipeg, MB, Canada: Dept. of Electrical and Computer Engineering, University of Manitoba, Oct. 12, 2001 (extended abstract).

[Huan99] B. Huang, "ECG classification and compression in long-term monitoring," *Proc. of GRADCON99*, Winnipeg, MB, Canada: Dept. of Electrical and Computer Engineering, University of Manitoba, Oct. 1, 1999 (extended abstract).

[HuKi00] B. Huang and W. Kinsner, "Impact of low-rate sampling on the reconstruction of ECG in phase-space," *Proc. of IEEE CCECE'00*, Halifax, Canada, pp. 508-512, May 7-10, 2000.

[HuKi01a] B. Huang and W. Kinsner, "A new nonlinear transform for IFS compression of ECG and other signals," Proc. *IEEE-EMBS Conference on Biomedical Engineering*, Istanbul, Turkey, October 25-28, 2001.

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

[HuKi02a] B. Huang and W. Kinsner, "ECG frame classification using dynamic time warping,", *Proc. of IEEE CCECE'02*, Winnipeg, Canada, May 12-15, 2002.

[HuKi02b] B. Huang and W. Kinsner, "ECG compression with block encoding," *Proc. of IEEE CCECE'02*, Winnipeg, Canada, May 12-15, 2002.

[HuKi99a] B. Huang and W. Kinsner, "ECG signal compression and analysis in long-term monitoring," *Proc. of IEEE CCECE'99*, Edmonton, Canada, pp. 797-800, May 1999.

[HuKi99b] B. Huang and W. Kinsner, "Feature extraction and classification of ECG signal compression," *12th Intern. Conf. Math. & Comp. Modelling & Sci. Comp.*, Chicago, USA, Aug. 1999.

[Hutc81] J. Hutchinson, "Fractals and self-similarity," *Indiana Univ. J. Math.*, vol. 30, pp. 713-747, 1981.

[Hu62] M. K. Hu, "Visual pattern recognition by moment invariants," *IRE Trans. Info. Theory*, vol. IT-8, pp. 179-187, 1962.

[ImKY85] H. Imai, N. Kimura, and Y. Yoshida, "An efficient encoding method for electrocardiography using spline function," *Syst. Comput. Japan*, vol. 16, no. 3, pp. 85-94, 1985.

[ISHS83] M. Ishijima, S. B. Shin, G. H. Hostetter, and J. Sklansky, "Scan-along polygonal approximation for data compression of electrocardiograms," *IEEE Trans. Biomed. Eng.*, vol. BME-30, pp. 723-729, Nov. 1983.

[Itak75] F. I. Itakura, "Minimum prediction residual principle applied to speech recognition," *IEEE Trans. Acous., Speech, Signal Processing*, vol. ASSP-23, pp. 67-72, Feb. 1975.

[IwNS90] A. Iwata, Y. Nagasaka, and N. Suzumura, "Data compression of the ECG using neural network for digital holter monitor," *IEEE Eng. Med. Biol. Mag.*,

pp. 53-57, Sept. 1990.

[Jacq90] A. E. Jacquin, "A novel fractal block-coding technique for digital images," *Proc. ICASSP*, pp. 2225-2228, 1990.

[Jacq92] A. E. Jacquin, "Image coding based on a fractal theory of iterated contractive image transformations," *IEEE Trans. Image Processing*, vol. 1, no. 1, pp. 18-30, 1992.

[JaNo84] N. S. Jayant and P. Noll, *Digital Coding of Waveforms*. Englewood Cliffs, NJ: Prentice-Hall, pp. 486-509, 1984.

[JHCS88] S. M. S. Jalaleddine, C. G. Hutchens, W. A. Coberly, and R. D. Strattan, "Compression of Holter ECG data," *Biomedical Sci. Instrument.*, vol. 24, pp. 35-45, Apr. 1988.

[JHSC90] S. M. S. Jalaleddine, C. G. Hutchens, R. D. Strattan, and W. A. Coberly, "ECG data compression techniques - A unified approach," *IEEE Trans. Biomedical Eng.*, vol. 37, pp. 329-343, Apr. 1990.

[JoSm87] D. W. Jordan and P. Smith, *Nonlinear Ordinary Differential Equations*. New York, NY: Oxford University, 1987.

[KaYo78] J. L. Kaplan and J. A. Yorke, "Chaotic behaviour of multidimensional difference equations," in *Functional Differential Equations and Approximation of Fixed Points*. H.-O. Peitgen and H. O. Walther (eds.), New York, NY: Springer-Verlag, pp. 204-227, 1979.

[KeBA92] M. B. Kennel, R. Brown, and H. D. Abarbanel, "Determining embedding dimensions for phase space reconstruction using a geometric construction," *Physical Review A*, vol. 45A, no. 6, pp. 3403-3411, 1992.

[KiHC00] W. Kinsner, B. Huang, and J. Chen, "Multifractal characterization of ECG signals with denoising," *IEEE-EMBS Asia-Pacific Conference on Biomedical Engineering (APBME'2000)*, Hangzhou, China, Sept. 26-28, 2000.

[Kins91] W. Kinsner, "Review of data compression methods, including Shannon-Fano, Huffman, arithmetic, Storer, Lempel-Ziv-Welch, fractal, neural network, and wavelet algorithms," *Technical Report, DEL91-1*, Dept. of Electrical and Computer Engineering, University of Manitoba, Jan. 1991, 157 pp.

[Kins94a] W. Kinsner, "Fractal dimension: morphological, entropy, spectrum, and variance classes," *Technical Report, DEL 94-4,* Department of Electrical and Computer Engineering, University of Manitoba, May 1994, 146 pp.

[Kins94b] W. Kinsner, "Batch and real-time computation of a fractal dimension based

on variance of a time series," *Technical Report, DEL 94-6*, Department of Electrical and Computer Engineering, University of Manitoba, May 1994, 22 pp.

[Kins94c] W. Kinsner, "Noise, power laws, and the spectral fractal dimension," *Technical Report, DEL 94-1*, Department of Electrical and Computer Engineering, University of Manitoba, Jan. 1994, 51 pp.

[Kins95] W. Kinsner, "Self similarity: fractals, chaos, scaling and their application," *Technical Report, DEL95-2*, Dept. of Electrical and Computer Engineering, University of Manitoba, Jan. 1995, 113 pp.

[Kins98] W. Kinsner, *Signal and Data Compression*. Lecture Notes, Dept. of Electrical and Computer Engineering, University of Manitoba, Jan. 1998.

[KiCh00] W. Kinsner and J. Chen, "Classification of transients in power system," *Technical Report*, Department of Electrical and Computer Engineering, University of Manitoba, Oct. 2000.

[KoSc93] E. J. Kostelich and T. Schreiber, "Noise reduction in chaotic time-series data: a survey of common methods," *Physical Review E*, vol. 48, no. 3, pp. 1752-1763, Sept. 1993.

[Krey89] E. Kreyszig, *Introductory Functional Analysis with Applications*. New York, NY: John Wiley & Sons, 1989, 688 pp.

[Kukl83] W. S. Kuklinski, "Fast Walsh transform data-compression algorithm: e.c.g. applications," *Med. & Biol. Eng. & Comput.*, vol. 21, pp. 465-472, July 1983.

[Lang96] A. Langi, *Wavelet and Fractal Processing and Compression of Nonstationary Signals*. Ph.D. Thesis, Dept. of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 1996, 227 pp.

[LEBS86] G. Lachiver, J. M. Eichner, F. Bessette, and W. Seufert, "An algorithm for ECG data compression using spline functions," *Comput. Cardiol.*, Boston, MA, pp. 575-578, Oct. 1986.

[LeBu99] H. Lee and K. M. Buckley, "ECG data compression using cut and align beats approach and 2-D transform," *IEEE Trans. on Biomedical Engineering*, vol. 46, no. 5, pp. 556-564, May 1999.

[LeHa00] K. M. Lewis and K. A. Handal, *Sensible Analysis of the 12-Lead ECG*. Albany, NY: Delmar Thomson Learning, 2000, 241 pp.

[LeZi76] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 75-81, Jan. 1976.

[Lloy82] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Information Theory*, vol. IT-28, no. 2, pp. 129-137, March 1982.

[Lore63] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of the Atmospheric Sciences*, vol. 20(2), pp. 130-141, 1963.

[MaHa92] D. S. Mazel and M. H. Hayes, "Using iterated function systems to model discrete sequences," *IEEE Trans. on Signal Processing*, vol. 40, no. 7, pp. 1724-1734, July 1992.

[Mall89] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, no. 7, pp. 674-693, July 1989.

[Mall98] S. G. Mallat, *A Wavelet Tour of Signal Processing*. San Diego, CA: Academic Press, 1998, 577 pp.

[Mand82] B. B. Mandelbrot, *The Fractal Geometry of Nature*. New York, NY: W. H. Freeman, 1982, 468 pp.

[MaRa90] C. P. Mammen and B. Ramamurthi, "Vector quantization for compression of multichannel ECG," *IEEE Trans. Biomed. Eng.*, vol. 37, no. 9, pp. 821-825, Sept. 1990.

[Marr72] H. J. L. Marriott, *Practical Electrocardiography*. Baltimore MD: the Williams and Wilkins, 5th Edition, 1972, 338 pp.

[Mast93] T. Masters, *Practical Neural Network Recipes in C++*. San Diego, CA: Academic Press, 1993, 493 pp.

[Mast95] T. Masters, *Advanced Algorithm for Neural Networks: A C++ Sourcebook*. New York, NY: John Wiley & Sons, 1995, 431 pp.

[Math01] The MathWorks, Inc., *Matlab: the Language of Technical Computing*, Version 6.1.8.450 Release 12.1, May 2001.

[Max60] J. Max, "Quantizing for minimum distribution," *IRE Transactions on Information Theory*, IT-6, pp. 7-12, Jan. 1960.

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

[Meye86] Y. Meyer, "Principe d'incertitude, bases hilbertiennes et algèbres d'opérateurs," In *Séminaire Bourbaki*, volume 662, Paris, 1986.

[Meye93] Y. Meyer, *Wavelet Algorithm and Application*. Philadephia, PN: SIAM,

1993, 133 pp.

[Micr01] MicroImages, Inc., *Reference Manual for the TNT Products V6.50.* http://www.microimages.com/refman/html/PROCE058.HTM, 2001.

[MiYe00] S. G. Miaou and H. L. Yen, "Quality driven gold washing adaptive vector quantization and its application to ECG data compression," *IEEE Trans. Biomed. Eng.*, vol. 47, no. 2, pp. 209-218, Feb. 2000.

[Mood99] G. B. Moody, "MIT-BIH database distribution," *http://ecg.mit.edu/*, Dec., 1999.

[Muel78] W. C. Mueller, "Arrhythmia detection program for an ambulatory ECG monitor," *Biomed. Sci. Instrument.*, vol. 14, pp. 81-85, 1978.

[NaCo93] G. Nave and A. Cohen, "ECG compression using long-term prediction," *IEEE Trans. Biomed. Eng.*, vol. 40, pp. 877-885, Sept. 1993.

[NaIw93] Y. Nagasaka and A. Iwata, "Data compression of long time ECG recording using BP and PCA neural networks," *IEICE Trans. Inform. Syst.*, vol. E76-D, no. 12, pp. 1434-1442, Dec. 1993.

[NyMK00] R. Nygaard, G. Melnikov, and A. K. Katsaggelos, "A rate distortion optimal ECG coding algorithm," *Technical Report*, Department of Electrical and Computer Engineering, Stavanger University College, Norway, 2000.

[OMGL96] S. Olmos, M. Millán, J. García, and P. Laguna, "ECG data compression with the Karhunen-Loeve transform," in *Proc. Computers in Cardiology*, IEEE Computer Society Press, pp. 253-256, 1996

[Ott93] Edward Ott, *Chaos in Dynamical Systems.* New York, NY: Cambridge University Press, 1993, 385 pp.

[PaBW79] O. Pahlm, P. O. Börjesson, and O. Werner, "Compact digital storage of ECGs," *Comput. Programs Biomed.*, vol. 9, pp. 292-300, 1979.

[Papo65] A. Papoulis, *Probability, Random Variables, and Stocastic Processes.* New York, NY: McGraw-Hill, 1965, 583 pp.

[PaSc87] K. Pawelzik and H. G. Schuster, "Generalized dimensions and entropies from a measured time series," *Phys. Rev.*, vol. A35(1), pp. 481-84, Jan. 1987.

[Pawl99] M. Pawlak, *Pattern Recognition.* Lecture Notes, Dept. of Electrical and Computer Engineering, University of Manitoba, 1999.

[PCFS80] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw, "Geometry

from a time series," *Phys. Rev. Letters*, vol. 45, no. 9, pp. 712-715, 1980.

[PeJS92] H. Peitgen, H. Jürgens, and D. Saupe, *Chaos and Fractals: New Frontiers of Science.* New York, NY: Springer-Verlag, 1992, 894 pp.

[Phil93] W. Philips, "ECG data compression with time-warped polynomials," *IEEE Trans. Biomed. Eng.*, vol 40, no. 11, pp. 1095-1101, Nov. 1993.

[PhJo92] W. Philips and G. D. Jonghe, "Data compression of ECG's by high-degree polynomial approximation," *IEEE Trans. Biomed. Eng.*, vol 39, no. 4, pp. 330-337, Apr. 1992.

[Phys01] PhysioNet, "PhysioBank archive index," *http://www.physionet.org/physiobank/database/#ecg*, Oct 1, 2001.

[PiPl84] T. C. Pilkington and R. Plonsey, *Engineering Contributions to Biophysical Electrocardiography.* New York: IEEE, pp. 134-187, 1984.

[PlBa88] R. Plonsey and R. C. Barr, *Bioelectricity: A Quantitative Approach.* New York, NY: Plenum Press, 1988, 305 pp.

[RaAn92] F. Ravelli and R. Antolini, "Complex dynamics underlying the human electrocardiogram," *Biol. Cybern.*, vol. 67, pp. 57-65, 1992.

[RaRL78] L. R. Rabiner, A. E. Rosenberg, and S. E. Levinson, "Considerations in dynamic time warping algorithms for discrete word recognition," *IEEE Trans. Acous., Speech, Signal Processing*, vol. ASSP-26, No. 1, pp. 575-582, Dec. 1978.

[RaSa97] A. G. Ramakrishnan and S. Saha, "ECG coding by wavelet-based linear prediction," *IEEE Trans. Biomed. Eng.*, vol. 44, no. 12, pp. 428-434, Dec., 1997.

[Rawl91] C. A. Rawlings, *Electrocardiography.* Redmond, WA: SpacLabs Inc., 1991, 129 pp.

[ReMu86] B. R. S. Reddy and I. S. N. Murthy, "ECG data compression using Fourier descriptions," *IEEE Trans. Biomed. Eng.*, vol. 33, pp. 428-434, 1986.

[Riou93] O. Rioul, "Regular wavelets: A discrete time approach," *IEEE Trans. Signal Processing*, vol. 41, no. 12, pp. 3572-3579, Dec. 1993.

[RiVe91] O. Rioul and M. Vetterli, "Wavelets and signal processing," *IEEE Signal Processing Magazine*, vol. 8, no. 4, pp. 14-38, Oct. 1991.

[RuBP76] U. E. Ruttimann, A. S. Berson, and H. V. Pipberger, "ECG data compression by linear prediction," *Computers Cardiol.*, St. Louis, MO, pp. 323-354, 1976.

[RuNe76] S. Rush and C. Nelson, "The effect of electric inhomogeneity and anisotropy of thoracic tissue on the field of the heart," in *The Theoretical Basis of Electrocardiography*, C. V. Nelson and D. B. Geselowitz, Eds. Oxford, England: Clarendon, pp. 323-354, 1976.

[RuPi79] U. E. Ruttimann and H. V. Pipberger, "Compression of the ECG by prediction or interpolation and entropy encoding," *IEEE Trans. Biomed. Eng.*, vol BME-26, pp. 613-623, Nov. 1979.

[SaCh71] H. Sakoe and S. Chiba, "A dynamic programming approach to continuous speech recognition," in *Proc. Int. Cong. Acous.*, budapest, Hungary, Paper 20C-13, 1971.

[SaCh78] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. Acous., Speech, Signal Processing*, vol. ASSP-26, No. 1, pp. 43-49, Feb. 1978.

[Sayo00] K. Sayood, *Introduction to Data Compression*. San Francisco, CA: Morgan Kaufmann Publishers, 2nd Edition, 2000, 600 pp.

[Sayo96] K. Sayood, *Introduction to Data Compression*. San Francisco, CA: Morgan Kaufmann Publishers, 1996, 475 pp.

[Schr93] T. Schreiber, "Extremely simple nonlinear noise-reduction method," *Phys. Rev. E*, vol. 47, no. 4, pp. 2401-2404, Apr. 1993.

[ScKa96] T. Schreiber and D. T. Kaplan, "Nonlinear noise reduction for electrocardiograms," *Chaos*, no. 6, pp. 87-92, 1996.

[Shaw97] D. B. Shaw, *Classification of transmitter Transients Using Fractal Measures and Probabilistic Neural Networking.* M.Sc. Thesis, Dept. of Electrical and Computer Engineering, University of Manitoba, 1997.

[SkGo80] J. Sklansky and V. Gonalez, "Fast polygonal approximation of digitized curves," *Pattern Recog.*, vol. 12, pp. 327-331, 1980.

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

[StBD79] D. Stewart, D. Berghofer, and R. G. Dower, "Data compression of ECG signals," *Eng. Foundation Conf. Computerized Interpretation of the ECG*, Asi-

lomar, CA, pp. 162-177 & A1-A8, Jan. 1979.

[StPI86] P. C. Stanley, T. C. Pilkington, and R. E. Ideker, "Accuracy of torso extension method for anisotropic analysis," in *Proc. 8th Ann. Conf. IEEE Eng. Med. Biol. Soc.*, Dallas-Ft. Worth, TX, pp. 299-301, 1986.

[Tai91] S. C. Tai, "SLOPE—a real-time ECG data compresser," *Medical & Biological Engineering & Computing*, vol. 29, pp. 175-179, Mar. 1991.

[Tai92] S. C. Tai, "Six-band sub-band coder on ECG waveforms," *Medical & Biological Engineering & Computing*, vol. 30, pp. 187-192, Mar. 1992.

[Take81] F. Takens, "Detecting strange attractors in turbulence," *Lecture Notes in Mathematics 898*, Berlin, NY: Springer-Verlag, pp. 366-381, 1981.

[Trim89] D. W. Trim, *Engineering Mathematics.* Ruskin Publishing, pp. 469-470, 1989.

[TrJT96] Ø. D. Trier, A. K. Jain, and T. Taxt, "Feature extraction methods for character recognition—A survey," *Pattern Recognition*, vol. 29, no. 4, pp. 641-662, 1996.

[Veko95] M. Vetterli, and J. Kovecrvic. *Wavelets and Subband Coding.* Englewood Cliffs, NJ: Prentice Hall, 1995, 488 pp.

[Vera99] E. Vera, *Fractal Modelling of Residual in Linear Predictive Coding of Speech.* M. Sc. Thesis, Dept. of Electrical and Computer Engineering, University of Manitoba, Canada, 1999, 199 pp.

[ViHa93] G. Vines and M. H. Hayes, "Nonlinear address maps in a one-dimensional fractal model," *IEEE Trans. Signal Processing*, vol. 41, no. 4, pp. 1721-1724, Apr. 1993.

[Vine93] G. Vines, *Signal Modelling with Iterated Function Systems.* Ph.D. Thesis, Georgia Institute of Technology, 1993.

[Vouk95] P. C. Voukydis, "An artificial neural network system for classification of the cardiac rhythm," *ICSPAT*, vol. 1, pp. 248-252, Boston, MA, USA, Oct. 1995.

[WaKi93] L. M. Wall and W. Kinsner, "A fractal block coding technique employing frequency sensitive competitive learning," *Proc. IEEE Communications, Computers & Power Conf.*, Saskatoon, Canada, pp. 320-329, May 1993.

[Wall93] L. M. Wall, *Reduced Search Fractal Block Coding Using Frequency Sensitive Neural Networks.* M.Sc. Thesis, Dept. of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, 1993, 227 pp.

[WaYu97] B. Wang and G. Yuan, "Compression of ECG data by vector quantization," *IEEE Eng. Med. Biol. Mag.*, pp. 23-26, Jul.-Aug. 1997.

[WeSD98] C. D. Werner, F. B. Sachse, and O. Dössel, "Electrical excitation of the human heart: a comparison of electrical source distributions in models of different spatial resolution," *Proc. Computer in Cardiology*, Cleveland, Ohio, pp. 309-312, 1998.

[WHML77] M. E. Womble, J. S. Halliday, S. K. Mitter, M. C. Lancaster, and J. H. Triebwasser, "Data compression for storing and transmitting ECGs/VCGs," *Proc. IEEE*, vol. 65, pp. 702-706, May 1977.

[WhWo80] J. Whitman and H. K. Wolf, "An encoder for electrocardiogram data with wide range of applicability," in *Optimization of Computer ECG Processing*, H. K. Wolf and P. W. MacFarlane, Eds. NY: North-Holland, pp. 87-90, 1980.

[Will97] G. P. Williams, *Chaos Theory Tamed.* Washington, DC: Joseph Henry, 1997, 516 pp.

[WKDE89] P. Wach, R. Killmann, F. Dienstl, and C. Eichtinger, "A computer model of human ventricular myocardium for simulation of ECG, MCG, and activation sequence including reentry rhythms," *Basic Research in Cardiology*, vol. 4, no. 4, 1989.

[WOHH95] D. Wei, O. Ozazaki, K. Harumi, E. Harasawa, and H. Hosaka, "Comparative simulation of excitation and body surface elctrocardiogram with isotropic and anisotropic computer heart models," *IEEE Trans. Biomedical Engineering*, vol. 42, no. 4, pp. 343-357, 1995.

[Worn96] G. W. Wornell, *Signal Processing with Fractals: A Wavelet-Based Approach.* Upper Saddle River, NJ: Prentice-Hall, 1996, 177 pp.

[WoSR72] H. K. Wolf, J. Sherwood, and P. M. Rautaharju, "Digital in transmission of electrocardiograms-A new approach," *Pro. 4th Can. Med. Biol. Conf.*, pp. 39a-39b, 1972.

[WoZi80] M. E. Womble and A. M. Zied, "A statistical approach to ECG/VCG data compression," in *Optimization of Computer ECG Processing*, H. K. Wilf and P. W. MacFarlane, Eds. NY:North-Holland, pp. 91-101, 1980.

[Yano00] F. G. Yanowitz, *ECG Learning Center.* http://medstat.med.utah.edu/kw/ecg/mml/ecg_torso.html, Nov. 16, 2000.

[ZiWo79] A. M. Zied and E. Womble, "Application of a partitioned Karhonen-Loeve expansion scheme to ECG/VCG data compression," in *Proc. Eighth New*

England Bioeng. Conf., vol. 7, pp. 102-105, 1979.

[ZZTW99] X. S. Zhang, Y. S. Zhu, N. V. Thakor, and Z. Z. Wang, "Detecting ventricular tachycardia and fibrillation by complexity measure," *IEEE Trans. on Biomed. Eng.*, vol. 46, no. 5, pp. 548-555, May 1999.

APPENDIX A

SOME PERTINENT MATHEMATICAL BACKGROUND

A.1 Linear Representation of Function

A.1.1 Function in Vector Space

In mathematics, a signal can be represented by a function $f(x)$ and treated as a vector, denoted as *f*. In general, the signal has finite energy and is square integrable in an $L^2(\mathbf{R})$ space.

$$\int_{\mathbf{R}} |f(x)|^2 dx < \infty \tag{A.1}$$

where $\mathbf{R}$ is a set of real numbers.

Definition A.1: A functional space defined as: $\left\{ f: \mathbf{R} \to \mathbf{R} \Big/ \int_{\mathbf{R}} |f(x)|^2 dx < \infty \right\}$ is called as $L^2(\mathbf{R})$.

Definition A.2: A Hilbert space is an inner product space which, as a metric space, is complete.

In the Hilbert space $L^2(\mathbf{R})$, an inner product of real valued functions, $f_1(x)$ and $f_2(x)$, is defined as

$$\langle f_1, f_2 \rangle = \int_{\mathbf{R}} f_1(x) f_2(x) dx \tag{A.2}$$

A norm $\|\bullet\|$ that defines a length of a vector (function) in $L^2(\mathbf{R})$ is induced from the inner product [Krey89].

$$\|f\| = \sqrt{\langle f, f \rangle} \tag{A.3}$$

A.1.2 Orthonormal Basis and Function Decomposition

The inner product of the Hilbert space can be used to see if any pair of vectors are orthogonal.

Definition A.3: A vector $f_1 \in F$ is said to be orthogonal to another vector $f_2 \in F$ if

$$\langle f_1, f_2 \rangle = 0 \tag{A.4}$$

where F is a set.

Specially, the inner product can determine whether a vector is orthonormal.

Definition A.4: A set of vectors $\{f_k(x)\}_{k \in \mathbf{Z}}$ is said to be orthonormal if

$$\langle f_i, f_j \rangle = \delta(i-j) \tag{A.5}$$

where the delta function $\delta(x) = 0$ for $x \in \mathbf{R}$ except $\delta(0) = 1$.

Definition A.5: A basis of a vector space V is defined as a subset $v_1, v_2, \cdots, v_n$ of vectors in V that are linearly independent and span V. Consequently, if $v_1, v_2, \cdots, v_n$ is a

list of vectors in V, then these vectors form a basis if and only if every $v \in V$ can be uniquely written as

$$v = a_1 v_1 + a_2 v_2 + \cdots + a_n v_n \tag{A.6}$$

where $a_1, a_2, \cdots, a_n$ are elements of $\mathbf{R}$.

A vector space V will have many different bases, but each will have the same number of basis vectors. The number of basis vectors in V is called the dimension of V. Every spanning list in a vector space can be reduced to a basis of the vector space.

Definition A.6: The basis $v_1, v_2, \cdots, v_n$ is orthonormal basis if

$$\langle v_i, v_j \rangle = \delta(i - j) \tag{A.7}$$

Under the orthonormal basis, the component a_i can be obtained from the inner product

$$a_i = \langle v, v_i \rangle \tag{A.8}$$

The simplest example of a basis is the standard basis in $\mathbf{R}^n$ consisting of the coordinate axes. For example, in $\mathbf{R}^2$, the standard basis consists of two vectors $v_1 = (1, 0)$ and $v_2 = (0, 1)$. Any vector $v = (a, b)$ can be written uniquely as the linear combination $v = av_1 + bv_2$. Indeed, a vector is defined by its coordinates. The vectors $v_1 = (3, 2)$ and $v_2 = (2, 1)$ are also a basis for $\mathbf{R}^2$ because any vector $v = (a, b)$ can be uniquely written as $v = (-a + 2b)v_1 + (2a - 3b)v_2$.

Equation A.6 shows the important decomposition property of a vector under a basis. It is the foundation of orthogonal transform of functions, such as Fourier transform and wavelet transform.

A.2 Metric Space and Affine Transform for IFS

We have already introduced some knowledge about vector space in Section A.1. This section will further provide basic notation, definitions, and information relating to topological geometry. We begin from the definition of space. A space means a set with structure on it. A metric space, a map, and an affine transformation can be defined on the space.

Definition A.7: A metric space ($\mathbf{X}$, d) is a space, or a set, $\mathbf{X}$ together with a real-valued function d: $\mathbf{X} \times \mathbf{X} \rightarrow \mathbf{R}$, which measures the distance between pairs of points x and y in $\mathbf{X}$. Suppose that d has the following properties:

1) $d(x, y) = d(y, x), \quad \forall x, y \in \mathbf{X}$ (A.9)

2) $0 < d(x, y) < \infty, \quad \forall x, y \in \mathbf{X}, \quad x \neq y$ (A.10)

3) $d(x, x) = 0, \quad \forall x \in \mathbf{X}$ (A.11)

4) $d(x, y) \leq d(x, z) + d(z, y), \quad \forall x, y, z \in \mathbf{X}$ (A.12)

The d is called a metric on the space $\mathbf{X}$. Some examples of metric space include:

1) The set of all real number $\mathbf{R}$ with

$$d(x, y) = |x - y| \tag{A.13}$$

2) The Cartesian plane denoted $\mathbf{R}^2$ with the Euclidean metric given by

$$d_2(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \tag{A.14}$$

where the notation x_i and y_i are the ith elements of the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively.

It should be noted that different metrics, defined on the same set, may form different spaces. For example the function

$$d(\boldsymbol{x}, \boldsymbol{y}) = |x_1 - y_1| + |x_2 - y_2| \tag{A.15}$$

is also metric on the Cartesian plane but is a different metric space from $(\mathbf{R}^2, d_2)$. An optimal choice of a metric space is dependent on the application.

Definition A.8: A sequence $\{x_n\}_{n=1}^{\infty}$ of points in a metric space ($\mathbf{X}$, d) is called a Cauchy sequence if, for any given number $\varepsilon > 0$, there is an integer $N > 0$ such that

$$d(x_n, x_m) < \varepsilon, \quad \forall m, n > N \tag{A.16}$$

Definition A.9: A metric space ($\mathbf{X}$, d) is said to be complete if every Cauchy sequence in $\mathbf{X}$ converges.

Definition A.10: Let $\mathbf{X}$ be a space. A transform, map, or mapping on $\mathbf{X}$ is a function f: $\mathbf{X} \rightarrow \mathbf{R}$. If $S \subset \mathbf{X}$, then $f(S) = \{f(x)\ x \in S\}$. The function f is one-to-one if $x, y \in \mathbf{X}$ with $f(x) = f(y)$ implies $x = y$. It is onto if $f(\mathbf{X}) = \mathbf{X}$. It is called invertible

if it is one-to-one and onto: in this case, it is possible to define a transform $f^{-1}: \mathbf{X} \to \mathbf{X}$, called the inverse of f, by $f^{-1}(y) = x$, where $x \in \mathbf{X}$ is the unique point such that $y = f(x)$.

Definition A.11: Let f: $\mathbf{X} \to \mathbf{X}$ be a transform on a space. The forward iterates of f are transforms $f^{\circ n}: \mathbf{X} \to \mathbf{X}$ defined by $f^{\circ 0}(x) = x$, $f^{\circ 1}(x) = f(x)$, $f^{\circ(n+1)}(x) = f \circ f^{\circ(n)}(x) = f(f^{\circ(n)}(x))$ for $n = 0, 1, 2, \cdots$. If f is invertible, the backward iterates of f are transforms $f^{\circ(-m)}(x): \mathbf{X} \to \mathbf{X}$ defined by

$$f^{\circ(-1)}(x) = f^{-1}(x), \; f^{\circ(-m)}(x) = (f^{\circ m})^{-1}(x) \text{ for } m = 1, 2, 3, \cdots. \quad \text{(A.17)}$$

Definition A.12: A sequence $\{x_n\}_{n=1}^{\infty}$ of points in a metric space $(\mathbf{X}, d)$ is said to converge to a point $x \in \mathbf{X}$ if, for any given number $\varepsilon > 0$, there is an integer $N > 0$ such that

$$d(x_n, x) < \varepsilon, \; \forall n \geq N \quad \text{(A.18)}$$

The point $x \in \mathbf{X}$, to which such a sequence converges, is called the limit of the sequence.

Definition A.13: Let $B \in \mathbf{X}$ be a subset of a metric space $(\mathbf{X}, d)$. B is compact if every infinite sequence $\{x_n\}_{n=1}^{\infty}$ in B contains a subsequence having a limit in B.

Definition A.14: Let $(\mathbf{X}, d)$ be a compact metric space. Then a Hausdorff space $H(\mathbf{X})$ denotes the space whose points are the compact subsets of $\mathbf{X}$, other than the empty set.

Definition A.15: Let (**X**, d) be a compact metric space. Let A and B belong to $H(\mathbf{X})$. Define

$$d(A, B) = max\{d(x, B) : x \in A\} \tag{A.19}$$

Definition A.16: Let (**X**, d) be a compact metric space. Then the Hausdorff distance between the subset A and subset B in $H(\mathbf{X})$ is defined by

$$d_h(A, B) = max(d(A, B), d(B, A)) \tag{A.20}$$

APPENDIX B

SOURCE CODE

Table B.1: List of source code files.

File Name	Page	Description
LorenzSolution.m	B-3	numerical solution for the Lorenz system
LorenzReconstruct.m	B-4	strange attractor reconstruction of the Lorenz system
autocorrelation.m	B-5	autocorrelation function of a time series
FalseNearestNeighbour.c	B-7	false nearest neighbour calculation of a time series
attractor.m	B-9	strange attractor reconstruction of a time series
PairCorrelation.c	B-10	calculating the extended pair correlation of a time series
NoisyECGChaosFilter.cpp	B-14	denoising a time series by principal component projection
Jacobi.h	B-18	include file header of NoisyECGChaosFilter.cpp
Jacobi.cpp	B-19	function for solving Jacob matrix
NoisyECGWaveletFilter.m	B-22	denoising a time series in wavelet domain
WienerFilter53.m	B-23	linear Wiener filter with 53 orders
FilterEvaluate.m	B-24	SNR gain and noise reduction factor calculation
ColorNoiseGen.m	B-25	coloured noise generator
NIFS.cpp	B-27	NIFS compression with or without signal partitioning
IFSReconstruct.m	B-37	signal reconstruction based on NIFS compression
NonuniformQuantizer.cpp	B-43	nonuniform quantizer design
FitCoef.m	B-46	main routine of finding distribution parameters for a data set

Table B.1: List of source code files.

File Name	Page	Description
FitCoef1.m	B-47	function for finding a mean and variance from a data set
hist55.m	B-48	function for giving the histogram of the coefficients from NIFS.cpp
ReadMeanVar.m	B-49	design of a nonuniform quantizer according to distribution parameters
VFDT_Segment.m	B-51	function for calculating the VFDT
segment.m	B-52	main routine for partitioning a signal according to complexity measures
moment.m	B-53	main routine for calculating the moment-invariant
moment_fun.m	B-54	function for calculating the moment-invariant
ISODATA.m	B-56	an unsupervised clustering algorithm for the training set
pnn1_error.m	B-61	PNN training and classification error test
MomentResidual.m	B-62	reconstruction error calculation based on PNN classification and single template reconstruction
recons1_error.m	B-64	reconstruction error calculation based on PNN classification and template-averaged reconstruction
DTW_Classification.c	B-68	DTW classification based on minimal average residual
h_d_p.m	B-76	template partitioning
pwln_residual.m	B-77	residual calculation for piecewise linear segments
waveform_detect.m	B-80	function for detecting waveforms in variance domain
linear_nml.m	B-82	function for linear contracting/dilating a segment

B.1 Strange Attractor Reconstruction

B.1.1 Numerical Solution for Lorenz System and the Reconstruction

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Name        : LorenzSolution.m
Procedure   : numerical solution for the Lorenz system
Date        : 17/Oct./1999
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
a=10; b=8/3; r=28;
N=5000;
dt=0.02; dt2=dt/2; dt3=1/3;
x=0; y=0.5; z=20;

for i=1:N
   d0x=a*(y-x)*dt2;
   d0y=(x*(r-z)-y)*dt2;
   d0z=(x*y-b*z)*dt2;
   xt=x+d0x;yt=y+d0y;zt=z+d0z;
   d1x=a*(yt-xt)*dt2;
   d1y=(xt*(r-zt)-yt)*dt2;
   d1z=(xt*yt-b*zt)*dt2;
   xt=x+d1x;yt=y+d1y;zt=z+d1z;
   d2x=a*(yt-xt)*dt;
   d2y=(xt*(r-zt)-yt)*dt;
   d2z=(xt*yt-b*zt)*dt;
   xt=x+d2x;yt=y+d2y;zt=z+d2z;
   d3x=a*(yt-xt)*dt2;
   d3y=(xt*(r-zt)-yt)*dt2;
   d3z=(xt*yt-b*zt)*dt2;
   xx(i)=x+(d0x+d1x+d2x+d3x)*dt3;
   yy(i)=y+(d0y+d1y+d2y+d3y)*dt3;
   zz(i)=z+(d0z+d1z+d2z+d3z)*dt3;
   x=xx(i);y=yy(i);z=zz(i);
end

plot_attractor=1;
if  plot_attractor==1
   set(axes,'fontsize',20);
   plot3(xx,yy,zz);
   xlabel('x');ylabel('y');zlabel('z')
else
   set(axes,'fontsize',20);
   plot(yy);
   xlabel('t');ylabel('y');
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Name        : LorenzReconstruct.m
Procedure   : strange attractor reconstruction of the Lorenz system
Date        : 17/Oct./1999
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Lag=L;
length=size(xx);
T1=length(1,2)
specimen=xx;

clear D;
clear D1;
clear D2;
clear DD;
clear DD1;
clear DD2;

D=specimen;
D1([1:(T1-Lag+1)])=D([Lag:T1]);
D2([1:(T1-2*Lag+1)])=D([2*Lag:T1]);

DD([1:(T1-2*Lag+1)])=D([1:(T1-2*Lag+1)]);
DD1([1:(T1-2*Lag+1)])=D1([1:(T1-2*Lag+1)]);
DD2([1:(T1-2*Lag+1)])=D2([1:(T1-2*Lag+1)]);

set(axes,'fontsize',20);
plot3(DD,DD1,DD2);
xlabel('x(n)');
if  Lag==2
   ylabel('x(n+2)');
   zlabel('x(n+4)');
elseif  Lag==5
   ylabel('x(n+5)');
   zlabel('x(n+10)');
elseif  Lag==10
   ylabel('x(n+10)');
   zlabel('x(n+20)');
end
```

B.1.2 Reconstruction of Strange Attractor of ECG and the Multifractal

```
**************************************************************************************
Name        : autocorrelation.m
Procedure   : autocorrelation function of a time series
Date        : 20/Oct./1999
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**************************************************************************************
% ------------------------------------------------------------------------------------------------
%- Calculating autocorrlation function of the ECG signal. When the function value drops to 1-1/e, we take
%- the corresponding time as time delay of the time series.
%------------------------------------------------------------------------------------------------
No_estimation_of_period=380;Start_point_addition=50;  R_S_distance_coe=1.5;
set(axes,'fontsize',20);

for i=sp:No_estimation_of_period+sp,
    j=i-sp+1;
    b(j)=a(i);
end

maxima=max(b);
for j=1:No_estimation_of_period,
    if b(j) ==maxima
        T0=j+sp
        A0=maxima;
    end
end

while1
    clear c;
    for i=T0+Start_point_addition:T0+No_estimation_of_period,
        j=i-T0;
        c(j)=a(i);
    end
    maxima_0=max(c);
    for j=1:No_estimation_of_period,
        if c(j) ==maxima_0
            Start_point=j+T0;
            specimen_peak=maxima_0;
        end
    end
    clear specimen;
    for i=1+T0:Start_point,
        j=i-T0;
        specimen(j)=a(i);
    end
    T1=Start_point-T0;T0=Start_point;
    clear cycle_data;
    cycle_data=specimen-mean(specimen);cycle_length=size(cycle_data);

    clear correlate_coe;
    for tt=1:cycle_length(1,2)-200
```

```
        t(tt)=tt-1;
        square_sum_x=0;
        for i=1:cycle_length(1,2)-t(tt)
            square_sum_x=square_sum_x+cycle_data(i).^2;
        end
        autocorrelate_sum_x=0;
        for i=1:cycle_length(1,2)-t(tt)
            autocorrelate_sum_x=autocorrelate_sum_x+cycle_data(i)*cycle_data(i+t(tt));
        end
        correlate_coe(tt)=autocorrelate_sum_x/square_sum_x;
    end
    plot(t,correlate_coe);

    Threshold=1-exp(-1);
    for t=1:cycle_length(1,2)-200
        if correlate_coe(t) <= Threshold
            Lag=t-1;
            text(10,0.9,sprintf('Lag=%d',Lag),'fontsize',18);
            break;
        end
    end
    xlabel('Lag');ylabel('Autocorrelation');
end
```

```
*****************************************************************************
Name        : FalseNearestNeighbour.c
Procedure   : false nearest neighbour calculation of a time series
Date        : 23/Nov./1999
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
*****************************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
#include <iostream.h>

const intLag=1;
const intR=10;
const intM=7;
const intN1=10000;

int main()
{
    FILE *fin, *fout;
    floattime[N1+1];
    doubleD1[N1+1],D2[N1+1],D3[N1+1];
    doubleresult[M+1][R+1], Rtol[R+1];
    int  m,n,i,j,r,N,i_n[N1+100];
    doublesum, mindist, minind, nextdist, distance, diff_time;
    doubleRv, Ra, Rb, Rt, Rr, Atol;
    int  embedmin, embedmax;

    embedmin=1; Rv=0.6; Atol=1;
    Rtol[0]=5;
    for  (i=1;i<=R;i++){
        Rtol[i]=i-0.99;
    }

    fin=fopen("length.dat","r+");
    fscanf(fin,"%d\n",&embedmax);
    fscanf(fin,"%d\n",&N);
    fclose(fin);
    cout<<"embedmax="<<embedmax<<"N="<<N<<endl;

    for  (i=embedmin;i<=embedmax;i++){
    for  (j=1;j<=R;j++){
        result[i][j]=0;
    }
    }

    fin=fopen("/home/ee/hbin/database/data/x_101.dat","r+");
    Ra=0.0;Rr=0.0;
    for  (i=1;i<=N;i++){
        fscanf(fin,"%f\n",&time[i]);
        Ra+=time[i];
        Rr+=time[i]*time[i];
    }
```

```
    fclose(fin);
    Ra=Ra/N;
    Ra=sqrt(Rr/N-Ra*Ra);
    cout<<"RA="<<Ra<<endl;

    for (m=embedmin;m<=embedmax;m++){
        for (i=1;i<=N;i++){
            mindist=10000;
            nextdist=0;
            for (j=0;j<=m;j++){
                i_n[j]=i+j;
            }
            for (j=1;j<=N;j++){
            if  (i!=j){
                distance=0.0;
                for (n=0;n<=m-1;n++){
                    diff_time=time[i_n[n]]-time[j+n];
                    distance+=(double) diff_time*diff_time;
                }
                if  (distance<mindist){
                    mindist=distance;
                    diff_time=time[i_n[m]]-time[j+m];
                    nextdist=(double) diff_time*diff_time;
                }
            }
            }
            D1[i]=mindist; D2[i]=nextdist;
        }

        fout=fopen("False_NN.dat","w");
        for (i=1;i<=N;i++){
            Rr=pow(D2[i],0.5)/pow(D1[i],0.5);
            Rb=(double) pow(D1[i]+D2[i],0.5);
            for (r=1;r<=R;r++){
                if      (Rr>Rtol[r] || Rb>Ra*Atol){
                        result[m][r]++;
                }
            }
        }
    }
    for (j=1;j<=R;j++){
        for (i=embedmin;i<=embedmax;i++){
            result[i][j]=result[i][j]/N;
            fprintf(fout,"%f",result[i][j]);
        }
        fprintf(fout,"\n");
    }
    fclose(fout);
}
```

```
**********************************************************************************
Name        : attractor.m
Input       : a[]
Procedure   : strange attractor reconstruction of a time series
Date        : 23/Sept./1999
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**********************************************************************************
%---------------------------------------------------------------------------
%  Reconstructing attractor for ECG time series with lag of 4.
%---------------------------------------------------------------------------
    Lag=4;
    length=size(a);
    T1=length(1,1)
    T1=60000
    specimen([1:T1])=a([1:T1]);
    clear D;
    clear D1;
    clear D2;
    clear DD;
    clear DD1;
    clear DD2;

    D=specimen;
    clear specimen;
    D1([1:(T1-Lag+1)])=D([Lag:T1]);
    D2([1:(T1-2*Lag+1)])=D([2*Lag:T1]);

    DD([1:(T1-2*Lag+1)])=D([1:(T1-2*Lag+1)]);
    clear D;
    DD1([1:(T1-2*Lag+1)])=D1([1:(T1-2*Lag+1)]);
    clear D1;
    DD2([1:(T1-2*Lag+1)])=D2([1:(T1-2*Lag+1)]);
    clear D2;

    set(axes,'fontsize',20);;
    plot3(DD,DD1,DD2,'.');
    xlabel('x(n) [mV]');
    ylabel('x(n+4) [mV]');
    zlabel('x(n+8) [mV]');
```

```
*************************************************************************************
Name        : PairCorrelation.c
Input       : length.dat, x_101.dat
Output      : Renyi_dim.dat
Procedure   : calculating the extended pair correlation of a time series for different boxes
Date        : 22/Oct./1999
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
*************************************************************************************
/*-----------------------------------------------------------------------------------------------------------------------
This program tests whether Renyi dimension spectrum is affected by the radii of a box or not. The time lag
of 4 of the ECG signal comes from autocorrelation.m. Embedding dimension is determined by this program
-----------------------------------------------------------------------------------------------------------------------*/
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>

const intsp=1;
const intLag=4;
const intembed=2;
const inteuc=2;
const intfile_length_1=30000;
const intmax_period=500;

const intNo_estimation_of_period=380;
const intStart_point_addition=50;
const floatR_S_distance_coe=1.5;
const floatinitial_probability=1.0;
const intvelnum1=50;

int main()
{
    FILE    *fin, *fout;
    float   a[file_length_1+1],cycle_data[file_length_1+1];//float  a[216000];
    float   b[max_period];
    double  c[max_period];
    long int Number_M[velnum1],Lag_embedding[50],integer_Lag,Total_points;
    long int total_points;
    double  Hq[velnum1][80];
    float   x[80], probability_threshold;

    double  distance,sum,maxima,maxima_0,A0,Invers_M,Delta_step;
    float   probs[velnum1][file_length_1],vels[velnum1];
    int     i1[file_length_1],Embedding_step;
    long int i,j,k,n,t,v,embed,time_series_length,dimension_size,Lag_length;
    float   q,qq;
    long int T0,T1,M,Start_point,size_c,velnum,Q_number;
    float   Delta_start,Delta_end,Q_step;

    fin=fopen("length.dat","r+");
    fscanf(fin,"%d\n",&dimension_size);
    fscanf(fin,"%d\n",&time_series_length);
```

```
fscanf(fin,"%d\n",&velnum);
fscanf(fin,"%f\n",&Delta_start);
fscanf(fin,"%f\n",&Delta_end);
fscanf(fin,"%d\n",&Q_number);
fscanf(fin,"%f\n",&Q_step);
fscanf(fin,"%d\n",&Embedding_step);
cout<<"dimension_size="<<dimension_size<<"  time_series_length="<<time_series_length;
cout<<"velnum="<<velnum<<"Delta_start="<<Delta_start<<"Delta_end="<<Delta_end<<endl;
cout<<"Q_number="<<Q_number<<"Q_step="<<Q_step<<endl;
fclose(fin);

Delta_step=(Delta_end-Delta_start)/(velnum-1);
for  (i=0;i< velnum;i++){
     distance=i*Delta_step+Delta_start;
     vels[i]=pow(2,2*distance);
     cout<<distance<<endl;
}

fin=fopen("/home/ee/hbin/database/data/x_101.dat","r+");
for  (i=0;i<=file_length_1;i++){
     fscanf(fin,"%f\n",&a[i]);
}
fclose(fin);
cout<<a[file_length_1]<<endl;
fout=fopen("Renyi_dim.dat","w");

for (i=sp;i<=No_estimation_of_period+sp;i++)
{    j=i-sp+1;
     b[j]=a[i];
}

maxima=(b[1]);
for (i=1;i<=No_estimation_of_period;i++)
{    if   (maxima<b[i]){
          maxima=b[i];
     }
}
printf("maxima=%f\n",maxima);

for (j=1;j<=No_estimation_of_period;j++)
{    if   (b[j]==maxima)
     {    T0=j+sp;
          A0=maxima;
     }
}

for (i=T0+Start_point_addition;i<=T0+No_estimation_of_period;i++)
{    j=i-T0;
     c[j]=a[i];
}

maxima_0=c[Start_point_addition];
for (i=Start_point_addition;i<=No_estimation_of_period;i++)
{    if   (maxima_0<c[i]){
```

```
            maxima_0=c[i];
        }
    }

    for (j=1;j<=No_estimation_of_period;j++)
    {   if   (c[j]==maxima_0)
        {   Start_point=j+T0;
            //specimen_peak=maxima_0;
        }
    }

    for (i=T0;i<=time_series_length+T0;i++)
    {   j=i-T0;
        cycle_data[j]=a[i];
    }
    T1=time_series_length;
    T0=Start_point;
    printf("T0=%d  T1=%d\n",T0,T1);
    size_c=T1;
    for  (embed=1;embed<=dimension_size;embed=embed+Embedding_step){
        cout<<"Embedding dimension="<<embed<<endl;
        Lag_length=(embed-1)*Lag;
        for  (i=0;i<velnum;i++){
            Number_M[i]=0;
        }
        for  (i=0;i<=embed-1;i++){
            Lag_embedding[i]=i*Lag;
        }
        for  (i=0;i<velnum;i++){
        for  (j=0;j<=size_c;j++){
            probs[i][j]=initial_probability;
        }
        }
        total_points=0;
        for  (i=(Lag_length+1);i<=size_c;i++){
            for  (n=0;n<embed;n++){
                i1[n]=i-Lag_embedding[n];
            }
            for  (j=(i+1);j<=size_c;j++){
                sum=0.0;
                for  (n=0;n<embed;n++){
                    distance=cycle_data[i1[n]]-cycle_data[j-Lag_embedding[n]];
                    sum=sum+distance*distance;
                }
                for  (k=0;k<velnum;k++){
                    if   (sum <= vels[k]){
                        probs[k][i]++;
                        probs[k][j]++;
                    }
                }
            }
        }
        Total_points=size_c-Lag_length;
```

```
        for  (k=0;k<velnum;k++){
             distance=0.0;
             integer_Lag=0;
             for  (j=(Lag_length+1);j<=size_c;j++){
                  probs[k][j]=(double)probs[k][j]/(double)Total_points;
             }
        }

        k=0;
        probability_threshold=initial_probability/(double) Total_points;
        for  (q=-Q_number;q<=Q_number;q=q+Q_step){
             qq=q-1.0;
             for  (v=0;v<velnum;v++){
                  Hq[v][k]=0.0;
                  for  (i=Lag_length+1;i<=size_c;i++){
                       Hq[v][k]=Hq[v][k]+pow(probs[v][i],qq);
                  }
                  Hq[v][k]=Hq[v][k]/(double)Total_points;
                  Hq[v][k]=log(Hq[v][k])/(qq*log(2));
                  fprintf(fout,"%f  ",Hq[v][k]);
             }
             k=k+1;
             fprintf(fout,"\n");
        }
    }
    cout<<"Total_points="<<Total_points<<endl;
    fclose(fout);
}
```

B.2 SNR Gain Improvement of ECG by Denoising

```
*************************************************************************************
Name         : NoisyECGChaosFilter.cpp
Input        : parameters.dat
Include file : Jacobi.h
Function     : Jacobi.cpp
Procedure    : denoising a time series by principal component projection
Date         : 18/June/1999
Version      : 1.0
Designer     : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
*************************************************************************************
/*-------------------------------------------------------------------------------------------------------------------
Step: 1. Embedding a time series x(i), by Xi={x(i-Lag*k),...,x(i+Lag*k)};
      2. Find the neighbourhood with radius r in the phase space; and
      3. Project the neighbourhood to a new coordinate system based on principal components
-------------------------------------------------------------------------------------------------------------------*/

#include <stdio.h>
//#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <malloc.h>
#include "Jacobi.h"

onst int      file_length= 66000;//216000;
const int     Max_embedding_dimension=100;
const int     Scale=10000;
const int     Remove_base=0;
const int     r=1;              //for penalty
int           Limit_of_neibour,Embedding_dimension;
int           N, Lag, DimensionLag, Period;
float         *Mean, *d;
int           *nrot;
float         **Q=NULL, **Covariance=NULL, **v=NULL;
nt            test_variable, R[Max_embedding_dimension];
void          Principal_Component_Transform(short [],short [],short [],int,long int [],int [], float [], int);

oid main()
{
    int       Size_of_neibourhood, NeighbourhoodSize[100];
    short     *x, *y, *x1, *x2;
    int       *neighbours;
    long      *m_Lag;
    short     embedding_space[Max_embedding_dimension][Max_embedding_dimension];
    int       i,j, k,m,iterate, IterateTimes;
    float     Data_String, center_point[Max_embedding_dimension];
    char      DataFile[80], InputData[80], OutputData[80]={""};

    ifstream fin("c:/home/ee/hbin/ECG/ECGModel/IdealECGDenoising/parameters.dat", ios::nocreate);
    if(!fin){cerr<<"cannot open input file"<<endl; exit(1);}
```

```
fin>>N;
fin>>Embedding_dimension;
fin>>Lag;
fin>>iterate;
fin>>Limit_of_neibour;
fin>>DataFile;
fin>>Data_String; NeighbourhoodSize[0]=int (Scale*Data_String);
fin>>Data_String; NeighbourhoodSize[1]=int (Scale*Data_String);
fin>>Data_String; NeighbourhoodSize[2]=int (Scale*Data_String);
fin>>Data_String; NeighbourhoodSize[3]=int (Scale*Data_String);

x = new short [N];y = new short [N];
x1 = new short [Embedding_dimension];x2 = new short [Embedding_dimension];
neighbours = new int [Limit_of_neibour+2];
m_Lag = new long [Max_embedding_dimension];
Mean=new float[Embedding_dimension];
d =new float[Embedding_dimension];
nrot=new int[200];
Q=(float **)malloc(sizeof(float *)*Embedding_dimension);
Covariance=(float **)malloc(sizeof(float *)*Embedding_dimension);
v=(float **)malloc(sizeof(float *)*Embedding_dimension+1);
for  (m=0;m<Embedding_dimension;m++){
     Q[m]=(float *)malloc(sizeof(float *)*Embedding_dimension);
     Covariance[m]=(float *)malloc(sizeof(float *)*Embedding_dimension);
     v[m]=(float *)malloc(sizeof(float *)*Embedding_dimension+1);
}

for  (i=1;i<Embedding_dimension-1;i++)
     R[i]=1;
R[0]=R[Embedding_dimension-1]=r;

InputData[0]=NULL;strcat(InputData, DataFile);strcat(InputData,".dat");
cout<<"InputData="<<InputData<<endl;

ifstream fin1(InputData, ios::nocreate);
if(!fin1){cerr<<"cannot open InputData"<<endl; exit(1);}
for  (i=0;i<N;i++){
     fin1>>Data_String;
     x[i]=int ((Data_String-Remove_base)*Scale);
}
cout<<"Last data="<<Data_String<<""<<x[N-1]<<endl;

strcat(OutputData, DataFile);
strcat(OutputData,"_index.dat"); cout<<"OutputData="<<OutputData<<endl;

k=(Embedding_dimension/2+1);//*Lag;
for  (m=0;m<Embedding_dimension;m++){
     m_Lag[m]=m*Lag;
}
DimensionLag=Embedding_dimension*Lag;
Period=(Embedding_dimension-1)*Lag;

for  (IterateTimes=0;IterateTimes<iterate;IterateTimes++)
{
```

```
            Size_of_neibourhood=NeighbourhoodSize[IterateTimes];

            for  (i=0;i<Period;i++){
                 Principal_Component_Transform(x,x1,x2,i,m_Lag,neigh-
bours,center_point,Size_of_neibourhood);
                 for  (m=0;m<Embedding_dimension;m++){
                      embedding_space[m][i]=int (center_point[m]);
                 }
                 y[i]=(short) center_point[0];
            }
            for  (i=Period;i<N-Period;i++){
                 Principal_Component_Transform(x,x1,x2,i,m_Lag,neigh-
bours,center_point,Size_of_neibourhood);
                 Data_String=0;j=k=i % Period;
                 for  (m=Embedding_dimension-1;m>0;m--){
                      Data_String+=embedding_space[m][j];
                      j=(j+Lag)%Period;
                 }
                 y[i]=(short) ((Data_String+center_point[0])/(float) Embedding_dimension);
                 for  (m=0;m<Embedding_dimension;m++){
                      embedding_space[m][k]=int (center_point[m]);
                 }
            }
            for  (i=N-Period;i<N;i++){
                 j=i % Period;
                 y[i]=(short) embedding_space[Embedding_dimension-1][j];
            }
            for  (i=0;i<N;i++){
                 x[i]=y[i];
            }
       }
       FILE*fout;
       fout=fopen(OutputData,"w");
       for  (i=0;i<N;i++){
            fprintf(fout,"%f\n",(float) y[i]/(float) Scale);
       }
       fclose(fout);
       cout<<"Denoising has been finished"<<endl;

       delete [] x;delete [] x1;delete [] x2;delete [] y;
       delete [] neighbours;delete [] m_Lag;
       delete [] Mean;
       delete [] d;
       delete [] nrot;
       delete [] Q;
       delete [] Covariance;
       delete [] v;
       if (!v)
            cout<<"Out of memory"<<endl;
}


void Principal_Component_Transform(short x[], short x1[], short x2[], int i1,long int m_Lag[], int neigh-
                                        bours[], float center_point[], int Size_of_neibourhood)
```

```
{
    int  j,m,n,Q_smallest_eigenvalues;

    test_variable=i1;
    int  no_neibour,number_of_neighbours;
    int  embedding_i[Max_embedding_dimension],embedding_j;

    for  (m=0;m<Embedding_dimension;m++){
         embedding_i[m]=i1+m_Lag[m];
         x1[m]=x[embedding_i[m]]-Size_of_neibourhood;
         x2[m]=x[embedding_i[m]]+Size_of_neibourhood;
         if   ((i1==N-1-Period) && (m==Embedding_dimension-1))
         {    cout<<"embedding_i[m]="<<embedding_i[m]<<"x="<<x[embedding_i[m]]<<endl;}
    }

    number_of_neighbours=0;
    for  (j=0;j<N-Period;j++){
         no_neibour=0;
         for  (m=0;m<Embedding_dimension;m++){
              embedding_j=j+m_Lag[m];
              if ((x1[m]>x[embedding_j]) || (x[embedding_j]>x2[m]))// |x-x(i)|>r
              {    no_neibour=1;
                   m=Embedding_dimension;
              }
         }
         if   (no_neibour==0){
              neighbours[number_of_neighbours]=j;//There is a neibour.
              if   (++number_of_neighbours>Limit_of_neibour){
                   j=N;
              }
         }
    }

    //------Get means of embedding space of the neighbourhoods-------;
    for  (m=0;m<Embedding_dimension;m++){
         Mean[m]=0.0;
         for  (j=0;j<number_of_neighbours;j++){
              Mean[m]+=x[neighbours[j]+m_Lag[m]];
         }
         Mean[m]=Mean[m]/number_of_neighbours;
    }
    //------Get covariance of embedding space of the neighbourhoods-------;
    for  (m=0;m<Embedding_dimension;m++){
    for  (n=0;n<Embedding_dimension;n++){
         Covariance[m][n]=0.0;
         for  (j=0;j<number_of_neighbours;j++){
              Covariance[m][n]+=x[neighbours[j]+m_Lag[m]]*x[neighbours[j]+m_Lag[n]];
         }
         Covariance[m][n]=Covariance[m][n]/number_of_neighbours-Mean[m]*Mean[n];
         Covariance[m][n]=R[m]*Covariance[m][n]*R[n];
    }
    }
    jacobi(Covariance,Embedding_dimension,d,v,nrot);
```

```
    eigsrt(d,v,Embedding_dimension);
    Q_smallest_eigenvalues=(int) (Embedding_dimension*0.8);
    for  (m=0;m<Embedding_dimension;m++){
    for  (n=0;n<Embedding_dimension;n++){
         Q[m][n]=0.0;
         for  (j=Embedding_dimension-Q_smallest_eigenvalues;j<Embedding_dimension;j++){
              Q[m][n]+=v[j][m]*v[j][n];//Q[m][n]+=v[m][j]*v[n][j];
         }
    }
    }
    for  (n=0;n<Embedding_dimension;n++){
         center_point[n]=0.0;
         for  (m=0;m<Embedding_dimension;m++){
              center_point[n]+=Q[n][m]*R[m]*(Mean[m]-x[i1+m_Lag[m]]);
         }
         center_point[n]=x[i1+m_Lag[n]]+center_point[n]/R[n];
    }
}
```

Jacobi.h

```
//---------------------------------------------------------------
extern jacobi(float **, int, float [], float **, int *);
extern void eigsrt(float [], float **, int);
//---------------------------------------------------------------
```

Jacobi.cpp

```
//-----------------------------------------------------------------------------------------------------------------
/* W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, Numerical Recipes in C:
The Art of Scientific Computing. NY: New York, Cambridge University, pp. 360-376, 1988.
-----------------------------------------------------------------------------------------------------------------*/
#include <math.h>
#include <malloc.h>
#include <stdio.h>
#include "Jacobi.h"

defineROTATE(a,i,j,k,l)g=a[i][j]; h=a[k][l];a[i][j]=g-s*(h+g*tau);a[k][l]=h+s*(g-h*tau);

void nrerror(char []);

extern jacobi(float **a, int n,float d[], float **v, int *nrot)
/*Computes all eigenvalues and eigenvectors of a real symmetric matrix a[1..n][1..n]. On output, elements of
  a array above the diagonal are destroyed. d[1..n] returns the eigenvalues of a array. v[1..n][1..n] is a matrix
  whose columns contain, on output, the normalized eigenvectors of a array. nrot returns the number of
  Jacobi rotations that were required.*/
{
    const doubleTheshold=0.000001;
    int  j,iq,ip,i;
    floattresh,theta,t,c,g,tau,s,h,*b,*z;
    doublesm;

    b= new float [n];
    z= new float [n];
    for  (ip=0;ip<n;ip++){//Initiall to the identity matrix.
        for  (iq=0;iq<n;iq++)v[ip][iq]=0.0;
        v[ip][ip]=1.0;
    }
    for  (ip=0;ip<n;ip++){
        b[ip]=d[ip]=a[ip][ip];//Initialize b and d to the diagonal of a.
        z[ip]=0.0;//This vector will accumulate terms of the form ta(qp) as
    }
    *nrot=0;

    for  (i=1;i<=50;i++){
        sm=0.0;
        for  (ip=0;ip<n-1;ip++){//Sum off-diagonal elements.
            for  (iq=ip+1;iq<n;iq++)
                sm+=fabs(a[ip][iq]);
        }
        if   (sm < Theshold){
            delete z;
            delete b;
            return(0);
        }
        if   (i < 4)
            tresh=(float) (0.2*sm/(n*n));
        else
            tresh=0.0;
        for  (ip=0;ip<n-1;ip++){
```

```
            for (iq=ip+1;iq<n;iq++){
                g=(float) (100.0*fabs(a[ip][iq]));
                //After four sweeps, skip the rotation if the pff-diagonal element is small.
                if   ( (i > 4) && ((float)(fabs(d[ip]+g)) == (float)(fabs(d[ip])))
                        && ((float)(fabs(d[iq]+g)) == (float)(fabs(d[iq]))) ){
                    a[ip][iq]=0.0;
                }
                else if(fabs(a[ip][iq]) > tresh){
                    h=d[iq]-d[ip];
                    if ((float)(fabs(h)+g) == (float)fabs(h))
                        t=(a[ip][iq])/h;//t=1/(2theta)
                    else {
                        theta=(float) 0.5*h/a[ip][iq];//Equation (11.1.10).
                        t=(float) (1.0/(fabs(theta)+sqrt(1.0+theta*theta)));
                        if(theta < 0.0) t=-t;
                    }
                    c=(float) (1.0/sqrt(1+t*t));
                    s=t*c;
                    tau=(float) (s/(1.0+c));
                    h=t*a[ip][iq];
                    z[ip]-=h;
                    z[iq]+=h;
                    d[ip]-=h;
                    d[iq]+=h;
                    a[ip][iq]=0.0;
                    for  (j=0;j<=ip-1;j++){//Case of rotation 1<j<p.
                        ROTATE(a,j,ip,j,iq)
                    }
                    for  (j=ip+1;j<=iq-1;j++){//Case of rotation p<j<q.
                        ROTATE(a,ip,j,j,iq)
                    }
                    for  (j=iq+1;j<n;j++){//Case of rotation q<j<n.
                        ROTATE(a,ip,j,iq,j)
                    }
                    for  (j=0;j<n;j++){
                        ROTATE(v,j,ip,j,iq)
                    }
                    ++(*nrot);
                }
            }
        }
        for  (ip=0;ip<n;ip++){
            b[ip] += z[ip];
            d[ip]=b[ip];  //Update d with the sum of ta(pq).
            z[ip]=0.0;    //ans reinitialize z.
        }
    }
    nrerror("Too many iterations in routine JACOBI");
}

void eigsrt(float *d, float **v, int n)
/*Given the eigenvalues d[1..n] and eigenvectors v[1..n][1..n], this routine sorts the eigenvalues into
  descending order, and rearranges the columns of v coorespondingly. The method is straight insertion.*/
{
```

```
    int k,j,i;
    floatp;

    for (i=0;i<n;i++){
        p=d[k=i];
        for (j=i+1;j<n;j++){
            if   (d[j] >= p){
                p=d[k=j];
            }
        }
        if   (k != i){
            d[k]=d[i];
            d[i]=p;
            for (j=0;j<n;j++){
                p=v[j][i];
                v[j][i]=v[j][k];
                v[j][k]=p;
            }
        }
    }
}

void nrerror(char error_text[])
//Numerical Recipes standard error handler.
{
    fprintf(stderr,"Numerical Recipes run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
//  return(0);
}
```

```
*********************************************************************************
Name        : NoisyECGWaveletFilter.m
Input       : ECG-ideal-NT.dat, ECGWhiteNoise_, ECGPinkkNoise_,  ECGBrownNoise_,
              ECGBlackNoise_, a[]
Output      : NoisyECGWaveletFilter.dat
Procedure   : denoising a time series in wavelet domain
Date        : 29/Jan./2002
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
*********************************************************************************
clear;

  fid =fopen('\home\ee\hbin\ECG\ECGModel\ECG-ideal-NT.dat', 'r');
  [PureSignal,DataLength]=fscanf(fid, '%f');fclose(fid);
  PureSignal=PureSignal';
  SignalVariance=sqrt(var(PureSignal,1));

  SNR(1)=5;  SNR(2)=10;  SNR(3)=20;
  NoiseTypePath(1,:)='\home\ee\hbin\ECG\ECGModel\IdealECGDenoising\ECGWhiteNoise_';
  NoiseTypePath(2,:)='\home\ee\hbin\ECG\ECGModel\IdealECGDenoising\ECGPinkkNoise_';
  NoiseTypePath(3,:)='\home\ee\hbin\ECG\ECGModel\IdealECGDenoising\ECGBrownNoise_';
  NoiseTypePath(4,:)='\home\ee\hbin\ECG\ECGModel\IdealECGDenoising\ECGBlackNoise_';
  fid1 =fopen('\home\ee\hbin\ECG\ECGModel\IdealECGDenoising\NoisyECGWaveletFilter.dat', 'w');
  for NoiseType=1:4
    fprintf(fid1,'%s\n',NoiseTypePath(NoiseType,:));
    fprintf(fid1,' %s     %s     %s\n','OriginalSNR','SNRGain','SNRRatio');
    for Iterate=1:3
      OriginalSNR=SNR(Iterate);
      NoisyDataFile=strcat(NoiseTypePath(NoiseType,:),num2str(OriginalSNR),'dB.dat')
      fid =fopen(NoisyDataFile, 'r');
      [a,N]=fscanf(fid, '%f');   fclose(fid);     a=a';
      X = wden(a,'sqtwolog','h','sln',4,'bior6.8');
      FilterEvaluate;
      fprintf(fid1,'        %d       %f     %f\n',OriginalSNR,SNRGain,r);
    end
    fprintf(fid1,'\n');
  end
  fclose(fid1);
```

```
%*************************************************************************************
Name        : WienerFilter53.m
Input       : ECG-ideal-NT.dat, ECGWhiteNoise_, ECGPinkkNoise_,  ECGBrownNoise_,
              ECGBlackNoise_, a[]
Output      : WienerFilter.dat
Procedure   : linear Wiener filter with 53 orders
Date        : 02/Feb./2002
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
%*************************************************************************************
clear;

  fid =fopen('\home\ee\hbin\ECG\ECGModel\ECG-ideal-NT.dat', 'r');
  [PureSignal,DataLength]=fscanf(fid, '%f');fclose(fid);
  PureSignal=PureSignal';
  SignalVariance=sqrt(var(PureSignal,1));

  [b,err,res]=gremez(53, [0 0.08 0.12 1],...
  {'taperedresp',[1 1 0 0]}, [1 1]);
  [H,F]=freqz(b,1,DataLength,180);

  SNR(1)=5;  SNR(2)=10;  SNR(3)=20;
  NoiseTypePath(1,:)='\home\ee\hbin\ECG\ECGModel\IdealECGDenoising\ECGWhiteNoise_';
  NoiseTypePath(2,:)='\home\ee\hbin\ECG\ECGModel\IdealECGDenoising\ECGPinkkNoise_';
  NoiseTypePath(3,:)='\home\ee\hbin\ECG\ECGModel\IdealECGDenoising\ECGBrownNoise_';
  NoiseTypePath(4,:)='\home\ee\hbin\ECG\ECGModel\IdealECGDenoising\ECGBlackNoise_';
  fid1 =fopen('\home\ee\hbin\ECG\ECGModel\IdealECGDenoising\WienerFilter.dat', 'w');
  for NoiseType=1:4
    fprintf(fid1,'%s\n',NoiseTypePath(NoiseType,:));
    fprintf(fid1,' %s     %s     %s\n','OriginalSNR','SNRGain','SNRRatio');
    for Iterate=1:3
      OriginalSNR=SNR(Iterate);
      NoisyDataFile=strcat(NoiseTypePath(NoiseType,:),num2str(OriginalSNR),'dB.dat')
      fid =fopen(NoisyDataFile, 'r');
      [a,N]=fscanf(fid, '%f');   fclose(fid);     a=a';

      FFTNoisySignal=fft(a,N);
      for i=1:N
        X1(i)=H(i)*FFTNoisySignal(i);
      end
      X=ifft(X1,N);  clear X1;   X1=2*real(X);   clear X;
      Shift=14;   X([1:N-Shift+1])=X1([Shift:N]);
      DataLength=N-200;

      FilterEvaluate;
      fprintf(fid1,'        %d       %f     %f\n',OriginalSNR,SNRGain,r);
    end
    fprintf(fid1,'\n');
  end
  fclose(fid1);
```

```
**************************************************************************************
Name        : FilterEvaluate.m
Input       : ECG-ideal-NT.dat, parameters.dat
Procedure   : SNR gain and noise reduction factor calculation
Date        : 22/Feb./2002
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**************************************************************************************
ChaoDenoising=1;                    % 1 for Chaos denoising

hold off;     N1=800; N2=1100;
if  ChaoDenoising==1
   fprintf('Now it is chaos denoising');

   fid =fopen('\home\ee\hbin\ECG\ECGModel\ECG-ideal-NT.dat', 'r');
   [PureSignal,DataLength]=fscanf(fid, '%f');fclose(fid);
   PureSignal=PureSignal';
   SignalVariance=sqrt(var(PureSignal,1));

   k=1;
   aa=PureSignal([k:DataLength]); clear PureSignal;  PureSignal=aa;  DataLength=DataLength-k+1;
   plot(PureSignal([N1:N2]));
   FileName='\home\ee\hbin\ECG\ECGModel\IdealECGDenoising\parameters.dat';
   [Param]=textread(FileName,'%s');
   Param1=char(Param(6));
   InputFile=strcat(Param1,'.dat'); %fprintf('InputFile=%s\n',InputFile);
   OutputFile=strcat(Param1,'_index.dat'); %fprintf('OutputFile=%s\n',OutputFile);
   FileNameLength=max(size(Param1));  clear dB;
   if  Param1(FileNameLength-3)=='_'
      dB(1)=Param1(FileNameLength-2);
   else
      dB(1)=Param1(FileNameLength-3);
      dB(2)=Param1(FileNameLength-2);
   end
   OriginalSNR=str2num(dB);
   hold on;

   fid=fopen(OutputFile,'r');   [aa,DataLength] = fscanf(fid,'%f\n'); fclose(fid);
   clear X;   X=aa([1:DataLength],1); X=X';   plot(X([N1:N2]));
   clear aa;clear a;
   fid=fopen(InputFile,'r');   [aa,length] = fscanf(fid,'%f\n'); fclose(fid);
   a=aa([k:length],1);          %+k-1
   a=a';
   plot(a([N1:N2]),'r--');
   clear aa;  aa=X([1:DataLength])-a([1:DataLength]);
else
   plot(X([N1:N2]));
   hold on;
   plot(PureSignal([N1:N2]));
   fprintf('Now it is non-chaos denoising     ');
end
   N1=100; N2=DataLength-100;
   clear EstimatedNoise; clear Noise;
   Noise([1:N2-N1+1])=a([N1:N2])-PureSignal([N1:N2]);
```

```
NoiseVariance=sqrt(var(Noise,1));
EstimatedNoise([1:N2-N1+1])=X([N1:N2])-PureSignal([N1:N2]);
EstimatedNoiseVariance=sqrt(var(EstimatedNoise,1));
EstimatedSNR=20*log10(SignalVariance/EstimatedNoiseVariance);
SNRGain=EstimatedSNR-OriginalSNR;
r=NoiseVariance/EstimatedNoiseVariance;
fprintf('SNRGain=%f    r=%f\n',SNRGain,r);


**********************************************************************************
Name        : ColorNoiseGen.m
Output      : WhiteNoise.dat, PinkNoise.dat, BrownNoise.dat, BlackNoise.dat
Procedure   : coloured noise generator
Date        : 05/Feb./2002
Version     : 1.0
**********************************************************************************
% This program:
% - displays the Fourier transform of the Gaussian white noise to check that Matlab's random number
%   generator is acceptable;
% - generates, displays, and saves white, pink, brown, and black noise and the corresponding power
%   spectrum densities; and
%-----------------------------------------------------------------------------------------------------------------
clear all variables;
fs = 16384-1;                % sampling frequency
t = [0:1/fs:1];              % time scale
tmax = 2000;                 % time slice to graph (tmax < fs)
wgn = randn(size(t));        % generate white Gaussian noise
figure(9);                   % check to see how good the random
wgn_ft = fft(wgn);           % number generation is (by looking
loglog(abs(real(wgn_ft)));   % at its Fourier transform)
xlabel('Frequency'); ylabel('Amplitude');
% Mr. Tom Bruhns from HP provided the following pole / zero
% locations for a white-to-pink noise filter for audio signals.
poles = [.9986823 .9914651 .9580812 .8090598 .2896591]';
zzeros = [.9963594 .9808756 .9097290 .6128445 -.0324723]';

[b,a] = zp2tf(zzeros,poles,1);  % coefficients for the filter
pnk = filter(b,a,wgn);       % generate pink noise
summ = 0;                    % generate brown noise
for i = 1 : length(wgn)      % (by integrating white noise)
  summ = summ + wgn(i); brn(i) = summ;
end % for

summ = 0;                    % generate black noise
for i = 1 : length(pnk)      % (by integrating pink noise)
  summ = summ + pnk(i); blk(i) = summ;
end % for

wgn = wgn/max(abs(wgn));     % normalize amplitudes
pnk = pnk/max(abs(pnk));brn = brn/max(abs(brn));blk = blk/max(abs(blk));

figure(1);                   % display white noise
plot(t(1:tmax)*1000,wgn(1:tmax));
title('White Noise');  xlabel('Time (msec)'); ylabel('Voltage');
```

```
figure(2);                  % display pink noise
plot(t(1:tmax)*1000,pnk(1:tmax));
title('Pink Noise');   xlabel('Time (msec)'); ylabel('Voltage');
figure(3);                  % display brown noise
plot(t(1:tmax)*1000,brn(1:tmax));
title('Brown Noise');  xlabel('Time (msec)'); ylabel('Voltage');
figure(4);                  % display black noise
plot(t(1:tmax)*1000,blk(1:tmax));
title('Black Noise');  xlabel('Time (msec)'); ylabel('Voltage');
% save project_1.mat fs t tmax wgn pnk brn blk; % save data!
x = wgn;                    % white noise
N = length(x);
f = (1/N * (1:N));          % frequency
x_ft = fft(x);              % calculate fast Fourier Transform
x_ft_conj = conj(x_ft);     % calculate conjugate of x_ft
PSD = (x_ft .* x_ft_conj) / N;  % calculate Power Spectrum Density
f_log = log10(f);           % take the log (base 10) of both ararys
PSD_log = log10(PSD);
x1 = 1;    x2 = N;          % min & max values for graphing
figure(5);  loglog(f(x1:x2),PSD(x1:x2));   % plot PSD
xlabel('Frequency'); ylabel('Power Spectrum Density');
x = pnk;                    % pink noise
x_ft = fft(x);              % calculate fast Fourier Transform
x_ft_conj = conj(x_ft);     % calculate conjugate of x_ft
PSD = (x_ft .* x_ft_conj) / N;  % calculate Power Spectrum Density
PSD_log = log10(PSD);
figure(6);  loglog(f(x1:x2),PSD(x1:x2));   % plot PSD
xlabel('Frequency'); ylabel('Power Spectrum Density');
x = brn;                    % brown noise
x_ft = fft(x);              % calculate fast Fourier Transform
x_ft_conj = conj(x_ft);     % calculate conjugate of x_ft
PSD = (x_ft .* x_ft_conj) / N;  % calculate Power Spectrum Density
PSD_log = log10(PSD);

figure(7);  loglog(f(x1:x2),PSD(x1:x2));   % plot PSD
xlabel('Frequency'); ylabel('Power Spectrum Density');
x = blk;                    % black noise
x_ft = fft(x);              % calculate fast Fourier Transform
x_ft_conj = conj(x_ft);     % calculate conjugate of x_ft
PSD = (x_ft .* x_ft_conj) / N;  % calculate Power Spectrum Density
PSD_log = log10(PSD);
figure(8);  loglog(f(x1:x2),PSD(x1:x2));   % plot PSD
xlabel('Frequency'); ylabel('Power Spectrum Density');

%-----Save normanized white, pink, brown, and black noises to----------
%-----WhiteNoise.dat, PinkNoise.dat, BrownNoise.dat, and BlackNoise.dat.---
wgn=wgn-mean(wgn);  Variance=sqrt(var(wgn,1));  wgn=wgn/Variance;
pnk=pnk-mean(pnk);  Variance=sqrt(var(pnk,1));  pnk=pnk/Variance;
brn=brn-mean(brn);  Variance=sqrt(var(brn,1));  brn=brn/Variance;
blk=blk-mean(blk);  Variance=sqrt(var(blk,1));  blk=blk/Variance;
fout =fopen ('WhiteNoise.dat', 'wt');   fprintf(fout,'%f\n', wgn);   fclose(fout);
fout =fopen ('PinkNoise.dat', 'wt');    fprintf(fout,'%f\n', pnk);   fclose(fout);
fout =fopen ('BrownNoise.dat', 'wt');   fprintf(fout,'%f\n', brn);   fclose(fout);
fout =fopen ('BlackNoise.dat', 'wt');   fprintf(fout,'%f\n', blk);   fclose(fout);
```

B.3 Data Compression by the NIFS with Segmentation

B.3.1 NIFS Compression

```
*****************************************************************************
Name        : NIFS.cpp
Input       : UnixOrWindows.dat, IFS_Parameter.dat, IFS255Param.dat
Procedure   : NIFS compression with or without signal partitioning
Date        : 06/May/2001
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
*****************************************************************************
#include <fstream.h>
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

int       RangeSegments;
int       SegStart;
int       SegEnd;
int       MaximalSize;
int       MinimalSize;
int       Size_Ratio;
float     ErrorThreshold;
int       LowestQuantizer, HighestQuantizer;

const int     MaxQuantizationLevel=32768;
const int     NumberofCoeType=6;
const int     MaxNumRangeBlock=500;
const int     MaxNumofResolution=14;
const int     FileLength=216002;

FILE    *fid;
long int  DataLength;
short int *aa, *dimension_index;
float     *InputFile, *InputFile1;
float     *Quantizer,*Quantizer_c,*Quantizer_d,*Quantizer_f;
float     *Quantizer_g,*Quantizer_h,*Quantizer_i;
float     *Interval,*Interval_c,*Interval_d,*Interval_f,*Interval_g,*Interval_h,*Interval_i;
float     *MeanSeg, *VarianceSeg;
int       QuantizerLevel, QuantizerLevel_1;
int       length,i,j,UnixOrWindows;
int       SegmentLength;
int       Start_Position,SegStart1,Length;
int       TotalSegmentCount,*tr,*tr_1,*trlr,*lr,*ld,td,i_td;
int       RangeBlock,RangeBlockPosition,RangeBlockSize;
int       BreakDown,Error2RangeSize,ContractionRatio,DomainStep,i_tr;
int       Segmenting, DomainSegment, kd, td_end, k, *SegmentedPoint, *d_i, *tdld, *complexity;
int       Quantizing;

float     sum_t,sum_t_square,sum_t_newx,sum_newx_square,sum_newx,sum_x_t;
```

```
float     sum_x,sum_x_newx;
float     A_k,E_k,c_k,d_k,f_k,rms,rms1,mean,rms2;
float     Error, BestDomain,DomainLength;
float     AA,EE,dd,cc,ff;
float     Center_c,Distance_c,Center_d,Distance_d,Center_f,Distance_f;
float     Center_g,Distance_g,Center_h,Distance_h,Center_i,Distance_i;
float     x_middle,*x_affine,MeanRangeSize;

char      FileName[100]={""}, Directory[100]={""}, Directory_1[100]={""}, DataFile[100];
char      SaveFile[100];
char      QuantizerName[100], MeanVarFile[100]={""}, SavedCoeName[100], TempCha[100];
//charQuResolution[MaxNumofResolution+1][6];

int       SelectQuantizer;
int       TotalSegments;
float     TotalMeanSegmentLength;
int       TotalRangeBlock, Segment_Number;
int       Start_Position_1;
int       *count;
float     *a,*x,*x1,*t1;

void fun_IFS255();
void fun_Quantizing(float &Output, float Quantizer[],float Interval[],int QuantizerLevel,float Input);
    //Prototype

void main() {
    aa = new short int[FileLength];
    dimension_index = new short int[FileLength];
    InputFile = new float[2*MaxQuantizationLevel];
    InputFile1= new float[2*NumberofCoeType];
    MeanSeg = new float[NumberofCoeType];
    VarianceSeg = new float[NumberofCoeType];
    Quantizer = new float[MaxQuantizationLevel];
    Quantizer_c = new float[MaxQuantizationLevel];
    Quantizer_d = new float[MaxQuantizationLevel];
    Quantizer_f = new float[MaxQuantizationLevel];
    Quantizer_g = new float[MaxQuantizationLevel];
    Quantizer_h = new float[MaxQuantizationLevel];
    Quantizer_i = new float[MaxQuantizationLevel];
    Interval = new float[MaxQuantizationLevel];
    Interval_c = new float[MaxQuantizationLevel];
    Interval_d = new float[MaxQuantizationLevel];
    Interval_f = new float[MaxQuantizationLevel];
    Interval_g = new float[MaxQuantizationLevel];
    Interval_h = new float[MaxQuantizationLevel];
    Interval_i = new float[MaxQuantizationLevel];
    SegmentLength=1024;                    // Define segment length;
    a = new float[SegmentLength+1];
    x = new float[SegmentLength+1];
    x_affine = new float[SegmentLength+1];
    x1 = new float[SegmentLength+1];
    t1 = new float[SegmentLength+1];
    count = new int[SegmentLength+1];
    ld = new int[SegmentLength+1];
```

```
lr = new int[SegmentLength];
tr = new int[SegmentLength+1];
tr_1 = new int[SegmentLength+1];
trlr = new int[SegmentLength+1];
SegmentedPoint = new int[SegmentLength+1];
d_i = new int[SegmentLength+1];
tdld = new int[SegmentLength+1];
complexity = new int[SegmentLength+1];

ifstream fid4("UnixOrWindows.dat",ios::nocreate);
fid4>>UnixOrWindows;
if    (UnixOrWindows==1){
      strcat(Directory,"//home//ee//hbin//Carol//hb//data//");
      strcat(Directory_1,"//home//ee//hbin//Carol//hb//ifs-l//ifs2//data//");
}
else {
      strcat(Directory,"\\Carol\\hb\\data\\");
      strcat(Directory_1,"\\Carol\\hb\\ifs-l\\ifs2\\data\\");
}
strcat(FileName,Directory);strcat(FileName,"IFS_Parameter.dat");
ifstream fid5(FileName,ios::nocreate);
fid5>>RangeSegments;
fid5>>SegStart;
fid5>>SegEnd;SegEnd+=SegStart;
fid5>>MaximalSize;
fid5>>MinimalSize;
fid5>>LowestQuantizer;
fid5>>HighestQuantizer;
fid5>>ErrorThreshold;
fid5>>ErrorThreshold;
//fid5>>ErrorThreshold;
//fid5>>ErrorThreshold;//for IFS455
cout<<"ErrorThreshold="<<ErrorThreshold<<endl;
Size_Ratio=1+(int) (log10(MaximalSize/MinimalSize)/log10(2));

FileName[0]=NULL;
strcat(FileName,Directory_1);strcat(FileName,"IFS255Param.dat");
ifstream fid5(FileName,ios::nocreate);
fid5>>DataFile;
fid5>>SaveFile;
fid5>>Segmenting;
fid5>>Quantizing;
cout<<"DataFile="<<DataFile<<"SaveFile="<<SaveFile<<"Segmenting="<<;
cout<<Segmenting<<"Quantizing="<<Quantizing<<endl;
if(Quantizing==0){
     LowestQuantizer=7;HighestQuantizer=7;//QuResolution[0]=NULL;
}
strcat(MeanVarFile,Directory_1);strcat(MeanVarFile,DataFile);strcat(MeanVarFile,SaveFile);
strcat(MeanVarFile,"MeanVar.dat");cout<<"MeanVarFile="<<MeanVarFile<<endl;
ifstream fid3(MeanVarFile,ios::nocreate);
if(!fid3){cerr<<"cannot open"<<MeanVarFile<<endl; exit(1);}
length=-1;
while(!fid3.eof()){
     fid3>>InputFile1[++length];
```

```
    }
    length--;
    j=0;
    for (i=0;i<=length;i+=2){
        MeanSeg[j]=InputFile1[i];
        VarianceSeg[j]=InputFile1[i+1];
        j++;
    }
    FileName[0]=NULL;
    strcat(FileName,Directory);
    strcat(FileName,DataFile);strcat(FileName,".dat");cout<<"DataFile="<<FileName<<endl;
    ifstream fid1(FileName,ios::nocreate);
    DataLength=0;
    while(!fid1.eof()){
        fid1>>A_k;aa[++DataLength]=(short int) (1000*A_k);
    }
    DataLength--;
    Length=DataLength;//size(aa);
    TotalSegments=(int) (DataLength/SegmentLength);
    cout<<"SegmentLength="<<SegmentLength<<"Length="<<Length<<"TotalSegments="<<TotalSeg-
ments<<endl;

    //%--------Read the segmented file--------------------------------
    if   (Segmenting==1){
        FileName[0]=NULL;
        strcat(FileName,Directory);strcat(FileName,DataFile);
        strcat(FileName,"_segment.dat");cout<<"Complexity file="<<FileName<<endl;
        ifstream fid5(FileName,ios::nocreate);
        for  (i=1;i<=DataLength;i++)
            fid5>>dimension_index[i];
    }
    DomainStep=1;
    for (SelectQuantizer=LowestQuantizer;SelectQuantizer<=HighestQuantizer;SelectQuantizer++){
        i=(int) pow(2,SelectQuantizer);k=0;//cout<<"i="<<i<<endl;
        while(i/(int) pow(10,++k)){}
        k--;
        for  (j=k;j>=0;j--){
            TempCha[j]=48+i%10; i=i/10;
        }
        TempCha[++k]='\0';
        QuantizerName[0]=NULL;
        strcat(QuantizerName,Directory);strcat(QuantizerName,"Laplacian_");
        strcat(QuantizerName,TempCha);
        strcat(QuantizerName,".txt");cout<<"QuantizerName="<<QuantizerName<<endl;
        SavedCoeName[0]=NULL;strcat(SavedCoeName,Directory_1);strcat(SavedCoeName,DataFile);
        strcat(SavedCoeName,SaveFile);
        if   (Quantizing==1)
            strcat(SavedCoeName,"Q");
        else TempCha[0]=NULL;
        if   (Segmenting==1)
            strcat(SavedCoeName,"Seg");
        strcat(SavedCoeName,"-");strcat(SavedCoeName,TempCha);
        strcat(SavedCoeName,".dat");cout<<"SavedCoeName="<<SavedCoeName<<endl;
        TotalMeanSegmentLength=0;
```

```
            TotalRangeBlock=0;
            Start_Position=1;
            Segment_Number=0;
            fun_IFS255();
        }

}

void fun_IFS255()
{
    fid = fopen(SavedCoeName,"wt");cout<<SavedCoeName<<endl;
    //%--------------ReadMeanVar.m-----------------------
    ifstream fid2(QuantizerName,ios::nocreate);
    QuantizerLevel=-1;
    while (!fid2.eof()){
        fid2>>InputFile[++QuantizerLevel];
    }
    QuantizerLevel--;
    QuantizerLevel=(QuantizerLevel)/2+1;QuantizerLevel_1=QuantizerLevel-1;
    for  (i=0;i<QuantizerLevel;i++){
        Quantizer[i]=InputFile[i];
        Quantizer_c[i]=Quantizer[i]*VarianceSeg[0]+MeanSeg[0];
        Quantizer_d[i]=Quantizer[i]*VarianceSeg[1]+MeanSeg[1];
        Quantizer_f[i]=Quantizer[i]*VarianceSeg[2]+MeanSeg[2];
        Quantizer_g[i]=Quantizer[i]*VarianceSeg[3]+MeanSeg[3];
//      Quantizer_h[i]=Quantizer[i]*VarianceSeg[4]+MeanSeg[4];
//      Quantizer_i[i]=Quantizer[i]*VarianceSeg[5]+MeanSeg[5];
    }
    for  (i=0;i<QuantizerLevel-1;i++){
        Interval[i]=InputFile[i+QuantizerLevel];
        Interval_c[i]=Interval[i]*VarianceSeg[0]+MeanSeg[0];
        Interval_d[i]=Interval[i]*VarianceSeg[1]+MeanSeg[1];
        Interval_f[i]=Interval[i]*VarianceSeg[2]+MeanSeg[2];
        Interval_g[i]=Interval[i]*VarianceSeg[3]+MeanSeg[3];
//      Interval_h[i]=Interval[i]*VarianceSeg[4]+MeanSeg[4];
//      Interval_i[i]=Interval[i]*VarianceSeg[5]+MeanSeg[5];
    }
    int RangeRatio=(int) (Quantizer[QuantizerLevel-1]+0.999);
    Center_c=MeanSeg[0];
    Distance_c=RangeRatio*VarianceSeg[0];
    Center_d=MeanSeg[1];
    Distance_d=RangeRatio*VarianceSeg[1];
    Center_f=MeanSeg[2];
    Distance_f=RangeRatio*VarianceSeg[2];
    Center_g=MeanSeg[3];
    Distance_g=RangeRatio*VarianceSeg[3];
//  Center_h=MeanSeg[4];
//  Distance_h=RangeRatio*VarianceSeg[4];
//  Center_i=MeanSeg[5];
//  Distance_i=RangeRatio*VarianceSeg[5];
//  %-------------------------------------------------------

    float    sum_t1_square,sum_x1_t1,sum_t1,sum_t1_t2,sum_x_t1,sum_x_t2;
    float    sum_x1_square,sum_x1,sum_x1_t2,sum_x_x1;
```

```
    float     sum_t2,sum_x;
    float     sum_t2_square;

    float     a11,a12,a13,a14;
    float     a22,a23,a24;
    float     a33,a34;
    //float   a41,a42,a43,a44,a45,a46;
    //float   a51,a52,a53,a54,a55,a56;
    double    b11,b12,b13;
    double    b21,b22,b23;
    //double b31,b32,b33,b34,b35;
    //double b41,b42,b43,b44,b45;
    //double c11,c12,c13,c14;
    //double c21,c22,c23;
    //double c31,c32,c33,c34;
    //double d11,d12,d13;
    //double d21,d22,d23;
    float     g_k;
    float     gg;

    const int     Poly2=2;
    const int     Poly3=3;
    const int     Poly4=4;
    float         *z2, *t2;
    z2 = new float[SegmentLength+1];
//  z3 = new float[SegmentLength+1];
//  z4 = new float[SegmentLength+1];
    t2 = new float[SegmentLength+1];
//  t3 = new float[SegmentLength+1];
//  t4 = new float[SegmentLength+1];

    for  (i=1;i<=SegmentLength;i++){
        z2[i]=(float) pow(i,Poly2);//z3[i]=(float) pow(i,Poly3);z4[i]=(float) pow(i,Poly4);
    }

    for  (TotalSegmentCount=1;TotalSegmentCount<=TotalSegments;TotalSegmentCount++){
        //%----------Get a segment from time series------
        Start_Position_1=Start_Position-1;;
        for  (i=Start_Position;i<Start_Position+SegmentLength;i++){
            a[i-Start_Position_1]=(float) (aa[i]/1000.);
        }
        Start_Position=Start_Position+SegmentLength;
        Segment_Number=Segment_Number+1;
        //%------------IFS.m-------------------------;
        SegmentLength=SegEnd-SegStart;
        SegStart1=SegStart-1;
        for  (i=1;i<=SegmentLength;i++){
            x[i]=a[i+SegStart1];
        }
        //%--------Get domain blocks according to the complexity--------
        if   (Segmenting==1){
            k=1;complexity[1]=dimension_index[SegStart];
            SegmentedPoint[k]=1;d_i[k]=dimension_index[SegStart];
            for  (i=2;i<=SegmentLength;i++){
```

```
        complexity[i]=dimension_index[i+SegStart1];
        if (d_i[k]!=complexity[i]){
          tdld[k]=i-1;
          k=k+1;SegmentedPoint[k]=i;d_i[k]=complexity[i];
        }
      }
      tdld[k]=SegmentLength;DomainSegment=k;
}
//%-------------------------------------------------------------

RangeBlock=1;RangeBlockPosition=1;
while  (RangeBlockPosition<SegmentLength){

      RangeBlockSize=MaximalSize;
      tr[RangeBlock]=RangeBlockPosition;tr_1[RangeBlock]=tr[RangeBlock]-1;

      for  (i=1;i<=Size_Ratio;i++){
           trlr[RangeBlock]=RangeBlockPosition+RangeBlockSize-1;
           if    (trlr[RangeBlock]<=SegmentLength){
                 break;
           }
           else {
                 RangeBlockSize=RangeBlockSize/2;
           }
      }

      Error=100;BreakDown=0;
      for  (Error2RangeSize=1;Error2RangeSize<=Size_Ratio;Error2RangeSize++){
        trlr[RangeBlock]=RangeBlockPosition+RangeBlockSize-1;
           lr[RangeBlock]=RangeBlockSize;
           A_k=1;
           for  (ContractionRatio=1;ContractionRatio<=3;ContractionRatio++){
                                          //%Corresponding to shrink factor a: 0.5, 0.25, 0.125
                 A_k=A_k/2;
                 ld[RangeBlock]=(int) (lr[RangeBlock]/A_k);

                 //------check the signal segmented or not-----------
                 if    (Segmenting==0)
                       td_end=SegmentLength-ld[RangeBlock];
                 else td_end=DomainSegment;
                 for  (kd=1;kd<=td_end;kd+=DomainStep){
                       if    (Segmenting==0)
                             td=kd;
                       else {
                             td=SegmentedPoint[kd];
                             if    ((td+ld[RangeBlock]-1) > SegmentLength){
                                   break;
                             }
                       }
                       i_td=td-1;
                       j=1;x1[j]=x[td];t1[j]=(float) td; count[j]=1;
                       t2[j]=z2[td];//t3[j]=z3[td];t4[j]=z4[td];
                       for(i=td+1;i<td+ld[RangeBlock];i++){
                       //i1=i-td;
```

```
if    (j==(int) ceil((i-td+1)*A_k)){
      x1[j]=x1[j]+x[i];t1[j]=t1[j]+(float) i;count[j]=count[j]+1;
      t2[j]=t2[j]+z2[i];//t3[j]=t3[j]+z3[i];t4[j]=t4[j]+z4[i];
}
else {
      j++;
      x1[j]=x[i];t1[j]=(float) i;count[j]=1;
      t2[j]=z2[i];//t3[j]=z3[i];t4[j]=z4[i];
}
}
for(j=1;j<=lr[RangeBlock];j++){
      x1[j]=x1[j]/(float) count[j];t1[j]=t1[j]/(float) count[j];
      t2[j]=t2[j]/(float) count[j];
      // t3[j]=t3[j]/(float) count[j]; t4[j]=t4[j]/(float) count[j];
}

sum_t1_square=0;sum_x1_t1=0; sum_t1=0;sum_t1_t2=0;sum_x_t1=0;
sum_x1_square=0;sum_x1=0;sum_x1_t2=0;sum_x_x1=0;
sum_t2=0;sum_x=0;
sum_t2_square=0;sum_x_t2=0;
for  (i=1;i<=lr[RangeBlock];i++){
      i_tr=tr_1[RangeBlock]+i;
      sum_t1_square=sum_t1_square+t1[i]*t1[i];
      sum_x1_t1=sum_x1_t1+x1[i]*t1[i];
      sum_t1=sum_t1+t1[i];
      sum_t1_t2=sum_t1_t2+t1[i]*t2[i];
      sum_x_t1=sum_x_t1+x[i_tr]*t1[i];
      sum_x1_square=sum_x1_square+x1[i]*x1[i];
      sum_x1=sum_x1+x1[i];
      sum_x1_t2=sum_x1_t2+x1[i]*t2[i];
      sum_x_x1=sum_x_x1+x[i_tr]*x1[i];
      sum_t2=sum_t2+t2[i];
      sum_x=sum_x+x[i_tr];
      sum_t2_square=sum_t2_square+t2[i]*t2[i];
      sum_x_t2=sum_x_t2+x[i_tr]*t2[i];
}
a11=sum_t1_square-sum_t1*sum_t1/(float) lr[RangeBlock];
a12=sum_x1_t1-sum_x1*sum_t1/(float) lr[RangeBlock];
a13=sum_t1_t2-sum_t1*sum_t2/(float) lr[RangeBlock];
a14=sum_x_t1-sum_x*sum_t1/(float) lr[RangeBlock];
a22=sum_x1_square-sum_x1*sum_x1/(float) lr[RangeBlock];
a23=sum_x1_t2-sum_x1*sum_t2/(float) lr[RangeBlock];
a24=sum_x_x1-sum_x*sum_x1/(float) lr[RangeBlock];
a33=sum_t2_square-sum_t2*sum_t2/(float) lr[RangeBlock];
a34=sum_x_t2-sum_x*sum_t2/(float) lr[RangeBlock];

b11=a11*a33-a13*a13;b12=a12*a33-a13*a23;b13=a14*a33-a13*a34;
b21=b12;b22=a22*a33-a23*a23;b23=a24*a33-a23*a34;

c_k=(float) ((b12*b23-b13*b22)/(b12*b21-b11*b22));
d_k=(float) ((b13*b21-b11*b23)/(b12*b21-b11*b22));
g_k=(float) ((a14-c_k*a11-d_k*a12)/a13);
f_k=(float)  ((sum_x-c_k*sum_t1-d_k*sum_x1-g_k*sum_t2)/(float)  lr[Range-
    Block]);
```

```
            if(Quantizing==0){
            rms=0;
            }
            else{
            rms=200;
            if   ((fabs(c_k-Center_c)<Distance_c) & (fabs(d_k-Center_d)<Distance_d) &
                 (fabs(f_k-Center_f)<Distance_f)){
            if (fabs(g_k-Center_g)<Distance_g){
                 fun_Quantizing(g_k,Quantizer_g,Interval_g,QuantizerLevel,g_k);
                 d_k=((a14-g_k*a13)*a12-a11*(a24-g_k*a23))/(a12*a12-a11*a22);
                 fun_Quantizing(d_k,Quantizer_d,Interval_d,QuantizerLevel,d_k);
                 c_k=(a14-d_k*a12-g_k*a13)/a11;
                 fun_Quantizing(c_k,Quantizer_c,Interval_c,QuantizerLevel,c_k);
                 rms=0;
            }
            }
            }
            if(Quantizing==1){
            mean=0.;
            for(i=1;i<=lr[RangeBlock];i++){
                  x_affine[i]=c_k*t1[i]+d_k*x1[i]+g_k*t2[i]-x[tr_1[RangeBlock]+i];
            x_middle=x_affine[i]+f_k;
            mean=mean+x_middle;
            }
            mean=mean/(float) lr[RangeBlock];
            f_k=f_k-mean;
            fun_Quantizing(f_k,Quantizer_f,Interval_f,QuantizerLevel,f_k);
            for(i=1;i<=lr[RangeBlock];i++){
            x_middle=x_affine[i]+f_k;
            rms=rms+x_middle*x_middle;
            }
            }
            else{
            for(i=1;i<=lr[RangeBlock];i++){
                  x_affine[i]=c_k*t1[i]+d_k*x1[i]+g_k*t2[i]-x[tr_1[RangeBlock]+i];
            x_middle=x_affine[i]+f_k;
            rms=rms+x_middle*x_middle;
            }
            }
            rms=(float) sqrt(rms/(float) lr[RangeBlock]);

            if(rms<Error){
          Error=rms;
            BestDomain=(float) td;DomainLength=(float) ld[RangeBlock];
         AA=A_k;//E[RangeBlock]=E_k;
            cc=c_k;dd=d_k;ff=f_k;
            gg=g_k;
            if(rms<ErrorThreshold){
BreakDown=1;
      break;
            }
            } //% ifNewError<Error(RangeBlock)
      }       //%kd
      if (BreakDown==1){
```

```
                    break;
                }
            }//          %ContractionRatio

            if   ((rms<ErrorThreshold) | (RangeBlockSize<=MinimalSize)){
                break;
            }
            else {
        RangeBlockSize=RangeBlockSize/2;
            }
        }//              %Error2RangeSize
        RangeBlockPosition=RangeBlockPosition+(int) (DomainLength*AA);

        fprintf(fid,"%5.0f\n", BestDomain);
        fprintf(fid,"%5.0f\n", DomainLength);
        fprintf(fid,"%5.3f\n", AA);
        fprintf(fid,"%14.12f\n", cc);
        fprintf(fid,"%14.11f\n", dd);
        fprintf(fid,"%14.11f\n", ff);
        fprintf(fid,"%14.12f\n", gg);
//      fprintf(fid,"%14.12f\n", hh);
//      fprintf(fid,"%16.14f\n", ii);
        RangeBlock=RangeBlock+1;
    }
    RangeBlock=RangeBlock-1;
    TotalRangeBlock=TotalRangeBlock+RangeBlock;
    MeanRangeSize=(float) SegmentLength/ (float) RangeBlock;
cout<<"TotalSegmentCount="<<TotalSegmentCount<<"MeanRangeSize="<<MeanRangeSize<<"Range-
Block="<<RangeBlock<<endl;
    TotalMeanSegmentLength=TotalMeanSegmentLength+SegmentLength;
    }
    TotalMeanSegmentLength=TotalMeanSegmentLength/(float) TotalRangeBlock;
    cout<<"TotalMeanSegmentLength="<<TotalMeanSegmentLength<<endl;
    fclose(fid);
}


void fun_Quantizing(float &Output, float Quantizer[],float Interval[],int QuantizerLevel,float Input)
{
int  fun_Q_i,Position,Level1;

  QuantizerLevel=QuantizerLevel/2;
  Level1=(int) (log10(QuantizerLevel)/log10(2));
  if  ((Interval[QuantizerLevel-1] <= Input) && (Input <= Interval[QuantizerLevel])){
    Output=Quantizer[QuantizerLevel];
    }
  else if  ((Interval[QuantizerLevel] < Input) && (Input <= Interval[QuantizerLevel+1])){
    Output=Quantizer[QuantizerLevel+1];
    }
  else{
    Position=QuantizerLevel;
    for (fun_Q_i=1;fun_Q_i<=Level1;fun_Q_i++){
      QuantizerLevel=QuantizerLevel/2;
      if  (Input < Interval[Position]){
```

```
            Position=Position-QuantizerLevel;
                }
          else{
            Position=Position+QuantizerLevel;
          }
        }
        if (Input < Interval[Position]){
          Output=Quantizer[Position];
        }
            else {
          Output=Quantizer[Position+1];
        }
    }
return;
}



*****************************************************************************************
Name        : IFSReconstruct.m
Input       : UnixOrWindows.dat, IFS_Parameter.dat
Procedure   : signal reconstruction based on NIFS compression
Date        : 29/June/2001
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
*****************************************************************************************
clear;

fid=fopen('UnixOrWindows.dat','r');
[flag]=fscanf(fid,'%d');
UnixAndWindowsFlag=flag(1);
if   (UnixAndWindowsFlag)
     Directory='/home/ee/hbin/Carol/hb/data/';
     Directory_1='/home/ee/hbin/Carol/hb/ifs-l/ifs2/data/';
else
     Directory='\Carol\hb\data\';
     Directory_1='\Carol\hb\ifs-l\ifs2\data\';
end
FileName=strcat(Directory,'IFS_Parameter.dat');
fid=fopen(FileName,'r');
[IFSParameters,length]=fscanf(fid, '%f');fclose(fid);
LowestQuantizer=IFSParameters(6);
HighestQuantizer=IFSParameters(7);
fprintf('Quantizer1=%d, Quantizer2=%d\n',LowestQuantizer, HighestQuantizer);
FileName=strcat(Directory_1,'IFS255Param.dat');
[Param]=textread(FileName,'%s');
Param1=char(Param(1));
Param2=char(Param(2));
Param3=char(Param(3));
Param4=char(Param(4));
Segmenting=str2num(Param3);    Quantizing=str2num(Param4);
fprintf('Segmenting=%d,    quantizing=%d\n', Segmenting, Quantizing);

DataFile=strcat(Directory,Param1,'.dat');  fprintf('DataFile=%s\n',DataFile);
```

```
fid=fopen(DataFile,'r');
[aa]=fscanf(fid,'%f\n');   aa=aa';

CoeFile=strcat(Directory_1,Param1,Param2);     MeanVarFile=strcat(CoeFile,'MeanVar.dat');
if (Quantizing==1)
  CoeFile=strcat(CoeFile,'Q');
else
  LowestQuantizer=7;
  HighestQuantizer=7;
end
if (Segmenting==1)
  CoeFile=strcat(CoeFile,'Seg');
end
CoeFile=strcat(CoeFile,'-');
SaveFile=strcat(CoeFile,'Re.dat'); fprintf('SaveFile=%s\n',SaveFile);
fout=fopen(SaveFile,'w');
fprintf(fout,'          %s  %s  %s','RMS','TotalEntropy','CompressionRatio');
fprintf(fout,' %s  %s   %s    %s    %s    %s    %s\r\n','DPosi','DLeng','A','c','d','f','g');

DistrName='Laplacian_';
for SelectQuantizer=LowestQuantizer:HighestQuantizer
  QuantizerName=num2str(2.^SelectQuantizer);
  if (Quantizing==1)
    InputFile=strcat(CoeFile,QuantizerName,'.dat'); %fprintf('InputFile=%s\n',InputFile);
  else
    InputFile=strcat(CoeFile,'.dat'); %fprintf('InputFile=%s\n',InputFile);
  end
  fprintf('\nQuantizerLevel=%s,   InputFile=%s\n',QuantizerName, InputFile);
  %-----Get parameters for Quantization----
  QuantizerName=strcat(DistrName,QuantizerName);
  fprintf(fout,'%s\r\n', QuantizerName);
  QuantizerName=strcat(Directory,QuantizerName,'.txt');
  %---------------------------------------
    [RMS,   TotalEntropy,   EntropyofCoe,   CompressionRatio,a]=fun_R255Q(aa,InputFile,Quantiz-
        erName,MeanVarFile);
  fprintf(fout,'          %5.2f%%',RMS);
  fprintf(fout,'   %f',TotalEntropy);
  fprintf(fout,'      %f',CompressionRatio);
  fprintf(fout,'   %5.2f  %5.2f  %5.2f  %5.2f  %5.2f  %5.2f  %5.2f\r\n',EntropyofCoe);
end
fclose(fout);


subroutine:
%-----------------------------------fun_R255Q.m-----------------------------------------------
% Read IFS coefficients given by IFS255.cpp for an complete ECG signal.
% Reconstructing it and calaulate RSM and Compression Ratio (also Entropy).
%
%                        29/06/2001 by Bin Huang
%-----------------------------------------------------------------------------------------------
function    [RMS, TotalEntropy, EntropyofCoe, CompressionRatio,a]=fun_R255Q(aa,InputFile,Quantiz-
              erName,MeanVarFile);
  fid=fopen(InputFile,'r');
  [temp,length] = fscanf(fid,'%f\n'); fclose(fid);
```

```
DimensionNumber=7;
CoeffientLength=length(1,1)/DimensionNumber; %fprintf('CoeffientLegth=%d\n',CoeffientLength);
k=0;
for i=1:DimensionNumber:length
  k=k+1;
  Segment_DomainPosition(k)=temp(i);
  Segment_DomainLength(k)=temp(1+i);
  Segment_a(k)=temp(2+i);
  Segment_c(k)=temp(3+i);
  Segment_d(k)=temp(4+i);
  Segment_f(k)=temp(5+i);
  Segment_g(k)=temp(6+i);
%   Segment_h(k)=temp(7+i);
%   Segment_i(k)=temp(8+i);
end

%----------------Do you want to show coefficient distribution----------
if 0
  Resolution=500;
  hist(Segment_DomainPosition,500);  xlabel('DomainPosition');  pause;
  hist(Segment_DomainLength,500);  xlabel('DomainLength');  pause;
  hist(Segment_a,Resolution);  xlabel('Coefficient a');  pause;
%  hist(Segment_e,Resolution);  xlabel('Coefficient e');  pause;
  hist(Segment_c,Resolution);  xlabel('Coefficient c');  pause;
  hist(Segment_d,Resolution);  xlabel('Coefficient d');  pause;
  hist(Segment_f,Resolution);  xlabel('Coefficient f');  pause;
  hist(Segment_g,Resolution);  xlabel('Coefficient g');  pause;
%   hist(Segment_h,Resolution);  xlabel('Coefficient h');  pause;
%   hist(Segment_i,Resolution);  xlabel('Coefficient i');  pause;
end

%------------Reconstructing the signal segment by segment----------
SegmentCount=0;
SegmentLength=1024;
i=0;
AccumulatedSegLg=0;
forCoePosition=1:CoeffientLength
  i=i+1;
  A(i)=Segment_a(CoePosition);
%E(i)=Segment_e(CoePosition);
  BestDomain(i)=Segment_DomainPosition(CoePosition);
  DomainLength(i)=Segment_DomainLength(CoePosition);
  c(i)=Segment_c(CoePosition);
  d(i)=Segment_d(CoePosition);
  f(i)=Segment_f(CoePosition);
  g(i)=Segment_g(CoePosition);
%  hh(i)=Segment_h(CoePosition);
%  ii(i)=Segment_i(CoePosition);

  AccumulatedSegLg=AccumulatedSegLg+Segment_DomainLength(CoePosition)*Segment_a(CoePosi-
      tion);
  if  (AccumulatedSegLg==SegmentLength)
      RangeBlock=i;
      iterate=100;y([1:SegmentLength])=1;Threshold=0.00001;
```

```
        k=1;   tr(k)=1;   E(k)=floor(tr(k)-ceil(A(k)*BestDomain(k)));
        for k=2:RangeBlock
         tr(k)=A(k-1)*DomainLength(k-1)+tr(k-1);
         E(k)=floor(tr(k)-ceil(A(k)*BestDomain(k)));
        end

      fori=1:iterate
            count([1:SegmentLength])=0;y_index([1:SegmentLength])=0;
            for  k=1:RangeBlock
                 BestDomain_1(k)=BestDomain(k)-1;
                 E1(k)=E(k)+floor(A(k)*BestDomain(k)-0.01);
                 for  j=1:DomainLength(k)
                      j1=j+BestDomain_1(k);
                      t_j=ceil(A(k)*j)+E1(k);
                      count(t_j)=count(t_j)+1;
%                     y_index(t_j)=y_index(t_j)+c(k)*j1+d(k)*y(j1)+f(k);
                      y_index(t_j)=y_index(t_j)+c(k)*j1+d(k)*y(j1)+f(k)+g(k)*(j1.^2);
                 end
            end

            for  j=1:SegmentLength
                 y_index(j)=y_index(j)/count(j);
            end
            ConError=std(y-y_index);
            if   ConError>Threshold
                  y=y_index;
            else
                  break;
            end
       end
       a([SegmentCount*SegmentLength+1:(SegmentCount+1)*SegmentLength])=y([1:Seg-
            mentLength]);
       SegmentCount=SegmentCount+1;     %fprintf('SegmentCount=%d\n', SegmentCount);
       i=0;
       AccumulatedSegLg=0;
  end    %if(AccumulatedSegLg==SegmentLength)
end      %forCoePosition=1:CoeffientLength

if 1
  DataLength=SegmentCount*SegmentLength;  fprintf('DataLength=%d\n', DataLength);
  clf;
  plot(aa([1:DataLength]),'-.');
  hold on;
  plot(a([1:DataLength]));
  xlabel(i);
   a1([1:DataLength])=aa([1:DataLength])-a([1:DataLength]);
   RMS=sqrt(cov(a1)/cov(a))*100;  %fprintf('RMS=%f\n', RMS);
end

fid=fopen(QuantizerName,'r');
[InputFile,length]=fscanf(fid,'%f');   fclose(fid);
QuantizerLevel=(length+1)/2;   QuantizerLevel_1=QuantizerLevel-1;
```

```
  fid=fopen(MeanVarFile,'r');
  [InputFile1,length]=fscanf(fid,'%f');
  fclose(fid);
  j=1;
  for i=1:2:length
    MeanSeg(j)=InputFile1(i);
    VarianceSeg(j)=InputFile1(i+1);
    j=j+1;
  end

  Quantizer=InputFile([1:QuantizerLevel],1); RangeRatio=ceil(max(Quantizer));
  Quantizer_c=Quantizer*VarianceSeg(1)+MeanSeg(1);
  Quantizer_d=Quantizer*VarianceSeg(2)+MeanSeg(2);
  Quantizer_f=Quantizer*VarianceSeg(3)+MeanSeg(3);
  Quantizer_g=Quantizer*VarianceSeg(4)+MeanSeg(4);
%   Quantizer_h=Quantizer*VarianceSeg(5)+MeanSeg(5);
%   Quantizer_i=Quantizer*VarianceSeg(6)+MeanSeg(6);

%------------------------------Entropy.m------------------------------
  clear EntropyofCoe;
  SignalSampling=11; %bits

  [EntropyofCoe(1)]=fun_Entropy(Segment_DomainPosition,SegmentLength);
  [EntropyofCoe(2)]=fun_Entropy(Segment_DomainLength,SegmentLength);
  [EntropyofCoe(3)]=fun_Entropy(Segment_a,SegmentLength);
  [EntropyofCoe(4)]=fun_Entropy_Q(Segment_c,Quantizer_c,QuantizerLevel);
  [EntropyofCoe(5)]=fun_Entropy_Q(Segment_d,Quantizer_d,QuantizerLevel);
  [EntropyofCoe(6)]=fun_Entropy_Q(Segment_f,Quantizer_f,QuantizerLevel);
  [EntropyofCoe(7)]=fun_Entropy_Q(Segment_g,Quantizer_g,QuantizerLevel);
%   [EntropyofCoe(8)]=fun_Entropy_Q(Segment_h,Quantizer_h,QuantizerLevel);
%   [EntropyofCoe(9)]=fun_Entropy_Q(Segment_i,Quantizer_i,QuantizerLevel);
  fprintf('EntropyofCoe=%f  %f  %f  %f  %f  %f  %f\n',EntropyofCoe);

  TotalEntropy=sum(EntropyofCoe);
  CompressionRatio=(DataLength*SignalSampling)/(TotalEntropy*CoeffientLength);
    fprintf('CoeffientLength=%d, TotalEntropy=%f, RMS=%f, CompressionRatio=%f\n',CoeffientLength,
          TotalEntropy, RMS, CompressionRatio);



%------------------------fun_Entropy.m-------------------------
% Calculate entropy for a quantized coefficient.
%
%                  05/06/2001 by Bin Huang
%--------------------------------------------------------------------
function [EntropyofCoe]=fun_Entropy(Coefficiet,Resolution)
  NumofCoe=size(Coefficiet);
  Prob=hist(Coefficiet,Resolution);
  Prob=Prob/NumofCoe(1,2); %ProbabilitySum=sum(Prob)
  EntropyofCoe=0.;   NumofCoe=size(Prob);
  for i=1:NumofCoe(1,2)
    if Prob(i)~=0.
      EntropyofCoe=EntropyofCoe-Prob(i)*log2(Prob(i));
    end
```

```
  end
  EntropyofCoe;


%-----------------------------------fun_Entropy_Q.m-----------------------------------
% Calculate entropy for a quantized coefficient based on probability distribution;
%                              01/07/2001 by Bin Huang
%-------------------------------------------------------------------------------------
function [EntropyofCoe]=fun_Entropy_Q(Coefficient,Quantizer,QuantizerLevel)
  Prob([1:QuantizerLevel])=0;
  NumofCoe=size(Coefficient);
  ProbConst=1./NumofCoe(1,2);
  for j=1:NumofCoe(1,2)
    QuantizerLeveL_2=QuantizerLevel/2;
    if (Coefficient(j)==Quantizer(QuantizerLeveL_2))
      Prob(QuantizerLeveL_2)=Prob(QuantizerLeveL_2)+ProbConst;
    else
      Position=QuantizerLeveL_2;
      for i=1:log2(QuantizerLeveL_2)
        QuantizerLeveL_2=QuantizerLeveL_2/2;
        if Coefficient(j) < Quantizer(Position)
          Position=Position-QuantizerLeveL_2;
        else
          Position=Position+QuantizerLeveL_2;
        end
      end
      Prob(Position)=Prob(Position)+ProbConst;
    end
  end
  EntropyofCoe=0;
  for i=1:QuantizerLevel
    if Prob(i)~=0.
      EntropyofCoe=EntropyofCoe-Prob(i)*log2(Prob(i));
    end
  end
```

B.3.2 Coefficient Quantization

```
*****************************************************************************
Name        : NonuniformQuantizer.cpp
Input       : parameter.dat,
Procedure   : nonuniform quantizer design based on the Gaussian or Laplacian distribution
Date        : 26/June/2001
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
*****************************************************************************
#include <fstream.h>
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#define DefRange 14.

const long int N = 10000000;//max 515000*2;
const long int N1= N/2;
doubleScaleCoe=65535;
int  Gaussian_curve=0;
doublepi=3.1415926;

void main() {

    FILE *fid;
    double  StepSize;
    double  sqrt_2=sqrt(2.), ConCoe;
    double  T_pi;
    double  i, x1,x2,x11,x22,Q_difference, xmiddle;
    long int  ii, j, Mean_interval, initial_position;
    int     flag_p;
    int     QuantizerLevel;
    unsigned short int*Coeffients;
    float   q_now, abs_q_Di, qM, qM1;
    char    FileName[50],FileName1[60]={""};
    char    FileLn[20];
    int     Level_1;
    float   *q, *q1, *x_interval, *x1_interval, *NormalizedQuanta, *NormalizedInterval;

    ifstream fid1("c:\\carol\\hb\\data\\parameter.dat",ios::nocreate);
    if(!fid1){cerr<<"cannot open fid"<<endl; exit(1);}
    fid1>>FileName;fid1>>FileLn;
    strcat(FileName, "_");strcat(FileName, FileLn);
    strcat(FileName1,FileName);
    strcat(FileName, ".txt");
    strcat(FileName1, "_index");strcat(FileName1, ".txt");
    cout<<FileName<<endl;cout<<FileName1<<endl;
    Level_1=atoi(FileLn);cout<<Level_1<<endl;
    q=new float[Level_1+1];q1=new float[Level_1+1];
    x_interval=new float[Level_1+1];x1_interval=new float[Level_1+1];
```

```
    NormalizedQuanta=new float[Level_1+1];NormalizedInterval=new float[Level_1+1];
    cout<<"N1="<<N1<<endl;
    StepSize=(double) (2.*DefRange)/(double) (N);
    cout<<"StepSize="<<StepSize<<endl;
    Coeffients = new unsigned short int [N1+1];
    if(Coeffients==NULL){cout<<"No enough memory!"; exit(1);}
    cout<<"1="<<Level_1<<endl;
    QuantizerLevel=Level_1;
    QuantizerLevel=QuantizerLevel/2;

    j=0;
/*  if (Gaussian_curve)     //Gausian
    {    T_pi=(double) ScaleCoe/sqrt(2*pi);
         for (i=-DefRange;i<DefRange;i=i+StepSize)
         {    Coeffients[j]=(unsigned short int) (exp(-i*i/2)*T_pi);
           j=j+1;
         }
    }
    cout<<"DefRange="<<DefRange<<""<<"StepSize="<<StepSize<<endl;
    if (~Gaussian_curve)    //Laplacian
    {*/ sqrt_2=sqrt(2); ConCoe=(double) ScaleCoe;///sqrt_2;
    cout<<"ScaleCoe="<<ScaleCoe<<"ConCoe="<<ConCoe<<endl;
         for (i=-DefRange;i<=0;i=i+StepSize)
         {    Coeffients[j]=(unsigned short int) (exp(-fabs(i)*sqrt_2)*ConCoe);
              j=j+1;
         }
    //}
    cout<<"j="<<--j<<endl;
    Mean_interval=(long int) (N1/2.26);//(float) QuantizerLevel);//2.3
    Q_difference=100000;
    x11=0.;  x22=1.e-40;ScaleCoe=ScaleCoe/StepSize;
    for  (ii=0;ii<Mean_interval;ii++){
         xmiddle=(double) Coeffients[ii]/ScaleCoe;
         x11=x11+ii*xmiddle;//*StepSize;
         x22=x22+xmiddle;
    }
    cout<<"3="<<Level_1<<endl;

    for (initial_position=Mean_interval;initial_position<(long int)(N1/2.1875);initial_position++){
         xmiddle=(double) Coeffients[initial_position]/ScaleCoe;
         x11=x11+initial_position*xmiddle;//*StepSize;
         x22=x22+xmiddle;
         q[1]=(float) (x11/x22);
         q[2]=(float) (2.*initial_position-q[1]);
         if  (q[1] < q[2]){
              x_interval[1]=(float) (initial_position);
              //%----Find every endpoint and quantum----------;
              for (j=2;j<=QuantizerLevel-1;j++){          //% for each intervals;
                   x1=0.;   x2=1.e-40;//%q_now=0;
                   flag_p=0;
                   for (ii=(int) (x_interval[j-1]+1);ii<N1;ii++){
                        xmiddle=(double) Coeffients[ii]/ScaleCoe;
                        x1=x1+ii*xmiddle;//*StepSize;
                        x2=x2+xmiddle;
```

```
                q_now=(float) (x1/x2);
                if (q[j] <= q_now){                    //% using monotonic increase property of x1/x2;
                    if (fabs(q[j]-((x1-ii*xmiddle)/(x2-xmiddle)))<fabs(q[j]-q_now))
                        x_interval[j]=(float) (ii-1);//*StepSize;
                    else
                        x_interval[j]=(float) (ii);//*StepSize;
                    q[j+1]=(float) (2.*x_interval[j]-q[j]);
                    if (q[j] < q[j+1])
                        flag_p=1;
                    break;
                }
            }
            if (flag_p==0)
                break;
        }

        //%-----Get and compare the last quantum--------;
        if (flag_p==1){
            x1=0.;  x2=1.e-40;
            for (ii=(long int) (x_interval[QuantizerLevel-1]+1);ii<N1;ii++){
                xmiddle=(double) Coeffients[ii]/ScaleCoe;
                x1=x1+ii*xmiddle;//*StepSize;
                x2=x2+xmiddle;
            }
            qM = (float) (x1/x2);
            abs_q_Di = (float) (fabs(qM-q[QuantizerLevel]));
            if (abs_q_Di < Q_difference){
                Q_difference=abs_q_Di;
                for (ii=1;ii<=QuantizerLevel;ii++){
                    q1[ii]=q[ii];
                    x1_interval[ii]=x_interval[ii];
                    qM1=qM;
                }
            }
        }
    }   //%if q(1) < q(2)
}
fid = fopen(FileName1, "w");
for (ii=1;ii<=QuantizerLevel;ii++){
    fprintf(fid,"%f\n", q1[ii]);
}
cout<<"6="<<Level_1<<endl;
for (ii=1;ii<QuantizerLevel;ii++){
    fprintf(fid,"%f\n", x1_interval[ii]);
}
fclose(fid);
cout<<"7="<<Level_1<<endl;
for (ii=1;ii<=QuantizerLevel;ii++){
    NormalizedQuanta[ii]=(float) (q1[ii]*StepSize)-DefRange;
    NormalizedInterval[ii]=(float) (x1_interval[ii]*StepSize)-DefRange;
    NormalizedQuanta[2*QuantizerLevel+1-ii]=-NormalizedQuanta[ii];
}
NormalizedInterval[QuantizerLevel]=0;//only for even quantization level;
for (ii=1;ii<QuantizerLevel;ii++){
```

```
        NormalizedInterval[2*QuantizerLevel-ii]=-NormalizedInterval[ii];
    }
    QuantizerLevel=QuantizerLevel*2;
    fid = fopen(FileName, "w");
    for  (ii=1;ii<=QuantizerLevel;ii++){
        fprintf(fid,"%f\n", NormalizedQuanta[ii]);
    }
    for  (ii=1;ii<QuantizerLevel;ii++){
        fprintf(fid,"%f\n", NormalizedInterval[ii]);
    }
    fclose(fid);
}


**********************************************************************************
Name        : FitCoef.m
Output      : CoeMeanVar.dat
Procedure   : main routine of finding distribution parameters for a data set
Date        : 23/May/2001
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**********************************************************************************
clear;
hist55;
i_CoefficientsInpput=0;
if 1
   VarianceScale=1000;    LengthRatio=20;
   CoefficientsInpput=Segment_c;
   FitCoef1;
end
if 1
   VarianceScale=6; LengthRatio=200;
   CoefficientsInpput=Segment_d;
   FitCoef1;
end
if 1
   VarianceScale=1000000; LengthRatio=40;
   CoefficientsInpput=Segment_f;
   FitCoef1;
end
if 1
   VarianceScale=23;    LengthRatio=800;
   CoefficientsInpput=Segment_g;
   FitCoef1;
end
if 1
   VarianceScale=100;     LengthRatio=7000;
   CoefficientsInpput=Segment_h;
   FitCoef1;
end
if 1
   VarianceScale=600;    LengthRatio=100000;
   CoefficientsInpput=Segment_i;
   FitCoef1;
```

```
end

i_CoefficientsInpput                    %should be 6
fid =fopen ('\carol\hb\data\CoeMeanVar.dat', 'wt');
for i=1:i_CoefficientsInpput
   fprintf(fid,'%10.8f    ', MeanSeg(i));
   fprintf(fid,'%10.8f\n', VarianceSeg(i));
end
fclose(fid);


**********************************************************************************
Name        : FitCoef1.m
Input       :
Output      :
Procedure   : function for finding a mean and variance from a data set
Date        : 23/May/2001
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**********************************************************************************
MaxInput=max(CoefficientsInpput);
MinInput=min(CoefficientsInpput);
N=LengthRatio*(MaxInput-MinInput)
Step1=(MaxInput-MinInput)/N;
CoefficientsSeg=hist(CoefficientsInpput,N);
N1=1000;
%--------Find peak in histogram (Correspond to mean)-----
MaxValue=-1000; MaxValePosition=0;
for i=1:N
   if  CoefficientsSeg(i) > MaxValue
      MaxValue=CoefficientsSeg(i);   MaxValuePosition=i;
   end
end
Mean=Step1*MaxValuePosition+MinInput

SegLength=size(CoefficientsInpput);
Variance=0;
for i=1:SegLength(1,2)
   Variance=Variance+(CoefficientsInpput(i)-Mean).^2;
end
Variance=Variance/SegLength(1,2);
Variance=Variance/VarianceScale;
SqrtVariance=sqrt(Variance);
EstimateWidth=2*sqrt(Variance)
DistributionWidth=floor(EstimateWidth/Step1);
x1_i1=floor(-DistributionWidth+MaxValuePosition);%-MinInput/Step1)
x1_i2=floor(DistributionWidth+MaxValuePosition);%-MinInput/Step1)
for  i=x1_i1:x1_i2
    x1_axis(i)=(i)*Step1+MinInput;
end
%-----curve generating-----------------------------------------------
Gaussian_curve=0;      % Gaussian_curve=1 for Gaussian, 0 for Laplacian;
Step2=2*EstimateWidth/N1;
Beta=Mean;   Alfa=sqrt(2/Variance);
```

```
j=1;
if Gaussian_curve    %Gausian

   Const1=2*Variance; Const2=sqrt(2*pi*Variance);
   j=1;
   for i=-EstimateWidth+Mean:Step2:EstimateWidth+Mean
      Coefficients(j)=exp(-(i-Mean).^2/Const1)/Const2;
      x_axis(j)=i+Step2;
      j=j+1;
   end
   MaxSeg=max(CoefficientsSeg)*Const2
end
if ~Gaussian_curve    %Laplacian
   sqrt_2=sqrt(2);
   for i=-EstimateWidth+Mean:Step2:EstimateWidth+Mean
      Coefficients(j)=(Alfa/2)*exp(-Alfa*abs(i-Beta));
      x_axis(j)=i+Step2;
      j=j+1;
   end
   MaxSeg=max(CoefficientsSeg)/(Alfa/2);
end
hold off;
plot(x_axis,Coefficients);  hold on;
[sum(CoefficientsSeg([x1_i1:x1_i2])),SegLength(1,2),sum(CoefficientsSeg([x1_i1:x1_i2]))/Seg-
Length(1,2)]
CoefficientsSeg=CoefficientsSeg/MaxSeg;
plot(x1_axis([x1_i1:x1_i2]), CoefficientsSeg([x1_i1:x1_i2]));
i_CoefficientsInpput=i_CoefficientsInpput+1;
MeanSeg(i_CoefficientsInpput)=Mean;
VarianceSeg(i_CoefficientsInpput)=SqrtVariance;


*********************************************************************************
Name        : hist55.m
Input       : Coeffie.dat, IFS_Coeffie.dat
Procedure   : function for giving the histogram of the coefficients from NIFS.cpp
Date        : 17/May/2001
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
*********************************************************************************
if 0
   fid =fopen ('\carol\hb\data\Coeffie.dat', 'wt');
   fprintf(fid,'%f\n', Segment_DomainPosition);
   fprintf(fid,'%f\n', Segment_DomainLength);
   fprintf(fid,'%f\n', Segment_a);
   fprintf(fid,'%f\n', Segment_e);
   fprintf(fid,'%f\n', Segment_c);
   fprintf(fid,'%f\n', Segment_d);
   fprintf(fid,'%f\n', Segment_f);
   fprintf(fid,'%f\n', Segment_g);
   fprintf(fid,'%10.8f\n', Segment_h);
   fprintf(fid,'%10.8f\n', Segment_i);
   fclose(fid);
end
```

```
clear temp;
load \carol\hb\data\IFS_Coeffie.dat;
temp=IFS_Coeffie;
length=size(temp);
DimensionNumber=10;
CoeffientLength=length(1,1)/DimensionNumber;
for i=1:CoeffientLength
   Segment_DomainPosition(i)=temp(i);
   Segment_DomainLength(i)=temp(CoeffientLength+i);
   Segment_a(i)=temp(2*CoeffientLength+i);
   Segment_e(i)=temp(3*CoeffientLength+i);
   Segment_c(i)=temp(4*CoeffientLength+i);
   Segment_d(i)=temp(5*CoeffientLength+i);
   Segment_f(i)=temp(6*CoeffientLength+i);
   Segment_g(i)=temp(7*CoeffientLength+i);
   Segment_h(i)=temp(8*CoeffientLength+i);
   Segment_i(i)=temp(9*CoeffientLength+i);
end

if 0
   Resolution=500;
   hist(Segment_DomainPosition,500);  xlabel('DomainPosition');  pause;
   hist(Segment_DomainLength,500);    xlabel('DomainLength');    pause;
   hist(Segment_a,Resolution);   xlabel('Coefficient a');   pause;
   hist(Segment_e,Resolution);   xlabel('Coefficient e');   pause;
   hist(Segment_c,Resolution);   xlabel('Coefficient c');   pause;
   hist(Segment_d,Resolution);   xlabel('Coefficient d');   pause;
   hist(Segment_f,Resolution);   xlabel('Coefficient f');   pause;
   hist(Segment_g,Resolution);   xlabel('Coefficient g');   pause;
   hist(Segment_h,Resolution);   xlabel('Coefficient h');   pause;
   hist(Segment_i,Resolution);   xlabel('Coefficient i');
end


*************************************************************************************
Name        : ReadMeanVar.m
Input       :
Output      :
Procedure   : given distribution parameters to design a nonuniform quantizer
Date        : 18/May/2001
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
*************************************************************************************
clear MeanSeg;  clear VarianceSeg;
fid=fopen(QuantizerName,'r');
[InputFile,length]=fscanf(fid,'%f');   fclose(fid);
QuantizerLevel=(length+1)/2;   QuantizerLevel_1=QuantizerLevel-1;
fid=fopen('\carol\hb\data\CoeMeanVar.dat','r');
[InputFile1,length]=fscanf(fid,'%f');   fclose(fid);
j=1;
for i=1:2:length
   MeanSeg(j)=InputFile1(i);
   VarianceSeg(j)=InputFile1(i+1);
```

```
   j=j+1;
end
MeanSeg
VarianceSeg
Quantizer=InputFile([1:QuantizerLevel],1);  RangeRatio=ceil(max(Quantizer));
Interval=InputFile([QuantizerLevel+1:2*QuantizerLevel-1],1);
Quantizer_c=Quantizer*VarianceSeg(1)+MeanSeg(1);
Interval_c=Interval*VarianceSeg(1)+MeanSeg(1);
Center_c=MeanSeg(1);
Distance_c=RangeRatio*VarianceSeg(1);
Quantizer_d=Quantizer*VarianceSeg(2)+MeanSeg(2);
Interval_d=Interval*VarianceSeg(2)+MeanSeg(2);
Center_d=MeanSeg(2);
Distance_d=RangeRatio*VarianceSeg(2);
Quantizer_f=Quantizer*VarianceSeg(3)+MeanSeg(3);
Interval_f=Interval*VarianceSeg(3)+MeanSeg(3);
Center_f=MeanSeg(3);
Distance_f=RangeRatio*VarianceSeg(3);
Quantizer_g=Quantizer*VarianceSeg(4)+MeanSeg(4);
Interval_g=Interval*VarianceSeg(4)+MeanSeg(4);
Center_g=MeanSeg(4);
Distance_g=RangeRatio*VarianceSeg(4);
Quantizer_h=Quantizer*VarianceSeg(5)+MeanSeg(5);
Interval_h=Interval*VarianceSeg(5)+MeanSeg(5);
Center_h=MeanSeg(5);
Distance_h=RangeRatio*VarianceSeg(5);

Quantizer_i=Quantizer*VarianceSeg(6)+MeanSeg(6);
Interval_i=Interval*VarianceSeg(6)+MeanSeg(6);
Center_i=MeanSeg(6);
Distance_i=RangeRatio*VarianceSeg(6);
```

B.3.3 Signal Segmentation

```
**************************************************************************************
Name        : VFDT_Segment.m
Input       : UnixOrWindows.dat, VDT-Parameter1.dat
Procedure   : function of calculating the VFDT
Date        : 06/Oct./2001
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**************************************************************************************
function VFDT_Segmrnt

clear;
fid=fopen('UnixOrWindows.dat','r');
[flag]=fscanf(fid,'%d');
UnixAndWindowsFlag=flag(1);
if   (UnixAndWindowsFlag)
     Directory='/home/ee/hbin/Carol/hb/data/';
else
     Directory='\Carol\hb\data\';
end
fclose(fid);
fid=fopen('VDT-Parameter1.dat','r');
[InputFile]=fscanf(fid,'%s')
fclose(fid);
fid=fopen('VDT-Parameter2.dat','r');
[window_width]=fscanf(fid,'%d')
fclose(fid);
InputFile=strcat(Directory,InputFile)
OutputFile=strcat(InputFile,'_','segment.dat');
InputFile=strcat(InputFile,'.dat');
fid=fopen(InputFile,'r');[a,length]=fscanf(fid,'%f');fclose(fid);
x=a;count=length
%Window in which variance dimension estimated.
%window_width=32;
HalfWindow=window_width/2;HalfWindow_1=1-HalfWindow;
1=1; K2=10;%starting point and ending point for linear slope region.
K_total=K2-K1+1; % P is the power of 2 which ...
last_location=(count-HalfWindow-K2);
ll=1;E=1;
for L=HalfWindow:last_location
     s1=0;s2=0;s3=0;s4=0;
     for k=K1:K2%(formular: 108-1)
          sum1=0;sum2=0;L1=L+k;
          for n=HalfWindow_1:HalfWindow
               deltB=x(L1+n)-x(L+n); sum1=sum1+deltB*deltB; sum2=sum2+deltB;
          end
          sum2=sum2/window_width;varB3=sum1/window_width-sum2*sum2;
          Y(k)=log2(varB3/window_width);
          X(k)=log2(k);
          s1=s1+X(k)*Y(k);s2=s2+X(k);s3=s3+Y(k);s4=s4+X(k)*X(k);
     end
     H=0.5*(K_total*s1-s2*s3)/(K_total*s4-s2*s2);%(formular: 109-3)
```

```
    dimension(L)=E+1-H;%(formular: 109-6)
end
['VFDT finished']


**********************************************************************************
Name        : segment.m
Procedure   : main routine of partitioning a signal according to complexity measures
Date        : 24/Feb./2001
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**********************************************************************************
if      window_width==64
        Threshold1=1.35;Threshold2=1.71;
elseif  window_width==32
        Threshold1=1.34;Threshold2=1.70;
else
        Threshold1=1.34;Threshold2=1.70;
        ['Attendtion: no proper threshold']
end
for  i=1:last_location
    if      dimension(i)<Threshold1
            dimension_index(i)=0;
    elseif  dimension(i)<Threshold2
            dimension_index(i)=1;
    else
            dimension_index(i)=2;
    end
end
for  segment_width=4:4:16
  PreviousSegmentValue=dimension_index(1);
  CurrentSegmentLength=1;
  fori=2:last_location
    if(dimension_index(i)==dimension_index(i-1))
      CurrentSegmentLength=CurrentSegmentLength+1;
    elseifCurrentSegmentLength<segment_width
      forj=1:CurrentSegmentLength
        dimension_index(i-j)=PreviousSegmentValue;
      end
      PreviousSegmentValue=dimension_index(i-1);
      CurrentSegmentLength=1;
    else
      PreviousSegmentValue=dimension_index(i-1);
      CurrentSegmentLength=1;
    end
  end
end
fid =fopen (OutputFile, 'wt');
fprintf(fid,'%d\n', dimension_index);
fclose(fid);
start_point=10;N=2800;N=N+start_point;
plot(a(start_point:N)-3.5,'-.');hold on;
plot(dimension_index(start_point:N));
```

B.4 Moment-Invariant and Classification of ECG

B.4.1 Moment-Invariant Feature Extraction

```
**************************************************************************************
Name        : moment.m
Input       : a[]
Output      : MOMENT.DAT
Procedure   : main routine for calculating the moment-invariant
Date        : 28/Nov./2001
Version     : 1.1
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**************************************************************************************
%-------------------------------------------------------------------------------------------------
%  Step:
%  1.  Find the first peak in initialization
%  2.  Find second peak to segmant a beat of the ECG
%  3.  Calculate the 7 MIs (call moment_fun.m) by treating the beat
%      as a character image and save the MIs in one line (MOMENT.DAT).
%  4.  Set the second peak as the first peak and jump to Step 2.
%-------------------------------------------------------------------------------------------------
    L=0;
    normal_L=1;
    sp=1;
    fid = fopen('MOMENT.DAT','w');
     SampleFrequency=360;
    No_estimation_of_period=370;%400;
    End_point_addition=50;
    R_S_distance_coe=1.5;
    Q_R_distance_coe=0.33;
    Period_error_limit=5;
    Moment_error_limit=10;

    length=size(a);
    File_length=length(1,1)
    peak1=0;
    for i=1+sp:No_estimation_of_period+sp,
    if   peak1<a(i)
         T0=i;
         peak1=a(i);
         end
    end
    sp=T0+End_point_addition;
while1
  if  T0+No_estimation_of_period>File_length
     break;
  end
  L=L+1;
  peak2=0;
  for i=1+T0+End_point_addition:T0+No_estimation_of_period,%+End_point_addition,
    if  peak2<a(i)
        End_point=i;
```

```
            peak2=a(i);
        end
    end
    clear specimen;
    clear ff;
    for i=1+T0:End_point,
        j=i-T0;
        specimen(j)=a(i);
    end
    specimen=specimen-min(specimen);
    specimen_Length=End_point-T0;
    T0=End_point;
    clear img;
    for j=1:specimen_Length
    for i=1:specimen(j)*100
        img(i,j)=1;
    end
    end
    moment_fun;
    fori=1:7
            fprintf(fid,'%6.4f  ',phi(i));
    end
    fprintf(fid,'%d\r\n',specimen_Length);
end
      fclose(fid);
      L


*************************************************************************************
Name          : moment_fun.m
Procedure     : function for calculating the moment-invariant
Date          : 28/Nov./2001
Version       : 1.1
Designer      : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
*************************************************************************************
    size_img=size(img);
    Ordinate_length=size_img(1,1);
    Abscissa_length=size_img(1,2);
    M00=0;  M01=0;  M10=0;
    for j=1:Ordinate_length
    for i=1:Abscissa_length
        if  img(j,i)~=0
            M00=M00+1;
            M01=M01+j;
            M10=M10+i;
        end
    end
    end
    mean_x=M10/M00;     mean_y=M01/M00;     eta00=M00;
    eta20=0;  eta02=0;  eta11=0;
    eta30=0;  eta03=0;  eta12=0;    eta21=0;
    for j=1:Ordinate_length
    for i=1:Abscissa_length
        if  img(j,i)~=0
```

```
        i_centre=i-mean_x; j_centre=j-mean_y;
        i2=i_centre*i_centre;  j2=j_centre*j_centre;
        i3=i2*i_centre;       j3=j2*j_centre;
        i1j2=i_centre*j2;     i2j1=i2*j_centre;
        eta20=eta20+i2;       eta02=eta02+j2;    eta11=eta11+i_centre*j_centre;
        eta30=eta30+i3;       eta03=eta03+j3;
        eta12=eta12+i1j2;     eta21=eta21+i2j1;
      end
    end
    end
    eta20=eta20/eta00.^2;     eta02=eta02/eta00.^2;     eta11=eta11/eta00.^2;
    eta30=eta30/eta00.^2.5;   eta03=eta03/eta00.^2.5;
    eta12=eta12/eta00.^2.5;   eta21=eta21/eta00.^2.5;
    clear phi;
    phi(1)=eta20+eta02;
    phi(2)=(eta20-eta02).^2+4*eta11.^2;
    phi(3)=(eta30-3*eta12).^2+(eta03-3*eta21).^2;
    phi(4)=(eta30+eta12).^2+(eta03+eta21).^2;
    phi(5)=(eta30-3*eta12)*(eta30+eta12)*((eta30+eta12).^2-3*(eta03+eta21).^2);
    phi(5)=phi(5)+(3*eta21-eta03)*(eta03+eta21)*(3*(eta30+eta12).^2-(eta03+eta21).^2);
    phi(6)=(eta20-eta02)*((eta30+eta12).^2-(eta03+eta21).^2)+4*eta11*(eta30+eta12)*(eta03+eta21);
    phi(7)=(3*eta21-eta03)*(eta30+eta12)*((eta30+eta12).^2-3*(eta21+eta03).^2);
    phi(7)=phi(7)+(3*eta12-eta30)*(eta21+eta03)*(3*(eta30+eta12).^2-(eta21+eta03).^2);
    [L, phi]
%   image(img);pause;
```

B.4.2 Clustering Training Set

```
**************************************************************************************
Name        : ISODATA.m
Input       : MOMENT.DAT
Output      : Moment_cluster.dat
Procedure   : an unsupervised clustering algorithm for the training set
Date        : 16/Dec./2001
Version     : 1.1
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**************************************************************************************
%-------------------------------------------------------------------------------
% ISODATA---Iterative Self-Organizing Data Analysis Techniques
% 2/3 of the data set is used for training and 1/3 for classification
%-------------------------------------------------------------------------------
% x--samples for training
% Di--The dimension of samples
% N--The number of samples
% u--mean of cluster
% I--allowable iterating number
% C--Initial cluster number
% Cd--Cluster number demanded
% TN--Minimum number of samples in a cluster
% Te--Standard error theshold
% Td--Distance theshold between the centres of clusters
% CP--Maximum Clustering pairs
% Sc--Separate coeffieciency

C=1;
load MOMENT.DAT
xx=MOMENT(:,[1:7]);
x1_value=size(xx);
NN=x1_value(1,1);
Dimension=x1_value(1,2);
N=floor(NN*2/3);
x([1:N],:)=xx([1:N],:);
u=mean(x);
break_out=0;

%--Step 2  Allocating all samples to clusters according to |X-uj|<|X-ui|, i=1...C, i<>j
while 1
    Step([3:14])=0;
    separat_occured=0;
    clear samples_in_cluster;  samples_in_cluster([1:C])=0;
    clear new_clusters;
    for  j=1:N
        current_cluster=0;  xj_value=100000;
        for  i=1:C
            xij_sum_square=sum((x(j,:)-u(i,:)).^2);
            if   xij_sum_square <= xj_value
                 current_cluster=i;
                 xj_value=xij_sum_square;
            end
```

```
        end
        samples_in_cluster(current_cluster)=samples_in_cluster(current_cluster)+1;
        new_clusters(current_cluster,samples_in_cluster(current_cluster),[1:Dimension])=x(j,[1:Dimen-
                sion]);
        Tc(j)=current_cluster;
    end
    Class(2)=C;

%--Step 3  If the number of samples in some clusters is too small, removing them.
    k=0;
    clear samples_in_cluster_index;
    clear new_clusters_index;
    for  i=1:C
        if   samples_in_cluster(i) >= TN
            k=k+1;
            samples_in_cluster_index(k)=samples_in_cluster(i);
            new_clusters_index(k,:,:)=new_clusters(i,:,:);
        end
    end
    C=k;
    clear samples_in_cluster;
    clear new_clusters;
    samples_in_cluster=samples_in_cluster_index;
    new_clusters=new_clusters_index;
    if  break_out==1          %Function for Step 14, program out from here!!!!!!!!!!!!!!!1
        k=0;   clear u_index;
        for  i=1:C
            if   samples_in_cluster(i) >= TN
                k=k+1;
                u_index(k,:)=u(i,:);
            end
        end
        C=k;   clear u;   u=u_index;
        for  j=1:N
            current_cluster=0;  xj_value=100000;
            for  i=1:C
                xij_sum_square=sum((x(j,:)-u(i,:)).^2);
                if   xij_sum_square <= xj_value
                    current_cluster=i;
                    xj_value=xij_sum_square;
                end
            end
            Tc(j)=current_cluster;
        end
        break;
    end %if break_out==1

%--Step 4  Renew the centre of the clusters
    clear u;
    Maxmum_cluster_size=size(new_clusters);
    for  i=1:C
     for k=1:Dimension
        u(i,k)=sum(new_clusters(i,:,k))/samples_in_cluster(i);
    end
```

```
    end
%--Step 5  Evaluate average distance between centre to samples in a cluster and total distance
    Total_D=0;
    for  i=1:C
        D(i)=0;
        for  j=1:samples_in_cluster(i)
            x_value=0;
            for  k=1:Dimension   %Di
                x_value=x_value+(new_clusters(i,j,k)-u(i,k)).^2;
            end
            D(i)=D(i)+sqrt(x_value);
        end
        Total_D=Total_D+(D(i));
        D(i)=D(i)/samples_in_cluster(i);
    end
    Total_D=Total_D/N;
%--Step 6  Evaluate total average distance D

%--Step 7  Make decision
    while 1
        if   I==0      %check iteration number
            Td=0;
            break;
        end
        if   C>(Cd/2)%b) isn't satisfactory, check c)
            if   ((C >= 2*Cd) | (2*floor(I/2)==I))   %even
                break;
            end
        end
%--Step 8 and 9: Evaluate standard error and find the maxmum values
        Step(8)=1;  Step(9)=1;  Sigma_i=0;Sigma_j=0;
        clear some_cluster; some_cluster([1:C],[1:Dimension])=0;    %Di
        for  i=1:C
            for  j=1:samples_in_cluster(i)
            for  k=1:Dimension   %(Di)
                some_cluster(i,k)=some_cluster(i,k)+(new_clusters(i,j,k)-u(i,k)).^2;
            end
            end
            some_cluster(i,:)=sqrt(some_cluster(i,:)/samples_in_cluster(i));
        end
        Sigma=max(max(some_cluster));
        break_flag=0;
        for  i=1:C
            for  j=1:Dimension   %(Di)
                if   Sigma==some_cluster(i,j)
                    Sigma_i=i;   Sigma_j=j;break_flag=1;
                    break;
                end
            end
            if   break_flag==1
                break;
                break_flag=0;
            end
        end
```

```
        Class(8)=C;Class(9)=C;

    %--Step 10 Check whether or not adding a new cluster
        C1=C;
        if   Sigma>Te
             Number_Clusters=C;
             for i=1:C
                 if   ((D(i)>=Total_D) & (samples_in_cluster(i)>2*(TN+1))) | (C<=(Cd/2))
                      Number_Clusters=Number_Clusters+1;
                      u(Number_Clusters,:)=u(i,:);
                      u(i,Sigma_j)=u(i,Sigma_j)-Sc*Sigma;
                      u(Number_Clusters,Sigma_j)=u(Number_Clusters,Sigma_j)+Sc*Sigma;
                      separat_occured=1;
                 end
             end
             C=Number_Clusters;
        end
        break;
    end %while  1 at Step 7
%--step 11 Evaluate the distance between clusters
    while 1
        if   separat_occured==1
             break;
        end
        clear d;clear dd;d([1:C],[1:C])=0;  k=0;
        for  i=1:C-1
        for  j=i+1:C
             for  kDim=1:Dimension
                  d(i,j)=d(i,j)+(u(i,kDim)-u(j,kDim)).^2;
             end
             k=k+1;       dd(k)=d(i,j);
        end
        end
         Step(11)=1; Class(11)=C;

%--Step 12 Comparing d(i,j) with Cp
%--Step 13 Combining cluster pairs and evaluating new centres
        dd_length=size(dd);
         if  dd_length(1,2)>Cp
             dd_length(1,2)=Cp;
        end
        Number_Clusters=C;
        for  k=1:dd_length(1,2)
                 if  dd(k)>Td
                      break
                 end
             for  i=1:C-1
             for  j=i+1:C
                 if   dd(k)==d(i,j)
                      u(i,:)=(samples_in_cluster(i)*u(i,:)+samples_in_cluster(j)*u(j,:))/
(samples_in_cluster(i)+samples_in_cluster(j));
                      u(j,1)=100000;      %(Di)
                      Number_Clusters=Number_Clusters-1;
```

```
                    break;
                end
            end
            end
        end

        k=0; clear uu;
        for i=1:C
            if  u(i,1)~=100000
                k=k+1;          uu(k,:)=u(i,:);
            end
        end
        clear u;  u=uu;  C=k;break;
    end  %while 1 at Step 11

%--Step 14 Goto step 2 or end
    I=I-1;
    if  I==0
        break_out=1;   % Stop program at Step 3 for clearing small classes.
    end
end
    if  Display_plot==1
        clear ClusteredTrainingSet;    clear SamplesInClassOfTrainingSet;
        ClusteredTrainingSet=new_clusters;
        SamplesInClassOfTrainingSet=samples_in_cluster;
        ISODATA_plot;
    end
    separat_occured=0;
    clear samples_in_cluster;
    for i=1:C
        samples_in_cluster(i)=0;
    end
    x=xx; clear new_clusters;
    for j=(N+1):NN
        current_cluster=0;xj_value=100000;
        for i=1:C
            xij_sum_square=sum((x(j,:)-u(i,:)).^2);
            if  xij_sum_square <= xj_value
                current_cluster=i;xj_value=xij_sum_square;
            end
        end
        samples_in_cluster(current_cluster)=samples_in_cluster(current_cluster)+1;
        for k=1:Dimension     %Di
            new_clusters(current_cluster,samples_in_cluster(current_cluster),k)=x(j,k);
        end
        Tc(j)=current_cluster;
    end
    fid = fopen('Moment_cluster.dat','w');
    for i=1:NN
        fprintf(fid,'%2d\r\n',Tc(i));
    end
    fclose(fid);
[Cd,C]
```

B.4.3 Classification by PNN

```
**********************************************************************************
Name        : pnn1_error.m
Input       : MOMENT.DAT, MOMENT_CLUSTER.DAT
Output      : MOMENT_CLUSTER1.ERR
Procedure   : PNN training and classification error test
Date        : 10/Dec./2001
Version     : 1.1
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**********************************************************************************
%---------------------------------------------------------------------------------
%2/3 of samples is taken as training set and 1/3 for classification.
%---------------------------------------------------------------------------------

clear P;   clear T;   clear Tc;
load MOMENT.DAT
P1=MOMENT(:,[1:7])';
P11=P1';
x1_value=size(P1);
NN=x1_value(1,2);
N=floor(NN*2/3);
load MOMENT_CLUSTER.DAT
Tc1=MOMENT_CLUSTER(:);
P(:,[1:N])=P1(:,[1:N]);
Tc([1:N])=Tc1([1:N]);
Tc_error=Tc;
error=0;
spread = 0.04;      %0.04;
T = ind2vec(Tc);
net = newpnn(P,T,spread);
for  k=(N+1):NN
     p=P11(k,:)';
     Tcp=Tc1(k);
     classified_point = sim(net,p);
     Tc_error(k)=vec2ind(classified_point);
     if   Tc_error(k)~=Tcp
          error=error+1;
     end
end

error;
NN-N;
Hold_out_error=100*error/(NN-N)
fid = fopen('MOMENT_CLUSTER1.ERR','w');
for  i=1:NN
     fprintf(fid,'%2d\n',Tc_error(i));
end
fclose(fid);
```

B.4.4 Reconstruction and Error

```
**********************************************************************************
Name        : MomentResidual.m
Input       : MOMENT.DAT, MOMENT_CLUSTER1.DAT
Output      :
Procedure   : reconstruction error calculation based on PNN classification and single template reconstruc-
              tion
Date        : 10/Dec./2001
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**********************************************************************************
   PeakMoveAhead=0;      %  15
   clear Sample;  clear SampleNumber;
   load MOMENT.DAT;Tc=MOMENT(:,8);
   load MOMENT_CLUSTER1.ERR;Cc=MOMENT_CLUSTER1(:);
   Cc_max=max(Cc); x1_value=size(Cc);NN=x1_value(1,1);  N=floor(NN*2/3);
   file_length=size(a);
   mean_sqare_Specimen_index=0;sum_square_specimen=0;Root_mean_sqare=0;
   PRD_ori=0; PRD_dif=0;   MPRD_ori=0; NPRD_ori=0;
   cycle_number([1:Cc_max])=0;
   sp=1;
   No_estimation_of_period=370;   %400
   Start_point_addition=50;
   peak1=0;
   for i=1+sp:No_estimation_of_period+sp,
     if peak1<a(i)
     T0=i;
          peak1=a(i);
     end
   end
   T0=T0-PeakMoveAhead;
for position_count=1:N
   clear Specimen;
   Start_point=T0+Tc(position_count);
   for i=1+T0:Start_point,
     Specimen(i-T0)=a(i);
   end
   T0=Start_point;
   Specimen=Specimen-min(Specimen);      %????????????????????????????????
   Specimen=Specimen/max(Specimen);
   cycle_number(Cc(position_count))=cycle_number(Cc(position_count))+1;
   SamplePosition=cycle_number(Cc(position_count));
   SampleNumber(Cc(position_count),SamplePosition)=Tc(position_count);
   Sample(Cc(position_count),SamplePosition,[1:Tc(position_count)])=Specimen([1:Tc(position_count)]);
end
   TT0=T0;
   for i=1:1000
     x_axis(i)=i/360;
   end
for position_count=1+N:NN
   clear Specimen;
   Start_point=T0+Tc(position_count);
```

```
for i=1+T0:Start_point,
  Specimen(i-T0)=a(i);
end
T0=Start_point;
Max_Specimen=max(Specimen);
Min_Specimen=min(Specimen);
MinResidual=100000;
Cluster=Cc(position_count);
Specimen_cycle=Tc(position_count);
for SamplePosition=1:cycle_number(Cluster)
   cluster_average_cycle(Cluster)=SampleNumber(Cluster,SamplePosition);
  cluster_average_waveform(Cluster,[1:SampleNumber(Cluster,SamplePosition)])=Sample(Cluster,Sam
                              plePosition,[1:SampleNumber(Cluster,SamplePosition)]);
   LinearRegistration;
   Specimen_index=Specimen_index*(Max_Specimen-Min_Specimen)+Min_Specimen;
   clear Specimen_index1;
   Specimen_index1=Specimen_index-Specimen;
   MeanDifference=mean(Specimen_index1);
   Specimen_index1=Specimen_index1-MeanDifference;
   Residual=sum(Specimen_index1.^2);
   if  Residual<MinResidual
       MinResidualPosition=SamplePosition;
       MinResidual=Residual;
       clear MinSpecimen_index;
       MinSpecimen_index=Specimen_index;  MinMean=MeanDifference;
   end
end
x11=MinResidual;   x12=sum(Specimen.^2);
x13=sum((Specimen-mean(Specimen)).^2);
mean_sqare_Specimen_index=mean_sqare_Specimen_index+x11;
sum_square_specimen=sum_square_specimen+x12;
Root_mean_sqare=Root_mean_sqare+x11/x12;
PRD_dif=PRD_dif+x11;   %mean of Specimen_index-specimen can be put into mean of specimen;
PRD_ori=PRD_ori+x12;
NPRD_ori=NPRD_ori+sqrt(x11/Specimen_cycle)/(Max_Specimen-Min_Specimen);
MPRD_ori=MPRD_ori+x13;

if plot_display
  set(axes,'fontsize',20);hold on;
  axis([0 1 4.5 6.2]);
  xlabel('Time [s]');
  ylabel('Voltage [mV]');

   Specimen_index=MinSpecimen_index-MinMean;
  plot(x_axis([1:Specimen_cycle]),Specimen_index([1:Specimen_cycle]),'--');
  plot(x_axis([1:Specimen_cycle]),Specimen([1:Specimen_cycle]));
   text(0.05,6.10,sprintf('Frame: %3d',position_count),'fontsize',18);
   text(0.05,5.95,sprintf('Class: %2d  PRD: %4.2f%%%',Cluster,100*sqrt(x11/x12)),'fontsize',18);
   plot(Legend_x,solid_line,'--');
   plot(Legend_x,dashed_line);
   text(beginning_solide+0.05,5.65  ,sprintf(' original'),'fontsize',18);
   text(beginning_solide+0.05,5.80,sprintf(' reconstructed'),'fontsize',18);

   figure(2);
```

```
    set(axes,'fontsize',20);hold on;
    axis([0 1 -0.9 0.9]);
    xlabel('Time [s]');
    ylabel('Voltage [mV]');
    plot(x_axis([1:Specimen_cycle]),Specimen_index1([1:Specimen_cycle]));
    text(beginning_solide+0.05,0.8 ,sprintf('Reconstruction error'),'fontsize',18);
    pause;
    hold off;
  end

  if 0
    text(20,6.45,sprintf('%3d      Class: %d',position_count,Cluster),'fontsize',14);
    text(20,6.35,sprintf('MPRD: %4.1f%%',100*sqrt(x11/x13)),'fontsize',14);
    pause;
    hold off;
  end
end   %for   position_count=1+N:NN
Root_mean_sqare1=100*sqrt(Root_mean_sqare/(NN-N-1))
%Root_mean_sqare=100*sqrt(mean_sqare_Specimen_index/(sum_square_specimen))

PRD= 100*sqrt(PRD_dif/PRD_ori)
NPRD=100*NPRD_ori/(NN-N)
MPRD=100*sqrt(PRD_dif/MPRD_ori)


*******************************************************************************
Name        : recons1_error.m
Input       : MOMENT.DAT, MOMENT_CLUSTER1.DAT
Output      :
Procedure   : reconstruction error calculation based on PNN classification and template-averaged recon-
              struction
Date        : 09/Dec./2001
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
*******************************************************************************
    plot_display=0;
    load MOMENT.DAT;Tc=MOMENT(:,8);
    load MOMENT_CLUSTER1.ERR;Cc=MOMENT_CLUSTER1(:);
    Cc_max=max(Cc); x1_value=size(Cc);NN=x1_value(1,1);  N=floor(NN*2/3);
    file_length=size(a);
    mean_sqare_Specimen_index=0;sum_square_specimen=0;Root_mean_sqare=0;
    PRD_ori=0;  PRD_dif=0;
    cluster_average_cycle([1:Cc_max])=0;
    cycle_number([1:Cc_max])=0;
    for  i=1:N
        cluster_average_cycle(Cc(i))=cluster_average_cycle(Cc(i))+Tc(i);
        cycle_number(Cc(i))=cycle_number(Cc(i))+1;
    end
    for  i=1:Cc_max
        cluster_average_cycle(i)=floor(cluster_average_cycle(i)/cycle_number(i));
    end
    cluster_average_waveform([1:Cc_max],[1:max(cluster_average_cycle)])=0;
    sp=1;
    No_estimation_of_period=370;   %400
```

```
    Start_point_addition=50;
    peak1=0;
    for i=1+sp:No_estimation_of_period+sp,
     if  peak1<a(i)
       T0=i;
            peak1=a(i);
        end
    end
    TT0=T0;

for position_count=1:N
    clear specimen;
    Start_point=T0+Tc(position_count);
    for i=1+T0:Start_point,
        specimen(i-T0)=a(i);
    end
    specimen=specimen-min(specimen);
    T0=Start_point;
    Cluster=Cc(position_count);
    cluster_average_cycle(Cluster);
    Specimen_cycle=Tc(position_count);
    differece_period1=cluster_average_cycle(Cluster)-Specimen_cycle;
    differece_period=abs(differece_period1)+1;
    period_ratio=floor(Specimen_cycle/differece_period);
    clear Specimen_index;
    if   differece_period1<0
        delta=0;
        for  i=1:differece_period-1
            i1=(i-1)*period_ratio;
            for  j=1:period_ratio
                Specimen_index(j+i1-delta)=specimen(j+i1);
            end
            delta=delta+1;
        end
        j1=j+i1;
        j=j+i1-delta;
        for  k=j1:Specimen_cycle
            Specimen_index(j)=specimen(k);
            j=j+1;
        end
   small=0;
   elseifdifferece_period1>0
        delta=0;
        for  i=1:differece_period-1
            i1=(i-1)*period_ratio;
            for  j=1:period_ratio
                Specimen_index(delta+j+i1)=specimen(j+i1);
            end
            delta=delta+1;
            Specimen_index(delta+j+i1)=specimen(j+i1);
        end
        j1=j+i1+delta;
        j=j+i1+1;
        for  k=j:Specimen_cycle
```

```
            j1=j1+1;
            Specimen_index(j1)=specimen(k);
        end
        great=1;
        equal=2;
    else Specimen_index=specimen;
    end

cluster_average_waveform(Cluster,[1:cluster_average_cycle(Cluster)])=cluster_average_waveform(Cluster,
[1:cluster_average_cycle(Cluster)])+Specimen_index([1:cluster_average_cycle(Cluster)]);
end   %for   position_count=1:N

TT0=T0
k=position_count;
  %-------Average waveforms for all samples in a class----------
    for  i=1:Cc_max
        cluster_average_waveform(i,:)=cluster_average_waveform(i,:)/cycle_number(i);
        cluster_average_waveform(i,:)=cluster_average_waveform(i,:)/
        max(cluster_average_waveform(i,:));
    end

for position_count=1+N:NN
    clear specimen;
    Start_point=T0+Tc(position_count);
    for i=1+T0:Start_point,
        specimen(i-T0)=a(i);
    end
    T0=Start_point;
    Cluster=Cc(position_count);
    cluster_average_cycle(Cluster);
    Specimen_cycle=Tc(position_count);
    differece_period1=Specimen_cycle-cluster_average_cycle(Cluster);
    differece_period=abs(differece_period1)+1;
    period_ratio=floor(cluster_average_cycle(Cluster)/differece_period);
    clear Specimen_index;
    if   differece_period1<0
        delta=0;
        for  i=1:differece_period-1
            i1=(i-1)*period_ratio;
            for  j=1:period_ratio
                Specimen_index(j+i1-delta)=cluster_average_waveform(Cluster,j+i1);
            end
            delta=delta+1;
        end
        j1=j+i1;
        j=j+i1-delta;
        for  k=j1:cluster_average_cycle(Cluster)
            Specimen_index(j)=cluster_average_waveform(Cluster,k);
            j=j+1;
        end
        small=0;
    elseifdifferece_period1>0
        delta=0;
        for  i=1:differece_period-1
```

```
            i1=(i-1)*period_ratio;
            for  j=1:period_ratio
                Specimen_index(delta+j+i1)=cluster_average_waveform(Cluster,j+i1);
            end
            delta=delta+1;
            Specimen_index(delta+j+i1)=cluster_average_waveform(Cluster,j+i1);
        end
        j1=j+i1+delta;
        j=j+i1+1;
        for  k=j:cluster_average_cycle(Cluster)
            j1=j1+1;
            Specimen_index(j1)=cluster_average_waveform(Cluster,k);
        end
        great=1;
        equal=2;
    else
        Specimen_index([1:Specimen_cycle])=cluster_average_waveform(Cluster,[1:Specimen_cycle]);
    end
    Specimen_index=Specimen_index*(max(specimen)-min(specimen))+min(specimen);
  if  plot_display
     set(axes,'fontsize',24);hold on;
    axis([0 330 4.5 6.5]);
    xlabel('Time (sample/3ms)');
    ylabel('Voltage (mV)');
    plot(Specimen_index,'--');
    plot(specimen);
  end
  Specimen_index=Specimen_index-specimen;
  x11=sum((Specimen_index-mean(Specimen_index)).^2);
  x12=sum(specimen.^2);
  mean_sqare_Specimen_index=mean_sqare_Specimen_index+x11;
  sum_square_specimen=sum_square_specimen+x12;
  Root_mean_sqare=Root_mean_sqare+x11/x12;
  PRD_dif=PRD_dif+x11;    %mean of Specimen_index-specimen can be put into mean of specimen;
  PRD_ori=PRD_ori+x12;
  if  plot_display
    text(20,6.45,sprintf('%3d      Class: %d',position_count,Cluster),'fontsize',14);
    text(20,6.35,sprintf('PRD: %4.1f%%',100*sqrt(x11/x12)),'fontsize',14);
    pause;
    hold off;
  end
end    %for   position_count=1+N:NN
Root_mean_sqare1=100*sqrt(Root_mean_sqare/(NN-N-1))
%Root_mean_sqare=100*sqrt(mean_sqare_Specimen_index/(sum_square_specimen))
PRD=100*sqrt(PRD_dif/PRD_ori)
```

B.5 DTW Classification and Block Encoding for ECG Frame

B.5.1 DTW Classification for ECG Frame

```
**************************************************************************************
Name        : DTW_Classification.c
Input       : x_101.dat
Output      : classification.dat
Procedure   : DTW classification based on minimal average residual
Date        : 18/Aug./1999
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**************************************************************************************
/*----------Dynamic Time Warping----------------------------------------------------------
---DTW is very sensitive to window_size;
---Move peak point back (Peak_move_ahead);
---Find minimal g(i,j) from adjancent three points;
---Begin from (I,J);
---Do not search forward; and
---G(I,J) and residual are obtained.----------------------------------------------------*/
/*-------------------------------------------------------------------------------------------------------------------
Average Residual is employed to classify ECG frames. Period difference threshold is another criterion to
classify ECG frames. First frame is taken as a template. A threshold is setup to classify ECG frames. If the
threshold condition is not satisfactory, a new class is generated. Therefore, we can train and classify a sys-
tem on-line. We place an input frame into a class with the minimal residual which is lower than the thresh-
old. The frame is extracted by forwarding the time position by Peak_move_ahead points.
-------------------------------------------------------------------------------------------------------------------*/

#include <stdio.h>
#include <sys/ddi.h>
#include <math.h>
#include <iostream.h>
#include <stdlib.h>

int main()
{
    FILE *fin, *fout;
    const int     frame_class=60;
    const int     number_of_frames=760;
    const int     frame_cycle=400;
    const int     mapping_length=500;
    const int     frame_class1=61;
    const int     frame_cycle1=401;
    const int     mapping_length1=501;

    int  file_flag;
    int  window_size,Peak_move_ahead,sp,No_estimation_of_period,non_rational;
    int  pattern_X_length,pattern_Y_length,mapping_function_length;
    int  have_been_classified_flag,Start_point_addition;
    int  last_class,frame_length,frame_number,residual_length,classes;
    long int  i,j,i1,j1,k,ii,jj,ii1,class_i,T0,T00,Start_point_1,Start_point;
    float     R_S_distance_coe,maximal,minimal,maxima_0,A0,specimen_peak;
```

```
    float     fabsf_sum,last_minimal_residual;
    int  classification[number_of_frames-1],period[frame_class1],I_J[3];
    float     templet[frame_class1][frame_cycle1],g[frame_cycle1][frame_cycle1];
    float     *aPtr,aPPtr;
    float     DTW_g[number_of_frames];
    float     pattern_X[frame_cycle1],pattern_Y[frame_cycle1];
    float     pattern_XX1[mapping_length1],pattern_YY1[mapping_length1];
    float     gg[4],residual[mapping_length1];
//%---Successive frame comparison.Shift and normalization of frame.
    window_size=30;//%it is r, required by DTW
    const float residual_threshold=0.01;
    Peak_move_ahead=15;
    //%----------Initial data-----------------------
    sp=1;
    No_estimation_of_period=380;
    Start_point_addition=50;
    R_S_distance_coe=1.5;

    non_rational=1000;
    frame_length=400;
    residual_length=500;
    cout<<endl<<"ECG frame classification by Dynamic Time Warping  1"<<endl;
    for  (i=1;i<=frame_class;i++){
        classification[i]=0;
        period[i]=0;
        for  (j=1;j<=frame_cycle;j++)
        {    templet[i][j]=0;
        }
    }
//%--GET First ECG CYCLE DATA pattern_Y-----------------------------------
//  %-----First, get first peak A0 and its position T0---------
    for  (i=1;i<=400;i++)
    {
        pattern_Y[i]=0;
    }
    fin=fopen("/home/ee/hbin/database/data/x_101.dat","r");
    maximal=0;
    for (i=sp;i<=No_estimation_of_period+sp;i++)
    {
        aPtr=&aPPtr;
        fscanf(fin,"%f", aPtr);
        if    (maximal < *aPtr)
        {    T0=i;
            maximal=*aPtr;
        }
    }
    cout<<"initial maximum="<<maximal<<"   Position="<<T0<<endl;
//  %----Second, get specimen_peak, specimen--
    maximal=0;
    fseek(fin,6*(T0+Start_point_addition-1),SEEK_SET);
    for (i=T0+Start_point_addition;i<=T0+No_estimation_of_period;i++)
    {
        aPtr=&aPPtr;
        fscanf(fin,"%f", aPtr);
```

```
            if (maximal < *aPtr)
            {   Start_point=i;
                maximal=*aPtr;
            }
        }
//      %-------Cut pattern_Y from data file----------
        minimal=non_rational;
        maximal=0;
        T00=T0-Peak_move_ahead;
        Start_point_1=Start_point-Peak_move_ahead;
        fseek(fin,6*(1+T00-1),SEEK_SET);
        for (i=1+T00;i<=Start_point_1;i++)
        {   j=i-T00;
            aPtr=&aPPtr;
            fscanf(fin,"%f", aPtr);
            pattern_Y[j]=*aPtr;
            if  (minimal > pattern_Y[j])
            {   minimal=pattern_Y[j];}
            else
            {   if  (maximal < pattern_Y[j])
                {   maximal=pattern_Y[j];}
            }
        }
        pattern_Y_length=Start_point-T0;
        T0=Start_point;
        maximal=maximal-minimal;
        classes=1;
        for  (i=1;i<=pattern_Y_length;i++)
        {   pattern_Y[i]=(pattern_Y[i]-minimal)/maximal;
            templet[classes][i]=pattern_Y[i];
        }
        frame_number=1;
        classification[frame_number]=classes;
        period[classes]=pattern_Y_length;
/*-----Loop: a. take a frame from data file;
                b. find optimal matching mapping by DTW;
                c. find mapping function.
                d. reconctrution
*/
        frame_number=frame_number+1;

while(1){
//      %-------GET sample cycle pattern_X----------------------------------------
        printf("frame_number=%3d\n",frame_number);
        for  (i=1;i<=frame_length;i++)
        {   pattern_X[i]=0;}
        maximal=0;
        fseek(fin,long (6*(T0+Start_point_addition-1)),SEEK_SET);
        for  (i=T0+Start_point_addition;i<=T0+No_estimation_of_period;i++)
        {   j=i-T0;
            aPtr=&aPPtr;
            file_flag=fscanf(fin,"%f", aPtr);
            pattern_X[j]=*aPtr;
            if  (maximal < pattern_X[j])
```

```
        {   Start_point=i;
            maximal=pattern_X[j];
        }
    }
    minimal=non_rational;
    maximal=0;
    T00=T0-Peak_move_ahead;
    Start_point_1=Start_point-Peak_move_ahead;
    fseek(fin,long (6*(1+T00-1)),SEEK_SET);
    for (i=1+T00;i<=Start_point_1;i++)
    {   j=i-T00;
        aPtr=&aPPtr;
        file_flag=fscanf(fin,"%f", aPtr);
        pattern_X[j]=*aPtr;
        if  (minimal > pattern_X[j])
        {   minimal=pattern_X[j];}
        else
         if  (maximal < pattern_X[j])
        {   maximal=pattern_X[j];}

    }
    if  (file_flag!=1)
        {   cout<<"    Read file end"<<endl;
            break;}       // end
    pattern_X_length=Start_point-T0;
    T0=Start_point;
    maximal=maximal-minimal;
    for  (i=1;i<=pattern_X_length;i++)
    {   pattern_X[i]=(pattern_X[i]-minimal)/maximal;
    }
//  %-------Evaluate frame average residual between current observation and templets
    last_minimal_residual=1000;
    have_been_classified_flag=0;//% invalid
    last_minimal_residual=non_rational;
    for  (class_i=1;class_i<=classes;class_i++)
    {
        for  (i=1;i<=period[class_i];i++)
        {   pattern_Y[i]=templet[class_i][i];}
        pattern_Y_length=period[class_i];
//      %------check period length-----------------------------
        if  (fabsf(pattern_Y_length-pattern_X_length) < window_size)
        {
//      %-------Getting the residual between two succesive patterns by DTW----
            for  (i=1;i<=pattern_X_length;i++){
            for  (j=1;j<=pattern_Y_length;j++)
            {   g[i][j]=0;}
            }
            j=1;                    //%correspond to pattern_Y(j), Y axis
            g[1][1]=2*fabsf(pattern_X[1]-pattern_Y[1]);
            for  (i=2;i<=(j+window_size);i++)
            {   g[i][j]=g[i-1][j]+fabsf(pattern_X[i]-pattern_Y[j]);}
            i=1;                    //% i = pattern_X(i), X axis,
            for  (j=2;j<=(i+window_size);j++)
            {   g[i][j]=g[i][j-1]+fabsf(pattern_X[i]-pattern_Y[j]);}
```

```
g[2][2]=g[1][1]+2*fabsf(pattern_X[1]-pattern_Y[1]);
j=2;
for  (i=3;i<=(j+window_size);i++)
{    gg[2]=g[i-1][j-1]+fabsf(pattern_X[i]-pattern_Y[j]);
     gg[3]=g[i-2][j-1]+2*fabsf(pattern_X[i-1]-pattern_Y[j]);
     g[i][j]=min(gg[2],gg[3])+fabsf(pattern_X[i]-pattern_Y[j]);
}
i=2;
for  (j=3;j<=(i+window_size);j++)
{    gg[1]=g[i-1][j-2]+2*fabsf(pattern_X[i]-pattern_Y[j-1]);
     gg[2]=g[i-1][j-1]+fabsf(pattern_X[i]-pattern_Y[j]);
     g[i][j]=min(gg[1],gg[2])+fabsf(pattern_X[i]-pattern_Y[j]);
}
i=3;
j=3;
while(1)
{    i=i+1;
     if   (i==(j+window_size))
     {    if   (i>pattern_X_length)
          {    i=pattern_X_length;}
          gg[2]=g[i-1][j-1]+fabsf(pattern_X[i]-pattern_Y[j]);
          gg[3]=g[i-2][j-1]+2*fabsf(pattern_X[i-1]-pattern_Y[j]);
          g[i][j]=min(gg[2],gg[3])+fabsf(pattern_X[i]-pattern_Y[j]);
          j=j+1;
          if   (j>pattern_Y_length)
          {    I_J[1]=i;
               I_J[2]=j-1;
               break;
          }
          else
          {    i=j-window_size;
               while(i<2)
               {    i=i+1;}
               if   (i>pattern_X_length)
               {    i=pattern_X_length;
                    gg[2]=g[i-1][j-1]+fabsf(pattern_X[i]-pattern_Y[j]);
                    gg[3]=g[i-2][j-1]+2*fabsf(pattern_X[i-1]-pattern_Y[j]);
                    g[i][j]=min(gg[2],gg[3])+fabsf(pattern_X[i]-pattern_Y[j]);
               }
               else
               {    gg[1]=g[i-1][j-2]+2*fabsf(pattern_X[i]-pattern_Y[j-1]);
                    gg[2]=g[i-1][j-1]+fabsf(pattern_X[i]-pattern_Y[j]);
                    g[i][j]=min(gg[1],gg[2])+fabsf(pattern_X[i]-pattern_Y[j]);
               }
          }
     }
     else
          if   (i>pattern_X_length)
          {
          }
          else
          {    gg[1]=g[i-1][j-2]+2*fabsf(pattern_X[i]-pattern_Y[j-1]);
               gg[2]=g[i-1][j-1]+fabsf(pattern_X[i]-pattern_Y[j]);
               gg[3]=g[i-2][j-1]+2*fabsf(pattern_X[i-1]-pattern_Y[j]);
```

```
                    gg[2]=min(gg[2],gg[3]);
                    g[i][j]=min(gg[1],gg[2])+fabsf(pattern_X[i]-pattern_Y[j]);
                }
            }
            DTW_g[class_i]=100*g[pattern_X_length][pattern_Y_length]/
                        (pattern_X_length+pattern_Y_length);
/*      %---Find match path (mapping function) according to minimal residual path-*/
            for (i=1;i<=pattern_X_length;i++){
            for (j=1;j<=pattern_Y_length;j++){
                if  (g[i][j]==0)
                {   g[i][j]=100;}
            }
            }
            pattern_XX1[1]=pattern_X[1];
            pattern_XX1[2]=pattern_X[2];
            pattern_YY1[1]=pattern_Y[1];
            pattern_YY1[2]=pattern_Y[2];
            ii1=pattern_X_length-1;
            ii=ii1;
            jj=pattern_Y_length-1;
            k=1;
            fabsf_sum=0;
        while (1)
        {   if  ((I_J[1]>=pattern_X_length) & (I_J[2]>=pattern_Y_length))
            {   pattern_XX1[k]=pattern_X[pattern_X_length];
                pattern_YY1[k]=pattern_Y[pattern_Y_length];
                residual[k]=pattern_XX1[k]-pattern_YY1[k];
                fabsf_sum=fabsf_sum+fabsf(residual[k]);
                k=k+1;
                i=pattern_X_length-1;
                j=pattern_Y_length-1;
            }
            else    {break;
            }
            while(1)
            {   if  (window_size < fabsf(i-j))
                {   if  (i < j-window_size)
                    {   j=j-1;}
                    else {i=i-1;
                    }
                }
                else
                {   i1=i-1;
                    j1=j-1;
                    if  ((j1<2) || (i1<2))
                    {   break;}
                    if  ((g[i1][j1]<g[i1][j]) & (g[i1][j1]<g[i][j1]))
                    {   i=i1;
                        j=j1;
                    }
                    else
                    if  ((g[i1][j]<g[i][j]) & (g[i1][j]<g[i][j1]))
                        {   i=i1;}
                    else    {j=j1;
```

```
                    }
                }
                pattern_XX1[k]=pattern_X[i];
                pattern_YY1[k]=pattern_Y[j];
                residual[k]=pattern_XX1[k]-pattern_YY1[k];
                fabsf_sum=fabsf_sum+fabsf(residual[k]);
                k=k+1;
            }
            break;
        }
        mapping_function_length=k-1;
        fabsf_sum=fabsf_sum/mapping_function_length;
        printf("class=%3dX_length=%3dY_length=%3d
            ",class_i,pattern_X_length,pattern_Y_length);
        cout<<"mapping_function_length="<<mapping_function_length<<",    "<<"Average resid-
            ual="<<fabsf_sum<<endl;
//%----Classifying-----------------------------------------------------------
//  %----------Check residual threshold----------
        if  (fabsf_sum < residual_threshold)
//      %---Comparison with minimal residual-----
        {   if  (fabsf_sum < last_minimal_residual)
            {   last_minimal_residual=fabsf_sum;
                last_class=class_i;
                classification[frame_number]=class_i;
                have_been_classified_flag=1;//% valide
                cout<<"                    last_class="<<last_class<<endl;
            }
        }
    }   /*if (fabsf(pattern_Y_length-pattern_X_length) < window_size)
    else    {printf("X_length=%3d    Y_length=%3d    Frame    difference    is    too
                great!\n",pattern_X_length,pattern_Y_length);
    }
}   /* Class_i
printf("frame_number=%3d   5\n",frame_number);
//  %----------Check if the current frame has been classified successfully---
if  (have_been_classified_flag==0)
{   classes=classes+1;
    for (i=1;i<=pattern_X_length;i++)
    {   templet[classes][i]=pattern_X[i];}
        period[classes]=pattern_X_length;
        classification[frame_number]=classes;
        printf("A new class is added.  classes=%3d\n",classes);
}
else {printf("Frame is classified to %3d\n",last_class);
}
frame_number=frame_number+1;
}
    fclose(fin);

//------------------Save classification information---------------------------------
//-------classification.dat:class information-------
//-------period.datclass length------------
//-------templet.datclass series------------
    fout=fopen("classification.dat","w");
```

```
    for  (i=1;i<frame_number;i++)
         fprintf(fout,"%d\n",classification[i]);
    fclose(fout);

    fout=fopen("period.dat","w");
    for  (i=1;i<=classes;i++)
         fprintf(fout,"%d\n",period[i]);
    fclose(fout);

    fout=fopen("templet.dat","w");
    for  (i=1;i<=classes;i++){
    for  (j=1;j<=period[i];j++)
         fprintf(fout,"%f\n",templet[i][j]);
    }
    fclose(fout);

return(0);
}
```

B.5.2 ECG Frame Partitioning

```
%*************************************************************************************
% Name        : h_d_p.m
% Input       : x_101.dat, period.dat, classification.dat, templet.dat
% Procedure   : template partitioning
% Date        : 23/Feb./2002
% Version     : 1.1
% Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
%*************************************************************************************
    clear
%   load /home/ee/hbin/database/data/x_101.dat    for unix
    load \home\ee\hbin\database\data\x_101.dat    %for Windows
    a=x_101(:,1);
    sp=1;
    window_size=30;
    No_estimation_of_period=380;
    Start_point_addition=50;
    Peak_move_ahead=15;
    non_rational=1000;
    load period.dat
    period1=period(:,1);
    load classification.dat
    classification1=classification(:,1);
    load templet.dat
    temporal=templet(:,1);
    middle=size(period);
    classes=middle(1,1);
    k=1;
    for  i=1:classes
        for  j=1:period(i)
            templet1(i,j)=temporal(k);
            k=k+1;
        end
    end

    hold off;
    Position=5;
%-------detect waveform change of pattern_Y-----------------------------------------------
    clear end_point_Y;
    clear location_Y;
    for  class_i=1:classes
        pattern_Y([1:period1(class_i)])=templet1(class_i,[1:period1(class_i)]);
        pattern_Y_length=period1(class_i);
        [location_Y(class_i,:),end_point_Y(class_i,:)]=waveform_detect(pattern_Y,pattern_Y_length);
    end
```

```
**************************************************************************************
Name        : pwln_residual.m
Input       : a[]
Procedure   : residual calculation for piecewise linear segments
Date        : 19/Aug./1999
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**************************************************************************************
%-------------------------------------------------------------------------------------------------------------
%Piecewise linear normalization residual calculation. The classification is prepared by Dynamic Time
%Warping algorithm, DTW_Classification.c. This program calls two functions: waveform_detect.m and
linear_nml.m. %h_d_p.m is used to read data file x101.dat, classification.dat, period.dat, and templet.dat. It
also %extract QRS, T and P wave of classes.
%-------------------------------------------------------------------------------------------------------------
%-- 1. reconstructing class information;
%-- 2. detect QRS, T and P wave of classes;
%-- 3. fetch a frame from samples;
%-- 4. detect QRS, T and P wave of the frame;
%-- 5. piece-wise linear normalization; and
%-- 6. evaluate residual.
%-----------------------------------------------
frame_Position=12;
Yes=1;  No=0;           ShowResidual=Yes;
scale=360;
    %-----First, get first peak A0 and its position T0---------
    for i=sp:No_estimation_of_period+sp
        j=i-sp+1;
        b(j)=a(i);
    end
    maxima=max(b);
    for j=1:No_estimation_of_period
        if b(j) ==maxima
            T0=j+sp-1
            A0=maxima;
        end
    end

%---Loop: a. take a frame from data file; b. find optimal matching mapping by DTW;
%---------c. find mapping function. d. reconctrution-------------------------
cycle_number=1;
while1
    %-------GET sample cycle pattern_X---------------------------------------
    clear c;
    for i=T0+Start_point_addition:T0+No_estimation_of_period,
        j=i-T0;
        c(j)=a(i);
    end
    maxima_0=max(c);
    for j=1:No_estimation_of_period,
        if c(j) ==maxima_0
            Start_point=j+T0;
            specimen_peak=maxima_0;
        end
    end
```

```
    clear pattern_X;
    T00=T0-Peak_move_ahead;
    Start_point_1=Start_point-Peak_move_ahead;
    for i=1+T00:Start_point_1
        j=i-T00;
        pattern_X(j)=a(i);
    end
    pattern_X_length=Start_point-T0;
    T0=Start_point;
    pattern_X=pattern_X-min(pattern_X);
    pattern_X=pattern_X/max(pattern_X);
%---------detect QRS, T and P wave of the frame----------------------------------
    [location_X,end_point_X]=waveform_detect(pattern_X,pattern_X_length);
%---------piece-wise linear normalizating pattern_Y------------------------------
  if (cycle_number >= frame_Position)
    class=classification1(cycle_number)
    clear pattern_Y;
    pattern_Y([1:period1(class)])=templet1(class,[1:period1(class)]);
    pattern_Y_length=period1(class);
    [pattern_X_length,pattern_Y_length]
    series_length=end_point_Y(class,1);
    series([1:series_length])=pattern_Y([1:series_length]);
    [series1]=linear_nml(series_length,series,end_point_X(1));
    series1=series1/max(series1);
    pattern_YY=series1;
    mapping_length=location_X(2)-end_point_X(1);
    series_length=location_Y(class,2)-end_point_Y(class,1);
    series([1:series_length])=pattern_Y([end_point_Y(class,1)+1:location_Y(class,2)]);
    [series1]=linear_nml(series_length,series,mapping_length);
    pattern_YY([1+end_point_X(1):location_X(2)])=series1([1:mapping_length]);
    mapping_length=end_point_X(2)-location_X(2);
    series_length=end_point_Y(class,2)-location_Y(class,2);
    series([1:series_length])=pattern_Y([location_Y(class,2)+1:end_point_Y(class,2)]);
    [series1]=linear_nml(series_length,series,mapping_length);
    pattern_YY([1+location_X(2):end_point_X(2)])=series1([1:mapping_length]);
    mapping_length=location_X(3)-end_point_X(2);
    series_length=location_Y(class,3)-end_point_Y(class,2);
    series([1:series_length])=pattern_Y([end_point_Y(class,2)+1:location_Y(class,3)]);
    [series1]=linear_nml(series_length,series,mapping_length);
    pattern_YY([1+end_point_X(2):location_X(3)])=series1([1:mapping_length]);
    mapping_length=pattern_X_length-location_X(3);
    series_length=pattern_Y_length-location_Y(class,3);
    series([1:series_length])=pattern_Y([location_Y(class,3)+1:pattern_Y_length]);
    [series1]=linear_nml(series_length,series,mapping_length);
    pattern_YY([1+location_X(3):pattern_X_length])=series1([1:mapping_length]);
%---------evaluate residual between pattern_X and pattern_YY---------------------
    clear residual;   residual=pattern_X-pattern_YY;
    NPRD=100*sqrt(sum(residual.^2)/pattern_X_length);
    sum_residual=0;
    for  i=1:pattern_X_length
        sum_residual=sum_residual+abs(residual(i));
    end
    sum_residual=sum_residual/pattern_X_length;
    [max(abs(residual)),NPRD]%,sum_residual]
```

```
set(axes,'fontsize',20);
hold off;
beginning_solid=0.7;
clear Legend_x;
clear solid_line;
clear dashed_line;
solid_line_height=0.77;
dashed_line_height=0.69;
for  i=1:9
     Legend_x(i)=beginning_solid+i/200;
     solid_line(i)=solid_line_height;
     dashed_line(i)=dashed_line_height;
end
clear axis_x;
clear axis_y;
for  i=1:pattern_X_length
     axis_x(i)=i/scale;
end
for  i=1:pattern_Y_length
     axis_y(i)=i/scale;
end
hold off;
plot(axis_x,pattern_X,'--');
hold on;
plot(axis_y,pattern_Y);
xlabel('Time [s]');
ylabel('Normalized Amplitude');
'Original frames';
text(beginning_solid,0.93,sprintf('Frame: %3d',cycle_number),'fontsize',18);
text(beginning_solid,0.85,sprintf('Class:  %3d',class),'fontsize',18);
plot(Legend_x,solid_line,'--');
plot(Legend_x,dashed_line);
text(beginning_solid+0.05,solid_line_height,sprintf('original frame'),'fontsize',18);
text(beginning_solid+0.05,dashed_line_height,sprintf('ECG class '),'fontsize',18);
hold off;
figure(2);
set(axes,'fontsize',20);
plot(axis_x,pattern_X,'--');
hold on;
plot(axis_x,pattern_YY);
xlabel('Time [s]');
ylabel('Normalized Amplitude');
clear solid_line;
clear dashed_line;
clear Legend_x;
solid_line_height=0.69;
dashed_line_height=0.61;
for  i=1:9
     Legend_x(i)=beginning_solid+i/200;
     solid_line(i)=solid_line_height;
     dashed_line(i)=dashed_line_height;
end
text(beginning_solid,0.93,sprintf('Frame: %3d',cycle_number),'fontsize',18);
text(beginning_solid,0.85,sprintf('Class:  %3d',class),'fontsize',18);
```

```
    plot(Legend_x,solid_line,'--');
    plot(Legend_x,dashed_line);
    text(beginning_solid,0.77,sprintf('Error: %4.2f%%%',NPRD),'fontsize',18);
    text(beginning_solid+0.05,solid_line_height,sprintf('original frame'),'fontsize',18);
    text(beginning_solid+0.05,dashed_line_height,sprintf('ECG class '),'fontsize',18);
    'Original frame (dashed line) and reconstruted frame';
    hold off;
    if   ShowResidual==Yes
        figure(3);
        set(axes,'fontsize',20);hold on;
        axis([0 1 -0.2 0.2]);
        xlabel('Time [s]');
        ylabel('Normalized Amplitude');
        plot(axis_x,residual);
        clear Pattern_YYY;  clear residual_1;
        [pattern_YYY]=linear_nml(pattern_Y_length,pattern_Y,pattern_X_length);
        residual_1=pattern_X-pattern_YYY;
        figure(4);
        set(axes,'fontsize',20);hold on;
        axis([0 1 -0.2 0.2]);
        plot(axis_x,residual_1);
        xlabel('Time [s]');
        ylabel('Normalized Amplitude');
        NPRD_1=sqrt(sum(residual_1.^2)/pattern_X_length)*100;
        [max(abs(residual_1)),NPRD_1]
        pause;
    end
 end
cycle_number=cycle_number+1
end


**********************************************************************************
Name        : waveform_detect.m
Procedure   : function of detecting waveforms in variance domain
Date        : 19/Aug./1999
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**********************************************************************************
%   ECG frame waveform (QRS, T, P wave) detection function using windowed variance technique.

function [location,end_point]= waveform_detect(pattern,pattern_length)

    %---------Evaluate windowed-variance of pattern-X for its full length---
    window_size=28;
    Threshold=0.018;%for variance
    Threshold_QRS=0.2;
    half_window=window_size/2;
    width_of_subwave=15;
    clear windowed_var;
    for  i=1:half_window
        clear window;
        for  j=1:i+half_window
            window(j)=pattern(j);
```

```
    end
    windowed_var(i)=std(window);
end
for i=1+half_window:pattern_length-half_window-1
    clear window;
    i1=-i+half_window+1;
    for j=i-half_window:i+half_window
        window(j+i1)=pattern(j);
    end
    windowed_var(i)=std(window);
end
for i=pattern_length-half_window:pattern_length
    clear window;
    i1=-i+half_window+1;
    for j=i-half_window:pattern_length
        window(j+i1)=pattern(j);
    end
    windowed_var(i)=std(window);
end
clear wave_var;
width=0;
clear location;
clear end_point;
high_level_flag_1=0;% low level
%------------------Detect QRS complex--------------
for i=1:pattern_length
    if  windowed_var(i)> Threshold_QRS
        high_level_flag_1=1;
    elseifhigh_level_flag_1~=0
        locate_number=1;
        end_point(1)=i;%-back_bias_of_first_point;
        location(locate_number)=i;
        %[locate_number,location(locate_number),end_point(locate_number)];
        width=0;
        break;
    end
end
%-----------------Detect T wave and P wave-----------
windowed_var(pattern_length+1)=0;
for i=location(1):pattern_length+1
    if  windowed_var(i)> Threshold
        width=width+1;
    elseifwidth > width_of_subwave
        if  i-width-end_point(locate_number)>width_of_subwave
            locate_number=locate_number+1;
            location(locate_number)=i-width;
        end
        end_point(locate_number)=i;
        %[locate_number,location(locate_number),end_point(locate_number)]
        width=0;
    else width=0;
    end
end
if  (locate_number==1)
```

```
        location(2)=pattern_length-1;
        end_point(2)=pattern_length;
        location(3)=pattern_length-1;
        end_point(3)=pattern_length;
    elseif(locate_number==2)
        if   (end_point(2) > pattern_length)
            end_point(2)=pattern_length;
        end
        location(3)=pattern_length-1;
        end_point(3)=pattern_length;
    end


**********************************************************************************
Name        : linear_nml.m
Procedure   : function of linear contracting/dilating a segment
Date        : 19/Aug./1999
Version     : 1.0
Designer    : Bin Huang; Tel: 4746992, e-mail: hbin@ee.umanitoba.ca
**********************************************************************************
function [Reconstructed_series]= linear_nml(series_length,series,mapping_length);

    %---Reconstruction: linear transform series to series1
    while1
        differece_period1=mapping_length-series_length;
        differece_period=abs(differece_period1)+1;
        period_ratio=floor(series_length/differece_period);
        if   (period_ratio < 2) & (differece_period1 < 0)
            clear series_index;
            j=floor(series_length/2);
            for  i=1:j
                series_index(i)=series(2*i);
            end
            clear series;
            series=series_index;
            series_length=j
        elseif(period_ratio < 2) & (differece_period1 > 0)
            clear series_index;
            for  i=1:series_length
                series_index(2*i)=series(i);
            end
            series_index(1)=series_index(2);
            for  i=2:series_length
                series_index(2*i-1)=(series_index(2*i-1)+series_index(2*i-2))/2;
            end
            clear series;
            series=series_index;
            series_length=2*series_length;
        else break;
        end
    end
    clear Reconstructed_series;
    if   differece_period1<0
    %------delete points---------
```

```
    delta=0;
    for  i=1:differece_period-1
        i1=(i-1)*period_ratio;
        if   delta~=0
            Reconstructed_series(1+i1-delta)=(series(1+i1)+Reconstructed_series(1+i1-delta))/2;
        else
            Reconstructed_series(1+i1-delta)=series(1+i1);
        end
        for  j=2:period_ratio
            Reconstructed_series(j+i1-delta)=series(j+i1);
        end
        delta=delta+1;
    end
    j1=j+i1;
    j=j+i1-delta;
    if   differece_period~=2
        for  k=j1:series_length
            Reconstructed_series(j)=series(k);
            j=j+1;
        end

    else
        Reconstructed_series(j)=(Reconstructed_series(j)+series(j1))/2;
        j=j+1;
        for  k=j1+1:series_length
            Reconstructed_series(j)=series(k);
            j=j+1;
        end
    end
elseifdifferece_period1>0
    delta=0;
    for  i=1:differece_period-1
        i1=(i-1)*period_ratio;
        if   delta~=0
            Reconstructed_series(delta+1+i1)=(series(1+i1)+series(2+i1))/2;
        else
            Reconstructed_series(delta+1+i1)=series(1+i1);
        end
        for  j=1:period_ratio
            Reconstructed_series(delta+j+i1)=series(j+i1);
        end
        delta=delta+1;
        Reconstructed_series(delta+j+i1)=series(j+i1);
    end
    j1=j+i1+delta;
    j=j+i1+1;
    if   differece_period~=2
        for  k=j:series_length
            j1=j1+1;
            Reconstructed_series(j1)=series(k);
        end
    else
        j1=j1+1;
        Reconstructed_series(j1)=(series(j)+series(j+1))/2;
```

```
        for  k=j+1:series_length
             j1=j1+1;
             Reconstructed_series(j1)=series(k);
        end
    end
else Reconstructed_series([1:series_length])=series([1:series_length]);
end
```